ORIGINAL RESEARCH PAPER

Yoyo trick on type-II generalised Feistel networks

Tao Hou | Ting Cui

PLA SSF Information Engineering University, Zhengzhou, China

Correspondence
Ting Cui, PLA SSF Information Engineering University, Zhengzhou, 450000, China.
Email: cutting_1209@hotmail.com

Abstract
This work presents a structural attack against the type-II generalised Feistel network (GFN) with secret internal functions. First, equivalent structures of the 7-round type-II GFN are provided, which helps reduce the first guess of the secret round functions. Then, two yoyo game distinguishers are simultaneously employed for these structures to reduce the data complexity by half. Based on these two distinguishers, it is found that the original yoyo game algorithm, proposed to attack the 5-round Feistel structure, is not suitable for these structures, owing to the characteristics of the yoyo game cycle. To solve this problem, the partial look-up table recycling technique is presented, which can utilise collision cycles with insufficient information. This technique performs better as the width of each branch ‘n’ grows. For yoyo game attacks, this study systematically investigates its cycle characteristics to determine the reason for the short collision cycle. For 7-round type-II GFNs, this work presents the first decomposition thus far, which can be executed within a time complexity of $O(2\times n^3 + \frac{2}{3})$ and a data complexity of $O(2\times n^3 + \frac{2}{3})$. We believe this work enriches the yoyo game attack and the application of type-II GFNs.

1 | INTRODUCTION

In block-cipher designs, the architecture not only directly influences safety but also affects implementation performance. The Feistel network (FN), substitution-permutation network (SPN) and the Lai–Massey scheme are three of the most famous structures. In [1], the authors extended the FN to generalised Feistel networks (GFNs). Later, Bogdanov et al. [2] provided a clear definition of the GFN and analysed its classification, security and efficiency. They proposed that there were only four non-contracting representatives in the class of four-line GFNs up to equivalence, namely the type-I and -II GFNs and their inverses. Among them, the type-II GFN is representative and can encrypt half plaintexts in one round. It has a smaller inner function scale than the FN. Until now, type-II GFNs have been used in numerous famous ciphers, such as Rivest cipher 6 [3], Sony’s CLEFIA [4] and HIGHT [5]. Considering their significance, therefore, the cryptanalysis of type-II GFNs is invaluable for understanding them better and exploring their weaknesses.

Most cipher designs apply the Kerckhoffs’ principle, which assumes that attackers know all the details of the cipher except for the key. More precisely, apart from the key, all the details of such ciphers are made public. However, in practice, quite a few ciphers choose to employ secret inner components, such as Khufu [6], Blowfish [7] and the Advanced Encryption Standard (AES) with a secret S-box [8]. In this kind of cipher, we cannot exploit the weaknesses of specific internal functions. Therefore, it is difficult to use traditional cryptanalysis, such as differential cryptanalysis [9] and linear cryptanalysis [10]. To take advantage of the well-established cryptanalysis tools used to estimate such ciphers, we must fill the gaps between such designs and the Kerckhoffs’ principle by first recovering all the details of the cipher.

Because we must perform $2^n$ exhaustion to confirm a single $n$-bit secret bijection, when $n$ increases to 16 or greater, it is impractical to recover the secret functions via exhaustion. Therefore, faster recovery would be quite helpful in evaluating the security of secret-component-based ciphers.

The method of recovering inner secret functions is called a structural attack [11]. Its target is the secret internal function. For most cases, we simply put forward the look-up tables (LUT) of such functions. The following conditions are available to the attackers:

- The architecture of such cipher (e.g. FN, SPN, or other structures).
- The number of cascade rounds.
- The encryption/decryption oracle of the cipher.
Related work. In 2001, Biryukov and Shamir proposed a multiset attack for an SPN with 128-bit plaintexts and 8-bit S-boxes [11]. They recovered five layers (i.e., the SASAS scheme, where the substitutions and permutations are all key-dependent) using $2^{16}$ chosen plaintexts and a time complexity of $2^{26}$. Later in 2011, Boghloff et al. proposed a slender-set attack to recover the secret S-boxes of the PRESENT-like cipher, where the permutation layers consisted of a fixed-set permutation [12]. In FSE 2014, Guo-qiang Liu et al. enriched the linear cryptanalysis against PRESENT-like ciphers, which is the improvement and sequel of Boghoff’s attack. And in 2015, they reconstructed the differential distinguisher to make slender-set attacks more efficient [13, 14]. In 2015, Tiessen et al. showed that employing a secret S-box did not significantly increase the security of 5-round AES. It was found to still be vulnerable to integral attacks [15].

At SAC 2015, Biryukov et al. recovered 7-round FN inner functions using a new method, ‘yoyo game’ [16], which was originally proposed as a key-recovery attack against the 16-round Skipjack [17]. In their work, they constructed a 5-round yoyo game distinguisher and extended it to recover the 7-round Feistel with additional guesses. They also compared other structural attacks in [18–20]. To the best of our knowledge, yoyo is the most effective recovery attack against the 5-round FN structure. Later, Cui et al. extended yoyo on a 10-round type-I GFN in 2019 [21]. They improved yoyo game attacks by rejecting a large group of start guesses with one single wrong guess. This method greatly helped reduce complexity. For the SPN, the yoyo trick was also effective. In 2015, Ronjom et al. employed this attack on the 5-round AES [22], and Bardeh et al. extended this attack to the secret-S-box AES [23] in 2019. Recently at EUROCRYPT 2020, Dunkelman et al. proposed a retracing attack, which can be seen as a generalised yoyo game with the lowest computational complexity for both usual AES and the secret-S-box AES [24].

Because the yoyo game attack performs quite well when decomposing the FN and SPN, it is of practical significance to investigate the security of other structures under such attacks. In this work, we focus on the problem of 7-round decomposition against the type-II GFN. It is worth noting that this 7-round attack can be extended to a 9-round attack using a very similar technique, such as that of [16]. We are only concerned about decomposing the 7-round structure here.

Our Contribution. First, we construct two yoyo game distinguishers for the 7-round type-II GFN, which can be used to establish linear equations for the secret functions. For one yoyo cycle, we independently detect that which distinguishes this cycle. Thus, the data complexity can be reduced by half. Furthermore, this method applies to other structures as long as multiple distinguishers exist.

Second, we study the characteristics of yoyo game cycles. Because the choice of mappings acting on the plaintext and ciphertext pairs, denoted by $\phi_p$ and $\rho_p$, respectively, determine the length of the yoyo game cycle or the number of efficient equations, to a great extent, selecting independent $\phi_p$ and $\rho_p$ that provide longer yoyo cycles with a much higher probability is a more rational choice. This also reveals significant technical differences between the yoyo game against the FN and type-II GFN. Namely, the former uses independent mappings, whereas the latter does not.

Finally, owing to the differences between the FN and type-II GFN, it is difficult to obtain the whole recovered inner functions using the original yoyo game when $n \geq 8$. We propose the partial LUT recycling technique to revitalise the incomplete LUTs. For the 7-round type-II GFN, the partial LUT survives when it passes four checks at least. Then, the yoyo game is continued based on partial LUTs until we obtain the complete LUT. The experimental results show that when $n < 6$, our improvement is not significant. However, when $n \geq 6$, our improved algorithm overcomes the problem of insufficient equations and recovers the LUTs having less computation and fewer data than the original one. When $n$ increases, the original yoyo game will be invalid. However, our attack still performs well. As a result, the recovery of the 7-round type-II GFN can be executed with a computational complexity $O(2^{3n} + 3)$ and a data complexity $O(2^{3n} + 3)$ by our improved yoyo game attack.

Organisation. The remainder of this work is organised as follows: Section 2 provides some preliminaries. Section 3 presents some properties of the 7-round type-II GFN. In Section 4, we focus on the outermost inner-function recovery. The rest of the 7-round type-II GFN is recovered in Section 5. Finally, Section 6 concludes this work.

2 | PRELIMINARIES

This section shows some notations and concepts that will be used throughout the rest of this work.

- $P = \{p_1, p_2, p_3, p_4\}$: the plaintext of type-II GFN,
- $C = \{c_1, c_2, c_3, c_4\}$: the ciphertext of type-II GFN,
- $F_i, G_i$: $i$-th secret-round functions,
- $E$: encryption oracle of type-II GFN,
- $D$: decryption oracle of type-II GFN,
- $\oplus$: bitwise XOR,
- $n$: width of one branch of the type-II GFN,
- $I$: identical transformation.

Type-II GFN. Let $P = \{p_1, p_2, p_3, p_4\} \in \{0,1\}^4$ be the input of the $i$th round. The $i$th round output of type-II GFNs is defined by

$$(F_i(p_1) \oplus p_2, p_3, p_4 \oplus G_i(p_3), p_1),$$

where $F_i$ and $G_i$ denote the round functions over $\{0,1\}^n$ (see Figure 1).

Since the shuffle layer of the last round does not affect the attack described in this work, we omit the switch of the last round for simplicity.

**Figure 1** The $i$th round function of type-II generalised Feistel networks
3 | PROPERTIES OF THE 7-ROUND TYPE-II GFN

3.1 | Equivalent structure of the 7-round type-II GFN

In [16], Biryukov et al. mentioned that there existed equivalent structures for the 5-round FN. In such structures, even if the internal functions are different, the encryption results are always consistent. These equivalent structures are quite helpful in reducing the computational cost. For the 7-round type-II GFN, we have obtained its equivalent structures.

Definition 1 Let \( T_1 \) and \( T_2 \) be two iterated block-cipher structures over \( \{0, 1\}^n \). If we always have \( T_1(x) = T_2(x) \) for any input, \( T_1 \) and \( T_2 \) are said to be equivalent.

For the type-II GFN, we obtained a kind of equivalent structure, which is concluded by the next theorem.

Theorem 1 Let \( T \) be a 7-round type-II GFN whose inner function is \( R_1(x_1, x_2, x_3, x_4) = (x_2 \oplus F_1(x_1), x_2, x_4) \). For any \( \text{con}_{1, 2, 3, 4} \in \{0, 1\}^n \), construct

### Equivalent structure of the 7-round type-II generalised Feistel network

\[
F_1'(x) = \begin{cases} 
F_1(x) \oplus \text{con}_1, & i = 1; \\
F_2(\text{con}_1 \oplus x), & i = 2; \\
F_3(x) \oplus \text{con}_3, & i = 3; \\
F_4(x), & i = 4; \\
F_5(x) \oplus \text{con}_2, & i = 5; \\
F_6(\text{con}_2 \oplus x), & i = 6; \\
F_7(x) \oplus \text{con}_3, & i = 7.
\end{cases}
\]

and

\[
G_1'(x) = \begin{cases} 
G_1(x) \oplus \text{con}_4, & i = 1; \\
G_2(\text{con}_4 \oplus x), & i = 2; \\
G_3(x) \oplus \text{con}_1, & i = 3; \\
G_4(x), & i = 4; \\
G_5(x) \oplus \text{con}_3, & i = 5; \\
G_6(\text{con}_3 \oplus x), & i = 6; \\
G_7(x) \oplus \text{con}_2, & i = 7.
\end{cases}
\]

If \( T' \) is another type-II GFN whose inner function is

\[
R_1'(x_1, x_2, x_3, x_4) = (x_2 \oplus F_1'(x_1), x_3, x_4) \oplus G_2'(x_3)),
\]

then \( T' \) is equivalent to \( T \).

Proof: Referring to Figure 2, we conclude that, for any choice of \( \text{con}_{1, 2, 3, 4} \in \{0, 1\}^n \), the encryptions will receive the same ciphertexts from the same arbitrary plaintexts.

According to Theorem 1, for a 7-round type-II GFN, we can freely fix one single arbitrary entry of \( F_1 \) without changing the encryption result.

Corollary 1 In the 7-round type-II GFN, if we fix \( F_1(c) = 0 \) for any \( c \in \{0, 1\}^n \), then we can always find a structure equivalent to the original one.

Proof: Suppose that the real value of \( F_1(c) \) is \( a \), then we end the proof by setting \( \text{con}_{1, 2, 3, 4} = (a, 0, 0, 0) \).

3.2 | Yoyo game property of the type-II GFN

Let us consider the differential trail of encryption. If the difference in positions \( (A, B, D) \) (see Figure 3) is equal to \( (0, 0, \gamma) \) for some \( \gamma \neq 0 \), we call this differential trail the \( L' \)-trail. Similarly, if the difference in positions \( (a, b, d) \) is equal to \( (0, 0, \gamma(\gamma \neq 0)) \), we call it \( R' \)-trail.

Definition 2 If the encryption of a plaintext pair \( (P, P') \) follows the \( L'-trail (R'-trail) \), this plaintext pair is said to be \( PL'-connected (PR'-connected) \). Similarly, if the decryption of a ciphertext pair \( (C, C') \) follows the \( L'-trail (R'-trail) \), we say that this ciphertext pair is \( CL'-connected (CR'-connected) \).

Theorem 2 The pair of plaintexts \( (P, P') \) is \( PL'-connected \) if and only if it satisfies the following equations:
decr yption direction, we can obtain (2), we conclude the next property.

\[(P, P') = ((p_1, p_2, p_3, p_4), (p_1', p_2', p_3', p_4'))\]

and

\[(C, C') = ((c_1, c_2, c_3, c_4), (c_1', c_2', c_3', c_4')).\]

We define two mappings \(\phi_f\) and \(\rho_f\) over \(\{0,1\}^n\) such that

\[
\phi_f(P, P') = ((p_1', p_2' + \gamma, p_3, p_4), (p_1, p_2 + \gamma, p_3', p_4'))
\]

and

\[
\rho_f(C, C') = ((c_1, c_2, c_3', c_4 + \gamma), (c_1', c_2', c_3, c_4 + \gamma)).
\]

If \((P, P')\) is \(PL_f\)-connected, then \(\phi_f(P, P')\) is also \(PL_f\)-connected. Similarly, if \((C, C')\) is \(CL_f\)-connected, then \(\rho_f(C, C')\) is also \(CL_f\)-connected.

**Proof:** The first equation in (1) can be transformed into

\[
F_1(p_1) + p_2 = F_1(p_1') + p_2' + \gamma
\]

and

\[
F_1(p_1') + p_2' = F_1(p_1) + p_2 + \gamma.
\]

From these two equations above, we can transform (1) into

\[
\begin{align*}
F_1(p_1') + F_1(p_1) &= (p_2' + \gamma) + (p_2 + \gamma) + \gamma, \\
G_2(p_1 + G_1(p_1)) + G_2(p_1' + G_1(p_1')) &= p_1' + p_1, \\
F_3(p_1 + F_2(p_1 + F_1(p_1))) + F_3(p_1' + F_2(p_1' + F_1(p_1'))) &= p_1 + p_1' + G_1(p_3) + G_1(p_3') \tag{2}
\end{align*}
\]

Consider the plaintext pair

\[(\hat{P}, \hat{P'}) = (p_1', p_2' + \gamma, p_3, p_4), (p_1, p_2 + \gamma, p_3', p_4')).\]

According to Theorem 2, \((\hat{P}, \hat{P'})\) is \(PL_f\)-connected.

We can also rewrite (2) as

\[
\begin{align*}
G_1(c_3) + G_2(c_3) &= c_4 + c_3' + \gamma, \\
G_6(c_2 + F_3(c_1)) + G_6(c_2' + F_3(c_1')) &= c_1' + c_3', \\
G_3(c_1 + F_6(c_4 + c_3) + G_3(c_1' + F_6(c_4' + G_7(c_1')))) &= c_2' + F_1(c_1') + F_1(c_1').
\end{align*}
\]
Likewise, \((\hat{C}, \hat{C'}) = [(c_1, c_2, c_3, c_4 \oplus \gamma), (c'_1, c'_2, c_3, c_4 \oplus \gamma)]\) is \(CL_r\)-connected when \((C, C')\) is \(CL_r\)-connected.

Similarly, we obtain the equivalent equations for the pairs satisfying the right trail in Figure 3.

**PR\(_r\)**-equivalent equations

\[
\begin{align*}
\bigg\{ & \; G_1(p_3) + G_1(p'_3) = p_4 + p'_4 + \gamma, \\
& \; \bigg\}; \\
\bigg\{ & \; F_2(p_2 + F_1(p_1)) + F_2(p'_2 + F_1(p'_1)) = p_3 + p'_3, \\
& \; \bigg\}; \\
\bigg\{ & \; G_3(p_1 + G_2(p_4 + G_1(p_3))) + G_3(p'_1 + G_2(p'_4 + G_1(p'_3))) \\
& \; = p_2 + p'_2 + F_1(p_1) + F_1(p'_1). \\
\end{align*}
\]

**CR\(_r\)**-equivalent equations

\[
\begin{align*}
\bigg\{ & \; F_7(c_1) + F_7(c'_1) = c_2 + c'_2 + \gamma, \\
& \; \bigg\}; \\
\bigg\{ & \; F_0(c_4 + G_7(c_1)) + F_0(c'_4 + F_7(c'_1)) = c_1 + c'_1, \\
& \; \bigg\}; \\
\bigg\{ & \; F_3(c_1 + G_6(c_2 + F_3(c_1))) + F_3(c'_1 + G_6(c'_2 + F_3(c'_1))) \\
& \; = c_4 + c'_4 + G_7(c_3) + G_7(c'_3). \\
\end{align*}
\]

By analysing these two systems of equations, we conclude a new property. The process for proving this property is parallel to that of Property 1.

**Property 2** Let \(\gamma \in \{0,1\}^n\),

\[(P, P') = ((p_1, p_2, p_3, p_4), (p'_1, p'_2, p'_3, p'_4))\]

and

\[(C, C') = ((c_1, c_2, c_3, c_4), (c'_1, c'_2, c'_3, c'_4)).\]

We define two mappings, \(\phi\'_r\) and \(\rho\'_r\), over \(\{0,1\}^{8n}\), such that

\[\phi\'_r(P, P') = ((p_1, p_2, p_3, p_4, p'_4 + \gamma), (p'_1, p'_2, p_3, p_4 + \gamma))\]

and

\[\rho\'_r(C, C') = ((c'_1, c'_2 + \gamma, c_3, c_4), (c_1, c_2 + \gamma, c'_3, c'_4)).\]

If \((P, P')\) is \(PR_r\)-connected, then \(\phi\'_r(P, P')\) is also \(PR_r\)-connected. Similarly, if \((C, C')\) is \(CR_r\)-connected, then \(\rho\'_r(C, C')\) is also \(CR_r\)-connected.

With the preparations above, we start our decomposition in the remainder of this article.

### 4. Basic idea of the attack

The basic idea of our attack is that, starting from a \(PL_r\)-connected pair \((P_0, P'_0)\), by Property 1, \(D \circ \rho \circ E \circ \phi\_r(P, P')\) is also a new \(PL_r\)-connected pair. Therefore, we can obtain many \(PL_r\)-connected pairs by repeating a process, which is also referred to as the ‘yoyo game’. According to Theorem 2, we can obtain one linear equation related to \(F_1\) from one single \(PL_r\)-connected pair. After collecting enough equations, the secret \(F_1\) can be recovered by the Gauss elimination method. In a similar way, we can recover \(F_7\), \(G_1\) and \(G_7\).

The main idea of recovering \(F_1\) can be roughly concluded by the following steps:

Step 1. Start from a random plaintext pair \((P_0, P'_0)\) (in the rest of this work, we call it a **start point**). Set \(\mathcal{L} \leftarrow \emptyset\) and \(i \leftarrow 0\).

Step 2. Add the equation, \(\phi\_r(P_{i-1}) + F_1(p'_i) = p'_3 + p_2 + \gamma\)
which is extracted from the pair \((P_{i-1}, P'_i)\), to \(\mathcal{L}\).

Step 3. Compute \((P_{i+1}, P'_{i+1}) = D \circ \rho \circ E \circ \phi\_r(P_i, P'_i)\). If \((P_{i+1}, P'_{i+1}) \neq (P_0, P'_0)\), set \(i \leftarrow i + 1\) and go to Step 2; else go to Step 4.

Step 4. Get the entries of \(F_1\) by solving all the linear equations in \(\mathcal{L}\). If \(F_1\) is fully recovered, output \(F_1\) and end the yoyo trick. Otherwise, change a new start point and continue the above steps.

For the sake of clarity, Steps 1–3 comprise the equation-accumulating phase, and Step 4 is the equation-solving phase.

In the equation-accumulating phase, if a newly computed plaintext pair \((P_0, P'_0)\) has already appeared from the previous yoyo trick, we will receive an exactly identical state after its subsequence. None of the subsequent plaintext pairs offer new equations. Therefore, we set \((P_{i+1}, P'_{i+1}) = (P_0, P'_0)\) in Step 3 as the termination condition.

In the equation-solving phase, the main task is to obtain the LUT of \(F_1\) from \(\mathcal{L}\). Any possibility falls into one of the following three cases:

1) The equations have no solution. The only possibility is that the start point is not \(PL_r\)-connected (in the remainder of this work, we call it a **wrong point**; otherwise it is a **right point**). Such a wrong point will lead to contradictory entries. For example, conflicting solutions \(F_1(0) = 0\) and \(F_1(0) = 1\) may appear simultaneously.

2) The equations have more than one solution. This includes two cases: one in which the start point is true, but we cannot collect enough equations (i.e. \(\mathcal{L}\) is not full rank). The other is where the start point is wrong, but no contradictory entry is yet detected.

3) The equations have a unique solution, which means that \(F_1\) is fully recovered.

Next, we take a closer look at the equation-solving phase. Note that, for each of these equations in set \(\mathcal{L}\), there exist
exactly two variables on the left-hand side. In [16], Biryukov et al. provided a very efficient way to solve the above equations. Hence, we provide an overview of their excellent idea.

A direct application of Biryukov’s yoyo solver.

The main idea for solving such equations is to launch a step-by-step method while playing the yoyo trick. Since there are always two variables on the left side of the equations, as soon as one is determined, the other is also determined. In the yoyo trick, we can deal with the new equation, which is extracted from the newly generated point. First, we need to prepare a set denoted by \( \mathcal{L} \) to store the equations and choose a starting point, \((P_i, P'_i) = (\langle q_1, q_2, q_3, q_4, q_5, q_6, q_7 \rangle, \langle q'_1, q'_2, q'_3, q'_4 \rangle)\). Then, we fix \( F_1(p_i) = 0 \) and \( F_1(p'_i) = q'_2 \oplus q'_2 \oplus \gamma \) in advance (according to Corollary 1, this step will not affect the correctness of our attack). Next, we iterate the yoyo game for one step to get a new point and extract one new equation. Without loss of generality, we assume that the current equation extracted is \( F_1(p_i) \oplus F_1(p'_i) = p'_2 \oplus p'_2 \oplus \gamma \).

This falls into one of the three cases:

- **Case 1** Neither \( F_1(p_i) \) nor \( F_1(p'_i) \) was fixed before.
- **Case 2** Either \( F_1(p_i) \) or \( F_1(p'_i) \) has been fixed.
- **Case 3** Both \( F_1(p_i) \) and \( F_1(p'_i) \) have already been fixed.

In Case 1, such an equation cannot provide enough information to fix these two variables, and we add this equation to \( \mathcal{L} \).

In Case 2, without loss of generality, suppose that the fixed variable is \( F_1(p_i) \). Then, the other variable, \( F_1(p'_i) \), can be determined directly. We update the LUT of \( F_1 \) by adding \( F_1(p'_i) \) to the \( p'_i \)th entry. Next, we check whether \( F_1(p'_i) \) can determine any uncertain value of \( F_1 \) by equations in the \( \mathcal{L} \) set and turn to dispose these related equations in the same way. Finally, all the disposed equations from \( \mathcal{L} \) are removed. This process is repeated until we can no longer fix any new entry of \( F_1 \).

In Case 3, we only need to compare \( p'_2 \oplus p'_2 \oplus \gamma \oplus F_1(p_i) \) with the existing value \( F_1(p'_i) \) in the LUT. If there is a contradiction between the existing one and the newly computed one, it means that the start point is incorrect.

To illustrate the procedures in Case 2 more clearly, we provide an example as follows:

**Example 1** Let \( n = 3 \); the function, \( F_1 \), is now partially recovered (i.e. \( F_1(0) = 7, F_1(3) = 4, F_1(5) = 0, F_1(7) = 1 \), denoted by \( F_1 = (\langle 7, ?, ?, 4, ?, ? \rangle, 0, 0, ?, 1) \). There are two equations in \( \mathcal{L} : F_{1}(1) \oplus F_{1}(2) = 3, F_{1}(4) \oplus F_{1}(6) = 6 \). According to the classification rule above, the newly extracted equation, \( F_{1}(3) \oplus F_{1}(6) = 1 \), falls into Case 2. Therefore, we need to update the LUT of \( F_1 \) by checking \( \mathcal{L} \).

First, \( F_1(6) \) can be determined instantly by \( F_1(6) = F_1(3) \oplus 1 = 5 \). Then, we check all equations in \( \mathcal{L} \) and find that \( F_1(4) \oplus F_1(6) = 6 \) is related to \( F_1(6) \). Therefore, we renew the LUT by fixing \( F_1(4) = F_1(6) \oplus 6 = 3 \) and removing this equation from \( \mathcal{L} \). Now, \( \mathcal{L} = \{ F_{1}(1) \oplus F_{1}(2) = 3 \} \), and the LUT is updated into \((7, ?, ?, 4, 3, 0, 5, 1)\). For the newly updated \( \mathcal{L} \), there is no equation containing \( F_1(6) \) or the newly fixed \( F_1(4) \) as its variable. Thus, we can no longer expand \( F_1 \).

According to the three cases above, we can constantly update the LUT by checking the newly introduced equations. When we meet with a contradiction in our updating process, we change a new start point and perform this algorithm again until we obtain the complete LUT. For a system of \( 2^r \) equations, we use the quick-sort algorithm on \( \mathcal{L} \) (because of each equation having two variables, we need two lists for sorting each variable) and require \( 2 \log_2 n \) lookups at most for each equation. In this way, the computational complexity is \( O(2^{r^2} + 1) \) lookups.

Based on the algorithm by Biryukov et al., we perform experiments for the 7-round type-II GFN. However, we find one strange phenomenon: as \( n \) grows, there is increasing difficulty in finding a fully recovered \( F_1 \) by using the original yoyo [16]. However, for the 5-round FN, no matter how much \( n \) increases, almost all right LUTs are fully recovered. Therefore, it may not be the insufficiency of the yoyo itself but the lack of universality of the original yoyo. To address this problem, we present a theoretical analysis in Section 4.2. Then, we develop an improved algorithm in Section 4.3.

### 4.2 Characteristic of the yoyo game cycle

Let us recall the procedures of the yoyo game, shown in Figure 4. We start from a plaintext pair \((P_0, P'_0)\) and repeatedly use \( \phi_r, \mathcal{E}, \rho_r \) and \( \mathcal{D} \) to obtain a sequence of plaintext/ciphertext pairs \((P_0, P'_0), (P_1, P'_1), (P_2, P'_2), \ldots/(C_0, C'_0), (C_1, C'_1), (C_2, C'_2), \ldots\). For simplicity, we use \([P_i] \) and \([C_i] \), respectively, to denote \((P_1, P'_1) \) and \((C_1, C'_1)\). We also use \([P_i]_\gamma \) and \([C_i]_\gamma\), respectively, to denote \( \phi_r(P_2, P'_2) \) and \( \rho_r(C_2, C'_2) \). The aim of this subsection is to determine how many equations can be obtained from a full iteration of the yoyo game on average. The main tool we use is the collision cycle.

**Definition 3** Let \( \Omega \) be a finite set and \( f, g \) be two bijections defined on \( \Omega \). For any \( a \in \Omega \), we define its collision cycle as \( O_a = (a, f(a), g \circ f(a), \ldots, (g \circ f)^{-1}(a), f \circ (g \circ f)^{-1}(a)) \),
where $r$ is the minimum positive integer such that $a = (g \circ f)^r(a)$. We call $r$ the collision length and $a = (g \circ f)^r(a)$ a collision.

**Example 2** In the yoyo game, we define two mappings $f := E \circ \phi_r$ and $g := D \circ \rho_r$ based on the pre-defined four bijections, namely, $D, \rho_r, E$ and $\phi_r$. We choose a plaintext pair $(P, P')$ and play a yoyo game. Then, all plaintext/ciphertext pairs of this game are in the collision cycle, $O_{\rho, P'}$, that is,

$$(P, P'), f(P, P'), g \circ f(P, P'), \ldots, f \circ (g \circ f)^{-1}(P, P').$$

If we define $\mathcal{Y} := g \circ f$ ($\mathcal{Y}$ is definitely a bijection leading a branchless yoyo cycle), then the collision length value, that is,

$$\min\{r : \mathcal{Y}^r(P, P') = (P, P'), \text{ for some } r > 0\}$$

denotes the number of plaintext pairs in this game.

The original yoyo game [16] is somewhat inefficient for the type-II GFN because the collision length is often too short to yield a sufficient number of equations to obtain a complete LUT. Most of the time, we only obtain partial LUTs from such short collisions. Owing to the pre-setting rules, we must abandon these incomplete candidates. We find that there are always two special pairs in these short collision cycles, such that $[P_i] = [P_i]_r$ or $[C_i] = [C_i]_r$, which may be the main reason for the short length. We describe such points in Definition 4.

**Definition 4** Let $h$ be a mapping over a finite set $\Omega$, then we call $x \in \Omega$ a fixed point of $h$ if $h(x) = x$.

We have already defined two functions in the yoyo game:

$$\phi_r(P, P') = ((p_1', p_2' \oplus r, p_3', p_4'), (p_1, p_2 \oplus r, p_3, p_4'))$$

and

$$\rho_r(C, C') = ((c_1, c_2, c_3, c_4 \oplus \gamma), (c_1', c_2', c_3', c_4' \oplus \gamma)).$$

In the rest of the work, the fixed points of $\phi_r$ and $\rho_r$ are unified by the name of simple fixed points.

Notably, both of $\phi_r$ and $\rho_r$ are involutory. If one yoyo cycle contains a simple fixed point, then the points in both sides of such a simple fixed point are axisymmetric (see Figure 5). Therefore, one simple fixed point divides the cycle into two symmetric parts.

An interesting question is, if a yoyo cycle contains two or more simple fixed points, how will such a symmetric feature change? In Theorem 4, we focus on the case of one yoyo cycle containing two simple fixed points.

**Theorem 4** Let $O_{\rho, P'}$ be an $r$-length yoyo cycle containing two simple fixed points. If integer $k, t$ $(0 \leq k < t \leq r - 1)$ are the coordinates of two simple fixed points, then we have

$$r = \begin{cases} 
2t - 2k : & \text{if } [P_k] = [P_k]_r \text{ and } [P_t] = [P_t]_r; \\
2t - 2k : & \text{if } [C_k] = [C_k]_r \text{ and } [C_t] = [C_t]_r; \\
2t - 2k + 1 : & \text{if } [P_k] = [P_k]_r \text{ and } [C_t] = [C_t]_r; \\
2t - 2k - 1 : & \text{if } [C_k] = [C_k]_r \text{ and } [P_t] = [P_t]_r.
\end{cases}$$

**Proof:** We only prove the first case: with $\begin{cases} [P_k] = [P_k]_r, \\
[P_t] = [P_t]_r, \end{cases}$

our target is to show that the cycle length is $r = 2t - 2k$. The other three cases can be accomplished similarly.

According to Definition 3, we have

$$O_{\rho, P'} = ([P_0], [C_0], [P_1], [C_1], \ldots, [P_{r-1}], [C_{r-1}]).$$

Since all intermediate states in $O_{\rho, P'}$ form an $r$-length cycle loop, it is convenient to introduce negative footmarks, $[P_{-i}] = [P_{r-i}]$ and $[C_{-i}] = [C_{r-i}]$, for any $0 \leq i \leq r - 1$.

The basic idea of the proof is to find a positive integer $r$ such that $[P_0] = [P_r]$ and then to prove that $r$ is the smallest positive integer making this equation hold. Our proof begins

![Figure 5](image-url) - Axes with respect to the simple fixed points

\[ \bullet : \text{fixed point} \]
with the observation that $[P_i] = [P_k]$. On account of the involuntary property of $\phi_i$ and $\rho_j$, the following equation holds:

\[
[P_{k+1}] = D \circ \rho_j \circ E \circ [P_k] = D \circ \rho_j \circ (E \circ D) \circ [P_{k-1}] = D \circ (\rho_j \circ I \circ \rho_j) \circ [P_{k-1}] = [P_{k-1}].
\]

(1)

In the same manner, we obtain a sequence of equations.

\[
\begin{aligned}
[P_k] &= [P_k], \\
[P_{k+1}] &= [P_{k-1}], \\
[P_{k+2}] &= [P_{k-2}], \\
& \vdots \\
[P_{k-1}] &= [P_{2k-1}], \\
[P_k] &= [P_{2k-1}].
\end{aligned}
\]

(2)

Next, we focus on $[P_1] = [P_{2k-1}]$. Since $[P_1] = [P_1]$, we conclude that $[P_1] = [P_{2k-1}]$, which means

\[
\phi_1 \circ (g \circ f)[P_0] = \phi_1 \circ (g \circ f)^{2k-1}[P_0].
\]

By recalling that $f$ and $g$ in the yoyo are all bijections, $[P_0] = [P_{2k-2}]$ follows from

\[
(g \circ f)^{2k-2}[P_0] = [P_0].
\]

Consequently, $[P_0]$ to $[P_{2k-2}]$ is a complete yoyo cycle.

The task is now to show that for any $r' \in \mathbb{N}^*$, $0 < r' < 2t - 2k$,

\[
[P_0] \neq [P_{r'}].
\]

Suppose that there exists $r' \in \mathbb{N}^*$, $0 < r' < 2t - 2k$ such that

\[
[P_0] = (f \circ g)^{r'}[P_0].
\]

On account of $[P_0] = [P_{2t-2k}]$, it follows that $[P_0] = [P_{2t-2k-r'}]$. Combining $[P_0] = [P_{2t-2k-r'}]$ and $[P_0] = [P_r]$, we can assert that

\[
r \leq \min\{2t - 2k - r', r'\} \leq t - k.
\]

This contradicts our assumption that $0 < k < t < r$.

In conclusion, according to the definition of the collision cycle length, if $\{[P_0] = [P_k], [P_1] = [P_{2k-1}]\}$ holds, we obtain $r = 2t - 2k$. \hfill $\square$

Figure 6 shows Cases 1 and 3 in Theorem 4. The red point $[P_k]$ is a simple fixed point, and those of the same colour are equal. In Case 1, $[P_1] = [P_1] = [P_{2k-1}]$, holds, meaning that the points on both ends of the red line merge into a single point. Hence, the successor of $[P_1]$ is exactly the same as the successor of $[P_{2k-1}]$. A $(2t - 2k)$-length yoyo cycle is generated. In Case 3, the successor of $[C_1]$ and $[C_{2k-1}]$, is also exactly the same point (both ends of the green line). It follows easily that the yoyo cycle length is $2t - 2k + 1$.

By the discussion above, the length of a two-simple fixed point yoyo cycle is completely determined by the distance between the two simple fixed points. Furthermore, we find that there are only two kinds of yoyo cycles: one has and only has two simple fixed points, and the other contains no simple fixed point. We conclude this property in Corollaries 2 and 3.

**Corollary 2** There are at most two simple fixed points in a yoyo cycle.

**Proof:** We will prove this by contradiction. Suppose that there are more than two simple fixed points in an $r$-length yoyo cycle. For the sake of simplicity, we only prove the case in which the first three consecutive simple fixed points are all plaintext pairs. Then, the other cases can be accomplished similarly. Without loss of generality, we assume that the first three simple fixed points are $[P_k]$, $[P_1]$ and $[P_3]$, respectively, where $0 \leq k < t < s < r$.

Notice that $[P_1]$ is a simple fixed point. By Equation (2) in Theorem 4, we obtain $[P_1] = [P_3]$. Combining with $[P_1] = [P_1]$, we have $[P_1] = [P_{2k-1}]$, which yields $t \equiv 2s - t \mod r$. Then, $2s = 2t + n_1r$, holds, where $n_1 \in \mathbb{Z} \setminus \{0\}$. Likewise, we obtain $[P_3] = [P_{2t-2k}]$ by the two simple fixed points, $[P_k]$ and $[P_1]$. Then, we deduce that $2t = 2k + n_2r$, $n_2 \in \mathbb{Z} \setminus \{0\}$. Thus, we obtain $2s = 2k + (n_1 + n_2)r$. According to the assumption...
When \( r < \) is a simple fixed point different from \([P_k]\), that is, there exist \(k \in \mathbb{N}, k < r\) such that

\[ [P_k] = [P_k]^r. \]

The proof falls naturally into two parts: \( r \) is odd or \( r \) is even.

When \( r \) is odd, by Equation (2) in Theorem 4, we obtain

\[ [P_k^\frac{1}{2}] = [P_k^\frac{3}{2}]^r. \]

According to \([P_k] = [P_k + 1]\),

\[ \gamma^{-\frac{1}{2}} \circ [P_k] = \gamma^{\frac{3}{2}} \circ [P_k] \]

holds. Then,

\[ [P_k - \frac{1}{2}] = [P_k + \frac{3}{2}]^r. \]

From this, we have

\[ [P_k + \frac{1}{2}] = [P_k - \frac{1}{2}]^r, \]

\[ [P_k + \frac{3}{2}] = [P_k + \frac{1}{2}]^r \]

\[ [P_k + \frac{5}{2}] = \rho_{\gamma} \circ \phi_{\gamma} \circ \gamma \circ [P_k + \frac{3}{2}] \]

\[ \rho_{\gamma} \circ \phi_{\gamma} \circ [P_k + \frac{3}{2}] = \gamma \circ [P_k + \frac{1}{2}] \]

\[ [C_k + \frac{1}{2}] = [C_k + \frac{3}{2}]^r, \]

which means \([C_k + \frac{1}{2}]\) is a simple fixed point. This contradicts our assumption.

When \( r \) is even, similarly, we obtain \([P_k + \frac{1}{2}] = [P_k - \frac{1}{2}]^r\) by (2) and \([P_k + \frac{3}{2}] = [P_k - \frac{1}{2}]^r\) by Definition 4, which implies \([P_k + \frac{1}{2}] = [P_k - \frac{1}{2}]^r\). From the assumption that

\[ 0 < k < r, \ k - \frac{1}{2} \equiv k \mod r \]

is impossible. Hence, \([P_k + \frac{1}{2}]\) is a simple fixed point different from \([P_k]\), which leads to a contradiction.

Thus, any yoyo cycle cannot contain only one simple fixed point. \( \square \)

Theorem 4 and Corollary 2 show that when we meet the second simple fixed point in a yoyo game, the length of this cycle is determined. Thus, the mathematical expectation of the yoyo cycle length is greatly influenced by the probability of simple fixed points. Corollaries 2 and 3 provide us a classification of all the yoyo game cycles: those containing exactly two simple fixed points and those without any simple fixed point, denoted by Class 1 and Class 2, respectively. Next, we are going to determine how often simple fixed points occur and what the expected values of the lengths of the two types of cycles are.

The cycle length of Class 1: For the cycles with two simple fixed points, we find that the probability of simple fixed points depends on whether the start point is right or wrong. We also divide them into two parts. One is derived from wrong start points and the other from right points.

**Part 1.** If one cycle is formed by a wrong point, the probability of a simple fixed point appears, that is, \([P] = [P]^r\) or \([C] = [C]^r\), which implies

\[ \left\{ \begin{array}{l} p_1 = p_1' \\ p_2 = p_2' \oplus \gamma \end{array} \right. \quad \text{or} \quad \left\{ \begin{array}{l} c_3 = c_3' \\ c_4 = c_4' \oplus \gamma \end{array} \right. \]

is \(2^{-2n} + 1\). Let \( k_w \) be the average length between the start point and the first simple fixed point. First, we need to determine the probability of the event that \((P_r, P'_r)\) is the first simple fixed point. Since the probability that the first \(i - 1\) points are not simple fixed points is \((1 - 2^{-2n+1})^{i-1}\), the probability of this event is \((1 - 2^{-2n+1})^{i-1} \times 2^{-2n+1}\). Then, we obtain

\[ k_w = \sum_{i=1}^{2n} i \times (1 - 2^{-2n+1})^{i-1} \times 2^{-2n+1} \approx 2^{2n-1}. \]

Suppose the first simple fixed point is \((P_{k_w}, P'_{k_w})\). Then,

\[ [P_{2k_w - 1}] = [P_i], (i = 1, 2, 3, \ldots, k_w), \]

which means \([P_{k_w+1}], [P_{k_w+2}] \cdots [P_{2k_w-1}]\) are determined. As per the analysis of Theorem 4, none of these points is a simple fixed point. The average length between \((P_{2k_w}, P'_{2k_w})\) and the second simple fixed point is also \( k_w \) where the second simple fixed point is assumed to be a plaintext pair. Thus, we have

\[ t = 2k_w - 1 + k_w = 3k_w - 1. \]

According to the first proposition in Theorem 4, the mathematical expectation of the length of the cycle with two plaintext simple fixed points is
\[ 2t - 2k_w = 4k_w - 2. \]

By a similar analysis, we also obtain \( t = 3k_w, t = 3k_w - 1 \) and \( t = 3k_w \) for the other three cases, respectively. Thus, the average cycle length of the Class I cycles is

\[
\frac{1}{4} \times (4k_w - 2) + \frac{1}{4} \times 4k_w + \frac{1}{4} \times (4k_w - 1) + \frac{1}{4} \times (4k_w - 1) = 2^{2r+1} - 1
\]

**Part 2.** If the collision cycle is derived from a right point that satisfies \( F_i(p_i) \oplus F_i(p'_i) = p_2 \oplus p'_2 \oplus \gamma \), the probability of

\[ [P] = [P]_{\gamma}, \text{ that is } \begin{cases} p_1 = p'_1 \\ p_2 = p'_2 \oplus \gamma \end{cases} \]

is \( 2^{-n} \), which is also true for \( [C] = [C]_{\gamma} \). Therefore, the probability of the event, \([P] = [P]_{\gamma} \) or \([C] = [C]_{\gamma} \), is \( 2^{-n+1} \).

Let \( k_r \) be the average length between the start point and the first simple fixed point, where \( k_r = 2^n - 1 \). In this case, the average cycle length is \( 2^n + 1 - 1 \).

The **probability of Class 2:** For cycles in Class 2 where simple fixed points do not appear, because the probability of a random point equal to the start point is 2, simple fixed points do not appear, because the probability of a random point equal to the start point is 2, simple fixed points do not appear, because the probability of a random point equal to the start point is 2. The average cycle length is \( 2^n \).

This means the number of this kind of cycle is extremely small compared with that of Class 1. Thus, the collision cycles used in recovery are almost all of the first class.

Each cycle from a right point only offers \( 2^n + 1 - 1 \) equations on average. However, not all of them provide valid recovery. For example, one equation is \( F_i(p_i) \oplus F_i(p'_i) = \delta \). Thus, the equation provided by the point with the simple fixed point as the axis of symmetry is \( F_i(p_i) \oplus F_i(p_i) = \delta \), which does not provide any new information. Therefore, only half of the points are available for recovery, owing to the repetition.

**Experimental verification of the cycle length.** In order to verify the correctness of the above analysis, we show the average length of 30,000 right-point and wrong-point cycles for \( n = 4, 5, 6, 7 \) in Table 1.

| Table 1 | The average length of 30,000 right-point and wrong-point cycles |
|---------|---------------------------------------------------------------|
| **Start points** | **n = 4** | **n = 5** | **n = 6** | **n = 7** |
| Wrong points | Reality prediction | 512.6 | 2049.4 | 8189.0 | 32,878.0 |
| Right points | Reality prediction | 511 | 2047 | 8191 | 32,767 |
| | | 31.5 | 63.2 | 127.1 | 254.7 |
| | | 31 | 63 | 127 | 255 |

In each yo-yo game collision cycle, half of the plaintexts provide repeated equations. On average, there are only \( 2^n \) effective equations for recovery. Therefore, it is more difficult to obtain complete LUTs as \( n \) increases.

### 4.3 Recycling-based yo-yo game

It would be wasteful to disregard the partial cycles from the right points. Therefore, when we get a collision cycle, a better choice is to judge whether it is from a right point. As long as a right collision cycle is screened out, we can ensure that the current LUT of \( F_i \) is correct, and our subsequent recovery will be based on this LUT. Otherwise, if it is undetermined, we have to abandon it and return to the former LUT, which is identified as correct for continued recovery. Before we introduce the **Partial LUT Recycling** technique, we first take a closer look at the symmetrical property of the yo-yo cycle.

In Section 4.1, we clarify that almost all the yo-yo cycles of the 7-round type-II GFN contain exactly two simple fixed points. Without loss of generality, we assume that \([P_0] \) and \([P_1] \) are two simple fixed points satisfying \( k < t \) (for the remaining three cases, we can launch similar arguments). Recalling the process of the yo-yo game, we start from \([P_1] \) and then iterate the mapping \( \mathcal{Y} \) until we meet with the first simple fixed point \([P_0] \). By the symmetrical property, \([P_{k+1}] = [P_{k-1}] \) holds for \( 1 \leq s \leq k - 1 \). So the points from \([P_{k+1}] \) to \([P_{2k-1}] \) provides us with the exact same equations as those points from \([P_1] \) to \([P_{k-1}] \). Likewise, we have the same property for the second simple fixed points. Subsequently, we can divide the yo-yo cycle into four trails by these two 'symmetrical axes' (see Figure 7).

As a result, in an iteration of the yo-yo game we never need to run a full cycle. Instead, we do the following:

1. (1) run Trail 1 (from the start point to the first simple fixed point).
2. (2) skip over Trail 2 and run Trail 3 (from \( \mathcal{Y}[P_{2k-1}] = \mathcal{Y} \circ [P_1] \) to the second simple fixed point).

In this way, we collect all the equations from the yo-yo cycle at half the cost of running.

**Partial LUTs recycling.** We now consider a special case. Assume that \( F_i(p_i) \oplus F_i(p'_i) = \delta \) is a newly extracted equation; if both \( F_i(p_i) \) and \( F_i(p'_i) \) have already been fixed before, then such an equation provides us a verification:
2) If it provides a verification and there is no contradiction, increase \( \tau \) by 1.
3) If it is the first simple fixed point, take \( Y \circ \phi_y(P_1, P'_1) \) as the next point.
4) If it is the second simple fixed point, go to Step 4 and then check the times of verification, \( \tau \).
5) Otherwise, we dispose the equation according to Case 1 and Case 2 proposed above, then we go to Step 2.

**Step 4. Judge the LUT:**
1) If the LUT has been fully recovered, output this LUT as the result.
2) If it is just partially recovered, and \( \tau \geq 4 \), we determine this LUT to be correct. Record the current LUT, then continually choose a new start point.
3) Otherwise, we must return to the last LUT that was determined correct.

We divide the above process into five algorithms. See the Appendix for details.

**Experiment results.** In this section, we apply the basic algorithm in [16] and our improved algorithm on the 7-round type-II GFN. Our first experiment is based on \( 2^{18} \) random start points. We use these start points to recover \( F_1 \) of the 7-round type-II GFN, where \( n = 4, 5, 6 \) and 7. Table 2 shows the number of start points that can recover \( F_1 \) (on a standard Intel i5-8250u 1.8 GHz).

In the second experiment, we used 100,000 right points for recovering \( F_1 \) to see how many fully recovered \( F_1 \) could be obtained. According to the results, all points can be classified into three parts:

Part 1. \( F_1 \) can be fully recovered starting from this point.
Part 2. In the yoyo cycle derived from this point, the number of verifications is greater than three, meaning that \( F_1 \) is not fully recovered. But we determine it as the right LUT. In other words, this point is available in the *Partial LUTs’ Recycling* technique.
Part 3. In the cycle starting from this point, the number of verifications is lower than four.

Table 3 shows the proportion of each part.

According to Table 1, our improved algorithm performs better than the original algorithm. In particular, when \( n \) increases, it becomes more difficult to obtain a fully recovered \( F_1 \) by the algorithm in [16]. This is because, as \( n \) increases, fewer LUTs can be fully recovered by \( 2^n \) effective equations on average. However, by employing the *Partial LUTs’ Recycling* technique, we are able to recover the LUT with the following advantages.
technique, our algorithm can find partial LUTs from right points and obtain the fully recovered $F_i$ based on them. According to Table 2, as $n$ increases, the proportions of right points in Parts 1 and 3 decrease but that in Part 2 increases. When $n \geq 5$, we obtain a right point that is available for recycling by every $2^n$ guesses on average with a probability of more than 50%. To summarise, our algorithm overcomes the problem described at the beginning of this section.

We require approximately $2^{2n+1}$ start points to screen out one available right point ($\gamma$ controlled by ourselves). In the experiments, most of the wrong start points are abandoned before producing $2^n$ equations. When a partial LUT is obtained, we can judge the correctness of each start point after a few steps of the subsequent recovery. Thus, recovering $F_i$ requires $O(2^{3n+1})$ data complexity and $O(n2^{3n+\frac{1}{2}})$ computational complexity. Employing two distinguishers simultaneously, we recover the outermost four functions with $O(2^{3n+\frac{1}{2}})$ data complexity and $O(n2^{3n+\frac{1}{2}})$ computational complexity.

Based on the analysis above, we provide two tips for the yoyo game design:

1) Try to select independent $\phi_f$ and $\rho_f$. This will help us utilise the type-S cycle to reduce computational complexity.
2) After determining $\phi_f$ and $\rho_f$, we need to calculate the probabilities of $\phi_f(P, P') = (P', P)$, $\phi_f(P, P') = (P', P')$, $\rho_f(C, C') = (C, C)$ and $\rho_f(C, C') = (C', C)$ for right and wrong points, respectively. Then, we can obtain the average collision length by these probabilities and decide whether to use the Partial LUTs’ Recycling technique.

\section{Decomposing the Remaining Rounds of the Type-II GFN}

In Section 4, we recovered $F_1$, $G_3$, $F_7$ and $G_1$. Next, we focus on recovering 5-round inner functions. Similar to the 7-round structure, the difference trail in Figure 8 was chosen.

\begin{table}[h!]
\centering
\caption{The rate of three parts in 100,000 random right points}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
 & \text{Part 1} & \text{Part 2} & \text{Part 3} \\
\hline
\text{n = 4} & 24.4\% & 22.3\% & 53.3\% \\
\text{n = 5} & 14.0\% & 35.5\% & 50.5\% \\
\text{n = 6} & 8.2\% & 45.3\% & 46.5\% \\
\text{n = 7} & 4.8\% & 52.1\% & 43.1\% \\
\text{n = 8} & 2.7\% & 57.0\% & 40.3\% \\
\hline
\end{tabular}
\end{table}

Table 3 The rate of three parts in 100,000 random right points

\begin{figure}[h!]
\centering
\includegraphics[width=\textwidth]{figure8.png}
\caption{Two differential properties of the 5-round type-II GFN. GFN, generalised Feistel network}
\end{figure}
problem, we studied the collision cycle's characteristic, which enabled us to obtain a clear view of yoyo game collision cycles. It was $\phi_f$ and $\rho_p$, which was different from that employed in Feistel, which limited the length of the collision cycles in the type-II GFN.

Finally, we proposed the **Partial LUTs' Recycling** technique to make yoyo games more applicable to type-II GFNs and provided two tips for the yoyo game design. Our experiments showed that for 7-round type-II GFNs, the improved algorithm overcame the insufficient equation problem. This technique is more suitable for type-II GFNs and other structures having short collision cycles.

This work broadens the application of yoyo games and enriches such attacks. It will serve as a reference for further yoyo game designs.

**ACKNOWLEDGEMENT**

The authors are grateful to the anonymous referees for their valuable comments. The work in this article is supported by the National Natural Science Foundation of China (Grant nos. 61772547, 61902428, 61802438, 61402523).

**REFERENCES**

1. Zheng, Y., Matsumoto, T., Imai, H.: On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In: Conference on the Theory and Application of Cryptology, pp. 461–480. Springer (1989)
2. Bogdanov, A., Shiibutani, K.: Generalized Feistel networks revisited. Des. Codes Cryptogr. 66(1-3), 75–97 (2013)
3. Rivest, R.L., et al.: The RC6 block cipher. In: First Advanced Encryption Standard (AES) Conference (1998)
4. Shirai, T., et al.: The 128-bit blockcipher CLEFIA. In: International Workshop on Fast Software Encryption, pp. 431–450. Springer (2007)
5. Hong, D., et al.: HIGHT: a new block cipher suitable for low-resource device. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 46–59. Springer (2006)
6. Merkle, R.C.: Fast software encryption functions. In: Conference on the Theory and Application of Cryptography, pp. 477–501. Springer (1990)
7. Schneier, B.: Description of a new variable-length key, 64-bit block cipher (Blowfish). In: International Workshop on Fast Software Encryption, pp. 191–204. Springer (1993)
8. Kavalauskas, K., Kavalauskas, J.: Key-dependent S-box generation in AES block cipher system. Informatica. 20(1), 23–34 (2009)
9. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. 4(1), 3–72 (1991)
10. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Workshop on the Theory and Application of Cryptographic Techniques, pp. 386–397. Springer (1993)
11. Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 395–405. Springer (2001)
12. Boghossian, J., et al.: Cryptanalysis of PRESENT-like ciphers with secret S-boxes. In: International Workshop on Fast Software Encryption, pp. 270–289. Springer (2011)
13. Liu, G., Jin, C.H., Qi, C.D.: Improved slender-set linear cryptanalysis. In: International Workshop on Fast Software Encryption, pp. 431–450. Springer (2014)
14. Liu, G.Q., Jin, C.H.: Differential cryptanalysis of PRESENT-like cipher. Des. Codes Cryptogr. 76(3), 385–408 (2015)
15. Tiessen, T., et al.: Security of the AES with a secret s-box. In: Fast Software Encryption, pp. 175–189. Springer (2015)
16. Biryukov, A., Leurent, G., Perrin, L.: Cryptanalysis of Feistel networks with secret round functions. In: International Conference on Selected Areas in Cryptography, pp. 102–121. Springer (2015)
17. Biham, E., et al.: Initial observations on skipjack: cryptanalysis of skipjack-3XOR. In: International Workshop on Selected Areas in Cryptography, pp. 362–375. Springer (1998)
18. Patarijn, J.: Generic attacks on Feistel schemes. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 222–238. Springer (2001)
19. Knudsen, L.: DEAL—a 128-bit block cipher. Complexity. 258(2), 216 (1998)
20. Biryukov, A., Perrin, L.: On reverse-engineering S-boxes with hidden design criteria or structure. In: Annual Cryptology Conference, pp. 116–140. Springer (2015)
21. Cui, T., Chen, S., Zheng, H.: A structural attack on type-I generalized Feistel networks. IEEE Access. 7, 60304–60310 (2019)
22. Ronjom, S., Bardh, N.G., Helleseth, T.: Yoyo tricks with AES. In: Advances in Cryptology - ASIACRYPT, pp. 217–243. Springer (2017)
23. Bardh, N.G., Ronjom, S.: Practical attacks on reduced-round AES. In: Progress in Cryptology - AFRICACRYPT, pp. 297–310. Springer (2019)
24. Dunkelman, O., et al.: The retracing boomerang attack. In: Advances in Cryptology - EUROCRYPT, pp. 280–309. Springer (2020)

**How to cite this article:** Hou, T., Cui, T.: Yoyo trick on type-II generalised Feistel networks. IET Inf. Secur. 1–14 (2021). https://doi.org/10.1049/ise2.12035

**APPENDIX**

Algorithm 1 is a general yoyo game framework. It returns the LUT of the internal function, $F_i$. The recovery is carried out on two auxiliary sets, $L'$ and $F_i'$. Only when $F_i'$ is determined to be right, can the real LUT $F_i$ receive values from $F_i'$. After many iterations of the **Partial LUTs Recycling** until the final one is fully filled, this algorithm outputs the recovered $F_i$.

**Algorithm 1**  Yoyo cryptanalysis against a 7-round type-II GFN: recovering the outermost round

$L \leftarrow $ empty set to store equations
$F_i \leftarrow $ empty LUT
$F_i' \leftarrow $ empty LUT $\triangleright$ used to receive updated LUT $L' \leftarrow \{\} \quad \triangleright$ used to receive updated $L$

while $F_i$ is not filled completely do

if Algorithm 2($F_i'$, $L'$) is SUCCESS then
$F_i \leftarrow F_i'$
$L \leftarrow L' \quad \triangleright F_i'$, $L'$ are right
else
$F_i' \leftarrow F_i$
$L' \leftarrow L \quad \triangleright F_i'$, $L'$ are wrong

end if
end while
return $F_i$
Algorithm 2 Partial LUT recycling judgement

**Inputs**: LUT of $F_1$; List of equations $L$;
**Outputs**: SUCCESS or FAIL

1: Select a random point, $(P_1, P_1') \in \{0, 1\}^{8n}$
2: for all $y \in \{0, 1\}^n$ and $y \neq 0$ do
3: \[ i \leftarrow 1, \text{count} \leftarrow 1, \tau \leftarrow 0; \]
4: if $F_1$ is empty then $F_1(P_1) = 0$ and $F_1(P_1') = p_2 \oplus p_2' \oplus y$
5: \[ \text{\textbf{\textcircled{\char140}}} \text{initialisation} \]
6: \[ \text{while count < 2 do} \]
7: $(P_{i+1}, P_{i+1}') \leftarrow \text{Algorithm 3}((P_i, P_i'))$
8: if $(P_i, P_i')$ is a simple fixed point then
9: \[ \text{count} \leftarrow \text{count} + 1 \]
10: if it is the first simple fixed point then
11: \[ i \leftarrow 2i \]
12: $(P_i, P_i') \leftarrow D \circ \rho_y \circ \mathcal{E}(P_1, P_1')$
13: else if $(C_i, C_i')$ is a simple fixed point then
14: \[ \text{count} \leftarrow \text{count} + 1 \]
15: i \leftarrow i + 1
16: end if
17: Algorithm 4$(L, F_1, (P_i, P_i'), y, \tau)$ \[ \text{\textbf{\textcircled{\char140}}} \text{renew LUT} \]
18: if it fails then return FAIL \[ \text{\textbf{\textcircled{\char141}}} \text{contradiction} \]
19: end while
20: if $\tau \geq 4$ then return SUCCESS \[ \text{\textbf{\textcircled{\char145}}} \text{correct partial verification} \]
21: else return FAIL \[ \text{\textbf{\textcircled{\char146}}} \text{insufficient verification} \]
22: end for

After inputting $(P_n, P_n')$, Algorithm 3 produces a new plaintext pair $(P_{i+1}, P_{i+1}')$ by the yoyo game and checks if $(P_n, P_n')$ or $(C_n, C_n')$ is a simple fixed point. It returns 1 or 2, indicating simple fixed point $(P_n, P_n')$ or $(C_n, C_n')$, respectively; otherwise, it returns 0.

Algorithm 3 Produce a new pair via yoyo and check the simple fixed point

**Input**: Plaintext pair $(P_i, P_i')$
**Outputs**: Flag for simple fixed point $x$; New plaintext pair $(P_{i+1}, P_{i+1}')$

1: $x \leftarrow 0$
2: if $\phi_y(P_i, P_i') = (P_i, P_i')$ then $x \leftarrow 1$
3: $(C, C') \leftarrow \mathcal{E} \circ \phi_y(P_i, P_i')$
4: if $\rho_y(C, C') = (C, C')$ then $x \leftarrow 2$
5: $(P_{i+1}, P_{i+1}') \leftarrow D \circ \rho_y(C, C')$
6: return $x$ and $(P_{i+1}, P_{i+1}')$

Algorithms 4 and 5 are used to refresh the LUT of $F_1$. We only make some changes based on the two original algorithms proposed in [16]. There are two improvements in our algorithm. One is employing $\tau$ to record the verifications in order to judge the correctness of the current partial LUT. The other is skipping the part of calculating plaintext pairs to reduce half of the computation.

Algorithm 4 Get entries from one plaintext pair

**Inputs**: List of equations $L$; Inner secret function $F_i$; Plaintext pair $(P_i, P_i')$; Difference $y$; Verification indicator $\tau$
**Outputs**: SUCCESS or FAIL; $\tau$

1: $\delta \leftarrow p_2 \oplus p_2' \oplus y$
2: if $F_1(p_1)$ and $F_1(p_1')$ are already known then
3: if $F_1(p_1) \oplus F_1(p_1') \neq \delta$ then
4: return FAIL, $\tau$ \[ \text{\textbf{\textcircled{\char141}}} \text{contradiction} \]
5: else
6: $\tau \leftarrow \tau + 1$ \[ \text{\textbf{\textcircled{\char145}}} \text{an effective verification} \]
7: end if
8: else if $F_1(p_1)$ is known but not $F_1(p_1')$ then
9: AddEntry($F_1, p_1, F_1(p_1) \oplus \delta, L, \tau$
10: if it fails then return FAIL, $\tau$ \[ \text{\textbf{\textcircled{\char146}}} \text{contradiction} \]
11: else if $F_1(p_1')$ is known but not $F_1(p_1)$ then
12: AddEntry($F_1, p_1', F_1(p_1') \oplus \delta, L, \tau$
13: if it fails then return FAIL, $\tau$ \[ \text{\textbf{\textcircled{\char141}}} \text{contradiction} \]
14: else
15: Add $F_1(p_1) \oplus F_1(p_1') = \delta$ to $L$
16: end if
17: return SUCCESS, $\tau$
Algorithm 5 Add new entries to $F_1$ (AddEntry)

**Inputs:** Inner secret function $F_1$; Input $x$;
Output $y$; List of equations $L$; Verification indicator $\tau$

**Outputs:** fail or success

```
F_1(x) ← y
for all equations $F_1(x_i) \oplus F_1(x_i') = \Delta_i$ in $L$
    do
        if $F_1(x_i)$ and $F_1(x_i')$ are set then
            if $F_1(x_i) \oplus F_1(x_i') \neq \Delta_i$ then
                return fail
                ▷ contradiction
            else $\tau ← \tau + 1$; Remove this eq. from $L$
        end if
    end do
end for
return success
```

```
else if $F_1(x_i)$ is set but not $F_1(x_i')$ then
    Remove this eq. from $L$; AddEntry($F_1$, $x$

    if it fails; then return fail
    ▷ eq. gives new entry

else if $F_1(x_i')$ is set but not $F_1(x_i)$ then
    Remove this eq. from $L$; AddEntry($F_1$, $x$

    if it fails; then return fail
    ▷ eq. gives new entry
```

end if