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

Mingxing Wang[1,2] and Yonglin Hao[1⋆]

[1] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[2] The 6th Research Institute of China Electronics Corporation, Beijing 100083, China

Abstract. At ASIACRYPT 2019, Zhang proposes a near collision attack on A5/1. He claims that such an attack method can recover the 64-bit A5/1 state with a time complexity around 2^{32} cipher ticks and requires negligible memory complexities. Soon after its proposal, Zhang's near collision attack is severely challenged by Derbez et al. who claim 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. In order to make a fair comparison, we recover the state $\boldsymbol{s}^0$ using both methods. We propose a new guessing technique that can construct linear equation filters in a more efficient manner. When evaluating time complexities, we take the filtering strength of the linear equation systems into account making the complexities more convincing. According to our detailed analysis, the new guess-and-determine attack can recover the state $\boldsymbol{s}^0$ with a time complexity of $2^{43.91}$ simple operations. The time complexity for the near collision attack is $2^{50.57}$ simple operations.

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 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 proposed by Golic [1] at EUROCRYPT 1997. The basic guess-and-determine attack in [1] requires $2^{43.15}$

⋆ Corresponding author email: `haoyonglin@yeah.net`

steps, where each step in this attack is much more complicated, since it is based on the solution of a linear system as pointed out in [9]. The latest memoryless result on A5/1 is proposed at Asiacrypt 2019 where Zhang [8] claims that, by utilizing some near collision properties, the complexity of A5/1 state recovery can be lowered to only around 2^{32} cipher ticks with a negligible memory requirement. Soon after its proposal, Zhang's result is challenged severely: Derbez et al. point out in [12] that since the attack in [8] is not fully implemented, according to their practical verification, the non-randomness claimed by Zhang in [8] do not even exist. Therefore, Derbez et al. 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]. However, Derbez et al. has not fully implemented the Zhang's near collision attack either making it unknown whether the near collision method can still be regarded as an effective cryptanalysis tool for A5/1 state recovery.

It is also 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. Therefore, the complexities of both the guess-and-determine and near collision attacks should be reevaluated in a more detailed manner.

Our Contributions. In this paper, we revisit the memoryless attacks on A5/1 using both the guess-and-determine and the near collision methods. We first propose a new guessing technique: instead of guessing the clock bits directly, we guess the encoded move patterns so the linear systems can be constructed more efficiently. With this method, we are able to acquire a new guess-and-determine attack that can recover the initial state with a time complexity of $2^{43.91}$ simple operations. Then, we analyze the near collision attack given by Zhang in [8] only to find that the complexities 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 has a time complexity $2^{50.57}$ simple operations. The C++ source codes for computing the statistics in this paper are available online [3].

This paper is organized as follows. Sect. 2 provides brief introduction to the A5/1 stream cipher and general process of the two memoryless state-recovery attacks. Sect. 3 introduces a new guessing technique: the move pattern guessing technique. Sect. 4 introduces our move-guessing based guess-and-determine attack on A5/1. Sect. 5 revisits Zhang's near collision attack: we point out the mistakes made in [8] and provide new collision attacks with correct complexities. Sect. 6 conclude the paper and point out some future works.

2 Preliminary

In Sect. 2.1, we give a brief introduction to the keystream generation phase of the A5/1 stream cipher. In Sect. 2.2, we briefly review the main idea of Golic's guess-and-determine attack [1]. In Sect. 2.3, we give the general process of Zhang's near collision attack [8].

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

2.1 The Keystream Generation Procedure of A5/1

A5/1 has a 64-bit internal state consisting of 3 registers of sizes 19, 22, 23 respectively. We denote the 64-bit state at time t ($t = 0, 1, 2, \ldots$) as

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

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

$$\begin{aligned} maj^t =& (\boldsymbol{R1}^t[8] \cdot \boldsymbol{R2}^t[10]) \oplus (\boldsymbol{R1}^t[8] \cdot \boldsymbol{R3}^t[10]) \oplus (\boldsymbol{R2}^t[10] \cdot \boldsymbol{R3}^t[10]) \\ =& (\boldsymbol{s}^t[8] \cdot \boldsymbol{s}^t[29]) \oplus (\boldsymbol{s}^t[8] \cdot \boldsymbol{s}^t[51]) \oplus (\boldsymbol{s}^t[29] \cdot \boldsymbol{s}^t[51]) \end{aligned} \tag{2}$$

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

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

In the remainder of this paper, we uniformly use $\boldsymbol{s}^t[i]$ to represent the i-th bit of the whole state and avoid using $\boldsymbol{R1}[j], \boldsymbol{R2}[k], \boldsymbol{R3}[\ell]$'s.

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

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. For each step $i = 1, 2 \ldots$, whether the registers $\boldsymbol{R1}, \boldsymbol{R2}, \boldsymbol{R3}$ are updated or not depends on the three clock bits $\boldsymbol{s}^i[8, 29, 51]$. With the knowledge of $\boldsymbol{s}^i[8, 29, 51]$, each bit of $\boldsymbol{s}^{i+1}$

can be represented as a linear combination of $\boldsymbol{s}^i$ bits. For each guess $\boldsymbol{s}^i[8, 29, 51] = (\rho, \varrho, \sigma) \in \mathbb{F}_2^3$, the adversary can deduce 3 linear equations:

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

According to 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 clock bits $\boldsymbol{s}^i[8, 29, 51]$, the adversary can deduce 4 linear equations of state bits. Therefore, in [1], Golic propose a basic attack that guess $3t$ clock bits $\boldsymbol{s}^1[8, 29, 51] \ldots \boldsymbol{s}^t[8, 29, 51]$. Based on the output bits $z^0, \ldots, z^{t+1}$, 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 consisting of $1 + 3t + \frac{4}{3}t \geq 63.32$ equations can identify the correct guess uniquely with high probability. Although such a "high probability" is never actually evaluated, the complexity of Golic's attack is usually believed as $2^{3t} \geq 2^{43.15}$ steps where each step needs the solution of a linear system.

2.3 The General Process of Zhang's Near Collision Attack

Unlike Golic's recovering $\boldsymbol{s}^1$, Zhang's near collision attack in [8] aims at recovering the init state $\boldsymbol{s}^0$. They divide the 64 $\boldsymbol{s}^0$ bits into constraint part (CP) and the rest part (RP). The CP part consists of 33 bits related to the 5 output bits $z^0, \ldots, z^4$. The other 31 bits are all categorized as RP.

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 $z^0, \ldots, z^4$. Such a CP-recovery step can be summarized as the list-merging process in Fig. 1. In Fig. 1, the list $L_{z^i \ldots z^j}$ $(i < j)$ contains the state $\boldsymbol{s}^i$'s whose state bits are only partially known: for each $\boldsymbol{s}^i \in L_{z^i \ldots z^j}$, the known bits are at positions $\lambda \subseteq [0, 63]$ s.t. the knowledge of $\boldsymbol{s}^i[\lambda]$ can produce the consecutive output bits $z^i, \ldots, z^j$ following the A5/1 keystream generation process. Specifically, for a list $L_{z^i z^{i+1}}$ at Level 1, 2 consecutive keystream bits $z^i z^{i+1}$ can be related to at most 15 state bits at positions λ_0 in (7).

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

The number of known state bits for 3, 4 and 5 consecutive keystream bits will grow to at most 21, 27 and 33 respectively. Therefore, finally at Level 4, it is claimed by Zhang in [8] each element $\boldsymbol{s}^0 \in L_{z^0 z^1 z^2 z^3}$ should contain 33 bits. When merging two lists at the same level, the known bits at the overlapped positions are to be used as filters: only the elements share the same value at the overlapped positions can be merged as an element in the list in the next level. Since each element in the lists contains at most 33 known bits, it is estimated in

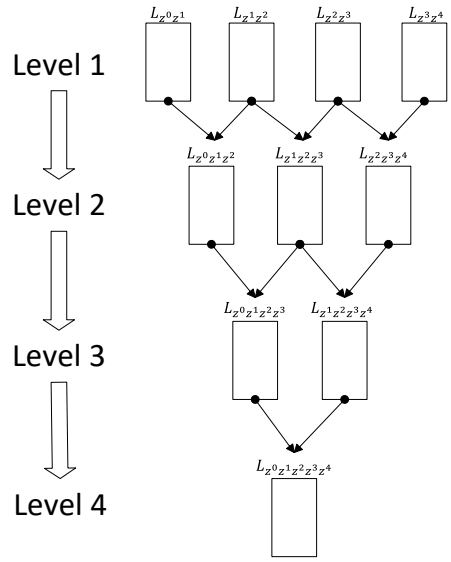

Fig. 1. The general process of Zhang's attack in [8]

[8] that the elements in all lists can be stored with 5 bytes of memory. However, the merging process in Fig. 1 is not fully implemented, only the process from Level 1 to 2 is implemented as can be seen from the source codes[4]. Without detailed analysis, Zhang theoretically estimated that that the whole list merging process could be finished with a time complexity $2^{28.3}$ cipher ticks and the final list $L_{z^0z^1z^2z^3z^4}$ only contains $2^{16.6}$ elements.

Another feature of Zhang's attack is the construction of the 4 initial lists: they employ the idea of near collision to construct the lists $L_{z^0z^1}, \ldots, L_{z^3z^4}$. They consider the low-hamming-weight internal state difference (ISD) $\Delta\boldsymbol{s}$ as follows:

$$\mathcal{D}_2 := \{\Delta\boldsymbol{s} | hw(\Delta\boldsymbol{s}) \leq 2 \text{ and } \Delta\boldsymbol{s}[i] = 0 \text{ for all } i \notin \lambda_1\} \tag{8}$$

Apparently, there are $\binom{15}{0} + \binom{15}{1} + \binom{15}{2} = 121$ elements in the ISD set $\mathcal{D}_2$ in (8). But only 99 ISDs in $\mathcal{D}_2$ can result in the 2-bit output difference `0x3`. Therefore, the adversary only needs to store such 99 low-hamming-weight ISD's in a table $\mathcal{T}$ defined in (9).

$$\mathcal{T} := \left\{\Delta\boldsymbol{s} \in \mathcal{D}_2 | \exists \boldsymbol{s}^0 \Rightarrow z^0z^1(\boldsymbol{s}^0) \oplus z^0z^1(\boldsymbol{s}^0 \oplus \Delta\boldsymbol{s}) = \texttt{0x3}\right\} \tag{9}$$

For a static 2-bit output z^0z^1, Algorithm 1 is proposed to generate list of states $L_{z^0z^1}$, making sure that all elements in the list can result in z^0z^1 directly. In Zhang's attack, the adversary sets the number limit $T = 4 \cdot 2^{15}/99 = 1323$ which results in the output list size 7963 and the correct state can be covered by $L_{z^0z^1}$ with probability $p_1 = 0.9835$. In order to improve the probability that the list

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

contains the correct state, a distilling process is further proposed. For positive integers η and ζ, the distilling process first generate $\eta \times \zeta$ lists with Algorithm 1; then, intersection and union operations are carried out as (10).

$$L_{z^0z^1} \leftarrow U(\eta,\zeta) = \bigcup_{i=1}^{\eta}\left(\bigcap_{j=1}^{\zeta} L_{z^0z^1}^{i,j}\right) \text{ where } L_{z^0z^1}^{i,j} \leftarrow \texttt{getList}(z^0z^1, T) \tag{10}$$

According to [8], when $\eta = 2, \zeta = 6$, the correct state can be covered by $L_{z^0z^1}$ with probability 0.9903.

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 $\hat{z}_0\hat{z}_1 \leftarrow z^0z^1 \oplus$ `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

After the CP-recovery phase, the adversary has already acquire $2^{16.6}$ $\boldsymbol{s}^0$'s in $L_{z^0z^1z^2z^3}$. 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 $3y$ clock bits in (11) and construct linear equations of unknown RP bits according both guesses and output bits $z^5, \ldots, z^{5+y-1}$.

$$\boldsymbol{s}^5[8,29,51], \ldots, \boldsymbol{s}^{5+y-1}[8,29,51] \tag{11}$$

They claim that the RP bits can be recovered with complexity approximately 2^{32} cipher tickes which is also the overall time complexity of the whole attack. But there is no further details on how the different y's can affect the complexities. It is unknown how effective the linear equation system can be in filtering wrong states. It is also unknown which y should be used to identify the uniquely correct internal state $\boldsymbol{s}^0$.

We are to show that Zhang's claims above are not accurate enough in Sect. 5.1. The analysis of y-effects are to be analyzed in detail in Sect. 5.2 only to find that Zhang's complexity analysis is underestimated due to the inaccurate parameters.

2.4 Unit of the Time Complexity

It is noticeable that the main operation of Golic's attack in [1] is solving the linear equation systems. Therefore, they use the time of solving the linear equation system solving by once as the unit of the time complexity. On the contrary, Zhang's near collision attack in [8] regard the cipher tick as the unit of time complexity. However, the linear equation system solving process in Zhang's attack is not strictly transformed to cipher ticks. In fact, the number of cipher ticks for solving a linear equation system depends on the numbers of both the variables and the equations in the system. The numbers of variables and equations are dynamic values depending on the bit guesses so an accurate transformation between the two units is quite difficult. Therefore, in this paper, we use Golic's time complexity unit in Golic's attack, which is the time of solving the linear equation system by once, and cipher ticks as time complexity units when talking about the attack in [8].

3 The Move Pattern Guesing Technique

In the Sect. 3.1, we introduce the concept of the move pattern and our move guessing technique. Sect. 3.2 compares the move guessing with the conventional clock guessing technique.

3.1 The Basic Concepts of the Move Pattern

For all 2^3 $\boldsymbol{s}^t[8, 29, 51]$ values, there are 4 possible move patterns denoted as Move 0, 1, 2 and 3. Each movement corresponds to 2 $\boldsymbol{s}^t[8, 29, 51]$ values and can also be represented as linear equations of state bits. Move 0-3 and their equations are defined as follows:

Move 0 `updateR1` in (3), `updateR2` in (4) and `updateR3` in (5) are all called. The $\boldsymbol{s}^t[8, 29, 51]$ values are $(0, 0, 0)$ and $(1, 1, 1)$. The linear equations are:

$$\begin{cases} \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[29] = 0 \\ \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[51] = 0 \end{cases} \Leftrightarrow \begin{cases} \boldsymbol{R1}^t[8] = \boldsymbol{R2}^t[10] \\ \boldsymbol{R1}^t[8] = \boldsymbol{R3}^t[10] \end{cases} \tag{12}$$

Move 1 Only `updateR2` and `updateR3` are called. The $\boldsymbol{s}^t[8, 29, 51]$ values are $(0, 1, 1)$ and $(1, 0, 0)$. The equations are:

$$\begin{cases} \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[29] = 1 \\ \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[51] = 1 \end{cases} \Leftrightarrow \begin{cases} \boldsymbol{R1}^t[8] = \boldsymbol{R2}^t[10] \oplus 1 \\ \boldsymbol{R1}^t[8] = \boldsymbol{R3}^t[10] \oplus 1 \end{cases} \tag{13}$$

Move 2 Only `updateR1` and `updateR3` are called. The $\boldsymbol{s}^t[8, 29, 51]$ values are $(1, 0, 1)$ and $(0, 1, 0)$. The equations are:

$$\begin{cases} \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[29] = 1 \\ \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[51] = 0 \end{cases} \Leftrightarrow \begin{cases} \boldsymbol{R1}^t[8] = \boldsymbol{R2}^t[10] \oplus 1 \\ \boldsymbol{R1}^t[8] = \boldsymbol{R3}^t[10] \end{cases} \tag{14}$$

Move 3 Only `updateR1` and `updateR2` are called. The $\boldsymbol{s}^t[8, 29, 51]$ values are $(1, 1, 0)$ and $(0, 0, 1)$. The equations are:

$$\begin{cases} \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[29] = 0 \\ \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[51] = 1 \end{cases} \Leftrightarrow \begin{cases} \boldsymbol{R1}^t[8] = \boldsymbol{R2}^t[10] \\ \boldsymbol{R1}^t[8] = \boldsymbol{R3}^t[10] \oplus 1 \end{cases} \tag{15}$$

We denote the movement $\boldsymbol{s}^t \to \boldsymbol{s}^{t+1}$ as $m^t \in \mathbb{F}_2^2 = \{0, 1, 2, 3\}$. So the movements before generating the output keystream bits $z^0, \ldots, z^t$ are $m^0, \ldots, m^t$. In our guess and determine attack, we first guess the movement m^t corresponding to $\boldsymbol{s}^t \to \boldsymbol{s}^{t+1}$ and maintains a linear equation set $\mathcal{BC}$ by adding new equations corresponding to the new movement m^t and the output z^t. For each step t, there are 3 linear equations: 2 are from one of (12), (13), (14), (15) according to the move guess and the rest is from the output z^t as

$$\boldsymbol{s}^{t+1}[18] \oplus \boldsymbol{s}^{t+1}[40] \oplus \boldsymbol{s}^{t+1}[63] = z^t \tag{16}$$

So each move guess can deduce 3 equations. In Sect. 4, we guess the moves $m^0, \ldots, m^{t-1}$ and maintain a linear equations system to distinguish the correct state $\boldsymbol{s}^0$ from the wrong ones.

3.2 Move Guessing v.s. Clock Guessing

Our move guessing method differs from the previous guessing strategies. In previous A5/1 cryptanalysis, the adversary guesses directly the 3 clock bits $\boldsymbol{s}^t[8, 29, 51]$ rather than the 2-bit move m^t. Apparently, our 2 move bits can be deduced from the 3 clock bits. Let $m^t[0, 1] = (\mu, \nu) \in \mathbb{F}_2^2$ and the corresponding clock bits are $\boldsymbol{s}^t[8, 29, 51] = (\rho, \varrho, \sigma) \in \mathbb{F}_2^3$, the two bits (μ, ν) can be deduced from (ρ, ϱ, σ) as (17)

$$\begin{aligned} \mu &= \bar{\rho}\bar{\varrho}\sigma \oplus \bar{\rho}\varrho\sigma \oplus \rho\bar{\varrho}\bar{\sigma} \oplus \rho\varrho\bar{\sigma} \\ \nu &= \bar{\rho}\bar{\varrho}\sigma \oplus \bar{\rho}\varrho\bar{\sigma} \oplus \rho\bar{\varrho}\sigma \oplus \rho\varrho\bar{\sigma} \end{aligned} \tag{17}$$

where $\bar{x}$ is the NOT operation equivalent to $x \oplus 1$. From the linear equation point of view, a 3-bit guess $\boldsymbol{s}^t[8, 29, 51] = (\rho, \varrho, \sigma)$ is naturally 3 equations. The 3 equations are also equivalent to 2 move-oriented equations and 1 bit value equation $\boldsymbol{s}^t[8] = \rho$. For example, if the clock bit values are of the form $\boldsymbol{s}^t[8, 29, 51] = (\rho, \rho, \rho)$, which corresponds to the Move 0, the 3 equations can also be regarded as adding $\boldsymbol{s}^t[8] = \rho$ to the Move 0 constraints in (12) as shown in (18).

$$\begin{cases} \boldsymbol{s}^t[8] = \rho \\ \boldsymbol{s}^t[29] = \rho \\ \boldsymbol{s}^t[51] = \rho \end{cases} \Leftrightarrow \begin{cases} \boldsymbol{s}^t[8] = \rho \\ \begin{cases} \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[29] = 0 \\ \boldsymbol{s}^t[8] \oplus \boldsymbol{s}^t[51] = 0 \end{cases} \end{cases} \tag{18}$$

Such an equivalence is also true for other clock bit values and move patterns. Adding the linear equations of the output in (16), we know that each 3-bit guess of the clock bits $\boldsymbol{s}^t[8, 29, 51]$ can deduce 4 equations while each of our 2-bit guess of m^t can deduce 3 equations. On average, 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. To make fair comparison between the two strategies, we apply both move and clock guessing strategy in the RP-recovery phase of Zhang's near collision attack in [8]. The results show that move guessing can result in slightly lowered complexities than its clock counterpart. Details can be seen later in Sect. 5.2 and Sect. 5.3.

4 Guess-and-Determine Attack based on the Move Guessing Technique

Instead of Golic's recovering $\boldsymbol{s}^1$, we propose a state recovery attack on A5/1 targeting at $\boldsymbol{s}^0$ so as to make a fair comparison with the near collision attack in [8]. As can be seen, the move equations (12), (13), (14), (15) and the output equation (16) correspond to the internal state at different time instances. But our attack is targeted to recovering the initial state $\boldsymbol{s}^0$. Therefore, we need to represent the internal states at different time instance t with $\boldsymbol{s}^0$ so that the equations are represented by $\boldsymbol{s}^0$ bits as well. With the knowledge of $m^0, \ldots, m^{t-1}$, $\boldsymbol{s}^t$ can be iteratively deduced from $\boldsymbol{s}^0$ and each $\boldsymbol{s}^t$ bit can be expressed as a linear combination of $\boldsymbol{s}^0$ bits. Since a linear combination of $\boldsymbol{s}^0$ bits can also be regarded as a inner-product of $\boldsymbol{s}^0$ and a 64-bit word $\boldsymbol{w} \in \mathbb{F}_2^{64}$, we can track each $\boldsymbol{s}^0, \ldots, \boldsymbol{s}^t$ bits with 64 64-bit words denoted as $W^0, \ldots, W^t \in (\mathbb{F}_2^{64})^{64}$. The initial W^0 corresponds to $\boldsymbol{s}^0$ is defined naturally as (19)

$$W^0 = (\boldsymbol{e}_0, \ldots, \boldsymbol{e}_{63}), \text{ where } \boldsymbol{e}_i[j] = \begin{cases} 1 & i = j \\ 0 & j \in [0, 63] \backslash \{i\} \end{cases} \text{ for } i = 0, \ldots, 63 \quad (19)$$

so as to make sure $W^0[i] \cdot \boldsymbol{s}^{\boldsymbol{0}} = \boldsymbol{e}_i \cdot \boldsymbol{s}^0 = \boldsymbol{s}^0[i]$ for $i = 0, \ldots, 63$. With $W^0, \ldots, W^{t-1}$, the word vector W^t can be deduced from W^{t-1} according to the movement $m^{t=1}$ by calling $W^t \leftarrow \texttt{UpdW}(m^{t-1}, W^{t-1})$ described in Algorithm 2. With the knowledge of W^t, each state bit of $\boldsymbol{s}^t$ can be uniformly expressed as a linear combination of $\boldsymbol{s}^0$ bits as

$$\boldsymbol{s}^t[i] = W^t[i] \cdot \boldsymbol{s}^0, i = 0, \ldots, 63 \quad (20)$$

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

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

where `getBC` is defined as Algorithm 3. The linear equations set $\mathcal{BC}$ can be regarded as a linear equation system in (21)

$$A\boldsymbol{x}^T = \boldsymbol{b}^T, \text{ where } A \in \mathbb{F}_2^{3t \times 64}, \boldsymbol{x} \in \mathbb{F}_2^{64}, \boldsymbol{b} \in \mathbb{F}_2^{3t} \quad (21)$$

and the solutions to the linear system in (21) is exactly the possible values of the internal state $\boldsymbol{s}^0$'s resulting in the output keystream bits $z^0, \ldots, z^{t-1}$. The number of solutions to (21) depends on the rank of the matrix A and its extended matrix

$$E = [A, \boldsymbol{b}^T] \tag{22}$$

If $rank(A) = rank(E)$, there will be $2^{64-rank(A)}$ solutions; otherwise, there will be no solutions at all. Apparently, the matrix A and the vector $\boldsymbol{b}$ are both deduced according to the move guesses $m^0, \ldots, m^{t-1}$ and the output bits $z^0, \ldots, z^{t-1}$. For the correct guess of $m^0, \ldots, m^{t-1}$, the relation $rank(A) = rank(E)$ holds constantly; for the wrong guesses, however, there should be a probability $1 - \alpha_t$ $(0 \leq \alpha_t \leq 1)$ that $rank(A) \neq rank(E)$. Based on such a finding, the general process of our state-recovery attack can be divided naturally into 3 main steps:

S1 Guess moves $m^0, \ldots, m^{t-1}$ and maintain a linear system in (21)
S2 Do the matrix rank test and discard the wrong guesses satisfying $rank(A) \neq rank(E)$
S3 Deduce 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

Therefore, the detailed description of our move-guessing-based state-recovery attack is as follows:

1. Query the A5/1 encryption oracle for ℓ keystream bits $z^0, \ldots, z^{\ell-1}$
2. Initialize an empty set $\mathcal{S}$ of $\boldsymbol{s}^0$ candidates
3. For some $t < \ell$, we guess the 2^{2t} movement values of $(m^0, \ldots, m^{t-1})$, acquire the equations $\mathcal{BC} \leftarrow \texttt{getBC}((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$ by calling Algorithm 3 (**S1**) and do the following substeps:
   (a) Deduce the A and $\boldsymbol{b}$ in (21) according to $\mathcal{BC}$ and compute the extended matrix E in (22)
   (b) Compute $rank(A)$ and $rank(E)$, if $rank(A) \neq rank(E)$, such a movement guess is wrong, go back to Step 2 for the next movement guess (**S2**)
   (c) 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 $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$
   (d) 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, there should be only 1 element in $\mathcal{S}$ which is exactly the correct internal state $\boldsymbol{s}^0$.

Complexity Analysis. In Step 3, there are 2^{2t} candidate moves $(m^0, \ldots, m^{t-1})$ and not all of them can pass the $rank(A) = rank(E)$ test Step 3.(b). We suppose that there is a positive number $0 \leq \alpha_t \leq 1$ that averaging $\alpha_t \cdot 2^{2t}$ candidate moves can pass the test. We further denote the averaging $rank(A)$ as β_t. With α_t, β_t, the averaging time complexity can be computed as follow:

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

We randomly select 2^{30} $((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$ pairs and do the 3.(b) test to compute the averaging α_t and β_t for t's. We find that when $t < 14$, α_t are larger than 0.5 ($\log \alpha_t \geq -1$) and $\beta_t \leq 3t$ so the overall complexity is constantly larger than 2^{50}. For $14 \leq t \leq 29$, the α_t, β_t and $Comp$ are listed in Table 1. As can be seen, the lowest time complexity appears at $t = 21$ with $Comp = 2^{43.91}$. As can be seen in Table 1, the order β_t has already climbed to almost 64 for $t = 27$. So we can safely set $\ell = 32$ to filter the wrong move guesses. According to our experiment, $\ell = 32$ is well enough to identify the correct $\boldsymbol{s}^0$ 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 (21) and (22). The memory complexity is only $O(t)$ and, to be more specific, $2 \cdot (64+1) \cdot 3t \leq 12480$ bits which is bounded by 2KB. So the memory complexity is practical and negligible in comparison with previous attacks.

The Effect of the Branching Technique. In both Golic's and Zhang's attacks, they claim to have used a "branching" technique when deducing equations [1,8]. The branching technique based on the fact that, with the current $\mathcal{BC}$, some of the state bits are known and the following clock bits can be deduced from such known bits. Such a technique can be applied at the beginning of Step 3 so that some of the 2^{2t} do not need to be guessed so Step 3.(a) may only need to process $\gamma 2^{2t}$ moves where $\gamma \leq 1$. Although different γ values may be deduces from different bit guesses, the number of candidate moves passing Step 3.(a) is still averaging $\alpha_t 2^{2t}$ so the size of γ is lower bounded by α_t. Therefore, taking the effect of branching technique into account, the complexity in (23) can reformulated as

$$Comp = 2^{2t+\log\gamma} + 2^{2t+64-\beta_t+\log\alpha_t} \geq 2^{2t+\log\alpha_t} + 2^{2t+64-\beta_t+\log\alpha_t} \tag{24}$$

Without doubt, the branching technique has some effects for lowering the complexity but, as can be seen in Table 1 it cannot change dominating factor of the overall complexities. In [1], based on the assumption that $\mathcal{BC}$ acts like a randomly constructed system, the α_t in Golic's attack is extremely small resulting in an overestimation to the effect of the branching technique. Such wrong evaluations are applied directly by Zhang in [8].

Table 1. The averaging α_t and β_t in (23) with 2^{30} random tests

t	β_t	$\log\alpha_t$	$\log Comp$	t	β_t	$\log\alpha_t$	$\log Comp$
14	41.959	-0.028	50.013	22	61.604	-3.971	44.417
15	44.868	-0.095	49.037	23	62.781	-5.915	46.055
16	47.683	-0.231	48.085	24	63.433	-8.420	48.006
17	50.381	-0.468	47.151	25	63.755	-11.173	50.001
18	52.955	-0.813	46.233	26	63.904	-14.060	52.000
19	55.409	-1.270	45.330	27	63.967	-17.027	54.000
20	57.734	-1.852	44.481	28	63.990	-20.021	56.000
21	59.860	-2.671	**43.914**	29	63.997	-23.117	58.000

5 Revisit Zhang's Near Collision Attack

As has been briefly mentioned in Sect. 2.3, the CP-recovery phase of the near collision attack has not been fully implemented: only the 1st step from Level 1 to 2 is implemented [8]. The details of the following CP-recovery steps and the whole RP-recovery phase are absent, leaving it unknown whether the whole attack can work as claimed.

To verify their attack, we fully implement it only to find several mistakes and the complexity evaluation of the whole attack is underestimated. Sect. 5.1 points out the inaccurate evaluations of several crucial parameters in [8]. Sect. 5.2 supplements Zhang's attack with all the missing details in the CP- and RP-recovery phases and gives a correct complexity evaluation to the original attack in [8]. Sect.5.3 replaces Zhang's clock-guess-based RP-recovery with our move-guess-based one to show the advantage of our technique.

5.1 Inaccurate Evaluations of Some Attack Parameters

There are 2 kinds of attack parameters being inaccurately evaluated in [8]: the p_1 used in the distilling phase along with the success probability deduced from p_1; the 4 parameters related to the complexities of the CP-Recovery process.

p_1 and the Success Probability. In [8], it is stated that, for $T = 4 \cdot 2^{15}/99$, a randomly constructed list $L_{z^0z^1} \leftarrow \texttt{getList}(z^0z^1, T)$ acquired by calling Algorithm 1 is of size 7963 and the correct internal state lies in $L_{z^0z^1}$ with probability $p_1 = 0.9835$. However, we repeat the experiment 10^6 times and the correct state lies in $L_{z^0z^1}$ for only 972436 times. So it is safe for us to claim that the actual p_1 is 0.9725: Zhang's evaluation in [8] is inaccurate. The reason why Zhang made such a mistake is unknown. Maybe their using RC4 as the source of randomness is not so qualified. Our experiments have tried various random generators including Snow-V [13], AES [14] etc. All these experiments reveal that the actual p_1 is 0.9725 rather than 0.9835.

With the corrected p_1, the probability for a distilled list U in (10) to cover the correct state should be reevaluated. Our corrected evaluation and Zhang's are both shown in Table 2. We read the source codes[5] corresponding to [8] carefully only to find that the $|U|$ parameter is wrong because of wrong implementation: 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. Since the attack can only succeed when the correct internal state lies in U, so the parameter `Prob.` in Table 2 is actually the success probability to the whole attack. Therefore, the success probability of Zhang's attack should be revised as well.

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

Table 2. 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. Since the attack in [8] has not been fully implemented, the estimation of 4 related parameters are inaccurate. Both our and Zhang's evaluations of the 4 parameters are listed in Table 3. We provide detailed explanations as follows.

Table 3. 4 Parameters Related to the CP-Recovery Process: Zhang Evaluation v.s. Ours

Parameter	Zhang's Eval. [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	33	30.14

†: quadratic time implementations

Since the 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, it is impossible to get an exact evaluation to the complexities without actually knowing the sizes of all the lists. According to our implementation, the sizes of the lists are as follows:

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

Therefore, the complexity of the merging process is dominated by Level 3 to 4 which is approximately $2^{20.46\times 2} = 2^{40.92}$ using the C++ implementation: far beyond $2^{28.3}$. The size of $L_{z^0z^1z^2z^3z^4}$ is $2^{24.21} > 2^{16.6}$. The reason is that the middle lists are used more than once in the merging phase, while in the original near collision attack, each list directly generated can only be used once.

The merging process in Fig. 1 takes two lists denoted as L_t and L_{t+1}. L_t contains the partial states of $\boldsymbol{s}^t$ while L_{t+1} consists of partial states of $\boldsymbol{s}^{t+1}$.

According to Sect. 3, a $\boldsymbol{s}^t$ should take a move $m^t \in \{0,1,2,3\}$ before reaching $\boldsymbol{s}^{t+1}$ and that move m^t is decided by the three clock bits $\boldsymbol{s}^t[8,29,51]$. So the merging step $\tilde{L}_t \leftarrow \texttt{merge}(L_t, L_{t+1})$ is as follows:

1. Initialize the merged list as empty $\tilde{L}_t \leftarrow \phi$.
2. For each $(\boldsymbol{s}^t, \boldsymbol{s}^{t+1}) \in L_t \times L_{t+1}$, do the following steps:
   (a) Identify the positions of the known bits in $\boldsymbol{s}^t$ denoted as $\lambda_0 \subseteq [0,63]$.
   (b) Determine the move m^t according to 3 known clock bits $\boldsymbol{s}^t[8,29,51]$.
   (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 $\hat{\boldsymbol{s}}^{\boldsymbol{t}}$ denoted as $\lambda_1 \subseteq [0,63]$
   (e) If $\hat{\boldsymbol{s}}^t[\lambda_0 \cap \lambda_1] = \boldsymbol{s}^t[\lambda_0 \cap \lambda_1]$, 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, any element $\boldsymbol{s} \in L$ should not only contain the value but the positions, denoted as λ, of the known bits as well. At Level 1, since all list are generated through Algorithm 1, all elements share the same known-bit positions in (7). But the following Example 1 shows that different moves will result in different known bit positions. Example 1 indicates that the known bits of the merged partial state $\tilde{\boldsymbol{s}}^t$ may not be exactly the 21 bits given in [8]: they are also likely to be subsets of the 21 bits.

Example 1. Let $(\boldsymbol{s}^0, \boldsymbol{s}^1) \in L_{z^0 z^1} \times L_{z^1 z^2}$. We have $\lambda_0 = \lambda$ in (7). If the move $m^0 = 0$ ($\boldsymbol{s}^0[8,29,51] \in \{(0,0,0),(1,1,1)\}$ according to (17) in Sect. 3.2), the known bit positions for $\boldsymbol{s}^1$ should be

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

and $\tilde{\lambda} = \lambda_0 \cup \lambda_1$ is of size $|\tilde{\lambda}| = 21$. If the move is $m^0 = 1$ ($\boldsymbol{s}^0[8,29,51] \in \{(1,0,0),(0,1,1)\}$), we have

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

and $|\tilde{\lambda}| = 19$.

In order to keep merging the lists in Level 2-4, $\tilde{\lambda}$ containing the known bit positions should also be stored which takes the same size of the partial states. Since the lists in Level 4 contain partial states of 33 bits, the $\tilde{\lambda}$'s are of the same size of 33 bits. So the elements in the lists requires at most $\lceil 2 \cdot 33/8 \rceil = 9$ bytes. Same with Example 1, the $\tilde{\lambda}$ for $\tilde{\boldsymbol{s}}^0 \in L_{z^0 z^1 z^2 z^3 z^4}$ are more likely to be subsets of the 33 bits. In fact, according to our experiments, the number of known bits of $L_{z^0 z^1 z^2 z^3 z^4}$ elements are usually $|\tilde{\lambda}| \approx 30.14$, which is below 33. $|\tilde{\lambda}|$ can only be 33 when $m^0 = \ldots = m^4 = 0$. Such an event can happen with probability 2^{-10}.

5.2 Near Collision Attack with Original Clock-Guess-Based RP-Recovery

We have supplemented all the details of the list merging operations in Sect. 5.1. The whole CP-recovery phase in Fig. 1 can now be carried out. At the end of

CP-recovery, the adversary have got the list $L_{z^0z^1z^2z^3z^4}$. Each $\tilde{\boldsymbol{s}}^0 \in L_{z^0z^1z^2z^3z^4}$ corresponds to a set $\tilde{\lambda} \subseteq [0, 63]$ containing the positions of known state bits and it guarantees that the first 5 keystream bits are exactly $z^0, \ldots, z^4$. The known bits $\tilde{\boldsymbol{s}}^0[\tilde{\lambda}]$ can deduce directly the first 5 moves $m^0, \ldots, m^4$. Therefore, the equations for each $\tilde{\boldsymbol{s}}^0$ is simply $\mathcal{BC} \leftarrow \texttt{getBC}((m^0, \ldots, m^4), (z^0, \ldots, z^4))$ adding the bit value constraints of $\tilde{\boldsymbol{s}}^0[\tilde{\lambda}]$. The number of equations is $15 + |\tilde{\lambda}|$.

Then, according to Zhang in [8], the RP part is to be recovered by guessing the unknown bits of $\tilde{\boldsymbol{s}}^0$ corresponding to the clock bits $\boldsymbol{s}^i[8, 29, 51]$ for $i = 5, 6, \ldots, t-1$ (equivalent to setting y in (11) as $y = t - 5$) and construct the corresponding linear equation system according to the clock bits and the output bits $z^0, \ldots, z^{t-1}$ as in (21). According to the analysis in Sect. 3.2, the clock guessing strategy is equivalent to adding bit value constraint to the corresponding move-oriented equations. Therefore, for the guesses of the $3(t-5)$ bits $\boldsymbol{s}^5[8, 29, 51], \ldots, \boldsymbol{s}^{t-1}[8, 29, 51]$, the number of deduced equations is $4(t-5)$. So the whole process of Zhang's near collision attack can now be summarized as follows:

1. Query the A5/1 encryption oracle for ℓ keystream bits $z^0, \ldots, z^\ell$
2. Run the merging process in Fig. 1 and acquire the list of candidates $L_{z^0z^1z^2z^3z^4}$ for the CP part of A5/1.
3. Initialize an empty set $\mathcal{S}$ of $\boldsymbol{s}^0$ candidates
4. For each $\tilde{\boldsymbol{s}}^0 \in L_{z^0z^1z^2z^3z^4}$ **(RP-Recovery)**
   (a) Deduce the 5 moves $(m_0, \ldots, m_4)$ and the known bit position set $\tilde{\lambda}$
   (b) For some $t \in [5, \ell - 1]$, we guess the $3(t-5)$ clock bits corresponding to $\boldsymbol{s}^5[8, 29, 51], \ldots, \boldsymbol{s}^{t-1}[8, 29, 51]$ and do the following substeps:
      i. Deduce the move guesses $m^5, \ldots, m^{t-1}$ and deduce the equations
      $$\mathcal{BC} \leftarrow \texttt{getBC}((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$$
      ii. For all $i \in \tilde{\lambda}$, add the linear equations $x_i = \tilde{\boldsymbol{s}}^0[i]$ to $\mathcal{BC}$
      iii. For $\boldsymbol{s}^i[8]$ $(i = 5, \ldots, t-1)$, add the bit value constraint $W^i[8] \cdot \boldsymbol{x} = \boldsymbol{s}^i[8]$ to $\mathcal{BC}$
      iv. Deduce the A and $\boldsymbol{b}$ in (21) according to $\mathcal{BC}$ and compute the extended matrix E in (22)
      v. 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 $\boldsymbol{s}^5[8, 29, 51], \ldots, \boldsymbol{s}^{t-1}[8, 29, 51]$
      vi. 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 $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$
      vii. 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 (25), there are $2^{24.21}$ candidate $\tilde{\boldsymbol{s}}^0$ in $L_{z^0z^1z^2z^3z^4}$. In Step 4.(b), there are 2^{3t-15} possible guesses and we assume that only $\alpha_t \cdot 2^{3t-15}$ of them can pass the $rank(A) = rank(E)$ test at Step 4.(b).v where $0 \leq \alpha_t \leq 1$. We denote the averaging $rank(A)$ as β_t. The analysis in Sect. 5.1, the merging process in Step 2 has complexity $2^{40.92}$ using the quadratic time

implementation, anyway this part is not dominated even with the linear time method in [8]. With α_t, β_t, the averaging time complexity can be computed as (27).

$$\begin{aligned} Comp &= 2^{40.92} + 2^{24.21+3t-15} + \alpha_t \cdot 2^{24.21+3t-15+64-\beta_t} \\ &= 2^{40.92} + 2^{9.21+3t} + 2^{73.21+3t+\log \alpha_t - \beta_t} \end{aligned} \tag{26}$$

Same with Sect. 3, we randomly select 2^{30} $\left(\tilde{\boldsymbol{s}}^{\mathbf{0}}, \boldsymbol{s}^5[8,29,51], \ldots, \boldsymbol{s}^{t-1}[8,29,51], (z^5, \ldots, z^m)\right)$ triplets and do the 4.(b).v test to compute the averaging α_t and β_t for different t's. For $5 \le t \le 21$, the α_t, β_t and $Comp$ are listed in Table 5. As can be seen, the complexities are constantly larger than 2^{52}, indicating that Zhang's complexity evaluation in [8] is inaccurate and is no better than the new guess-and-determine attack in Sect. 4. The lowest time complexity is $2^{52.159}$ and it appears at $t = 12$. The memory complexity is dominated by the size of $L_{z^0 z^1 z^2 z^3 z^4}$ which is $2^{24.21}$ according to (25). Zhang has already claimed that the attack can only succeed when the exact $\boldsymbol{s^i}$ lies in the corresponding list $L_{z^i z_{i+1}}$ for $i = 0, 1, 2, 3$ in Fig. 1. According to the Sect. 5.1, the success probability can be evaluated as $p_1^4 \approx 0.8942$.

The Effect of the Branching Technique. Same with the analysis in Sect. 4, the branching technique can also be applied to Step 4 and the complexity in (26) can be reformulated as

$$\begin{aligned} Comp &= 2^{40.92} + 2^{9.21+3t+\log \gamma} + 2^{73.21+3t+\log \alpha_t - \beta_t} \\ &\ge 2^{40.92} + 2^{9.21+3t+\log \alpha_t} + 2^{73.21+3t+\log \alpha_t - \beta_t} \end{aligned}$$

It does not affect the dominating factor of the whole attack so the overall complexity remains unchanged.

Table 4. The averaging α_t and β_t in (26) with 2^{30} random tests

t	β_t	$\log \alpha_t$	$\log Comp$	t	β_t	$\log \alpha_t$	$\log Comp$
6	33.443	-0.256	57.511	14	59.715	-0.989	54.646
7	37.424	-0.285	56.501	15	60.510	-1.455	56.560
8	41.423	-0.300	55.487	16	61.388	-2.586	58.223
9	45.421	-0.307	54.482	17	62.273	-4.668	60.387
10	49.415	-0.357	53.438	18	62.975	-7.000	63.233
11	53.470	-0.430	52.311	19	63.501	-9.483	66.213
12	56.494	-0.570	**52.159**	20	63.834	-12.293	69.210
13	58.436	-0.752	53.073	21	63.980	-15.476	72.210

5.3 Modified Near Collision Attack with Move-Based RP-Recovery

According to the relationship between our move guess and the conventional clock guess strategies revealed in Sect. 3.2, Zhang's clock-guess-based RP-recovery in Sect. 5.2 can be replaced with our move-guess-based strategy. We simply rewrite the RP-recovery phase of this modified attack as follows:

4. For each $\tilde{\boldsymbol{s}}^{\mathbf{0}} \in L_{z^0z^1z^2z^3z^4}$ **(RP-Recovery)**
   (a) Deduce the 5 moves $(m_0, \ldots, m_4)$ and the known bit position set $\tilde{\lambda}$
   (b) For some $t \in [5, \ell - 1]$, we guess the $2^{2(t-5)}$ movements $(m^5, \ldots, m^{t-1})$, we acquire the equations $\mathcal{BC} \leftarrow \texttt{getBC}((m^0, \ldots, m^{t-1}), (z^0, \ldots, z^{t-1}))$ and do the following substeps:
      i. For all $i \in \tilde{\lambda}$, add the linear equations $x_i = \tilde{\boldsymbol{s}}^0[i]$ to $\mathcal{BC}$
      ii. Deduce the A and $\boldsymbol{b}$ in (21) according to $\mathcal{BC}$ and compute the extended matrix E in (22)
      iii. Compute $rank(A)$ and $rank(E)$, if $rank(A) \neq rank(E)$, such a movement guess is wrong, go back to Step (b) for the next guess of moves $m^5, \ldots, m^{t-1}$
      iv. 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 $\hat{z}^0, \ldots, \hat{z}^{t-1}, \hat{z}^t, \ldots, \hat{z}^{\ell-1}$
      v. 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. In Step 4.(b), there are 2^{2t-10} candidate moves $(m^0, \ldots, m^t)$ and we assume that only $\alpha_t \cdot 2^{2t-10}$ moves can pass the $rank(A) = rank(E)$ test at Step 4.(b).iii where $0 \leq \alpha_t \leq 1$. We denote the averaging $rank(A)$ as β_t. According to (25), the size of $L_{z^0z^1z^2z^3z^4}$ is approximately $2^{24.21}$. The analysis in Sect. 5.1, the merging process in Step 2 has complexity $2^{40.92}$. With α_t, β_t, the averaging time complexity can be computed as (27).

$$\begin{aligned} Comp &= 2^{40.92} + 2^{24.21+2t-10} + \alpha_t \cdot 2^{24.21+2t-10+64-\beta_t} \\ &= 2^{40.92} + 2^{14.21+2t} + 2^{78.21+2t+\log \alpha_t - \beta_t} \end{aligned} \tag{27}$$

For $6 \leq t \leq 21$, the α_t, β_t and $Comp$ are listed in Table 5. As can be seen, the complexities are constantly larger than 2^{50} which is no better than our guess-and-determine attack in Sect. 4 but is lower than Zhang 's original one in Sect. 5.2. The lowest possible complexity is $2^{50.567}$ and it appears at $t = 16$. The memory complexity and the success probability are identical to those of Zhang 's original attack which are $2^{24.21}$ and 0.8942 respectively. It is noticeable that the β_t in Table 4 grows faster than that in Table 5, indicating that clock guess can result in a faster growth in the order of matrix in (21). But such a growth cannot guarantee a better filter when applied to state-recovery attacks: this is a fact that can only be discovered by solid experiments and accurate implementations.

The Effect of the Branching Technique. Same with the analysis in Sect. 4, the branching technique can also be applied to Step 4 and the complexity in (27) can be reformulated as

$$\begin{aligned} Comp &= 2^{40.92} + 2^{14.21+2t+\log \gamma} + 2^{78.21+2t+\log \alpha_t - \beta_t} \\ &\geq 2^{40.92} + 2^{14.21+2t+\log \alpha_t} + 2^{78.21+2t+\log \alpha_t - \beta_t} \end{aligned}$$

It does not affect the dominating factor of the whole attack so the overall complexity remains unchanged.

Table 5. The averaging α and β in (27) with 2^{30} random tests

t	β_t	$\log \alpha_t$	$\log Comp$	t	β_t	$\log \alpha_t$	$\log Comp$
6	31.957	-0.160	58.094	14	54.675	-0.524	51.016
7	34.683	-0.203	57.325	15	56.735	-0.848	50.643
8	37.584	-0.217	56.409	16	58.302	-1.415	**50.567**
9	40.548	-0.223	55.438	17	59.485	-2.202	50.789
10	43.515	-0.234	54.461	18	60.401	-3.194	51.427
11	46.457	-0.259	53.494	19	61.146	-4.403	52.635
12	49.371	-0.300	52.540	20	61.788	-5.736	54.330
13	52.159	-0.370	51.682	21	62.365	-7.287	56.238

6 Conclusion and Future Works

In this paper, we revisit 2 memoryless state-recovery methods on A5/1 stream cipher namely the guess-and-determine attack and the near collision attack. For the guess-and-determine attack, we propose a new guessing technique and provides a new attack with practically verified complexities. For the near collision attack, we revisit Zhang's attack in [8]. We point out the mistake in [8] and provide correct complexity evaluations. According to our analysis, the Zhang's near collision attack can work for A5/1 but does not have an advantage over the new guess-and-determine attack method.

About future works, it is noticeable that we propose a new guess-and-determine attack to recover the state $\boldsymbol{s}^0$ while Golic's original one targets at $\boldsymbol{s}^1$ [1]. According to the analysis in Sect. 4, the filtering strength of the deduced linear equation system $\mathcal{BC}$ may not be as good as that of a random system. Since Golic constantly regards the $\mathcal{BC}$'s of wrong guesses as random systems, it is highly likely that the time complexity of Golic's original guess-and-determine attack be wrongly evaluated, as pointed out by some previous literature. In order to acquire the correct time complexity evaluation, one has to practically compute the α_t and β_t's following Golic's guessing strategy, which is an obvious direction for future works.

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) `http://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. Derbez, P., Fouque, P., Mollimard, V.: Fake near collisions attacks. IACR Trans. Symmetric Cryptol. **2020**(4) (2020) 88–103
13. 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
14. Standards, N.: Specification for the advanced encryption standard (aes). FIPS-197 (2001)

Algorithm 2 Deduce the equation word set according to a movement

procedure `UpdW`(movement $m^t \in \{0,3\}$, words $W^t \in (\mathbb{F}_2^{64})^{64}$)
  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 (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 (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 (5)
  end if
  Return X
end procedure

Algorithm 3 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 $W^0 \leftarrow (\boldsymbol{e}_0, \ldots, \boldsymbol{e}_{63})$ according to (19)
  Initialize the linear equations set as empty: $\mathcal{BC} \leftarrow \phi$
  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**
    if $m^i = 0, 1, 2, 3$ **then**
      Update $\mathcal{BC}$ by adding the following equations corresponding to (12), (13), (14), (15):

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

      where $(\delta(m^i), \varrho(m^i)) = (0,0), (1,1), (1,0), (0,1)$ for $m_i = 0, 1, 2, 3$ respectively
    end if
    Deduce W^{i+1} according to W^i by calling $W^{i+1} \leftarrow$ `UpdW`(m^i, W^i) defined in Algorithm 2
    Update $\mathcal{BC}$ by adding the following linear equations corresponding to (16)

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

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