Revisit Two Memoryless State-Recovery Cryptanalysis Methods on A5/1

Yanbin Xu[2], Yonglin Hao[1⋆], and Mingxing Wang[1,3]

[1] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[2] College of Computer Science, Sichuan University, Chengdu 610065, China
[3] The 6th Research Institute of China Electronics Corporation, Beijing 100083, China

Abstract. At ASIACRYPT 2019, Zhang proposed a near collision attack on A5/1 claiming to recover the 64-bit A5/1 state with a time complexity around 2^{32} cipher ticks with negligible memory requirements. Soon after its proposal, Zhang's near collision attack was severely challenged by Derbez et al. who claimed that Zhang's attack cannot have a time complexity lower than Golic's memoryless guess-and-determine attack dating back to EUROCRYPT 1997. In this paper, we study both the guess-and-determine and the near collision attacks for recovering A5/1 states with negligible memory complexities. Firstly, we propose a new guessing technique called the *move guessing technique* that can construct linear equation filters in a more efficient manner. Such a technique can be applied to both guess-and-determine and collision attacks for efficiency improvements. Secondly, we take the filtering strength of the linear equation systems into account for complexity analysis. Such filtering strength are evaluated with practical experiments making the complexities more convincing. Based on such new techniques, we are able to give 2 new guess-and-determine attacks on A5/1: the 1st attack recovers the internal state $\boldsymbol{s}^0$ with time complexity $2^{43.92}$; the 2nd one recovers a different state $\boldsymbol{s}^1$ with complexity $2^{43.25}$. We also revisit Golic's guess-and-determine attack and Zhang's near collision attacks. According to our detailed analysis, the complexity of Golic's $\boldsymbol{s}^1$ recovery attack is no lower than $2^{46.04}$, higher than the previously believed 2^{43}. On the other hand, Zhang's near collision attack recovers $\boldsymbol{s}^0$ with the time complexity $2^{53.19}$: such a complexity can be further lowered to $2^{50.78}$ with our move guessing technique.

Keywords: stream ciphers, A5/1, guess-and-determine, near collision attack

1 Introduction

A5/1 is a typical LFSR-based stream cipher with an irregular clocking mechanism designed in 1980's for the GSM standard. Ever since its proposal, A5/1 has been attacked with various cryptanalytic methods such as time/memory/data

⋆ Corresponding author email: haoyonglin@yeah.net

tradeoff attacks, guess-and-determine attacks, near collision attack (NCA) etc. [1,2,3,4,5,6,7,8] Most of the practical attacks on A5/1 requires large precomputed rainbow table which significantly increases the memory complexities [9,10,11]. Since the implementation of high-memory-requirement attacks are usually quite expensive, the attacks with negligible memory complexities, which we refer in this paper as the "memoryless" attacks, are usually preferable.

The 1st memoryless state-recovery attack on A5/1 is the guess-and-determine attack proposed by Golic [1] at EUROCRYPT 1997. Such an attack requires $2^{43.15}$ steps and each step involves to solve a linear system constructed according to clock bit guesses which is much more complicated than a clock tick as pointed out in [9].

In addition to the guess-and-determine, the near collision attack proposed by Zhang at Asiacrypt 2019 [8] is another cryptanalytic method for recovering A5/1 states with negligible memory complexities. Utilizing some near collision properties in keystream bits and internal state bits, Zhang claimed to reduce the complexity of A5/1 state recovery to only around 2^{32} cipher ticks. The idea of utilizing near collision properties for lowering complexities originate from the cryptanalysis of Grain-v1 in EUROCRYPT 2018 [12]. However, both the Grain-v1 and A5/1 near collision attacks are severely challenged by Derbez et al. in [13]. According to theoretic analysis and practical experiments, Derbez et al. pointed out in [13] that the non-randomness claimed by Zhang in [8] can hardly exist so they draw the conclusion that Zhang's near collision attack in [8] cannot have a complexity lower than that of Golic's basic guess-and-determine attack in [1].

Motivations. Although [13] involves experiments disproving the non-randomness claimed in [8], Derbez et al. did not fully implement the near collision attack or specify an exact complexity leaving it unknown whether the near collision method can still be regarded as an effective cryptanalysis tool for A5/1 state recoveries. It is noticeable that both Golic's attack and Zhang's attack use a system of linear equations as a filter for wrong guesses. But neither Golic nor Zhang has ever evaluated the strength of such a filter in practice. The original complexity evaluation in [8] is based on the assumption that the linear system filter act randomly for wrong guesses with a rank growing linearly to 64: such an assumption is also the foundation of the $2^{43.15}$ complexity of Golic's guess-and-determine attack [1]. Therefore, Derbez et al.'s work has challenged not only the near collision attack but the complexity evaluation of the original memoryless guess-and-determine attack as well indicating that the previous assumption on wrong-guess based linear system may not be true. Therefore, the complexities of both the guess-and-determine and near collision attacks should be reevaluated in a more detailed and accurate manner by taking the filtering strength of the linear systems into account.

Our Contributions. In this paper, we revisit the 2 kinds of memoryless state recovery attacks on A5/1 namely the guess-and-determine attack and the near collision attack. We not only propose new attacks but thoroughly revisit Golic's and Zhang's existing results as well. We first propose a new state bit guessing

strategy called *the move guessing technique*: instead of the conventional practice of guessing 3-bit clocks, we encode the move pattern to 2-bit moves for deducing linear equations of state bits. Since both guess-and-determine and near collision attacks require the process of constructing linear equation systems for filtering, our move guessing technique can be applied to both methods for efficiency improvements. Our 2nd contribution is a more convincing complexity evaluation technique based on practical evaluations to the filtering strength of linear system. Previous works [1,8] implicitly regard the wrong-guess oriented linear system as random whose ranks are assumed to grow linearly. According to our practical experiments, the rank growth and filtering strength of the wrong-guess oriented linear systems are not so random as expected. Based on such techniques, we are able to give 2 new guess-and-determine attacks on A5/1: the 1st attack recovers the initial state $\boldsymbol{s}^0$ with complexity $2^{43.92}$ and the 2nd recovers $\boldsymbol{s}^1$–the state directly used for computing the 1st keystream bit–with complexity $2^{43.25}$. We also revisit Golic's guess-and-determine attack and our complexity evaluation reveals that Golic's method can recover the internal state $\boldsymbol{s}^1$ with a complexity $2^{46.04}$, higher than the previously believed $2^{43.15}$. Then, we analyze the near collision attack on A5/1 only to find that the complexity evaluations in [8] are somewhat optimistic. We point out the mistakes made in [8] and give corrections. According to our detailed analysis, the near collision attack in [8] has a time complexity $2^{53.19}$ which can be further reduced to $2^{50.78}$ using our move guessing technique. The C++ source codes for computing the statistics in this paper are available online [4].

Outline. This paper is organized as follows. Section 2 provides brief introduction to the A5/1 stream cipher and general process of the two categories of memoryless state-recovery attacks. Then, we introduce our move guessing technique in Section 3 and propose our 2 new guess-and-determine attacks on A5/1 in Section 4. After that, we thoroughly revisits Golic's guess-and-determine attack in Section 5: the effect of branching technique is taken into account for complexity evaluations. Section 6 revisits Zhang's near collision attack: we point out the mistakes made in [8], provide corrected complexity evaluations and make slight improvements using our move guessing technique. Section 7 concludes the paper.

1.1 Differences between This Paper and Its Conference Version

This paper is an expanded version of the conference paper [14] presented at Inscrypt 2021. In comparison with the conference version, this paper involves new results as follows:

1. New guess-and-determine attack on internal state $\boldsymbol{s}^1$ of A5/1 with the current lowest complexities.
2. Revisit Golic's guess-and-determine attack in [1] with practically verified complexity evaluations only to prove that the complexity of Golic's attack is higher than previously expected.

[4] `https://github.com/peterhao89/A51Attacks`

3. More detailed analysis to the effect of branching technique resulting in tighter complexity evaluations on Golic's and Zhang's attacks.

2 Preliminary

In this part, we first give a brief introduction to the keystream generation phase of the A5/1 stream cipher in Section 2.1. Then, we review the general processes of the guess-and-determine (Section 2.2) and the near collision (2.3) attacks. Section 2.4 discuss the unit of time for complexity evaluations in this paper.

2.1 The Keystream Generation Procedure of A5/1

A5/1 is a typical LFSR based stream cipher. Its 64-bit internal state consists of 3 LFSR registers $\boldsymbol{R1}, \boldsymbol{R2}, \boldsymbol{R3}$ of sizes 19, 22, 23 respectively. Therefore, we can define the A5/1 64-bit state at time t ($t = 0, 1, 2, \ldots$) as

$$\begin{aligned} \boldsymbol{s}^t =& (\boldsymbol{R1}^t, \boldsymbol{R2}^t, \boldsymbol{R3}^t) \\ =& (\boldsymbol{s}^t[0, \ldots, 18], \boldsymbol{s}^t[19, \ldots, 40], \boldsymbol{s}^t[41, \ldots, 63]) \\ =& (\boldsymbol{R1}^t[0, \ldots, 18], \boldsymbol{R2}^t[0, \ldots, 21], \boldsymbol{R3}^t[0, \ldots, 22]) \end{aligned} \tag{1}$$

We uniformly use $\boldsymbol{s}^t[i]$ to represent the i-th bit of the whole state hereafter. Before generating the output bit z^t, A5/1 round function will update the internal state $\boldsymbol{s}^t \to \boldsymbol{s}^{t+1}$ in a stop-and-go manner as follows:

1. Compute maj^t as
$$\begin{aligned} maj^t =& (\boldsymbol{R1}^t[8] \cdot \boldsymbol{R2}^t[10]) \oplus (\boldsymbol{R1}^t[8] \cdot \boldsymbol{R3}^t[10]) \oplus (\boldsymbol{R2}^t[10] \cdot \boldsymbol{R3}^t[10]) \\ =& (\boldsymbol{s}^t[8] \cdot \boldsymbol{s}^t[29]) \oplus (\boldsymbol{s}^t[8] \cdot \boldsymbol{s}^t[51]) \oplus (\boldsymbol{s}^t[29] \cdot \boldsymbol{s}^t[51]) \end{aligned} \tag{2}$$
where $\cdot$ is the AND of 2 bits.
2. If $\boldsymbol{R1}^t[8] = \boldsymbol{s}^t[8] \neq maj^t$, $\boldsymbol{R1}^{t+1} \leftarrow \boldsymbol{R1}^t$, otherwise, call `updateR1` as follows:
$$\boldsymbol{R1}^{t+1}[i] \leftarrow \begin{cases} \boldsymbol{R1}^t[i-1] & i \in [1, 18] \\ \boldsymbol{R1}^t[18] \oplus \boldsymbol{R1}^t[17] \oplus \boldsymbol{R1}^t[16] \oplus \boldsymbol{R1}^t[13] \end{cases} \tag{3}$$
3. If $\boldsymbol{R2}^t[10] = \boldsymbol{s}^t[29] \neq maj^t$, $\boldsymbol{R2}^{t+1} \leftarrow \boldsymbol{R2}^t$, otherwise, call `updateR2` as follows:
$$\boldsymbol{R2}^{t+1}[i] \leftarrow \begin{cases} \boldsymbol{R2}^t[i-1] & i \in [1, 21] \\ \boldsymbol{R2}^t[21] \oplus \boldsymbol{R2}^t[20] \end{cases} \tag{4}$$
4. If $\boldsymbol{R3}^t[10] = \boldsymbol{s}^t[51] \neq maj^t$, $\boldsymbol{R3}^{t+1} \leftarrow \boldsymbol{R3}^t$, otherwise, call `updateR3` as follows:
$$\boldsymbol{R3}^{t+1}[i] \leftarrow \begin{cases} \boldsymbol{R3}^t[i-1] & i \in [1, 22] \\ \boldsymbol{R3}^t[22] \oplus \boldsymbol{R3}^t[21] \oplus \boldsymbol{R3}^t[20] \oplus \boldsymbol{R3}^t[7] \end{cases} \tag{5}$$

Then, the output keystream bit z^t is generated as

$$\begin{aligned} z^t =& \boldsymbol{R1}^{t+1}[18] \oplus \boldsymbol{R2}^{t+1}[21] \oplus \boldsymbol{R3}^{t+1}[22] \\ =& \boldsymbol{s}^{t+1}[18] \oplus \boldsymbol{s}^{t+1}[40] \oplus \boldsymbol{s}^{t+1}[63] \end{aligned} \tag{6}$$

2.2 A Brief Review of Golic's Guess-and-Determine Attack

For each step $i = 0, 1, 2 \ldots$, whether the registers $\boldsymbol{R1}, \boldsymbol{R2}, \boldsymbol{R3}$ are updated or not depends on the three clock bits $\boldsymbol{s}^i[8, 29, 51]$. Such 3-bit clocks can also be regarded as a 3-bit integer $c^i \in \{0, \ldots, 7\}$ defined as the following Eq. (7):

$$c^i[0, 1, 2] = \boldsymbol{s}^i[8, 29, 51] = (\rho, \varrho, \sigma) \tag{7}$$

In Golic's guess-and-determine model [1], the adversary aims at recovering the initial state $\boldsymbol{s}^1$: the state right before the generation of z^0. So the to-be-guessed clocks are c^i's for $i = 1, 2, \ldots$. With the knowledge of c^i, each bit of $\boldsymbol{s}^{i+1}$ can be represented as a linear combination of $\boldsymbol{s}^i$ bits and, following Eq. (7), the adversary can deduce 3 linear equations:

$$\begin{cases} \boldsymbol{s}^i[8] = \rho \\ \boldsymbol{s}^i[29] = \varrho \\ \boldsymbol{s}^i[51] = \sigma \end{cases}$$

From the output z^i, the adversary can further deduce 1 linear equation:

$$z^i = \boldsymbol{s}^{i+1}[18] \oplus \boldsymbol{s}^{i+1}[40] \oplus \boldsymbol{s}^{i+1}[63]$$

In other words, by guessing 3-bit c^i, the adversary can deduce 4 linear equations of state bits. Therefore, in [1], Golic propose a basic attack that guess $3t$ clock bits $c^1, \ldots c^t$. Based on the $t+1$ output bits $z^0, \ldots, z^t$, the adversary can deduce a system of averaging $1+3t+\frac{4}{3}t$ linear equations. According to [1], for $t \geq 14.38$, the system can involve $1+3t+\frac{4}{3}t \geq 63.32$ equations which is sufficient for identifying the correct guess uniquely with "high probability". Although the number of equations and the "high probability" have never been verified, the complexity of Golic's attack is usually believed as $2^{3t} \geq 2^{43.15}$ steps where each step involves the solution of a linear system. Apparently, such a complexity evaluation is based on the assumption that the wrong-guess oriented linear equation system act randomly and its rank grows linearly with t to 63.32. It is later proved that such an assumption is not true for A5/1.

Besides, Golic also notices that not all $3t$ clock bits $c^1, \ldots, c^t$ are to be guessed independently. According to the sopt-and-go mechanism in Section 2.1, there are occasions where only 2 out of the 3 LFSRs are updated ($c^i \notin \{0, 7\}$) and 1 out of the 3 c^{i+1} bits are already known in c^i. In order to avoid such redundant bit guesses, Golic propose "branching technique" where a tree structure is applied to track the known bits so as to further lower the complexity. However, since the branching technique depends on the clock dynamic values, the complexities in[1] did not take the effect of the branching technique into the evaluations.

2.3 The General Process of Zhang's Near Collision Attack

Different from [1], Zhang's near collision attack in [8] aims at recovering the init state $\boldsymbol{s}^0$. The 64-bit $\boldsymbol{s}^0$ is divided into the 33-bit constraint part (CP) and the

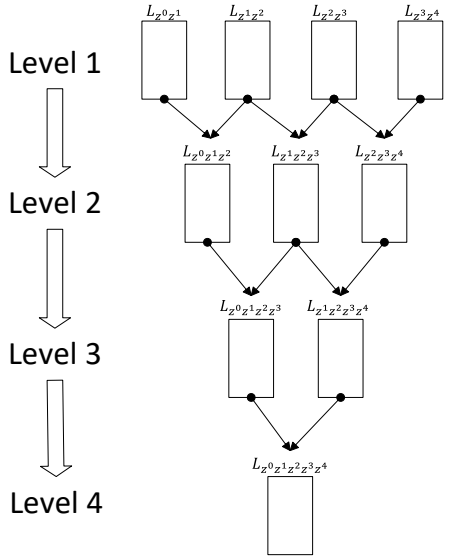

Fig. 1. The general process of Zhang's attack in [8]

31-bit rest part (RP). According to [8], the CP part are the bits related to the 5 output bits namely $z^0, \ldots, z^4$.

The most crucial step in Zhang's attack in [8] is the recovery of the 33-bit CP based on the first 5 keystream bits $z^0, \ldots, z^4$. The CP-recovery phase of Zhang's near collision attack can be summarized as the list-merging process in Fig. 1. Each list $L_{z^i \ldots z^j}$ $(i < j)$ consists of $\boldsymbol{s}^i$'s with partially known state bits: such known bits are determined by the output bits $z^i, \ldots, z^j$. The positions of known bits are denoted as the subset $\lambda \subseteq [0, 63]$ s.t. the knowledge of $\boldsymbol{s}^i[\lambda]$ guarantees the production of outputs $z^i, \ldots, z^j$. For example, at Level 1, $z^i z^{i+1}$ can be related at most 15 $\boldsymbol{s}^0$ denoted as $\boldsymbol{s}^0[\lambda_0]$ where

$$\lambda_0 = \{7, 8, 16, 17, 18, 28, 29, 38, 39, 40, 50, 51, 61, 62, 63\} \tag{8}$$

So the list $L_{z^i z^{i+1}}$ contains all the $\boldsymbol{s}^0[\lambda_0]$ candidates. Similarly, there are at most 21, 27 and 33 known bits for 3/4/5 consecutive keystream bits. Therefore, by the end of the CP-recovery, the list $L_{z^0 z^1 z^2 z^3 z^4}$ in Fig. 1 contains $\boldsymbol{s}^0$'s with 33 known bits at positions $\boldsymbol{s}^0[\lambda]$ where

$$\begin{aligned} \lambda = \{&4, 5, 6, 7, 8, 13, 14, 15, 16, 17, 18, 25, 26, 27, 28, 29, 35, \\ &36, 37, 38, 39, 40, 47, 48, 49, 50, 51, 58, 59, 60, 61, 62, 63\} \end{aligned} \tag{9}$$

It is noticeable that the known bits $\boldsymbol{s}^0[\lambda]$ include involve the clock bits $c^0, \ldots, c^4$. The list merging operations in Fig. 1 takes overlapped bits as filters and, according to [8], each elements in $L_{z^i \ldots z^j}$ can be represented with 5 bytes. Unfortunately, the CP-recovery of Fig. 1 is not fully implemented: the source codes in [8]

only involves Level 1. There is only a theoretic evaluation that the CP-recovery can be accomplished with $2^{28.3}$ cipher and $|L_{z^0z^1z^2z^3z^4}| = 2^{16.6}$.

The 4 initial lists $L_{z^0z^1}, \ldots, L_{z^3z^4}$ are constructed utilizing the low-hamming-weight internal state difference (ISD) $\Delta\boldsymbol{s}$ as follows:

$$\mathcal{D}_2 := \{\Delta\boldsymbol{s} | hw(\Delta\boldsymbol{s}) \leq 2 \text{ and } \Delta\boldsymbol{s}[i] = 0 \text{ for all } i \notin \lambda_0\} \tag{10}$$

where λ_0 is defined in Eq. (8). Among the $\binom{15}{0} + \binom{15}{1} + \binom{15}{2} = 121$ ISD elements, only the 99 can result in the output difference `0x3` which can be defined as the set $\mathcal{T}$ in Eq. (11).

$$\mathcal{T} := \left\{\Delta\boldsymbol{s} \in \mathcal{D}_2 | \exists \boldsymbol{s}^0 \Rightarrow z^0z^1(\boldsymbol{s}^0) \oplus z^0z^1(\boldsymbol{s}^0 \oplus \Delta\boldsymbol{s}) = \texttt{0x3}\right\} \tag{11}$$

For 2 consecutive output bits z^0z^1, the list $L_{z^0z^1}$ can be generated with Algorithm 1.

Algorithm 1 Generate the internal states resulting in the given 2-bit output

procedure `getList`(output bits $z^0z^1 \in \mathbb{F}_2^2$, the number limit T)
  Initialize an empty list $L_{z^0z^1} \leftarrow \phi$
  Declare $\hat{z}_0\hat{z}_1 \leftarrow z^0z^1 \oplus \texttt{0x3}$
  Generate T states $\hat{\boldsymbol{s}}^0$ that only have non-zero elements at positions λ_1 and can result in the output $\hat{z}_0\hat{z}_1$
  for $\Delta\boldsymbol{s} \in \mathcal{T}$ **do**
    Construct state $\boldsymbol{s}^0$
    if $\boldsymbol{s}^0$ can result in the output z^0z^1 **then**
      Update $L_{z^0z^1} \leftarrow L_{z^0z^1} \cup \{\boldsymbol{s}^0\}$
    end if
  end for
  Return $L_{z^0z^1}$
end procedure

According to [8], by setting $T = 4 \cdot 2^{15}/99 = 1323$, the output $L_{z^0z^1}$ is of size 7963 and there is a probability $p_1 = 0.9835$ for $L_{z^0z^1}$ to contain the correct state. In order to further improve the success probability, [8] proposes a distilling process. Let η and ζ be positive integers. The $L_{z^0z^1}$ generation with distilling can be summarized as $L_{z^0z^1} \leftarrow \texttt{distill}(z^0z^1, T, \eta, \zeta)$ as defined in Algorithm 2. By setting $\eta = 2, \zeta = 6$, the parameter p_1 can be improved to 0.9903 according to [8].

After the CP-recovery phase, the adversary has already acquired $2^{16.6}$ $\boldsymbol{s}^0$'s in $L_{z^0z^1z^2z^3z^4}$. For each such $\boldsymbol{s}^0$, the corresponding $\boldsymbol{s}^5$ can be directly computed with the knowledge of 33 bits in CP. For RP-recovery, Zhang guess the $3(t-5)$ clock bits $c^5, \ldots, c^{t-1}$ and observe the outputs $z^5, \ldots, z^{t-1}$ for deducing linear system for key filtering. According to [8], such RP-recovery can be accomplished within 2^{32} cipher ticks dominating the overall complexity of the near collision attack. However, [8] does not provide further details on the filtering strength of

Algorithm 2 Distilling process in [8]

procedure `distill`(output bits $z^0z^1 \in \mathbb{F}_2^2$, the number limit T, integers η, ζ.)
  Initialize $U \leftarrow \phi$
  for $i = 1, \ldots, \eta$ **do**
    for $j = 1, \ldots, \zeta$ **do**
      Call Algorithm 1 as $L^{i,j}_{z^0z^1} \leftarrow$ `getList`(z^0z^1, T)
    end for
    Compute the set $U_i \leftarrow \bigcap_{j=1}^{\zeta} L^{i,j}_{z^0z^1}$
  end for
  Compute $U \leftarrow \bigcup_{i=1}^{\eta} U_i$
  Return U
end procedure

the linear system or the effect of t settings on overall complexities. Therefore, the optimal t setting of the near collision attack is still unspecific.

According to our revisit in Section 6.1, the parameter evaluations in [8] has accuracy issues. The detailed evaluations of t settings further given in Section 6.2 for correct complexity evaluations .

2.4 Unit of the Time Complexity

In [1], Golic takes the solving process of the linear equation system as the unit of time complexities. On the contrary, the unit of time complexity in [8] is the cipher tick. It is noticeable that the near collision attack in [8] also involves the process of solving linear systems but such a process has not been transformed cipher ticks: each linear system solving is still regarded as 1 time unit.

For fair comparison, we uniformly regard the cipher tick and the linear system solving as the time complexity unit.

3 Move Guessing Technique

The introduction of the move guessing technique is in Section 3.1. We further compare the move guessing with the conventional clock guessing technique in Section 3.2.

3.1 Encoding Move Patterns

We define the 2-bit *move pattern* $m^t \in \{0, \ldots, 3\}$ according to the 3-bit clock $c^t = \boldsymbol{s}^t[8, 29, 51]$ in Eq. (7). Such move pattern can be equivalently regarded as binary vector of dimension 2 defined as follows:

$$m^t = m^t[0, 1] = (\boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[29], \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[51]) = (\mu, \nu) \in \mathbb{F}_2^2 \tag{12}$$

With the knowledge of m^t in Eq. (12), 2 equations are deduced as follows:

$$\begin{cases} \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[29] = \mu \\ \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[51] = \nu \end{cases} \tag{13}$$

The 4 possible values of m^t, referred as Move 0-3, corresponds to different movements in A5/1 LFSRs transforming $\boldsymbol{s}^t$ to $\boldsymbol{s}^{t+1}$.

Move 0 From the LFSR action aspect, `updateR1` in (3), `updateR2` in (4) and `updateR3` in (5) are all called. This corresponds to clock values $c^t \in \{0, 7\}$ or equivalently $\boldsymbol{s}^t[8, 29, 51] \in \{(0, 0, 0), (1, 1, 1)\}$.

Move 1 Only `updateR2` and `updateR3` are called corresponding to $c^t \in \{1, 6\}$ or equivalently $\boldsymbol{s}^t[8, 29, 51] \in \{(0, 1, 1), (1, 0, 0)\}$.

Move 2 Only `updateR1` and `updateR3` are called corresponding to $c^t \in \{2, 5\}$ or equivalently $\boldsymbol{s}^t[8, 29, 51] \in \{(1, 0, 1), (0, 1, 0)\}$.

Move 3 Only `updateR1` and `updateR2` are called corresponding to $c^t \in \{3, 4\}$ or equivalently $\boldsymbol{s}^t[8, 29, 51] \in \{(1, 1, 0), (0, 0, 1)\}$.

According to the definition, the LFSR actions before generating the output keystream bits $z^0, \ldots, z^t$ can be represented as $m^0, \ldots, m^t$. In our guess and determine attack, we first guess the movement m^t corresponding to the transformation $\boldsymbol{s}^t \to \boldsymbol{s}^{t+1}$ and maintains a linear equation set $\mathcal{BC}$ by adding new equations according to m^t and the output z^t. For each step t, there are 3 linear equations: 2 are from Eq. (13) according to the move guess and the rest is from the output z^t as

$$\boldsymbol{s}^{t+1}[18] \oplus \boldsymbol{s}^{t+1}[40] \oplus \boldsymbol{s}^{t+1}[63] = z^t \tag{14}$$

So each 2-bit move guess result in 3 equations. In Section 4, we guess the moves $m^0, \ldots, m^{t-1}$ and maintain a linear equations system for recovering correct state $\boldsymbol{s}^0$; we can also guess $m^1, \ldots, m^{t-1}$ for recovering $\boldsymbol{s}^1$.

3.2 Move Guessing v.s. Clock Guessing

We now draw links between the conventional clock guessing technique and our move guessing technique. For time instance t, with the knowledge of clock $c^t = (\rho, \varrho, \sigma)$ in Eq. (7), the corresponding 2-bit move pattern $m^t = (\mu, \nu) \in \mathbb{F}_2^2$ in Eq. (12) can be deduced as

$$\begin{cases} \mu = \rho \oplus \varrho \\ \nu = \rho \oplus \sigma \end{cases} \tag{15}$$

Adding the linear equations of the output in Eq. (14), we know that each 3-bit guess of c^t can deduce 4 equations while each of our 2-bit guess of m^t can deduce 3 equations. Therefore, our move guessing method seems more efficient because each move bit guess can result in 1.5 equations while the number of linear equations for each clock bit guess is no more than 1.34. But it remains to be checked whether the clock-oriented equations can also be better filters for eliminating wrong internal states. On the other hand, the consecutive clocks c^t, c^{t+1} are not independent: when $c^t \notin \{0, 7\}$, there is $m^t \neq 0$ and only 2 out of 3 LFSRs are updated so 1 out of the 3 c^{t+1} bits are equal to the corresponding c^t bit. For example, when $c^t \in \{1, 6\}$ there is $m = 1$, LFSR $\boldsymbol{R1}$ is not updated so $c^t[0] = c^{t+1}[0]$. From this aspect, the knowledge of the t consecutive clocks $c^0, \ldots, c^{t-1}$ do not need to guess all $3t$ bits independently. To identify the clock

guesses, both Golic and Zhang use a branch based method whose complexity is to be evaluated in detail as well. In order to make a fair comparison between the two strategies, we apply both move and clock guessing techniques to different cryptanalysis of A5/1 and provide detailed and convincing complexity evaluations. The results show that move guessing can result in slightly lowered complexities than its clock counterpart.

4 Move Guessing based Guess-and-Determine Attacks

Since Golic's attack aims at $\boldsymbol{s}^1$ and Zhang's attack recovers $\boldsymbol{s}^0$, we provide 2 guess-and-determine attacks recovering $\boldsymbol{s}^0$ (Section 4.1) and $\boldsymbol{s}^1$ (Section 4.2) respectively utilizing the move guessing technique.

4.1 Recovering s^0 State

As can be seen, the move equations Eq. (13) and the output equation Eq. (14) correspond to the internal states at different time instances. But our attack is targeted to recovering the initial state $\boldsymbol{s}^0$. Therefore, the internal states $\boldsymbol{s}^t$ at different time instance t should be represented by $\boldsymbol{s}^0$ bits for deducing $\boldsymbol{s}^0$-related equations. The state $\boldsymbol{s}^t$ is deduced from $\boldsymbol{s}^0$ by taking the moves $m^0, \ldots, m^{t-1}$ as

$$\boldsymbol{s}^0 \xrightarrow{m^0} \boldsymbol{s}^1 \xrightarrow{m^1} \ldots \xrightarrow{m^{t-2}} \boldsymbol{s}^{t-1} \xrightarrow{m^{t-1}} \boldsymbol{s}^t$$

The moves $m^0, \ldots, m^{t-1}$ corresponds to the linear transformations in LFSRs so each $\boldsymbol{s}^t$ bit is a linear combination of $\boldsymbol{s}^0$ bits: such a linear combination can be regarded as a inner-product of $\boldsymbol{s}^0$ and a 64-bit word $\boldsymbol{w} \in \mathbb{F}_2^{64}$. In order to track all state bits in $\boldsymbol{s}^0, \ldots, \boldsymbol{s}^t$ bits, we define 64×64 binary matrices $W^0, \ldots, W^t \in (\mathbb{F}_2^{64})^{64}$ s.t. $\boldsymbol{s}^i = W^i \boldsymbol{s}^0$ for all $i = 0, \ldots, t$. The row vector of W^i is denoted as $W^i[j]$ for $j = 0, \ldots, 63$. In this way, the state bit $\boldsymbol{s}^i[j]$ can be computed as the inner produce of initial $\boldsymbol{s}^0$ and row vector $W^i[j]$. Apparently, there is $W^0 = I$ s.t. $W^0[i] \cdot \boldsymbol{s^0} = \boldsymbol{e}_i \cdot \boldsymbol{s}^0 = \boldsymbol{s}^0[i]$ for $i = 0, \ldots, 63$. With the knowledge of W^{t-1} and m^{t-1} $(t \geq 1)$, the matrix W^t can be deduced by calling Algorithm 3 as $W^t \leftarrow \texttt{UpdW}(m^{t-1}, W^{t-1})$. In this way, each state bit of $\boldsymbol{s}^t$ can be uniformly expressed as a linear combination of $\boldsymbol{s}^0$ bits as

$$\boldsymbol{s}^t[i] = W^t[i] \cdot \boldsymbol{s}^0, i = 0, \ldots, 63, t = 0, 1, 2 \ldots \tag{16}$$

For t consecutive movements $m^0, \ldots, m^{t-1}$ and the corresponding output $z^0, \ldots, z^{t-1}$, the corresponding linear equations set $\mathcal{BC}$ can be deduced as

$$\mathcal{BC} \leftarrow \texttt{getBC}((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$$

where `getBC` is defined as Algorithm 4. Such $\mathcal{BC}$ can be regarded as a linear equation system in Eq. (17)

$$A\boldsymbol{x}^T = \boldsymbol{b}^T, \text{ where } A \in \mathbb{F}_2^{3t \times 64}, \boldsymbol{x} \in \mathbb{F}_2^{64}, \boldsymbol{b} \in \mathbb{F}_2^{3t} \tag{17}$$

and the solutions of Eq. (17) corresponds to all candidate $\boldsymbol{s}^0$'s. The number of solutions depends on the rank of the matrix A and its extended matrix in Eq. (18)

$$E = [A, \boldsymbol{b}^T] \tag{18}$$

- If $rank(A) = rank(E)$, there will be $2^{64-\beta_t}$ solutions where β_t is the positive integer defined in Eq. (19) as the rank of the matrix A;

$$\beta_t = rank(A) \tag{19}$$

- If $rank(A) \neq rank(E)$, there will no solution at all.

With the guessed moves $m^0, \ldots, m^{t-1}$ and the observed output bits $z^0, \ldots, z^{t-1}$, we are now able to acquire both A and $\boldsymbol{b}$ along with the extended matrix E in Eq. (18). We now discuss the probability of $rank(A) = rank(E)$:

- For the correct guess of $m^0, \ldots, m^{t-1}$, $rank(A) = rank(E)$ is constantly true;
- If $m^0, \ldots, m^{t-1}$, the probability of $rank(A) = rank(E)$ is defined as α_t $(0 \leq \alpha_t \leq 1)$. According to our analysis, such α_t's grows gradually with t and should be measured practically.

So the probability of $rank(A) = rank(E)$ can be formally represented as Eq. (20).

$$\Pr\left[rank(A) = rank(E)\right] = \begin{cases} 1 & m^0, \ldots, m^{t-1} \text{ is correctly guessed} \\ \alpha_t \in [0,1] & m^0, \ldots, m^{t-1} \text{ is wrongly guessed} \end{cases} \tag{20}$$

With techniques above, we propose our 1st guess-and-determine attack on A5/1 targeting at recovering $\boldsymbol{s}^0$ within the generic bound 2^{64}. The general process of such an attack can be summarized as follows:

S1 Guess $m^0, \ldots, m^{t-1}$, observe $z^0, \ldots, z^{t-1}$ and deduce the linear system represented as $A\boldsymbol{x}^T = \boldsymbol{b}^T$
S2 Do the rank test checking whether $rank(A) \neq rank(E)$
S3 Traversing the remaining $\boldsymbol{s}^0$ candidates and identify the correct $\boldsymbol{s}^0$ with additional output bits $z^t, \ldots, z^{\ell-1}$ generated by the encryption oracle

Attack Procedure. We further detail the attack as follows:

1. Acquire ℓ keystream bits $z^0, \ldots, z^{\ell-1}$ by querying the A5/1 oracle
2. Initialize $\mathcal{S} \leftarrow \phi$ for collecting $\boldsymbol{s}^0$ candidates
3. Guess $(m^0, \ldots, m^{t-1})$ and do the following substeps:
   (a) Acquire the equations $\mathcal{BC} \leftarrow \texttt{getBC}((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$ by calling Algorithm 4 (**S1**)
   (b) Deduce the A and $\boldsymbol{b}$ in (17) according to $\mathcal{BC}$ and compute the extended matrix E in (18)
   (c) Compute $rank(A)$ and $rank(E)$, if $rank(A) \neq rank(E)$, such a movement guess is wrong, go back to Step 3 for the next movement guess (**S2**)

(d) For all $2^{64-rank(A)}$ solutions to $A\boldsymbol{x}^T = b^T$, set $\hat{\boldsymbol{s}}^0 \leftarrow \boldsymbol{x}$ and generate the keystream bits $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$
(e) If $(\hat{z}^t, \ldots, \hat{z}^{\ell-1}) = (z^t, \ldots, z^{\ell-1})$, add such $\hat{\boldsymbol{s}}^0$ into $\mathcal{S}$ (**S3**)

4. Return $\mathcal{S}$

When ℓ is large enough ($\ell > 64$), there should be only 1 element in $\mathcal{S}$ which is exactly the correct internal state $\boldsymbol{s}^0$. We further evaluate the complexity of our attack as follows.

Complexity Analysis. In Step 3, there are 2^{2t} candidate $(m^0, \ldots, m^{t-1})$'s. According to Eq. (20), averaging $\alpha_t \cdot 2^{2t}$ move pattern candidates can pass the test. Adding β_t in Eq. (19), the averaging time complexity can be computed as Eq. (21).

$$Comp = 2^{2t} + \alpha_t \cdot 2^{2t+64-\beta_t} = 2^{2t} + 2^{2t+64-\beta_t+\log \alpha_t} \tag{21}$$

We randomly select 2^{30} $((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$ pairs and do the 3.(c) test to compute the averaging α_t and β_t for t's and present the statistics in Table 1. The lowest time complexity is $2^{43.92}$ corresponding to $t = 21$. As can be seen in Table 1, β_t has already climbed to almost 64 for $t = 27$ enabling us to set $\ell = 32$ for identifying the correct $\boldsymbol{s}^0$. According to our experiments, such $\ell = 32$ setting is well enough for $\boldsymbol{s}^0$-recovery so the data complexity of our attack is only 32 bits. The memory complexity is only $\mathcal{BC}$ and the corresponding matrix A as well as its extended matrix E in Eq. (17) and Eq. (18), which is $2 \cdot (64 + 1) \cdot 3t \leq 12480$ bits bounded by 2KB.

Table 1. The averaging α_t and β_t in Eq. (21) with 2^{30} random tests for our $\boldsymbol{s}^0$ recovery guess-and-determine attack

t	β_t	$\log \alpha_t$	$\log Comp$	t	β_t	$\log \alpha_t$	$\log Comp$
14	41.96	-0.03	50.01	21	59.86	-2.66	**43.92**
15	44.87	-0.09	49.04	22	61.60	-3.96	44.42
16	47.68	-0.23	48.08	23	62.78	-5.93	46.05
17	50.38	-0.47	47.15	24	63.43	-8.41	48.01
18	52.95	-0.81	46.24	25	63.76	-11.17	50.00
19	55.41	-1.27	45.33	26	63.90	-14.07	52.00
20	57.73	-1.86	44.48	27	63.97	-17.01	54.00

4.2 Recovering $\boldsymbol{s}^1$ State

For recovering $\boldsymbol{s}^1$ according to $z^0, \ldots, z^{t-1}$, we do not need to guess m^0. We guess directly the $t-1$ move patterns $m^1, \ldots, m^{t-1}$ and acquire the linear equation system $A\boldsymbol{x}^T = \boldsymbol{b}^T$ of sizes $A \in \mathbb{F}_2^{(2t-1)\times 64}, \boldsymbol{b} \in \mathbb{F}_2^{2t-1}$. Therefore, the general process has now become:

S1 Guess moves $m^1, \ldots, m^{t-1}$ and maintain a linear system $A\boldsymbol{x}^T = \boldsymbol{b}^T$

Algorithm 3 Deduce the matrix W^{t+11} according to W^t and $m^t \in \mathbb{F}_2^2$.

procedure UpdW(movement $m^t \in \{0,3\}$, the matrix $W^t \in (\mathbb{F}_2^{64})^{64}$)
  if $m^t = 0$ **then**
    $A^t \leftarrow$ UpdWR$(W^t, 1)$
    $B^t \leftarrow$ UpdWR$(A^t, 2)$
    $W^{t+1} \leftarrow$ UpdWR$(B^t, 3)$
  end if
  if $m^t = 1$ **then**
    $B^t \leftarrow$ UpdWR$(W^t, 2)$
    $W^{t+1} \leftarrow$ UpdWR$(B^t, 3)$
  end if
  if $m^t = 2$ **then**
    $A^t \leftarrow$ UpdWR$(W^t, 1)$
    $W^{t+1} \leftarrow$ UpdWR$(A^t, 3)$
  end if
  if $m^t = 3$ **then**
    $A^t \leftarrow$ UpdWR$(W^t, 1)$
    $W^{t+1} \leftarrow$ UpdWR$(A^t, 2)$
  end if
  Return W^{t+1}
end procedure

procedure UpdWR(words $W \in (\mathbb{F}_2^{64})^{64}$, register number $n \in \{1,2,3\}$)
  Initialize $X \in (\mathbb{F}_2^{64})^{64}$ as $X \leftarrow W$
  if $n = 1$ **then**
    for $i = 1, \ldots, 18$ **do**
      Update the i-th entry of X as $X[i] \leftarrow W[i-1]$
    end for
    Compute the 0-th entry of X as $X[0] \leftarrow W[18] \oplus W[17] \oplus W[16] \oplus W[13]$ according to Eq. (3)
  end if
  if $n = 2$ **then**
    for $i = 20, \ldots, 40$ **do**
      Update the i-th entry of X as $X[i] \leftarrow W[i-1]$
    end for
    Compute the 19-th entry of X as $X[19] \leftarrow W[40] \oplus W[39]$ according to Eq. (4)
  end if
  if $n = 3$ **then**
    for $i = 42, \ldots, 63$ **do**
      Update the i-th entry of X as $X[i] \leftarrow W[i-1]$
    end for
    Compute the 41-th entry of X as $X[19] \leftarrow W[63] \oplus W[62] \oplus W[61] \oplus W[48]$ according to Eq. (5)
  end if
  Return X
end procedure

S2 Do the matrix rank test and discard the wrong guesses satisfying $rank(A) \neq rank(E)$

S3 Traverse the remaining $\boldsymbol{s}^1$ candidates and identify the correct $\boldsymbol{s}^1$ with additional output bits $z^t, \ldots, z^{\ell-1}$

In **S1**, we start from $W^1 = I$ and acquire the bit conditions on $(m^1, \ldots, m^{t-1})$ and $(z^1, \ldots, z^{t-1})$ by calling Algorithm 4. Besides, letting $\boldsymbol{s}^1 = \boldsymbol{x} = (x_0, \ldots, x_{63})$, there is also an equation deduced from z^0 according to Eq. (14) as

$$x_{18} \oplus x_{40} \oplus x_{63} = z^0 \tag{22}$$

Same with our $\boldsymbol{s}^0$-recovery guess-and-determine attack in Section 4.1, we detail the procedure and complexity analysis of our $\boldsymbol{s}^1$-recovery attack as follows.

Algorithm 4 Deduce the set of equations according to the given moves and output bits

procedure `getBC`(movements $(m^0, \ldots, m^{t-1}) \in \{0,3\}^t$, output bits $(z^0, \ldots, z^{t-1}) \in \mathbb{F}_2^t$)
  Initialize the words $W^0 \leftarrow I$
  Initialize the linear equations set $\mathcal{BC} \leftarrow \phi$
  Initialize $\boldsymbol{x} = (x_0, \ldots, x_{63})$ as vector of 63 unknown boolean variables corresponding to the 64 state bits of $\boldsymbol{s}^0$
  for $i = 0, 1, \ldots, t-1$ **do**
    Represent $m^i = (\mu, \nu) \in \{0, \ldots, 3\}$ as Eq. (12)
    Update $\mathcal{BC}$ by adding the following equations

$$\begin{cases} (W^i[8] \oplus W^i[29]) \cdot \boldsymbol{x} = \mu \\ (W^i[8] \oplus W^i[51]) \cdot \boldsymbol{x} = \nu \end{cases}$$

    Deduce W^{i+1} according to W^i by calling $W^{i+1} \leftarrow \texttt{UpdW}(m^i, W^i)$ defined in Algorithm 3
    Update $\mathcal{BC}$ by adding the following linear equations corresponding to Eq. (14)

$$(W^{i+1}[18] \oplus W^{i+1}[40] \oplus W^{i+1}[63]) \cdot \boldsymbol{x} = z^i$$

  end for
  Return $\mathcal{BC}$
end procedure

Attack Procedure. So the detailed description of the $\boldsymbol{s}^1$ recovery attack has become:

1. Acqurie ℓ keystream bits $z^0, \ldots, z^{\ell-1}$ by calling the A5/1 oracle
2. Initialize $\mathcal{S} \leftarrow \phi$ for containing $\boldsymbol{s}^1$ candidates
3. Guess $(m^1, \ldots, m^{t-1})$ and do the following substeps:
   (a) Acquire $\mathcal{BC} \leftarrow \texttt{getBC}((m^1, \ldots, m^{t-1}), (z^1, \ldots, z^{t-1}))$ by calling Algorithm 4
   (b) Add an additional Eq. (22) to $\mathcal{BC}$ (**S1**)
   (c) Deduce the A and $\boldsymbol{b}$ in Eq. (17) according to $\mathcal{BC}$ and compute the extended matrix E in Eq. (18)
   (d) Compute $rank(A)$ and $rank(E)$, if $rank(A) \neq rank(E)$, such a movement guess is wrong, go back to Step 3 for the next movement guess (**S2**)
   (e) For all $2^{64-rank(A)}$ solutions to $A\boldsymbol{x}^T = b^T$, set $\hat{\boldsymbol{s}}^1 \leftarrow \boldsymbol{x}$ and generate the keystream bits $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$
   (f) If $(\hat{z}^t, \ldots, \hat{z}^{\ell-1}) = (z^t, \ldots, z^{\ell-1})$, add such $\hat{\boldsymbol{s}}^1$ into $\mathcal{S}$ (**S3**)
4. Return $\mathcal{S}$

Complexity Analysis. Among the $2^{2(t-1)}$ moves $m^1, \ldots, m^{t-1}$, there is a portion of α_t passing the rank test and the averaging $rank(A)$ is β_t in Eq. (19). So the complexity can be evaluated as:

$$Comp = 2^{2(t-1)} + \alpha_t 2^{2(t-1)+64-\beta_t} = 2^{2(t-1)} + 2^{2(t-1)+64-\beta_t+\log \alpha_t} \tag{23}$$

The α_t, β_t parameters are practically evaluated and the corresponding complexities are deduced accordingly as shown in Table 2. The lowest complexity is $2^{43.25}$ at $t = 22$.

Table 2. The averaging α_t and β_t in (21) with 2^{30} random tests for our $\boldsymbol{s}^1$ recovery guess-and-determine attack

t	β_t	$\log \alpha_t$	$\log Comp$	t	β_t	$\log \alpha_t$	$\log Comp$
13	37.00	0.00	51.00	21	58.65	-1.93	43.55
14	39.99	-0.01	50.00	22	60.62	-2.93	**43.25**
15	42.96	-0.03	49.01	23	62.09	-4.53	44.22
16	45.87	-0.09	48.04	24	63.00	-6.74	46.03
17	48.68	-0.24	47.08	25	63.51	-9.32	48.00
18	51.38	-0.47	46.15	26	63.78	-12.06	50.00
19	53.94	-0.82	45.24	27	63.91	-14.85	52.00
20	56.38	-1.29	44.35	28	63.97	-17.61	54.00

5 Revisit Golic's Memoryless Guess-and-Determine Attack

Golic's memoryless guess-and-determine attack guess state bits resembles our $\boldsymbol{s}^1$ recovery attack in Section 4.2. The differences lie in the guessing strategy: we applies the move pattern guessing technique while Golic use the clock guessing technique so the **S1** of Golic's attack has become

S1 Guess clocks $c^1, \ldots, c^{t-1}$ and maintain a linear system $A\boldsymbol{x}^T = \boldsymbol{b}^T$

Resembling our move pattern based equation deduction method `getBC` in Algorithm 4, we define the clock based equation deduction method `getClockBC` in Algorithm 5 where each 3-bit clock c^i $(i = 1, \ldots, t-1)$ and the corresponding output bit z^i results in 4 equations in total. Adding the z^0 based equation Eq. (22), the linear system $A\boldsymbol{x}^T = \boldsymbol{b}^T$ in **S1** are of sizes $A \in \mathbb{F}_2^{(4t-3)\times 64}$ and $\boldsymbol{b} \in \mathbb{F}_2^{4t-3}$. The difference of the detailed description lies in Step 3:

3. Guess $(c^1, \ldots, c^{t-1})$ and do the following substeps:
   (a) Acquire the equations $\mathcal{BC} \leftarrow$ `getClockBC`$((c^1, \ldots, c^{t-1}), (z^1, \ldots, z^{t-1}))$ by calling Algorithm 5
   (b) Add an additional Eq. (22) to $\mathcal{BC}$ (**S1**)
   (c) Deduce the A and $\boldsymbol{b}$ in Eq. (17) according to $\mathcal{BC}$ and compute the extended matrix E in Eq. (18)
   (d) Compute $rank(A)$ and $rank(E)$, if $rank(A) \neq rank(E)$, such a movement guess is wrong, go back to Step 3 for the next clock guess (**S2**)
   (e) For all $2^{64-rank(A)}$ solutions to $A\boldsymbol{x}^T = b^T$, set $\hat{\boldsymbol{s}}^1 \leftarrow \boldsymbol{x}$ and generate the keystream bits $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$

(f) If $(\hat{z}^t, \ldots, \hat{z}^{\ell-1}) = (z^t, \ldots, z^{\ell-1})$, add such $\hat{\boldsymbol{s}}^1$ into $\mathcal{S}$ (**S3**)

The Effect of the Branching Technique. In Golic's guess-and-determine attack, as well as Zhang's near collision attacks, the so-called "branching" technique when deducing equations [1,8]. The branching technique points out that not all $3(t-1)$ bits in $c^1, \ldots, c^{t-1}$ are to be guessed independently. However, the effect of branching techniques is closely related to the actual clock values making it hard to evaluate accurately. So all previous works still regard $c^1, \ldots, c^{t-1}$ as $3(t-1)$-bit guesses in complexity analysis. With the help of move pattern, we are able give a tighter complexity evaluation to the branching technique. According to Section 3.1, each clock c^i uniquely defines the corresponding movement m^i. Different from the move guessing strategy where exactly 2 bits are guessed for each move m^i $(i = 2, \ldots, t-1)$, the number of guessed bits for c^i can either be 2 or 3: when $c^{i-1} \notin \{0, 7\}$ $(m^{i-1} \neq 0)$, only 2 out of 3 LFSRs are updated so 1 out of the 3 c^i bits are already known from c^{i-1}; otherwise, all 3 LFSRs are updated so all 3 c^i bits should be guessed. Therefore, the branching technique utilizes a tree structure for tracking the clock bits that is already known, only averaging $\frac{7}{3}(t-1)$ bits are to be guessed for $c^1, \ldots, c^{t-1}$ and the remaining $\frac{2}{3}(t-1)$ clock bits are deduced from the guessed bits.

Complexity Analysis. According to Eq. (20), among the $2^{\frac{7}{3}(t-1)}$ candidate clocks $(c^1, \ldots, c^{t-1})$, only averaging $\alpha_t \cdot 2^{\frac{7}{3}(t-1)}$ of them can pass the rank test. Adding the knowledge of β_t in Eq. (19), we are able to give the complexity evaluation of Golic's attacks as Eq. (24)

$$Comp = 2^{\frac{7}{3}(t-1)} + \alpha_t \cdot 2^{\frac{7}{3}(t-1)+64-\beta_t} = 2^{\frac{7}{3}(t-1)} + 2^{\frac{7}{3}(t-1)+64-\beta_t+\log \alpha_t} \tag{24}$$

We practically evaluated the α_t, β_t parameters and deduced the corresponding complexities. As can be seen in Table 3, the optimal complexity is $2^{46.04}$ at $t = 20$, higher than the previously believed 2^{43} [1] and that of our $\boldsymbol{s}^1$ recovery attack in Section 4.2. Such a revisit indicates that our move guessing technique has advantages over the conventional clock guessing technique in guess-and-determine attacks on A5/1.

Table 3. The averaging α_t and β_t in Eq. (21) with 2^{30} random tests for Golic's $\boldsymbol{s}^1$ recovery guess-and-determine attack

t	β_t	$\log \alpha_t$	$\log Comp$	t	β_t	$\log \alpha_t$	$\log Comp$
14	43.99	-0.01	50.34	22	62.09	-6.76	49.05
15	47.14	-0.08	49.45	23	62.79	-9.14	51.34
16	50.09	-0.30	48.61	24	63.27	-11.73	53.67
17	52.77	-0.74	47.82	25	63.60	-14.36	56.00
18	55.23	-1.40	47.05	26	63.81	-16.88	58.33
19	57.49	-2.25	46.34	26	63.81	-16.90	58.33
20	59.48	-3.34	**46.04**	27	63.92	-19.33	60.67
21	61.04	-4.81	47.02	28	63.97	-21.48	63.00

Algorithm 5 Deduce the set of equations according to the given clocks and output bits

procedure `getClockBC`(movements $(c^1, \ldots, c^{t-1}) \in \{0, \ldots, 7\}^t$, output bits $(z^1, \ldots, z^{t-1}) \in \mathbb{F}_2^t$)
  Initialize the matrix $W^1 \leftarrow I$
  Initialize the linear equations set as empty: $\mathcal{BC} \leftarrow \phi$
  Initialize $\boldsymbol{x} = (x_0, \ldots, x_{63})$ as vector of 64 unknown boolean variables corresponding to the 64 state bits of $\boldsymbol{s}^1$
  for $i = 1, \ldots, t-1$ **do**
    Represent $c^i = (\rho, \varrho, \sigma)$ and deduce the movement $m^i = (\mu, \nu)$ according to Eq. (15)
    Update $\mathcal{BC}$ by adding the following equations:

$$\begin{cases} W^i[8] \cdot \boldsymbol{x} = \rho \\ W^i[29] \cdot \boldsymbol{x} = \varrho \\ W^i[51] \cdot \boldsymbol{x} = \sigma \end{cases}$$

    Deduce W^{i+1} according to W^i by calling $W^{i+1} \leftarrow \texttt{UpdW}(m^i, W^i)$ defined in Algorithm 3
    Update $\mathcal{BC}$ by adding the following linear equations corresponding to Eq. (14)

$$(W^{i+1}[18] \oplus W^{i+1}[40] \oplus W^{i+1}[63]) \cdot \boldsymbol{x} = z^i$$

  end for
  Return $\mathcal{BC}$
end procedure

6 Revisit Zhang's Near Collision Attack

In this part, we fully implement Zhang's near collision attack originally given in [8]. Firstly, Section 6.1 reveals that several crucial parameters are wrongly evaluated in [8] resulting in underestimated complexities. Then, we thoroughly revisit Zhang's near collision attack in Section 6.2. We further propose an improved near collision attack by replacing Zhang's RP-recovery with our move guessing technique.

6.1 Inaccurate Evaluations of Parameters

There are 2 categories of wrongly evaluated parameters: the p_1 used in the distilling phase along with corresponding success probability; the 4 parameters related to the complexities of the CP-Recovery process.

p_1 and the Success Probability. According to [8], the randomly constructed list $L_{z^0z^1} \leftarrow \texttt{getList}(z^0z^1, T)$ acquired by calling Algorithm 1 is of size 7963 when $T = 4 \cdot 2^{15}/99$, and the probability for $L_{z^0z^1}$ to contain correct $\boldsymbol{s}^0$ is $p_1 = 0.9835$. However, among 10^6 times' repeated experiments, the correct state

lies in $L_{z^0z^1}$ for only 972436 times. So we safely claim the actual p_1 as 0.9725: Zhang's evaluation in [8] is inaccurate. The reason of such inaccuracy is unknown. One possible explanation is the usage of RC4 as the source of randomness which is not so qualified. We have tried various random generators including SNOW-V [15], AES [16] etc. All experiments indicate that the actual p_1 is 0.9725 rather than 0.9835.

With p_1 corrected, the probability that the correct state lies in the distilled list $U \leftarrow \texttt{distill}(z^0z^1, T, \eta, \zeta)$ should be reevaluated as well. We compare our probability evaluations and the original one in Table 4. By analyzing the source codes [5] given in [8], we conclude that the wrong $|U|$ evaluations result from wrong implementations: when they try to get (η, ζ), they actually do ζ intersect operations so they actually acquire the $U(\eta, \zeta+1)$ instead. As a consequence, the size $|U|$'s are smaller than expected. The near collision in [8] can only succeed when the correct $\boldsymbol{s}^0$ lies in U so the `Prob` in Table 4 is the overall success probability which should be corrected to 0.9761 rather than the previously claimed 0.9903.

Table 4. Our evaluation (left) v.s. Zhang et al.'s (right, quoted from [8])

η	ζ	$\lvert U\rvert$	`Prob.`
2	3	8109	0.9935
2	4	8050	0.9887
2	5	8009	0.9830
2	6	7948	0.9761

η	ζ	$\lvert U\rvert$	`Prob.`
2	3	8065	0.9940
2	4	7989	0.9927
2	5	7934	0.9912
2	6	7835	0.9903

4 Parameters Related to the CP-Recovery Process. The original paper [8] does not involve the full implementation. We reveal that the theoretically deduced values of 4 CP-recovery related parameters are inaccurate. We list such parameters and compare their theoretic evaluated values with our practically acquired values in Table 5. As can be seen, the 2 evaluations are quite different and we provide detailed explanations as follows.

Merging of two lists L_1, L_2 requires $|L_1| \cdot |L_2|$ (the time complexity of the merging algorithm in [8] is $|L_1| + |L_2|$) operations so the exact complexity of the merging cannot be acquired without the knowledge of all list sizes. We fully implement the merging process in Fig. 1 and acquire the sizes of the lists are as follows:

$$\begin{aligned} |L_{z^i z_{i+1}}| &\approx 2^{12.95}, \quad \text{for } i = 0, 1, 2, 3 \\ |L_{z^i z_{i+1} z^{i+2}}| &\approx 2^{16.70}, \quad \text{for } i = 0, 1, 2 \\ |L_{z^i z_{i+1} z^{i+2} z^{i+3}}| &\approx 2^{20.46}, \quad \text{for } i = 0, 1 \\ |L_{z^0 z^1 z^2 z^3 z^4}| &\approx 2^{24.21} \end{aligned} \tag{25}$$

[5] `https://github.com/martinzhangbin/gsmencryption`

Table 5. 4 Parameters Related to the CP-Recovery Process: [8]'s evaluations v.s. Ours

Parameter	Eval. in [8]	Our Eval.
Cipher ticks for the merging process in Fig. 1	$2^{28.3}$	$2^{40.92}$†
The number of $L_{z^0z^1z^2z^3z^4}$ candidates	$2^{16.6}$	$2^{24.21}$
Bytes for storing a $L_{z^0z^1z^2z^3z^4}$ element	5	9
The number of known bits for each $L_{z^0z^1z^2z^3z^4}$ element ($\|\lambda\|$)	33	30.14

†: quadratic time implementations

The complexity of the merging process is dominated by the procedure from Level 3 to 4 which is approximately $2^{20.46\times 2} = 2^{40.92}$ using the C++ implementation which is far beyond $2^{28.3}$ claimed in [8]. The size of $L_{z^0z^1z^2z^3z^4}$ is $2^{24.21}$, also larger than the $2^{16.6}$ in [8]. The reason is that [8] ignored the fact that the middle lists are used more than once in the merging phase. [8] did not realize that such repeated use of lists cannot provide extra filtering strength.

Each merging step in Fig. 1 takes two lists denoted as L_t and L_{t+1} where L_t contains the partial states of $\boldsymbol{s}^t$ while L_{t+1} contains those of $\boldsymbol{s}^{t+1}$. According to Section 3, the $\boldsymbol{s}^t$ to $\boldsymbol{s}^{t+1}$ transformation takes a move pattern $m^t \in \{0,1,2,3\}$ and such m^t is decided by 3-bit clock $c^t = \boldsymbol{s}^t[8,29,51]$. So the merging step $\tilde{L}_t \leftarrow \texttt{merge}(L_t, L_{t+1})$ is as follows:

1. Initialize the list $\tilde{L}_t \leftarrow \phi$.
2. For all $(\boldsymbol{s}^t, \boldsymbol{s}^{t+1}) \in L_t \times L_{t+1}$, do the following substeps:
   (a) Identify the positions of the known bits in $\boldsymbol{s}^t$ denoted as $\lambda_0 \subseteq [0,63]$.
   (b) Compute m^t according to 3-bit clock $c^t = \boldsymbol{s}^t[8,29,51]$ as Eq. (15).
   (c) Determine the state $\hat{\boldsymbol{s}}^t$ s.t. $\hat{\boldsymbol{s}}^t \xrightarrow{m^t} \boldsymbol{s}^{t+1}$
   (d) Identify the positions of the known bits in $\hat{\boldsymbol{s}}^t$ denoted as $\lambda_1 \subseteq [0,63]$
   (e) If $\hat{\boldsymbol{s}}^t[i] = \boldsymbol{s}^t[i]$ for all $i \in \lambda_0 \cap \lambda_1$, store the vector $\tilde{\boldsymbol{s}}^t \leftarrow \hat{\boldsymbol{s}}^t \vee \boldsymbol{s}^t$ in $\tilde{L}_t$ where $\vee$ is bitwise OR. The known bits of the newly generated $\tilde{\boldsymbol{s}}^t$ is $\tilde{\lambda} \leftarrow \lambda_0 \cup \lambda_1$.
3. Return $\tilde{L}_t$

According to the description above, an element $\boldsymbol{s} \in L$ contains both the values of known bits but the set $\lambda \subseteq [0,63]$ containing the positions of known bits as well. At Level 1, all list are generated with Algorithm 1 so all $\boldsymbol{s}$'s share the same known-bit positions λ_0 in Eq. (8). But at Levels 2-4, different moves will result in different λ's. Example 1 indicates that the known bits of the merged partial state $\tilde{\boldsymbol{s}}^t$ may not be exactly the 21 bits as believed in [8]: the λ's are also likely to be subsets of the 21 bits.

Example 1. Let $(\boldsymbol{s}^0, \boldsymbol{s}^1) \in L_{z^0z^1} \times L_{z^1z^2}$. The known bits are at positions $\lambda = \lambda_0$ defined in Eq. (8). We consider the merged elements $\tilde{\boldsymbol{s}}^0 \in L_{z^0z^1z^2}$ as well as the known bit positions $\tilde{\lambda} \subseteq [0,63]$. We consider the 2 situations corresponding to $m^0 = 0$ and $m^0 = 1$ as follows:

1. For $m^0 = 0$ ($c^t \in \{0, 7\}$), the known bit positions for $\boldsymbol{s}^1$ should be $\lambda = \lambda_1$ where

   $$\lambda_1 := \{7-1, 8-1, 16-1, 17-1, 18-1, 28-1, 29-1, 38-1, 39-1, 40-1, 50-1, 51-1, 61-1, 62-1, 63-1\}.$$

   and $\tilde{\lambda} = \lambda_0 \cup \lambda_1$ is of size $|\tilde{\lambda}| = 21$.
2. For $m^0 = 1$ ($c^0 \in \{3, 4\}$), the λ_1 becomes

   $$\lambda_1 := \{7, 8, 16, 17, 18, 28-1, 29-1, 38-1, 39-1, 40-1, 50-1, 51-1, 61-1, 62-1, 63-1\}.$$

   and $\tilde{\lambda} = \lambda_0 \cup \lambda_1$ is of size $|\tilde{\lambda}| = 19$.

As can be seen from Example 1, different moves can result in different known bit position λ's so all elements in the merging lists in Level 2-4 should contain not only the bit values but the positions as well. Since the elements in $L_{z^0 z^1 z^2 z^3 z^4}$ are partial states of at most 33 bits, the corresponding λ can also be stored with 33 bits. So an element in the lists in Fig. 1 requires at most $\lceil 2 \cdot 33/8 \rceil = 9$ bytes. Same with Example 1, the λ for $\boldsymbol{s}^0 \in L_{z^0 z^1 z^2 z^3 z^4}$ are more likely to be subsets of the 33 bits. According to our experiments, the $L_{z^0 z^1 z^2 z^3 z^4}$ elements has averaging $|\lambda| \approx 30.14$ known bits: $|\lambda| = 33$ can only happen when $m^0 = \ldots = m^4 = 0$ (or equivalently $c^0, \ldots, c^4 \in \{0, 7\}$) which is of probability 2^{-10}.

6.2 The Original Near Collision Attack Revisit

With the supplementations in Section 6.1, the whole CP-recovery phase can be carried out. After CP-recovery, the adversary is equipped with the list $L_{z^0 z^1 z^2 z^3 z^4}$ containing $2^{24.21}$ elements. Each $\tilde{\boldsymbol{s}}^0 \in L_{z^0 z^1 z^2 z^3 z^4}$ is also equipped with a set $\tilde{\lambda} \subseteq [0, 63]$ containing the positions of known state bits. The known $\tilde{\boldsymbol{s}}^0[\tilde{\lambda}]$ bits guarantee the first 5 output bits $z^0, \ldots, z^4$. Let $\boldsymbol{x} = (x_0, \ldots, x_{63})$ be targeted state $\boldsymbol{s}^0$, the known bits $\tilde{\boldsymbol{s}}^0[\tilde{\lambda}]$ corresponds to $|\tilde{\lambda}|$ equations of the form:

$$x_i = \tilde{\boldsymbol{s}}^0[i], \text{ for } i \in \tilde{\lambda} \tag{26}$$

Note that the known bits $\tilde{\boldsymbol{s}}^0[\tilde{\lambda}]$ involve clocks $c^0, \ldots, c^4$ based on which can deduce directly the first 5 moves $m^0, \ldots, m^4$.

Then, according to [8], the RP-recovery process is carried out by guessing $c^5, \ldots, c^{t-1}$ and constructing the corresponding equation system according to clock bits and output bits. So the whole process of the original near collision attack in [8] can now be summarized as follows:

1. Acquire the keystream bits $z^0, \ldots, z^{\ell-1}$ by querying A5/1 oracle
2. Acquire the list $L_{z^0 z^1 z^2 z^3 z^4}$ by running the CP-recovery process in Fig. 1
3. Initialize $\mathcal{S} \leftarrow \phi$ for containing the $\boldsymbol{s}^0$ candidates
4. For each $\tilde{\boldsymbol{s}}^0 \in L_{z^0 z^1 z^2 z^3 z^4}$, do the following substeps **(RP-Recovery)**
   (a) Extract the 5 clocks $c^0, \ldots, c^4$ from the known bits $\tilde{\boldsymbol{s}}^0[\tilde{\lambda}]$
   (b) Guess the clock $c^5, \ldots, c^{t-1}$ and do the following substeps:
      i. Deduce equations $\mathcal{BC} \leftarrow \texttt{getClockBC}((c^0, \ldots, c^{t-1}), (z^0, \ldots, z^{t-1}))$
      ii. Add the linear equations in Eq. (26) to $\mathcal{BC}$

iii. Deduce the A and $\boldsymbol{b}$ in Eq. (17) according to $\mathcal{BC}$ and compute the extended matrix E in Eq. (18)
iv. Compute $rank(A)$ and $rank(E)$, if $rank(A) \neq rank(E)$, such a clock guess is wrong, go back to Step (b) for the next guess of $c^5, \ldots, c^{t-1}$
v. For all $2^{64-rank(A)}$ solutions to $A\boldsymbol{x}^T = \boldsymbol{b}^T$, set $\hat{\boldsymbol{s}}^0 \leftarrow \boldsymbol{x}$ and generate the keystream bits $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$
vi. If $(\hat{z}^t, \ldots, \hat{z}^{\ell-1}) = (z^t, \ldots, z^{\ell-1})$, add such $\hat{\boldsymbol{s}}^0$ into $\mathcal{S}$

5. Return $\mathcal{S}$

Complexity Analysis. According to Eq. (25), there are $2^{24.21}$ candidate $\tilde{\boldsymbol{s}}^0$ in $L_{z^0 z^1 z^2 z^3 z^4}$. Same with the analysis in Section 5, there are averaging $2^{\frac{7}{3}(t-5)}$ possible $c^5, \ldots, c^{t-1}$ guesses. According to Eq. (20), there are $\alpha_t \cdot 2^{\frac{7}{3}(t-5)+24.21}$ candidate $\tilde{\boldsymbol{s}}^0$'s can pass the rank test. According to the analysis in Section 6.1, the CP-recovery process has complexity $2^{40.92}$. Adding the β_t in Eq. (19), the time complexity can be computed as Eq. (28).

$$Comp = 2^{40.92} + 2^{24.21+\frac{7}{3}(t-5)} + 2^{24.21+\frac{7}{3}(t-5)+64-\beta_t+\log \alpha_t} \tag{27}$$

The practically evaluated α_t, β_t along with the corresponding complexities are listed in Table 7. The lowest complexity is $2^{53.19}$ at the $t = 14$ setting. Furthermore, the near collision attack needs to store the lists in Fig. 1 so the memory complexity is $2^{24.21}$ dominated by the size of $L_{z^0 z^1 z^2 z^3 z^4}$. The near collision attack can only succeed when the correct $\boldsymbol{s^i}$ lies in the corresponding list $L_{z^i z_{i+1}}$ for $i = 0, 1, 2, 3$ so the success probability is $p_1^4 \approx 0.8942$ according to the Section 6.1. In comparison, Golic's guess-and-determine attack recovers $\boldsymbol{s}^1$ with complexity $2^{46.19}$ which is lower than that of Zhang's near collision attack which supports the conclusion in [13].

Table 6. The averaging α_t and β_t in (27) with 2^{30} random tests for Zhang's near collision attack in [8].

t	β_t	$\log \alpha_t$	$\log Comp$	t	β_t	$\log \alpha_t$	$\log Comp$
6	31.98	-0.11	58.45	14	55.55	-0.47	**53.19**
7	34.84	-0.12	57.92	15	57.53	-0.75	53.29
8	37.95	-0.12	57.15	16	59.00	-1.28	53.70
9	41.14	-0.12	56.28	17	60.05	-2.08	54.43
10	44.31	-0.14	55.43	18	60.83	-3.10	55.58
11	47.38	-0.20	54.63	19	61.44	-4.33	57.25
12	50.34	-0.28	53.93	20	61.97	-5.72	59.32
13	53.10	-0.35	53.42	21	62.46	-7.25	61.57

6.3 Improved Near Collision Attack with Move-Based RP-Recovery

In this part, we propose an improved near collision attack by replacing the clock guessing based RP-recovery phase with a move guessing one. The improved and

original attacks share the same CP-recovery process so we only detail the RP-recovery phase of the improved attack as follows:

4. For each $\tilde{\boldsymbol{s}}^{\mathbf{0}} \in L_{z^0 z^1 z^2 z^3 z^4}$ **(RP-Recovery)**
   (a) Deduce the 5 moves $(m_0, \ldots, m_4)$ from the known bits $\tilde{\boldsymbol{s}}^{\mathbf{0}}[\tilde{\lambda}]$
   (b) Guess the move patterns $(m^5, \ldots, m^{t-1})$ and do the following substeps:
      i. Deduce the equations $\mathcal{BC} \leftarrow \texttt{getBC}((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$
      ii. Add the linear equations of the form Eq. (26) to $\mathcal{BC}$
      iii. Deduce the A and $\boldsymbol{b}$ in Eq. (17) according to $\mathcal{BC}$ and compute the extended matrix E in Eq. (18)
      iv. Compute $rank(A)$ and $rank(E)$, if $rank(A) \neq rank(E)$, go back to Step (b) for the next guess of moves $m^5, \ldots, m^{t-1}$
      v. For all $2^{64-rank(A)}$ solutions to $A\boldsymbol{x}^T = \boldsymbol{b}^T$, set $\hat{\boldsymbol{s}}^0 \leftarrow \boldsymbol{x}$ and generate the keystream bits $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$
      vi. If $(\hat{z}^t, \ldots, \hat{z}^{\ell-1}) = (z^t, \ldots, z^{\ell-1})$, add such $\hat{\boldsymbol{s}}^0$ into $\mathcal{S}$

Complexity Analysis. There are $2^{2(t-5)}$ moves $(m^5, \ldots, m^{t-1})$ in total. According to Eq. (20), there are $\alpha_t \cdot 2^{2(t-5)+24.21}$ candidate $\tilde{\boldsymbol{s}}^0$'s passing the rank test. So the averaging time complexity can be computed as Eq. (28).

$$Comp = 2^{40.92} + 2^{24.21+2(t-5)} + \alpha_t \cdot 2^{24.21+2(t-5)+64-\beta_t} \tag{28}$$

We list practical evaluated α_t, β_t along with the corresponding complexities in Table 7. The lowest complexity is $2^{50.78}$ at $t = 16$ setting: such a complexity is lower than that of the original near collision attack in Section 6.2 but higher than those of the guess-and-determine attacks in Section 4 and Section 5. The memory complexity and the success probability are identical to those of the original near collision attack which are $2^{24.21}$ and 0.8942 respectively.

Table 7. The averaging α_t and β_t with 2^{30} random tests for the new near collision attack improved with the move guessing technique.

t	β_t	$\log \alpha_t$	$\log Comp$	t	β_t	$\log \alpha_t$	$\log Comp$
6	31.95	-0.11	58.15	14	54.68	-0.37	51.17
7	34.68	-0.12	57.41	15	56.74	-0.66	50.83
8	37.58	-0.12	56.51	16	58.30	-1.19	**50.78**
9	40.55	-0.12	55.54	17	59.48	-1.96	50.99
10	43.52	-0.13	54.57	18	60.40	-2.95	51.57
11	46.46	-0.15	53.60	19	61.15	-4.14	52.71
12	49.37	-0.19	52.65	20	61.79	-5.50	54.35
13	52.16	-0.25	51.80	21	62.36	-7.01	56.24

7 Conclusion

In this paper, we propose new move guessing technique and revisit 2 categories of memoryless state-recovery methods on A5/1 stream cipher namely the guess-and-determine attack and the near collision attack. We propose 2 move guessing

based guess-and-determine attacks on A5/1 that can cover internal states $\boldsymbol{s}^0$ and $\boldsymbol{s}^1$ with the current lowest complexities. We further revisit Golic's guess-and-determine [1] and Zhang's near collision [8] attacks. With practical filtering strength evaluations, we are able to prove that the complexities of both Golic's and Zhang's attacks are lower than expected: the complexity of Golic's attack is $2^{46.04}$, higher than the previously believed 2^{43}; Zhang's attack has complexity $2^{53.19}$. Although its complexity is higher than the previously claimed 2^{32}, Zhang's near collision attack is still effective for recovering the internal state of A5/1. We further propose an improved near collision attack with complexity $2^{50.78}$. According to our analysis, we claim that the filtering strength of linear equations deduced from wrong guesses should be evaluated practically rather than simply regarded as random because the rank growth of wrong guess oriented linear systems may not be so fast as random linear systems.

Acknowledgement. Yonglin Hao is supported by National Natural Science Foundation of China (Grant No. 62002024), National Key Research and Development Program of China (No. 2018YFA0306404).

References

1. Golic, J.D.: Cryptanalysis of alleged A5 stream cipher. In Fumy, W., ed.: EUROCRYPT'97. Volume 1233 of LNCS., Springer, Heidelberg (May 1997) 239–255
2. Biham, E., Dunkelman, O.: Cryptanalysis of the A5/1 GSM stream cipher. In Roy, B.K., Okamoto, E., eds.: INDOCRYPT 2000. Volume 1977 of LNCS., Springer, Heidelberg (December 2000) 43–51
3. Shah, J., Mahalanobis, A.: A New Guess-And-Determine Attack On The A5/1 Stream Cipher. Cryptology ePrint Archive, Report 2012/208 (2012) `https://eprint.iacr.org/2012/208`.
4. Maximov, A., Johansson, T., Babbage, S.: An improved correlation attack on A5/1. In Handschuh, H., Hasan, A., eds.: SAC 2004. Volume 3357 of LNCS., Springer, Heidelberg (August 2004) 1–18
5. Li, Z.: Optimization of Rainbow tables for practically cracking GSM A5/1 based on validated success rate modeling. In Sako, K., ed.: CT-RSA 2016. Volume 9610 of LNCS., Springer, Heidelberg (February / March 2016) 359–377
6. Gendrullis, T., Novotný, M., Rupp, A.: A real-world attack breaking A5/1 within hours. In Oswald, E., Rohatgi, P., eds.: CHES 2008. Volume 5154 of LNCS., Springer, Heidelberg (August 2008) 266–282
7. Barkan, E., Biham, E.: Conditional estimators: An effective attack on A5/1. In Preneel, B., Tavares, S., eds.: SAC 2005. Volume 3897 of LNCS., Springer, Heidelberg (August 2006) 1–19
8. Zhang, B.: Cryptanalysis of GSM encryption in 2G/3G networks without Rainbow tables. In Galbraith, S.D., Moriai, S., eds.: ASIACRYPT 2019, Part III. Volume 11923 of LNCS., Springer, Heidelberg (December 2019) 428–456
9. Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of A5/1 on a PC. In Schneier, B., ed.: FSE 2000. Volume 1978 of LNCS., Springer, Heidelberg (April 2001) 1–18

10. Pornin, T., Stern, J.: Software-hardware trade-offs: Application to A5/1 cryptanalysis. In Koç, Çetin Kaya., Paar, C., eds.: CHES 2000. Volume 1965 of LNCS., Springer, Heidelberg (August 2000) 318–327
11. Lu, J., Li, Z., Henricksen, M.: Time-memory trade-off attack on the GSM A5/1 stream cipher using commodity GPGPU - (extended abstract). In Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M., eds.: ACNS 15. Volume 9092 of LNCS., Springer, Heidelberg (June 2015) 350–369
12. Zhang, B., Xu, C., Meier, W.: Fast near collision attack on the Grain v1 stream cipher. In Nielsen, J.B., Rijmen, V., eds.: EUROCRYPT 2018, Part II. Volume 10821 of LNCS., Springer, Heidelberg (April / May 2018) 771–802
13. Derbez, P., Fouque, P.A., Mollimard, V.: Fake near collisions attacks. IACR Trans. Symm. Cryptol. **2020**(4) (2020) 88–103
14. Wang, M., Hao, Y.: Revisit two memoryless state-recovery cryptanalysis methods on A5/1. In Yu, Y., Yung, M., eds.: Information Security and Cryptology - 17th International Conference, Inscrypt 2021, Virtual Event, August 12-14, 2021, Revised Selected Papers. Volume 13007 of Lecture Notes in Computer Science., Springer (2021) 191–211
15. Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: A new SNOW stream cipher called SNOW-V. IACR Trans. Symmetric Cryptol. **2019**(3) (2019) 1–42
16. Standards, N.: Specification for the advanced encryption standard (aes). FIPS-197 (2001)