Revisiting RC4 Key Collision: Faster Search Algorithm and New 22-byte Colliding Key Pairs

Amit Jana and Goutam Paul

Cryptology and Security Research Unit (CSRU),
R. C. Bose Centre for Cryptology and Security,
Indian Statistical Institute, Kolkata 700 108, India.
janaamit001@gmail.com, goutam.paul@isical.ac.in

Abstract. If two different secret keys of stream cipher RC4 yield the same internal state after the key scheduling algorithm (KSA) and hence generate the same sequence of keystream bits, they are called a colliding key pair. The number of possible internal states of RC4 stream cipher is very large (approximately 2^{1700}), which makes finding key collision hard for practical key lengths (i.e., less than 30 bytes). Matsui [FSE 2009] for the first time reported a 24-byte colliding key pair and one 20-byte near-colliding key pair (i.e., for which the state arrays after the KSA differ in at most two positions) for RC4. Subsequently, Chen and Miyaji [ISC 2011] designed a more efficient search algorithm using Matsui's collision pattern and reported a 22-byte colliding key pair which remains the only shortest known colliding key pair so far. In this paper, we show some limitations of both the above approaches and propose a faster collision search algorithm that overcomes these limitations. Using our algorithm, we are able to find three additional 22-byte colliding key pairs that are different from the one reported by Chen and Miyaji. We additionally give 12 new 20-byte near-colliding key pairs. These results are significant, considering the argument by Biham and Dunkelman in 2007, that for shorter keys there might be no instances of collision at all.

Keywords: Colliding Pair, Key Collision, Near Colliding Pair, RC4, Related Key Cryptanalysis, Stream Cipher.

Mathematics Subject Classification (2010) 94A60

1 Introduction

RC4 [2] has been one of the most widely-used real-world stream ciphers. It has been used in various practical software applications and network protocols such as Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA), Transport Layer Security (TLS), Microsoft Windows, Apple OCE, Secure SQL, etc.

There are some practical attacks [1, 7, 8, 11, 12, 15, 17, 18] on RC4 and on some of the protocols using RC4. A few of these protocols have replaced (such as WPA 2) or are going to replace (such as TLS) RC4 by other ciphers. In spite of all these, recently, the designers of RC4 have reiterated the usability of *RC4-like* ciphers in their recent proposal of Spritz [16]. Theoretically, RC4 serves as a good model for a software stream cipher in the shuffle-exchange paradigm with many nice combinatorial properties that are worth investigating. So even if use of RC4 becomes deprecated in network protocols, it will remain a model stream cipher in the academic community for years to come.

The RC4 stream cipher consists of two algorithms, namely, the key scheduling algorithm (KSA) and the pseudo-random generation algorithm (PRGA). In KSA, we give a key K of size $k(\leq 256)$ bytes as input and initialize a state array S of size $N(= 256)$ to the identity over $\mathbb{Z}_N$. The key K is implicitly assumed to be stretched to size $N = 256$ bytes by repeating the same key (if k does not divide N, then the last repetition is incomplete). The KSA then scrambles this permutation through N swaps between a running deterministic index i and a pseudo-randomly evolving (through key) index j. The PRGA uses the scrambled permutation S to generate pseudo-random bytes $Z_1, Z_2, \ldots$, from state S, that are bitwise XOR-ed with the next plaintext/ciphertext byte to perform encryption/decryption. The KSA and PRGA are formally described in Algorithm 1 and 2 respectively.

Algorithm 1: **KSA**

Input: Secret key K
Output: Internal state S

$j = 0$;
`// State Initialization`
for $i = 0$ *to* $N - 1$ **do**
    $S[i] = i$;
`// State Randomization`
for $i = 0$ *to* $N - 1$ **do**
    $j = (j + S[i] + K[i \bmod k]) \bmod N$;
    swap$(S[i], S[j])$;

Algorithm 2: **PRGA**

Input: Internal state S, generated by KSA
Output: keystream Z

$i = 0$;
$j = 0$;
for *each new message byte* **do**
    $i = (i + 1) \bmod N$;
    $j = (j + S[i]) \bmod N$;
    swap$(S[i], S[j])$;
    $Z = S[(S[i] + S[j]) \bmod N]$;
    output Z;

1.1 RC4 key collision and why it is hard to find

In this paper, we revisit the methods for finding RC4 colliding key pairs, i.e., two different keys yielding the same state after KSA and hence the same keystream output. This is the strongest form of related key cryptanalysis and can be used to mount key recovery attack as described in [5]. Moreover, key collision for a stream cipher is itself a combinatorially interesting problem.

RC4 internal state consists of a permutation over the bytes $0, \ldots, 255$. Thus, the total number of states is $256! \approx 2^{1684}$ and by birthday paradox, if the key size is (≥ 106) bytes, with high probability there exists a colliding key pair. But finding colliding keys with reduced key sizes is a challenging problem, since the key is repeated in key length boundaries, making it hard to maintain the conditions for collision. Due to the state recovery attack [14] of Maximov, the practical key length of RC4 is now between 16 to 30 bytes and so finding the colliding pairs for these key sizes is even harder.

In 2009, the work of Matsui [13] reported a 24-byte colliding key pair and a 20-byte near-colliding key pair, i.e., keys that generate the states differing only at two positions. In 2009, the work of Chen and Miyaji [4] broke the record of Matsui and reported a 22-bye colliding key pair. Until now, this is the shortest known colliding key pair of RC4.

It is interesting to note that Biham and Dunkelman [3] state the following: "However, for keys shorter than 22 bytes (176 bits), the probability of two keys to generate the same internal state in the manner described earlier is so small, such that the expected number of pairs of keys which satisfy the characteristic is smaller than 1. Thus, for short keys there might be no such instances". Thus, beating the result of Chen and Miyaji [4] is indeed very challenging and as per [3] may be even impossible.

1.2 Our Contributions

In this work, we have four main contributions.

1. We investigate the collision search algorithms of Matsui [13] and Chen-Miyaji [4] and discover some limitations in their techniques.
2. We remove these limitations and also use additional novel tricks to devise a more efficient collision search algorithm. We provide comparison of time complexity estimates of the three approaches.
3. We touch the record of [4] by reporting three new 22-byte colliding key pairs (note that, as per [3], colliding keys shorter than 22-byte might not even exist).
4. We also report 12 new 20-byte near-colliding key pairs different from the one reported in [13].

1.3 Notations

Here we summarize the general notations used in this paper. Other notations are described whenever they are introduced.

- k, N denote the sizes of the secret key and the internal state in bytes respectively.
- K_1, K_2 denote the two secret keys of RC4 with the same size.
- d $(< k)$ denotes the index in which the two keys K_1, K_2 differ.
- The closed interval notation $[a, b]$ means the set of integers i such that $a \leq i \leq b$. The open interval notation (a, b) means the set of integers i such that $a < i < b$. Half-open or half-closed intervals are defined accordingly.
- We divide the RC4 KSA iteration into mutually exclusive and exhaustive intervals called *round intervals* as follows. The t-th round interval is defined as

$$r_t = \begin{cases} [0, d], & \text{if} \quad t = 1 \\ [d + (t-2)k + 1, d + (t-1)k], & \text{if} \quad 2 \leq t \leq n \end{cases}$$

  where $n = \lfloor \frac{256+k-1-d}{k} \rfloor$ denotes the total number of *round intervals*. For example, when $k = 64$ with $d = k - 3$, we have $n = 4$ and $r_1 = [0, 61], r_2 = [62, 125], r_3 = [126, 189], r_4 = [190, 253]$.
- For $1 \leq t \leq n$, we also denote the t-th round interval as $[a_t, b_t]$, where $a_1 = 0$, $b_1 = d$ and for $2 \leq t \leq n$, $a_t = d + (t-2)k + 1$, $b_t = d + (t-1)k$.
- S_1, S_2 denote the two state arrays corresponding to the keys K_1, K_2 respectively.
- $S_1[x], S_2[x]$ denote the value of the state array S_1, S_2 at the state index $x \in [0, 255]$ corresponding to K_1, K_2 respectively.
- $S_{1,i}[x], S_{2,i}[x]$ denote the value of states $S_1[x], S_2[x]$ respectively, before the swap operations are done at the i-th iteration in KSA.
- The equality $S_{1,i} = S_{2,i}$ means that at the i-th iteration in KSA, $S_{1,i}[x] = S_{2,i}[x]$, $\forall x \in [0, 255]$, used in Table 1.
- J_1, J_2 denote the two arrays of j-values in KSA iteration corresponding to the keys K_1, K_2 respectively.
- $J_{1,i}, J_{2,i}$ denote the j-values at the i-th iteration in KSA corresponding to K_1, K_2 respectively. $J'_{1,i}, J'_{2,i}$ denote the new j-values at the i-th iteration in KSA, when KSA is repeated after some key modifications (specific modifications will be explained as and when they appear in the text).
- At any round interval t, for each $l = 1, 2$, $J_{l,i,t}$ and $J'_{l,i,t}$ denote the $J_{l,i}$-values before and after the key modification in KSA respectively.
- $J_{1,[a,b]} = J_{2,[a,b]}$ means $J_{1,i} = J_{2,i}$, $\forall i \in [a, b]$, used in Table 2.
- For any round interval $r \geq 2$, we denote $c_{r,x} = x + d + (r-2)k$, $x \in [0, k-1]$. For the first round, i.e., for $r = 1$, we define $c_{1,x} = x$.
- Throughout this paper, "index" means the state index, unless we explicitly mention it as a key index. Note that the key index is a state index modulo k. Also sometimes we use the phrase 'index i in t-th round interval' to mean a state index $i \in [a_t, b_t]$.

- All operations are done modulo 256 except operations with the key indices for which modulo k is used.
- $K\langle x, y\rangle$ denotes the key modifications on the state indices $x, x+1$ and are defined as $K_1[x \bmod k] \leftarrow K_1[x \bmod k]+y$, $K_2[x \bmod k] \leftarrow K_2[x \bmod k]+y$, $K_1[x+1 \bmod k] \leftarrow K_1[x+1 \bmod k]-y$, $K_2[x+1 \bmod k] \leftarrow K_2[x+1 \bmod k]-y$, where $0 \leq x < 255$, $1 \leq y < 256$.
- $K^{'}\langle x, y\rangle$ denote the key modifications on the key index x and is defined as: $K_1[x] \leftarrow K_1[x]+y$, $K_2[x] \leftarrow K_2[x]+y$.
- $MaxS$ is the maximal state index i such that the distance between S_1 and S_2 from index 0 to i is at most two, where distance is the number of different state byte values corresponding to each state index $x \in [0, 255]$.
- $MaxP$ is the state index i such that $\forall l \in [0, i], J_{1,l}, J_{2,l}$ will satisfy all the required conditions according to Table 2.
- MaxColStep(K_1, K_2) function uses the KSA iteration for both the keys and return the state index i in which the first difference in state byte occurs.

2 Previous Related Works

In this section, we briefly describe the previous works on key collisions in RC4. In 2009, Matsui proposed [13] a key collision pattern for two different secret keys that can differ from each other in only one position to generate the same internal state. He also gave a search algorithm that yields a 24-byte colliding pair.

Subsequently, Chen-Miyaji [4] proposed an improved search algorithm yielding a 22-byte colliding pair. Later, Chen-Miyaji claimed to make further improvement [6] of their earlier work [4], but surprisingly obtained the same 22-byte colliding key pair as in [4].

Recently, the work by Maitra et al. [10] has studied related keystream distinguishers on RC4, but it depends on near-colliding keys that lead to states with difference in more than two positions.

To better understand about key collisions in RC4, at first we give an example from Section 3 of [13] in Table 1 that describes how collision occurs. The following is a pair of 64-byte keys, written in decimal form, which differ in only one byte, $K_1[2] \neq K_2[2]$. These two keys create the same initial state and hence they are cryptographically indistinguishable.

K_1 = 69 61 125 61 201 69 87 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,

K_2 = 69 61 126 61 201 69 87 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.

Using our notations, Table 1 shows internal values in the state randomization loop for each key and state differences corresponding to each state index i. Here we see that the key differences occur at the state indices 2, 66, 130, 194. Now inside the state randomization loop, we have at $i = 2$ (after the swap operation), $J_{1,i} = 2, J_{2,i} = 3$, causing the states to differ at exactly two positions 2 and 3 (i.e., $S_{1,i}[2] = 2, S_{1,i}[3] = 3, S_{2,i}[2] = 3, S_{2,i}[3] = 2$). At the next step $i = 3$, $J_{1,i} = J_{2,i} = 66$, because the previous difference (i.e., $J_{1,2} = J_{2,2}-1$) must be absorbed due to state difference $S_{1,2}[3] = S_{2,2}[3]+1$. For $i \in [4, 65]$, $J_{1,i} = J_{2,i}(\neq 66)$. But at the state index $i = 66$, where $S_{1,i}[66] = 3, S_{2,i}[66] = 2$, we have $J_{1,i} = J_{2,i} = 130$, i.e., we swap the state values between 66 and the next state index ($=$ 130), where the keys will differ. Similarly we need, $\forall i \in [67, 129]$, $J_{1,i} = J_{2,i}(\neq 130)$. Proceeding in this way until we go to the state index ($= 130$) of the state array in the second last round, where the keys differ, i.e., at $i = 130$, we swap the state values between $130 (= d + (n-2)k)$ and

193($= d+(n-1)k-1$) (i.e., a state index just before the last state index, where the keys differ). Then $\forall i \in [131, 191]$, $J_{1,i} = J_{2,i} (\neq 193)$ and at $i = 192$, we have $J_{1,i} = J_{2,i} = r$, a state index such that $S_{1,i}[r] = 2, S_{2,i}[r] = 3$. Here in this example $r = 129$. At last for $i = 193$, we have $J_{1,i} = i, J_{2,i} = i-1$ and for $i = 194$, $J_{1,i} = J_{1,i} = 84$, that converts these two different states into the same state.

Our work is along the same line as those of [13] and [4]. Since both these works as well as our work use the same transition pattern as in [13, Table 4], we reproduce that in Table 2, where for each round interval r_t, $1 \leq t \leq n$ and $\forall i \in [a_t, b_t]$, we define the class 1 conditions to be those conditions where $J_{1,i}, J_{2,i}$-values at the i-th iteration in KSA refer to a fixed state index and the class 2 conditions are those conditions where $J_{1,i}, J_{2,i}$-values at the i-th iteration in KSA are not equal to some fixed state indices. Throughout this paper, we will use these conditions for each round interval and make frequent references to this table.

i	K_1	K_2	$J_{1,i}$	$J_{2,i}$	difference between S_1 and S_2
0	69	69	69	69	$S_{1,0} = S_{2,0}$
1	61	61	131	131	$S_{1,1} = S_{2,1}$
2	125	126	2	3	$(S_1[2] = 2,\ S_{2,i}[2] = 3),\ (S_{1,i}[3] = 3,\ S_{2,i}[3] = 2)$
3	61	61	66	66	$(S_{1,i}[2] = 2,\ S_{2,i}[2] = 3),\ (S_{1,i}[66] = 3,\ S_{2,i}[66] = 2)$
$4-64$					$(S_{1,i}[2] = 2,\ S_{2,i}[2] = 3),\ (S_{1,i}[66] = 3,\ S_{2,i}[66] = 2)$
65	61	61	2	2	$(S_{1,i}[65] = 2,\ S_{2,i}[65] = 3),\ (S_{1,i}[66] = 3,\ S_{2,i}[66] = 2)$
66	125	126	130	130	$(S_{1,i}[65] = 2,\ S_{2,i}[65] = 3),\ (S_{1,i}[130] = 3,\ S_{2,i}[130] = 2)$
$67-128$					$(S_{1,i}[65] = 2,\ S_{2,i}[65] = 3),\ (S_{1,i}[130] = 3,\ S_{2,i}[130] = 2)$
129	61	61	65	65	$(S_{1,i}[129] = 2,\ S_{2,i}[129] = 3),\ (S_{1,i}[130] = 3,\ S_{2,i}[130] = 2)$
130	125	126	193	193	$(S_{1,i}[129] = 2,\ S_{2,i}[129] = 3),\ (S_{1,i}[193] = 3,\ S_{2,i}[193] = 2)$
$131-191$					$(S_{1,i}[129] = 2,\ S_{2,i}[129] = 3),\ (S_{1,i}[193] = 3,\ S_{2,i}[193] = 2)$
192	69	69	129	129	$(S_{1,i}[192] = 2,\ S_{2,i}[192] = 3),\ (S_{1,i}[193] = 3,\ S_{2,i}[193] = 2)$
193	61	61	193	192	$S_{1,193} = S_{2,193}$
194	125	126	84	84	$S_{1,194} = S_{2,194}$
$195-255$					$S_{1,i} = S_{2,i}, \forall i \in [195, 255]$

Table 1. The state transition pattern of the 64-byte key pairs in [13]

Round	Round Interval	Class 1 conditions	Class 2 conditions
1	$[a_1, b_1]$	$J_{1,b_1} = d, J_{2,b_1} = b_1 + 1$	$J_{1,[a_1,b_1-1]} = J_{2,[a,b_1-1]}\ (\neq b_1, b_1+1)$
2	$[a_2, b_2]$	$J_{1,b_1+1} = J_{2,b_1+1} = b_2$, $J_{1,b_2} = b_3, J_{2,b_2} = b_3$	$J_{1,[a_2+1,b_2-1]} = J_{2,[a_2+1,b_2-1]}\ (\neq b_2)$
3	$[a_3, b_3]$	$J_{1,b_3} = J_{2,b_3} = b_4$	$J_{1,[a_3,b_3-1]} = J_{2,[a_3,b_3-1]}\ (\neq b_3)$
...	...	...	...
t	$[a_t, b_t]$	$J_{1,b_t} = J_{2,b_t} = b_{t+1}$	$J_{1,[a_t,b_t-1]} = J_{2,[a_t,b_t-1]}\ (\neq b_t)$
...	...	...	....
$n-1$	$[a_{n-1}, b_{n-1}]$	$J_{1,b_{n-1}} = J_{2,b_{n-1}} = b_n - 1$	$J_{1,[a_{n-1},b_{n-1}-1]} = J_{2,[a_{n-1},b_{n-1}-1]}\ (\neq b_{n-1})$
n	$[a_n, b_n]$	$S_{1,b_n-3}[x] = b_1, S_{1,b_n-3}[x] = b_1+1$, $J_{1,b_n-2} = J_{2,b_n-2} = x$, $J_{1,b_n-1} = b_n - 1, J_{2,b_n-1} = b_n - 2$	$J_{1,[a_n,b_n-3]} = J_{2,[a_n,b_n-3]}\ (\neq b_n - 1)$ $J_{1,b_n} = J_{2,b_n}$
	$[b_n+1, 255]$	No Condition	$J_{1,[b_n+1,255]} = J_{2,[b_n+1,255]}$

Table 2. Transition pattern involving j-conditions

2.1 Matsui's work [13]

In [13], it is clearly described how two keys K_1 and K_2 with the same key size k, where $K_1[i] = K_2[i] - 1$ if $i = d$ $(0 \leq d < k)$ and $K_1[i] = K_2[i]$ otherwise, can achieve a collision. Matsui's search algorithm takes such a key pair as input and calls itself recursively until it finds a colliding key pair and stops, otherwise it tries another key pair. If value of the $MaxS$ variable (the $MaxS$ notation is defined in Section 1.3) is less than or equal to its value in the previous iteration, the search function modifies the key as $K\langle x, y\rangle$ and tries to find the maximum state index q (i.e., the index in which the state difference is at most two) using $MaxColStep(K_1, K_2), \forall x \in [0, k-1], y \in [1, 255]$. If $q > MaxS$, then it identifies the corresponding $K\langle x, y\rangle$, and the recursive search continues. Recall that a key modification $K\langle x, y\rangle$ increments the keys values at key index x and decrements the keys values at key index $x+1$ by y. We present Matsui's approach in Algorithm 3.

Algorithm 3: Matsui's Search Algorithm

```
Input: K1, K2, d
Output: KSA(K1) = KSA(K2)
Generate random key pair K1 and K2, which differ at position d;
Set K1[d] = K2[d + 1] = k − d − 1;
Call Search(K1, K2) and repeat this until a (near-)colliding key pair is found;
----------------------------------------------------------------
Search(K1, K2):
P = MaxColStep(K1, K2);
if P = 255 then
    stop (found a (near-)collision!) or return (to find more);
MaxS = max_{x,y} MaxColStep(K⟨x, y⟩);
if MaxS ≤ P then
    return;
C = 0;
for x = 0 to 255 do
    for y = 1 to 255 do
        if MaxColStep(K⟨x, y⟩) = MaxS then
            call Search(K1, K2);
            C = C + 1;
            if C = MaxS then
                return;
            return;
```

Where **MaxColStep**$(K1, K2)$ is defined as maximal step i such that distance between K_1 and K_2 is at most two at all steps up to step i. $MaxC$ is a pre-defined value. In general $MaxS = 10$, but never exceeds 20.

Now we state a result that will be useful for the subsequent analysis.

Lemma 1. *The effect of the key modification as* $K\langle x, y\rangle$, *with* $x \bmod k \neq d$ *will change the value* $J_{1,x}$ *to* $J_{1,x} + y$, *whereas the value of* $J_{1,x+1}$ *still remains unchanged.*

Proof. Due to the key modification as $K\langle x, y\rangle$ i.e., $K_1[x] \leftarrow K_1[x]+y$, $K_2[x] \leftarrow K_2[x]+y$, $K_1[x+1] \leftarrow K_1[x+1] - y$, $K_2[x+1] \leftarrow K_2[x+1] - y$, we have,

$$J'_{1,x} = J_{1,x-1} + K_1[x \bmod k] + S_{1,x-1}[x] = J_{1,x-1} + K_1[x \bmod k] + y + S_{1,x-1}[x]$$

$$= (J_{1,x-1} + K_1[x \bmod k] + S_{1,x-1}[x]) + y = J_{1,x} + y.$$

$$J'_{1,x+1} = J'_{1,x} + K_1[(x+1) \bmod k] + S_{1,x}[x+1] = J_{1,x} + y + K[(x+1) \bmod k] - y + S_{1,x}[x+1]$$

$$= J_{1,x} + K_1[(x+1) \bmod k] + S_{1,x}[x+1] = J_{1,x+1}.$$

Again we know, $J_{1,x-1} = J_{2,x-1}, J_{1,x} = J_{2,x}, K_1[x \bmod k] = K_2[x \bmod k], K_1[(x+1) \bmod k] = K_2[(x+1) \bmod k]$, $S_{1,x-1}[x] = S_{2,x-1}[x], S_{1,x}[x+1] = S_{2,x}[x+1]$. Consequently we have,

$$J'_{1,x} = J'_{2,x} = J_{1,x} + y, J'_{1,x+1} = J'_{2,x+1} = J_{1,x+1}.$$

□

The main idea is to take care of the j-values in such a way that the difference between the two states becomes zero. Once all the conditions according to the transition pattern in Table 2 are satisfied, a collision is expected. According to this transition pattern, Matsui proposed an algorithm in [13, Section 6] that produced the following 24-byte colliding key pair.

K_1 = 000 066 206 211 223 221 182 157 065 061 189 058 177 022 090 051 237 162 205 031 226 140 001 118,

K_2 = 000 066 206 211 223 221 182 157 065 061 189 058 177 022 090 051 237 162 205 031 226 140 001 119.

He also reported the following 20-byte near-colliding key pair that yields two states differing in only two positions.

K_1 = 000 115 047 106 001 055 137 197 021 073 154 085 152 084 215 083 078 246 079 220,

K_2 = 000 115 047 106 001 055 137 197 021 073 154 085 152 084 215 083 078 246 079 221.

2.2 Chen-Miyaji's Work [4]

In [4], Chen-Miyaji improved Matsui's collision search algorithm using some new techniques. We briefly summarize their techniques here, since we will point out some weaknesses of these techniques shortly.

2.2.1 Bypassing the first round interval deterministically.

To bypass the first round, i.e., to satisfy the class 1 conditions for the first round interval, the following three steps are needed.

(1) Run the KSA algorithm up to $i = d - 1$ to get $J_{1,d-1}$ $(= J_{2,d-1})$ using the keys K_1, K_2 respectively.

(2) Modify the keys as $K_1[d] = 256 - J_{1,d-1}$, $K_2[d] = K_1[d] + 1$ will be resulting to $J_{1,d} = d, J_{2,d} = d + 1$. Because,

$$J_{1,d} = J_{1,d-1} + K_1[d] + S_{1,d}[d] = J_{1,d-1} + 256 - J_{1,d-1} + d = d$$

$$J_{2,d} = J_{2,d-1} + K_2[d] + S_{2,d}[d] = J_{2,d-1} + 256 - J_{2,d-1} + 1 + d = d + 1.$$

(3) To satisfy $J_{1,d+1}(= J_{2,d+1}) = d + k$, we modify the keys on the key index $d + 1$ as $K_1[d + 1] = K_2[d + 1] = k - d - 1$, which results us,

$$J_{1,d+1} = J_{1,d} + K_1[d + 1] + S_{1,d+1}[d + 1] = d + (k - d - 1) + (d + 1) = d + k$$

$$J_{2,d+1} = J_{2,d} + K_2[d + 1] + S_{2,d+1}[d + 1] = (d + 1) + (k - d - 1) + d = d + k.$$

2.2.2 Bypassing the second Round with high probability.

By choosing $d = k - 3$ and assuming all the class 2 conditions for the second round interval are satisfied, we need to satisfy the class 1 conditions for the second round interval, i.e., $J_{1,b_2} = J_{2,b_2} = b_3$. Now we can write,

$$J_{1,b_2} = (((((J_{1,a_2} + K_1[d+2] + S_{1,d+2}[d+2]) + K_1[d+3] + S_{1,d+3}[d+3]) + ..)...) + K_1[b_2] + S_{1,b_2}[b_2])$$

$$= J_{1,a_2} + K_1[d+2] + \sum_{i=0}^{i=d} K_1[i] + \sum_{i=d+2}^{i=b_2} S_{1,i}[i].$$

i.e.,

$$K_1[d+2] = J_{1,b_2} - J_{1,a_2} - \sum_{i=0}^{i=d} K_1[i] - \sum_{i=d+2}^{i=b_2} S_{1,i}[i]$$

which can be written as,

$$K_1[d+2] = J_{1,b_2} - J_{1,a_2} - \sum_{i=0}^{i=d} K_1[i] - \sum_{i=d+2}^{i=b_2} S_{1,a_2+1}[i] \tag{1}$$

i.e., at the $i = d+1$ iteration in KSA after swapping, if we modify the value of $K_1[d+2]$ using Equation (1) with the required conditions $\forall l \in [d+2, b_2]$: $J_{1,l} \notin [l+1, b_2]$, the second round interval will pass with some probability p_2.

Similarly, $J_{2,b_2} = J_{1,b_2}$, because

$$\sum_{i=0}^{i=d} K_1[i] + \sum_{i=d+2}^{i=b_2} S_{1,i}[i] = \sum_{i=0}^{i=d} K_2[i] + \sum_{i=d+2}^{i=b_2} S_{2,i}[i].$$

Now assume $J_{1,b_2} \neq b_3$. If the above mentioned conditions hold at the $i = d+1$ iteration in KSA after swapping, then the second round interval will pass probabilistically by setting the value of $K_1[d+2]$ using Equation (1). Thus it is always possible to get a unique value $\delta = b_3 - J_{b_2}$ and by modifying the keys as $K^{'}\langle d+2, \delta\rangle$, there will be a chance to satisfy the class 1 conditions for the second round interval. The probability to pass the second round interval is given by

$$p_2 = \frac{256-(d+1)}{256} \cdot \frac{256-d}{256} \cdots \frac{256-1}{256} = \prod_{i=1}^{d+1} \frac{256-i}{256}.$$

For short keys such as $k = 24$, $p_2 = 0.39$ and for $k = 22$, $p_2 = 0.43$.

2.2.3 Last round interval passing technique to reduce complexity. For the class 1 conditions for the last round interval, we have to satisfy: $J_{1,b_n-2} = J_{2,b_n-2} = x$ so that $S_{1,b_n-2}[x] = d, S_{2,b_n-2}[x] = d+1$ and $J_{1,b_n-1} = b_n - 1, J_{2,b_n-1} = b_n - 2$.

As we know,

$$J_{1,b_n-1} = J_{1,b_n-2} + K_1[b_n-1] + S_{1,b_n-1}[b_n-1] = J_{b_n-2} + K_1[d-1] + S_{1,b_n-1}[b_n-1],$$
$$J_{2,b_n-1} = J_{2,b_n-2} + K_2[b_n-1] + S_{2,b_n-1}[b_n-1] = J_{b_n-2} + K_2[d-1] + S_{2,b_n-1}[b_n-1].$$

Now from the above mentioned conditions and equations, we can set the value of $K[d-1]$ such that $J_{1,b_n-1} = b_n - 1, J_{2,b_n-1} = b_n - 2$ always hold. Also the probability to get the correct state index x is almost uniform. So if $\forall i \in [d+2, b_n-3]$, $J_{1,i} = J_{2,i}$ $(\neq d)$ holds and assuming $J_{1,b_n-2} = J_{2,b_n-2} = d$, then we can modify $K[d-1]$ before the swap at $i = b_n - 1$ iteration in KSA, as:

$$K_2[d-1] = K_1[d-1] = J_{1,b_n-1} - J_{1,b_n-2} - S_{1,b_n-1}[b_n-1]$$
$$= ((n-1)k + d - 1) - d - (d+1) = (n-1)k - d - 2.$$

This modification indicates that if some trial meets $J_{1,b_n-2} = d$ with $J_{1,i} \neq d$, $\forall i \in [d+2, b_n-3]$, then with probability 1 we can satisfy the conditions $J_{1,b_n-1} = b_n - 1, J_{2,b_n-1} = b_n - 2$.

2.2.4 Multi-key modification.

Chen-Miyaji proposed a multi-key modification technique to pass the class 1 conditions for all round intervals t, $3 \leq t < n-1$. The multi-key modification technique makes k number of key modifications to pass the t-th round. Let us consider a trial that passes all the first $t-1$ round intervals but fails to pass the t-th round interval $[a_t, b_t]$, i.e., $J_{1,b_t} (= J_{2,b_t}) \neq b_{t+1}$. One can write

$$J_{1,b_t} = J_{1,b_{t-1}} + \sum_{i=a_t}^{b_t} K_1[i] + \sum_{i=a_t}^{b_t} S_{1,i}[i]$$

$$J_{2,b_t} = J_{2,b_{t-1}} + \sum_{i=a_t}^{b_t} K_2[i] + \sum_{i=a_t}^{b_t} S_{2,i}[i] = J_{1,b_{t-1}} + \sum_{i=a_t}^{b_t} K_1[i] + \sum_{i=a_t}^{b_t} S_{1,i}[i]$$

As,

$$\sum_{i=a_t}^{i=b_t} K_1[i] + \sum_{i=a_t}^{i=b_t} S_{1,i}[i] = \sum_{i=a_t}^{i=b_t-1} K_1[i] + K_1[b_t] + \sum_{i=a_t}^{i=b_t-1} S_{1,i}[i] + S_{1,b_t}[b_t]$$

$$= \sum_{i=a_t}^{i=b_t-1} K_2[i] + K_2[b_t] - 1 + \sum_{i=a_t}^{i=b_t-1} S_{2,i}[i] + S_{2,b_t}[b_t] + 1 = \sum_{i=a_t}^{i=b_t} K_2[i] + \sum_{i=a_t}^{i=b_t} S_{2,i}[i],$$

where $b_t \bmod k = d$ and $S_{1,b_t}[b_t] = d+1, S_{2,b_t}[b_t] = d$.

So for each $i \in [a_t, b_t]$, one can calculate the exact state value from the above relation as

$$\Delta_i = S_{1,i}[i] = J_{1,b_t} - J_{1,b_{t-1}} - \sum_{i=a_t}^{b_t} K_1[i] - \sum_{\substack{l=a_t \\ l \neq i}}^{b_t} S_{1,l}[l].$$

Thus for each $i \in [a_t, b_t]$, if $\Delta_i \leq d + (t-2)k$, then it may pass the t-th round interval by modifying the key as

$$K_1[\Delta_i] \leftarrow K_1[\Delta_i] + (i - J_{1,\Delta_i}), \qquad K_1[\Delta_i + 1] \leftarrow K_1[\Delta_i + 1] - (i - J_{1,\Delta_i}),$$

$$K_2[\Delta_i] \leftarrow K_2[\Delta_i] + (i - J_{2,\Delta_i}), \qquad K_2[\Delta_i + 1] \leftarrow K_2[\Delta_i + 1] - (i - J_{2,\Delta_i}),$$

where J_{1,Δ_i} is the j-value at the Δ_i-th index before the key modification at Δ_i-th iteration in KSA.

Using the bypassing techniques and the multi-key modification, Chen-Miyaji proposed an algorithm in [4, Table 3] that we summarize in Algorithm 4 and produced the following 22-byte colliding key pair.

$K_1(K_2)$: 162 039 067 167 003 148 047 023 117 187 167 039 143 221 062 123 198 161 199 129 (130) 002 090.

3 Some Limitations of Matsui and Chen-Miyaji's Collision Search Algorithms and Our Remedies

In this section, we point out subtle flaws in Matsui and Chen-Miyaji's collision search algorithm and mention how to correct them. In the next section, we incorporate the corrections to circumvent these flaws and present an improved algorithm. Suppose that the search subroutine fails to satisfy the class 2 conditions on some index x in the t-th round. At this point, the previous algorithms try to adjust the key value at index x by adding $y \in [1, 255]$ (modification $K\langle x, y\rangle$) to satisfy the failed condition. Also if in the t-th round, the class 1 j-conditions are not satisfied, then Matsui's algorithm performs all key modifications $K\langle x, y\rangle$, for each $x \in [0, 255], y \in [1, 255]$ to pass this round interval and Chen-Miyaji algorithm employs multi-key modification to pass this round.

3.1 Limitations of Matsui's algorithm and our remedies

In the worst case, the search algorithm of Matsui makes 256×255 key modifications for each failed condition. Suppose there is a key modification $K\langle x_1, y_1\rangle$ and $MaxS$ is the current maximum of MaxColStep(K_1, K_2), i.e., the maximal step i such that the distance between S_1 and S_2 is at most two. Let $z = x_1 \bmod k$ and take a round interval l $(< n)$ such that $z + (l-1)k < MaxS$. Then the new key modification $K\langle x_1, y\rangle$, $y > y_1$ will work only if for each such l, any one of the following cases hold for each $i = 1, 2$:

1. Given $J_{i,z+(l-1)k} \notin (z + (l-1)k, MaxS]$,
   (a) either $J^{'}_{i,z+(l-1)k} \notin (z + (l-1)k, MaxS]$,
   (b) or $J^{'}_{i,z+(l-1)k} \in (z + (l-1)k, MaxS]$, and $\exists m \in (z + (l-1)k, J^{'}_{i,z+(l-1)k})$ such that $J_{i,m} = J^{'}_{i,z+(l-1)k}$.
2. Given $J_{i,z+(l-1)k} \in (z + (l-1)k, MaxS]$,
   (a) either $J^{'}_{i,z+(l-1)k} \in (z + (l-1)k, MaxS]$, and $\exists m \in (z + (l-1)k, J_{i,z+(l-1)k}), m^{'} \in (z + (l-1)k, J^{'}_{i,z+(l-1)k})$ with $J_{i,m} = J_{i,z+(l-1)k}, J_{i,m'} = J^{'}_{i,z+(l-1)k})$.
   (b) or $J^{'}_{i,z+(l-1)k} \notin (z + (l-1)k, MaxS]$, and $\exists m \in (z + (l-1)k, J_{i,z+(l-1)k})$ with $J_{i,m} = J_{i,z+(l-1)k}))$.

In Matsui's algorithm, the index z is modified irrespective of whether these conditions hold or not, thereby increasing the number of key modifications. To circumvent the extra key modifications, we take an array A of size k, which keeps track of those y values that increase the current $MaxS$ position on the state array corresponding to each key index x. This enables us to reduce the number of modifications from 255 to $255 - y$ (corresponding to a key index x) in the subsequent search function calls.

3.2 Limitations of Chen-Miyaji's algorithm and our remedies

In this section we discuss some problems in Chen-Miyaji's algorithm [4, 6], and also discuss how do we fix those problems.

3.2.1 Some algorithmic errors.

In the **newsearch**(K_1, K_2) function of Chen-Miyaji's search algorithm (Algorithm 4), at line number 9, $MaxR = t - 1$ and $r = b_t$. At line number 10, the while loop starts at $r = b_t$ and stops until $r = a_t$. Inside the loop, when r decreases, then the corresponding Δ_r value does not follow the multi-key modification technique because of the state sum expression $(\sum_{i=r-k+1}^{i=r-1} S_{1,i}[i])$ at line number 11. This state sum expression at line number 10 should be of the form $\sum\limits_{i=a_t}^{i=b_t} S_{1,i}[i] - S_{1,r}[r]$.

Algorithm 4: Chen-Miyaji's Search Algorithm

Input: K_1, K_2, d, n, k
Output: $\mathbf{KSA}(K_1) = \mathbf{KSA}(K_2)$

1 Store the following J^* values in the table, which are the conditions needed to be satisfied. $J^*_d = d, J^*_i = i + k$ for $i \in d + k, ..., d + (n - 2)k$;
2 Randomly generate a key K_1 with key length k. Modify $K_1[d-1] = (n-1)k - d - 2$, $K_1[d+1] = k - d - 1$. Set $K_2 = K_1$ and $K_2[d] = K_1[d] + 1$;
3 Run the KSA until $i = d - 1$ after the swap. Modify $K_1[d] = 256 - J_{1,d-1}$, $K_2[d] = K_1[d] + 1$;
4 Keep running the KSA until $i = d + 1$ after the swap. Modify
$K_1[d+2] = \left(k - \sum_{i=0}^{i=d} K_1[i] - \sum_{i=d+2}^{i=d+k} S_{1,d+2}[i]\right) \bmod 256$,
$K_2[d+2] = K_1[d+2]$;
5 Set the recursive depth variable $R = 0$;
6 **if** *newsearch*$(K_1, K_2) = n$ **then**
  Colliding key pair found. Output K_1, K_2;
7 **else**
  goto 2;

newsearch(K_1, K_2):
8 **if** ***Round***$(K_1, K_2) = n$ **then**
  return n;
9 $MaxR = \mathbf{Round}(K_1, K_2) = t - 1$, set $r = d + (t-1)k$;
10 **while** $r > d + (t-2)k$ **do**
11   set $\Delta_r = \left(k - \sum_{i=0}^{i=k-1} K_1[i] - \sum_{i=r-k+1}^{i=r-1} S_{1,i}[i]\right) \bmod 256$;
12   **if** $\Delta_r \leq d + (t-2)k$ **then**
    modify the key as follows:
    $K_1[\Delta_r] = K_1[\Delta_r] + r - J_{1,\Delta_r}$;
    $K_1[\Delta_r + 1] = K_1[\Delta_r + 1] - r + J_{1,\Delta_r+1}$;
    $K_2[\Delta_r] = K_2[\Delta_r] + r - J_{1,\Delta_r}$;
    $K_2[\Delta_r + 1] = K_2[\Delta_r + 1] - r + J_{1,\Delta_r+1}$;
13   **if** ***Round***$(K_1, K_2) \leq MaxR$ *or* $R = n$ **then**
    return $\mathbf{Round}(K_1, K_2)$
14   **else**
    $R = R + 1$;
    newsearch(K_1, K_2)
15   $r = r - 1$;

Round(K_1, K_2):: The number of round intervals that a key pair K_1, K_2 can pass.

Algorithm 5: Corrected Chen-Miyaji's Search Algorithm

Input: K_1, K_2, d, n, k
Output: $\mathbf{KSA}(K_1) = \mathbf{KSA}(K_2)$

1 Store the following J^* values in the table, which are the conditions needed to be satisfied. $J^*_d = d, J^*_i = i + k$ for $i \in d + k, ..., d + (n - 2)k$;
2 Randomly generate a key K_1 with key length k. Modify $K_1[d-1] = (n-1)k - d - 2$, $K_1[d+1] = k - d - 1$. set $K_2 = K_1$ and $K_2[d] = K_1[d] + 1$;
3 Run the KSA until $i = d - 1$ after the swap. Modify $K_1[d] = 256 - J_{1,d-1}$, $K_2[d] = K_1[d] + 1$;
4 Keep running the KSA until $i = d + 1$ after the swap. Modify
$K_1[d+2] = \left(k - \sum_{i=0}^{i=d} K_1[i] - \sum_{i=d+2}^{i=d+k} S_{1,d+2}[i]\right) \bmod 256$,
$K_2[d+2] = K_1[d+2]$;
5 Set the recursive depth variable $R = 0$;
6 **if** *newsearch*$(K_1, K_2) = n$ **then**
  Colliding key pair found. Output K_1, K_2;
7 **else**
  goto 2;

newsearch(K_1, K_2):
8 $MaxR = \mathbf{Round}(K_1, K_2) = t$, set $r = b_t$;
9 **if** $MaxR = n$ **then**
  return n;
10 **if** $MaxR < n - 1$ **then**
11   **while** $r \geq a_t$ **do**
12   set $\Delta_r = \left(k - \sum_{i=0}^{i=k-1} K_1[i] - \sum_{i=a_t}^{i=b_t} S_{1,i}[i] + S_{1,r}[r]\right) \bmod 256$;
13   **if** $\Delta_r \leq a_t - 2$ **then**
      $K\langle \Delta_r, r - J_{1,\Delta_r} \rangle$;
14       $MaxR_1 = \mathbf{Round}(K_1, K_2)$;
15   **if** $MaxR_1 > MaxR$ **then**
        break;
16     $r = r - 1$;
17   **if** ***Round***$(K_1, K_2) \leq MaxR$ *or* $R = n$ **then**
    return $MaxR$;
18   **else**
    $R = R + 1$;
    newsearch(K_1, K_2)
19 **else**
  $MaxR_2 = MaxR$;
20   **for** $x = 0$ *to* $d - 3$ **do**
21   **for** $y = 1$ *to* 255 **do**
      $K\langle x, 1 \rangle$, $MaxR_1 = \mathbf{Round}(K_1, K_2)$;
22   **if** $MaxR_1 > MaxR_2$ **then**
        $MaxR_2 = MaxR_1$, break;
23   **if** ***Round***$(K_1, K_2) \leq MaxR$ *or* $R = n$ **then**
    return $MaxR$;
24   **else**
    $R = R + 1$, **newsearch**(K_1, K_2);

Round(K_1, K_2):: The number of round intervals that a key pair K_1, K_2 can pass.

Also, after the key modifications on the key indices $\Delta_r, \Delta_r + 1$ at $r = b_t$ inside the while loop, if $\mathbf{Round}(K_1, K_2) \leq MaxR$ (line number 13) happens, then the $\mathbf{newsearch}(K_1, K_2)$ stops and the next random key pair is tried to get a colliding key pair. But according to the multi-key modification technique, this while loop will stop until $r = a_t$ and hence the if-else conditions (line 13 and 14) should be outside this while loop. Furthermore, the multi-key modification technique is applied for those round intervals t, where $2 < t < n-1$. But in Algorithm 4, there is no information about how to pass the last two round intervals.

We rectify these mistakes and submit a corrected Chen-Miyaji's search algorithm in Algorithm 5, where we use all the key modifications to pass each of the last two round intervals.

3.2.2 Flaw in the second round interval passing technique. To satisfy the class 1 conditions for the second round interval, Chen-Miyaji's algorithm first checks the condition $\forall l \in [b_1+1, b_2-1], J_{1,l}(=J_{2,l}) \notin [l+1, b_2]$ and then computes a unique value δ $(= b_3 - J_{1,b_2})$. The key modification $K^{'}\langle b_1+2, \delta\rangle$ may pass the class 1 conditions for the second round interval. However, if we modify the key as $K^{'}\langle b_1+2, \delta\rangle$ to satisfy the class 1 conditions for the second round interval, then for each $i = 1, 2$, it will also modify the $J_{i,l}$ values to $J^{'}_{i,l} = J_{i,l} + \delta$ for $l \in [b_1+2, b_2]$. Now it might happen that for some $l \in [b_1+2, b_2-1]$, $J^{'}_{1,l}(=J^{'}_{2,l}) \in [l+1, b_2]$ would violate the class 1 conditions for the second round interval and the Chen-Miyaji's condition as $J_{1,l} \notin [l+1, b_2]$. To avoid this, we modify their technique as follows:

1 If $\forall l \in [b_1+2, b_2-1], J_{1,l} \notin [l+1, b_2]$ holds, then we get a unique value $\delta = (b_3 - J_{1,b_2})$.

2 After adding δ to $K[b_1+2]$, we need to satisfy $\forall l \in [b_1+2, b_2-1] : J^{'}_{1,l} \notin [l+1, b_2]$.

3.2.3 Some problems in multi-key modification. Note that in multi-key modification, there are two cases: either $S_{p,\Delta_i}[\Delta_i] = \Delta_i$ or not, for each $p = 1, 2$. We show that both of these two cases are problematic for Chen-Miyaji's algorithm to pass the t-th round. So for each $p = 1, 2$,

1. If $S_{p,\Delta_i}[\Delta_i] \neq \Delta_i$, $\forall\ i \in [a_t, b_t]$, then one can pass the t-th round interval if $\exists$ an index $q \in [a_t, b_t - 1]$ such that $J^{'}_{p,q} = J_q + \Delta$, where $J^{'}_{p,b_t} = J_{p,b_t} + \Delta = d + tk$, with the required condition that $J^{'}_{p,r} \notin [r+1, b_t], \forall r \in [J^{'}_{p,q}+1, b_t]$.
2. If $S_{p,\Delta_i}[\Delta_i] = \Delta_i$ happens for some $i \in [a_t, b_t]$, then the key modification is performed on the Δ_i-th index corresponding to those i's. To ensure that the previous round intervals will not be affected due to these key modifications, we have to take care of the following conditions.
   (a) If $J_{p,\Delta_i - mk} \in (\Delta_i - mk, b_t]$, then one has to find that $\exists$ an index $r \in (\Delta_i - mk, J_{p,\Delta_i-mk}]$ such that $J_{p,r} = J_{p,\Delta_i-mk}$ and so $S_{p,r+1}[J_{p,\Delta_i-mk}] = S_{p,J_{p,\Delta_i-mk}}[J_{p,\Delta_i-mk}]$ and also one needs to check that $J^{'}_{p,\Delta_i-mk} \notin (\Delta_i - mk, b_t]$; or if $J^{'}_{p,\Delta_i-mk} \in (\Delta_i - mk, b_t]$, then to find that $\exists$ an index $r^{'} \in (\Delta_i - mk, J^{'}_{p,\Delta_i-mk}]$ so that $J_{p,r'} = J^{'}_{p,\Delta_i-mk}$, shown in Fig. 1.
   (b) If $J_{p,\Delta_i-mk} \notin (\Delta_i - mk, b_t]$ with $J^{'}_{p,\Delta_i-mk} \notin (\Delta_i - mk, b_t]$, then it creates no problem on that round. But if $J^{'}_{p,\Delta_i-mk} \in (\Delta_i - mk, b_t]$ then one needs to check that $\exists$ an index $r^{'} \in (\Delta_i - mk, b_t]$ so that $J_{p,r'} = J^{'}_{p,\Delta_i-mk}$, shown in Fig. 2.

Note that, we don't need to calculate Δ_{b_t} and the key modification at index b_t, because in the t-th round, we always have $\forall m \in [a_t, b_t - 1]$: $S_{1,m}[b_t] = (d+1), S_{2,m}[b_t] = d$ according to Matsui's transition pattern. It is obvious from our discussion that if the round interval number increases, then it will become less probable to pass that round. But there may be some other key modifications (other than considered here) which might help to pass that round. Our work is based on finding such other key modifications. Although Chen-Miyaji's algorithm spends less time on a random key, it has less probability of converting a random key to a colliding pair. Our algorithm spends slightly more time on a random key (to search for other key modifications), but it increases the probability

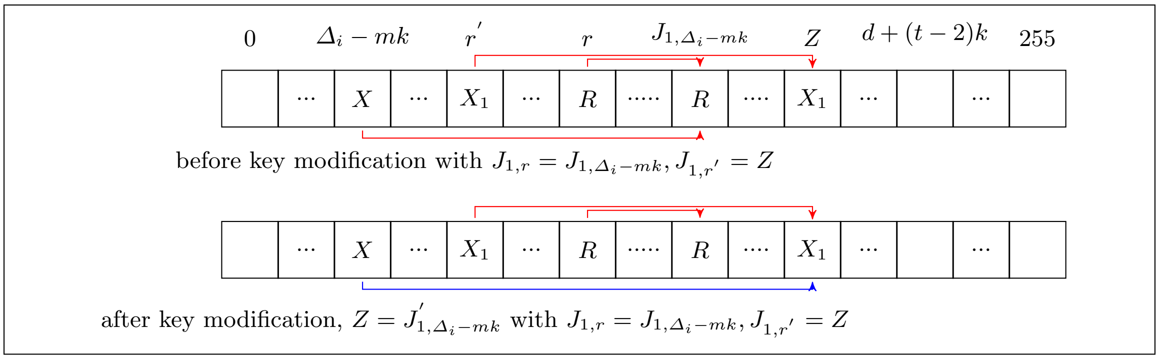

Fig. 1. Illustration of Case 2.(a) in Section 3.2.3

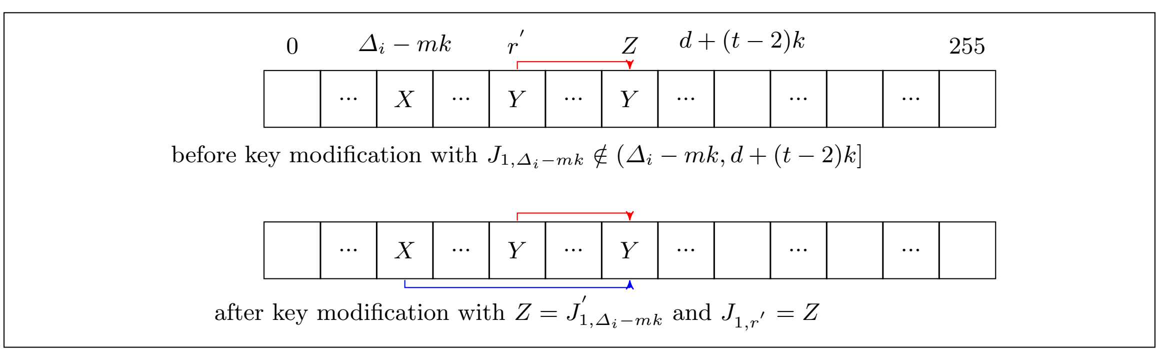

Fig. 2. Illustration of Case 2.(b) in Section 3.2.3

of converting a random key to a colliding pair. Our overall search complexity is less as compared to Chen-Miyaji's algorithm, which we will briefly analyze and show the complexity comparisons in Section 5.
In [6], Chen-Miyaji modify all $K\langle x, y\rangle, x \in [0, d-3], y \in [0, 255]$ instead of using the multi-key modification to pass the t-th $(2 < t < n-1)$ round interval and get the same 22-byte colliding key pair in five hours. For any round interval t (except the first round), there are $(k-1)$-positions in which we can change state values (differing from its previous value) such that it would help to pass this round. In [4, Table 3], the Chen-Miyaji technique uses all the key modifications on the key indices $x \in [0, d-3]$ and does not keep track of the y values corresponding to those x that would have caused additional key modifications. But in our algorithm we will take care of these extra key modifications.

4 Our New Collision Search Algorithm

Keeping both Matsui's and Chen-Miyaji's algorithms in mind, we design a new algorithm that works very fast and is efficient. In our algorithm, we use Chen-Miyaji's first and last round interval passing techniques, but we don't use their second round interval passing technique, because the efficiency of this technique decreases as the key size increases. Instead, we use a new technique (with $d = k-3$), called the *last key index increasing technique* as follows:

- Pass the first round interval deterministically and the class 2 conditions of the second round.
- Modify the key as $\forall y \in [1, 255]$: $K^{'}\langle k-1, y\rangle$.

In our algorithm, we use this technique along with the key modification $K\langle x, y\rangle$, $0 \le x < k, 1 \le y \le 255$ except at $x \ne d-2, d-1, d, d+1$, to pass the second round. Our experimental results in Table 3 indicate that for all key sizes our technique has a better chance of passing the second round interval as compared to the previous existing techniques.

Number of bytes	Expected Probability of Matsui's Algorithm	Expected Probability of Chen-Miyaji's Algorithm	Expected Probability of Last key index increasing technique	Expected Probability of Our Algorithm
30	0.43	0.16	0.40	0.55
28	0.40	0.25	0.42	0.57
25	0.31	0.27	0.44	0.59
24	0.30	0.35	0.47	0.62
23	0.29	0.40	0.49	0.64
22	0.27	0.42	0.53	0.65
20	0.23	0.49	0.59	0.67

Table 3. Second round interval passing comparison with 2^{30} samples

We see in Table 3 that the expected probability to pass the second round interval using Matsui's algorithm also decreases, as the key size decreases, because Matsui's algorithm uses a brute force search to pass any round interval and also there is no key modification on the last key index (i.e., $K^{'}\langle k-1, y\rangle, y \in [1, 255]$), which could have increased the probability to pass the second round. For each key modification as $K\langle x, y\rangle$, $0 \le x \le k-3$, $1 \le y \le 255$, there might occur a state difference δ at the state index $i \in [a_2, b_2 - 1]$ so that $J'_{1,b_2} = J_{1,b_2} + \delta = b_3$ holds. Hence the effectiveness to pass the second round interval decreases as the key size decreases for Matsui's algorithm. But for last key index increasing technique, which is a generalization of Chen-Miyaji's second round interval passing technique, we modify the key as $K^{'}\langle k-1, y\rangle$ for each $y \in [1, 255]$. So there are two cases,

– if $\forall i \in [a_2+2, b_2-1], J_{1,i} \notin [i+1, b_2]$ hold, then we will have $J'_{1,i} = J_{1,i} + y$, $\forall i \in [a_2+2, b_2]$,
– or if there occurs a state difference δ at any index $i \in [a_2+2, b_2-1]$, then $J'_{1,b_2} = J_{1,b_2} + y = b_3$, together with $J_{1,l} \notin [l+1, b_2], \forall l \in [i+1, b_2]$.

Consequently, there is more chance to satisfy $J_{1,b_2} = b_3$ as the key size decreases, because the probability that $J_{1,i} \notin [i+1, b_2], \forall i \in [a_2+2, b_2]$ increases. Furthermore, our algorithm uses both of the last key index increasing technique and the key modifications $K\langle x, y\rangle$, $0 \le x \le k-6$ and $1 \le y \le 255$. So using our algorithm, the probability to pass the second round interval should be inversely proportional to the key size.

We present our search strategy in Algorithm 6, Algorithm 7, Algorithm 8 and Algorithm 9. We use only one key K (instead of two keys K_1, K_2), as the other key can be computed by a difference in position d. Algorithm 9 generates a random key and passes the first and second round intervals (using Algorithm 7) deterministically, after that we call Algorithm 8, which in turn invokes Algorithm 6 to check all the class 1 and class 2 conditions in Table 2 which returns the corresponding failing state index, whenever j-values break either of the class 1 or class 2 conditions according to Table 2.

One naive approach to design a search algorithm (after passing the first and second round) would be that for each key index $x \in [0, k-1]$, we perform $K\langle x, 1\rangle$ for 255-times and each time we check whether the returned state index ($Temp$) is greater than the current $MaxS$ value (because according to Lemma 1, if any one of the class 2 conditions at the t-th round interval (say at the index x) is not satisfied, then we could satisfy that condition by performing the key modifications on $x, x+1$. If it holds, we save the corresponding indices as the key index x and the increased value y and stop modification on that key index subsequently. Finally, we check if our current $MaxS >$ previous $MaxS$ ($= P$, see Algorithm 8). If it is true, then we call this function recursively until we convert this key pair into a colliding key pair, otherwise we stop this recursive call and generate a new key pair to process. Now, it is possible that whenever we get a maximum returned state index $temp$ (so that $temp > MaxS$) by modifying the key on index x, $\forall y \in [1, 255]$, further key modifications on that key index might not help to pass the next round intervals. So to overcome this situation, we introduce a new search algorithm in Algorithm 8 (named as **OurNewSearch**(K_1)) that supervises to reduce some extra key modifications corresponding to each key indices for every recursive call. We need an array A with size k to keep track of y values corresponding to each key indexes.

To get near-colliding pair, we will do a slight modification in Algorithm 6 (**NewMaxColStep**(K_1)) as follows.

– Check all j-conditions according to Table 1 up to $i = b_{n-1} - 1$.
– At $i = b_{n-1}$, we change the condition as $J_{1,b_{n-1}} = J_{2,b_{n-1}} = b_n$.
– For $i \in [b_{n-1}+1, b_n-2]$, the j-conditions remain the same as in Table 1 and for $i = b_n - 1$, $J_{1,i}(= J_{2,i}) \neq i+1$.
– At $i = b_n$, we need to satisfy the condition $J_{1,b_n}(= J_{2,b_n}) \le b_n$.

Algorithm 6: **NewMaxColStep**(K_1)

Input: K_1, k, d
Output: State index $i \in [0, 255]$
1 **for** $i = 0$ *to* 255 **do**
  $S[i] = i$;
2 **for** $i = 0$ *to* 255 **do**
  $j = (j + S[i] + K[i \bmod k]) \bmod 256$;
  equip all j conditions according to Table 2 to check and return the index i if J_i does not satisfy these required conditions and with the additional conditions as $\forall i \in [d+1, d+(n-1)k-3] : J_i \neq d$ (Used for Chen-Miyaji's last round interval passing technique);
  swap$(S[i], S[j])$;
return 255;

Algorithm 7: **PassSecondRound**(K_1)

Input: K_1
Output: State index $i \in [0, 255]$
$P =$ **NewMaxColStep**(K_1);
while $P < d + k$ **do**
  $Q = S$;
  if P mod $k = d-2, d-1, d, d+1$ **then**
    return P;
  for $y = 1$ *to* 256 **do**
    $K_1\langle P \bmod k, y\rangle$;
    $R =$ **NewMaxColStep**(K_1);
    if $R > Q$ **then**
      $A[P \bmod k] = y$;
      $Q = R$;
    if $P > Q$ **then**
      return P;
    else
      $K_1\langle P \bmod k, A[P \bmod k]\rangle$;
    $P = Q$;
for $y = 1$ *to* 256 **do**
  $K_1^{'}\langle P \bmod k, y\rangle$;
  $P =$ **NewMaxColStep**(K_1);
  if $P > d + k$ **then**
    return P

/* In addition to, you can also use all the key modifications ($K\langle x, y\rangle, x \in [0, d-3], y \in [1, 255]$) to pass the second round, but this is optional. */

Algorithm 8: **OurNewSearch**(K_1)

Input: Key K_1
Output: Boolean
$P =$ **NewMaxColStep**(K_1);
if $P = 255$ **then**
  return True;
Set $MaxP = P$, $Flag =$ False;
for $x = 0$ *to* $d - 3$ **do**
  if $A[x] = 0$ **then**
    for $y = 1$ *to* 256 **do**
      Modify $K_1\langle x, 1\rangle$ and $Temp =$ **NewMaxColStep**(K_1);
      if $Temp = 255$ **then**
        return True;
      if $Temp > MaxP$ **then**
        $MaxP = Temp, A[x] = y$ and $Flag =$ True;
  if $Flag = True$ **then**
    Modify $K_1\langle x, A[x]\rangle$;
    $Flag =$ False;
if $MaxP \leq P$ **then**
  return False
return (**OurNewSearch**(K_1));

Algorithm 9: **New Collision Search Algorithm**

Input: Key K_1 with key length k and fixed d ($< k-2$)
Output: Colliding key K_1

while *True* **do**
  Generate a random key pair K_1;
  Set $K_1[d-1] = (n-1)k - d - 2$;
  Set $K_1[d+1] = k - d - 1$;
  $Temp = \textbf{NewMaxColStep}(K_1)$;
  while $Temp < d - 2$ **do**
    $Temp_1 = \textbf{NewMaxColStep}(K_1)$;
    for $y = 1$ *to* 255 **do**
      $K_1\langle y, 1\rangle$;
      $Temp = \textbf{NewMaxColStep}(K_1)$;
      if $Temp > Temp_1$ **then**
        break;
    $Temp = Temp_1$
  Modify $K_1[d]$ as: $K_1[d] = 256 - J_{1,d-1}$;
  for $i = 0$ *to* $k - 1$ **do**
    $A[i] = 0$;
  $Temp = \textbf{PassSecondRound}(Key1)$;
  if $Temp > d + k$ **then**
    if ***OurNewSearch***$(K_1) =$ *True* **then**
      stop (found a collision) or continue (to find more);

return;

4.1 Average Number of Key Modifications in Our and Previous Algorithms

By choosing each $k, d\ (= k-3)$, we see from our algorithm that the average number of key modifications is $(k-6)\cdot 2^8 + 2^8$ (2^8 for the last key index increasing technique) to pass first and second round. Also to pass the remaining round intervals, our algorithm performs on an average $(k-6)\cdot 2^8 \cdot \frac{n-2}{2}$ number of key modifications. So on an average our algorithm does total $(k-5)\cdot 2^8 + (k-6)\cdot 2^8 \cdot \frac{n-2}{2}$ number of key modifications. Now Chen-Miyaji's algorithm in [4] use multi-key modification technique, that uses k number of key modifications to pass each round. To pass the last two round intervals, Chen-Miyaji's algorithm needs $2\cdot(k-6)\cdot 2^8$ number of key modifications on an average. But for the remaining round intervals their algorithm uses $(n-4)\cdot k$ number of key modifications. So on an average Chen-Miyaji's algorithm needs $(n-4)\cdot k + (k-6)\cdot 2^9$ number of key modifications and Matsui's algorithm needs $2\cdot \frac{n}{2}\cdot k \cdot 2^8 = n\cdot k\cdot 2^8$ number of key modifications.

4.2 Our Results: New Colliding and Near-Colliding Key Pairs

We ran 3 instances (in parallel) of an unoptimized C-implementation of our new algorithm with $d = k-3$, on an Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2 core machine, obtained one 22-byte colliding pair in two days and also by running 10 instances using an Intel(R) Xeon(R) @ 2.5GHz 8-core machine we obtained two other 22-byte colliding pairs in around two days. The 3 new 22-byte colliding key pairs are as follows.

1st pair: 028 080 034 132 091 116 236 182 098 245 149 126 004 095 090 008 074 107 199 044 (045) 002 149,

2nd pair: 088 118 029 149 220 246 232 019 000 071 078 210 065 193 001 074 087 125 199 189 (190) 002 104, 3rd pair: 143 226 130 093 081 116 138 191 040 029 225 035 010 142 060 254 152 092 199 033 (034) 002 106.

Also by running 10 instances on the same machine with $d = k - 3$, we found the following 12 new 20-byte near-colliding pairs in around two days.

(1). 109 188 227 013 189 124 109 143 099 130 244 243 114 009 250 166 128 208 (209) 002 170,

(2). 043 233 031 249 175 055 138 047 228 208 021 000 002 163 018 201 001 099 (100) 002 203,

(3). 232 212 176 190 246 112 002 175 166 212 193 168 206 163 211 225 199 (200) 107 002 152,

(4). 064 222 223 068 078 161 071 024 124 185 053 129 027 068 152 189 007 067 (068) 002 185,

(5). 154 148 200 248 009 035 047 011 093 173 028 124 089 077 008 102 058 052 (053) 002 155,

(6). 222 085 178 146 189 187 220 029 208 091 165 122 182 098 222 108 125 103 (104) 002 155,

(7). 076 234 035 052 104 054 242 004 142 162 098 020 114 095 062 107 231 080 (081) 002 155,

(8). 043 030 083 010 175 079 124 244 019 184 123 029 167 123 251 157 175 155 (156) 002 152,

(9). 215 071 075 076 115 210 173 185 242 047 220 213 193 171 244 128 255 107 (108) 002 181,

(10). 082 146 174 123 021 246 218 069 244 063 055 226 002 017 170 022 226 065 (066) 002 170,

(11). 064 222 223 068 078 161 071 024 124 185 053 129 027 068 152 189 007 067 (068) 002 185,

(12). 154 148 200 248 009 035 047 011 093 173 028 124 089 077 008 102 058 052 (053) 002 155.

5 Complexity Estimates and Comparison with Previous Works

The *search complexity* of a collision search algorithm means how many random key pairs have to be tried so that the algorithm will output a colliding key pair for each fixed k, d. We also use the term *time complexity*, which means on an average how many key modifications will be required to get a colliding key pair for each given k, d. So for each algorithm, the *time complexity* is calculated as the *search complexity* multiplied by the average number of key modifications per new random key pair trial.

We present the main result related to the success probability of our algorithm in Theorem 1. Subsequently, in Theorem 2 we connect this probability with the search complexity to find a colliding key pair.

Theorem 1. *Suppose* $\mathrm{Pr}_{t,(x,y)}$ *defines the probability that a trial passes the* t*-th* $(2 < t < n-1)$ *round interval after modifying the secret key as* $K\langle x, y\rangle$ *or* $K^{'}\langle x, y\rangle$ *according to our new algorithm,* *given that the previous trial fails to pass the t-th round. Then*

$$\Pr_{t,(x,y)} = (1-p_3) + p_3 \cdot \Big[(1-q_1) + \sum_{i=2}^{t-1} \Big(\prod_{j=1}^{i-1} q_j\Big) \cdot (1-q_i)\Big],$$

where $p_3 = (\frac{255}{256})^{k-3} \cdot \frac{1}{256}$ *and*

$$q_i = p_{1,i}^2 + 2 \cdot p_{1,i} \cdot (1-p_{1,i}) \cdot p_{2,i}^2 + (1-p_{1,i})^2 \cdot p_{2,i}^2,$$

with $p_{1,i} = \frac{256-(b_{t-1}-c_{i,x}-1)}{256}$ *and* $p_{2,i} = \frac{b_{t-1}-c_{i,x}-1}{256}$.

Proof. Let us consider that the trial passes the first r $(< t)$ round intervals, but fails to pass the t-th round interval. Then the algorithm tries to modify the key as $K\langle x,y\rangle$ or $K^{'}\langle x,y\rangle$ in our proposed algorithm. Recall that for each $i = 1,2$, $J_{i,x,r}$, $J_{i,x+1,r}$ denote the j-values before the key modification at round interval r and $J^{'}_{i,x,r}$, $J^{'}_{i,x+1,r}$ denote the modified (new) j-values after the key modification at the same round interval r. Due to the key modification, we must have $J_{i,x+1,r} = J^{'}_{i,x+1,r}$, for $i = 1,2$. So for each $i = 1,2$, these two values $J_{i,x+1,r}, J^{'}_{i,x+1,r}$ do not affect the state. But the remaining two j-values $J_{i,x,r}$, $J^{'}_{i,x,r}$, for $i = 1,2$ cause difference in the state S for some $y \in [1,255]$. We see that after the key modification in a trial, the key sum $\sum_{x=0}^{k-1} K_1[x] (= \sum_{x=0}^{k-1} K_2[x])$ always remains the same, but the state may change for any round interval r $(\leq t-1)$, because the key modification on the key index x will change the previous state value (i.e., state value before key modification). Any change of the state value due to key modification will not be impacted by the current and the previous round intervals and the corresponding restrictions are as follows.

Suppose we are at round interval r and perform a key modification as $K\langle x,y\rangle$, then the previous round intervals will not be impacted if $\forall$ round intervals $r < t$, any of the following four cases hold for each $i = 1,2$.

- $J^{'}_{i,x,r} \notin [c_{r,x}+1, b_{t-1})$, given $J_{i,x,r} \notin [c_{r,x}+1, b_{t-1})$.
- $J^{'}_{i,x,r} \in [c_{r,x}+1, b_{t-1})$ and $\exists m^{'} \in (c_{r,x}+1, J^{'}_{i,x,r})$ such that $J_{i,m} = J^{'}_{i,x,r}$, given $J_{i,x,r} \notin [c_{r,x}+1, b_{t-1})$.
- $J^{'}_{i,x,r} \notin [c_{r,x}+1, b_{t-1})$ and $\exists m \in (c_{r,x}+1, J_{i,x,r})$ such that $J_{i,m} = J_{i,x,r}$, given $J_{i,x,r} \in [c_{r,x}+1, b_{t-1})$.
- $J^{'}_{i,x,r} \in [c_{r,x}+1, b_{t-1})$ and $\exists m \in (c_{r,x}+1, J_{i,x,r})$, $\exists m^{'} \in (c_{r,x}+1, J^{'}_{i,x,r})$ such that $J_{i,m} = J_{i,x,r}$, $J_{i,m'} = J^{'}_{i,x,r}$, given $J_{i,x,r} \in [c_{r,x}+1, b_{t-1})$.

For the r-th round interval, we define the following events.
$\mathcal{A}_r$: $J_{i,x,r} \in [c_{r,x}+1, b_{t-1})$.
$\mathcal{B}_r$: $J^{'}_{i,x,r} \in [c_{r,x}+1, b_{t-1})$.
$\mathcal{C}_r$: $\exists m^{'} \in [c_{r,x}+1, J^{'}_{i,x,r})$ such that $J_{i,m'} = J^{'}_{i,x,r}$.
$\mathcal{D}_r$: $\exists m \in [c_{r,x}+1, J_{i,x,r})$ such that $J_{i,m} = J_{i,x,r}$.
$\mathcal{E}_r$: the event that the key modification in round interval r will not be broken the class 1 and class 2 j-conditions, that have already been satisfied in the previous round intervals.
$\mathcal{F}_t$: the event that our algorithm will help to pass the t-th round interval.
$\mathcal{G}_t$: the trial fails to pass the t-th round interval.

We assume that all random variables used in these defined events are independent. Thus we have,

$$\Pr(\mathcal{E}_r) = \Pr(\mathcal{A}_r^c, \mathcal{B}_r^c) + \Pr(\mathcal{A}_r^c, \mathcal{B}_r, \mathcal{C}_r) + \Pr(\mathcal{A}_r, \mathcal{B}_r^c, \mathcal{D}_r) + \Pr(\mathcal{A}_r, \mathcal{B}_r, \mathcal{C}_r, \mathcal{D}_r), \quad (2)$$

where

$$\Pr(\mathcal{A}_r^c, \mathcal{B}_r^c) = \Pr(\mathcal{A}_r^c) \cdot \Pr(\mathcal{B}_r^c) = \frac{256 - (b_{t-1} - c_{r,x} - 1)}{256} \cdot \frac{256 - (b_{t-1} - c_{r,x} - 1)}{256}, \text{ and}$$

$$\Pr(\mathcal{A}_r^c, \mathcal{B}_r, \mathcal{C}_r) = \Pr(\mathcal{A}_r^c) \cdot \Pr(\mathcal{B}_r) \cdot \Pr(\mathcal{C}_r) = \frac{256 - (b_{t-1} - c_{r,x} - 1)}{256} \cdot \frac{b_{t-1} - c_{r,x} - 1}{256} \cdot \frac{b_{t-1} - c_{r,x} - 2}{256^2}.$$

Similarly,

$$\Pr(\mathcal{A}_r, \mathcal{B}_r^c, \mathcal{D}_r) = \frac{b_{t-1} - c_{r,x} - 1}{256} \cdot \frac{256 - (b_{t-1} - c_{r,x} - 1)}{256} \cdot \frac{b_{t-1} - c_{r,x} - 2}{256^2},$$

$$\Pr(\mathcal{A}_r, \mathcal{B}_r, \mathcal{C}_r, \mathcal{D}_r) = \frac{b_{t-1} - c_{r,x} - 1}{256} \cdot \frac{b_{t-1} - c_{r,x} - 1}{256} \cdot \frac{b_{t-1} - c_{r,x} + 1}{256^2} \cdot \frac{b_{t-1} - c_{r,x} - 2}{256^2}.$$

Plugging in these values in Equation (2), we get the value of $\Pr(\mathcal{E}_r)$. Thus the probability that the key modification in round interval r will break the class 1 and class 2 j-conditions, that have already been satisfied in the previous round intervals is denoted and defined as

$$\Pr(\mathcal{E}_r^c) = 1 - \Pr(\mathcal{E}_r).$$

Also,

$$\Pr(\mathcal{F}_t) = \left(\frac{255}{256}\right)^k \cdot \frac{1}{256}$$

and hence,

$$\Pr(\mathcal{F}_t^c) = 1 - \Pr(\mathcal{F}_t).$$

Therefore, the probability that the trial fails to pass the t-th round interval will be,

$$\Pr(\mathcal{G}_t) = \Pr(\mathcal{F}_t^c) + \Pr(\mathcal{F}_t) \cdot \Pr(\mathcal{E}_1^c) + \Pr(\mathcal{F}_t) \cdot \Pr(\mathcal{E}_1) \cdot \Pr(\mathcal{E}_2^c) + \cdots + \Pr(\mathcal{F}_t) \cdot \prod_{l=1}^{t-2} \big(\Pr(\mathcal{E}_l)\big) \cdot \Pr(\mathcal{E}_{t-1}^c).$$

So the probability that the trial will pass the t-th round interval using some key modification, while the trial before the key modification passes the previous round intervals is given by $Pr_{t,(x,y)} = 1 - \Pr(\mathcal{G}_t)$.

In our algorithm, we use Chen-Miyaji's last round interval passing technique, where the last round interval indicates the interval as $[a_n, b_n]$. So in the n-th (last) round interval,

$$\Pr(\mathcal{F}_n) = \left(\frac{254}{256}\right)^{k-3} \cdot \frac{1}{256} \cdot \mathcal{X}_n,$$

where $\mathcal{X}_n$ is the probability that $\forall i \in [a_2, b_{n-1}], J_{1,i} \neq b_1$ holds i.e., $\mathcal{X}_n \approx \left(\frac{254}{256}\right)^{b_{n-1} - a_2 + 1}$.

After substituting the individual component expressions, we get the result. □

Thus we have the following result on the search complexity.

Theorem 2. *If k and $d(< k)$ are the key size and the key difference index respectively, then the search complexity to find a colliding key pair using our algorithm is denoted and given by $\mathcal{COMP}^{our} \approx 1/\hat{\Pr}_n$, where $\hat{\Pr}_n$ is the average of $\Pr_{n,(x,y)}$ over all x, y and $n = \lfloor \frac{256+k-1-d}{k} \rfloor$.*

The search complexity of Matsui's algorithm can be found in [4, Theorem 3]. It is important to note that to get the actual complexity estimates, we need to multiply each search complexity figure with the number of key modifications involved for each trial key up to the last round interval. It is easy to see that if $\mathcal{COMP}^{our}$, $\mathcal{COMP}^{CM}$ and $\mathcal{COMP}^{Mat}$ denote the search complexities of our, Matsui's and Chen-Miyaji's algorithm respectively, then the corresponding overall (time) complexities in terms of the number of key modifications needed to get a colliding key pair is given by $\mathcal{COMP}^{our}\cdot\left[(k-5)\cdot 2^8+(k-6)\cdot 2^8\cdot\frac{n-2}{2}\right]$, $\mathcal{COMP}^{CM}\cdot\left[(n-4)\cdot k+(k-6)\cdot 2^9\right]$ and $\mathcal{COMP}^{Mat}\cdot k\cdot n\cdot 2^8$ respectively. For the time being, we provide a numerical comparison of the search complexities of the three algorithms for different key lengths in Table 4.

Key size (bytes)	Number of round intervals	Matsui's algorithm		Chen-Miyaji's algorithm		Our algorithm	
		Search complexity	Time complexity	Search complexity	Time complexity	Search complexity	Time complexity
20	12	$2^{56.7}$	$2^{72.6}$	$2^{51.6}$	$2^{64.5}$	$2^{33.7}$	$2^{47.7}$
21	12	$2^{60.6}$	$2^{76.5}$	$2^{56.9}$	$2^{69.9}$	$2^{35.7}$	$2^{49.7}$
22	11	$2^{52.7}$	$2^{68.7}$	$2^{47.7}$	$2^{61.2}$	$2^{30.7}$	$2^{44.7}$
24	10	$2^{47.8}$	$2^{63.7}$	$2^{42.6}$	$2^{56.2}$	$2^{27.5}$	$2^{41.5}$
26	9	$2^{42.6}$	$2^{58.5}$	$2^{36.5}$	$2^{50.1}$	$2^{23.6}$	$2^{37.6}$
28	9	$2^{45.8}$	$2^{61.7}$	$2^{41.7}$	$2^{55.4}$	$2^{25.6}$	$2^{39.6}$
30	8	$2^{39.5}$	$2^{55.4}$	$2^{33.6}$	$2^{47.3}$	$2^{21.9}$	$2^{35.9}$
36	7	$2^{36.6}$	$2^{52.6}$	$2^{31.7}$	$2^{45.7}$	$2^{20.7}$	$2^{34.7}$
41	6	$2^{31.6}$	$2^{47.6}$	$2^{25.7}$	$2^{40.2}$	$2^{18.0}$	$2^{32.0}$
64	4	$2^{23.8}$	$2^{40.3}$	$2^{17.7}$	$2^{32.6}$	$2^{14.0}$	$2^{28.0}$

Table 4. Complexity comparison of Matsui's, Chen-Miyaji's and Our collision search algorithms

Key size (bytes)	Matsui's algorithm (seconds)	Chen-Miyaji's algorithm (seconds)	Our algorithm (seconds)
64	39.07751	0.20979	0.01706
41	892.73275	2.16465	0.11378
36	2184.34910	197.81812	1.65500
30	8025.36729	5110.37252	29.48690
28	28551.76842	18970.05472	1563.82900
26	24141.91946	17518.71410	1235.80500
24	109520.08072	75079.41634	49519.62140

Table 5. Average processor time taken to get a single collision for different key sizes in Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2 core, by running a single instance 2^{20} times for the keys with $64, 41, 36$, 2^{10} times for the key 30, 2^5 times for the keys $26, 28$ and 5 times for the key 24

The full table of the time complexities for $k \in [20, 64]$ is given in Appendix A (Table 6). It shows that our algorithm is indeed faster than both Matsui's and Chen-Miyaji's algorithms and the improvement becomes more prominent with decreasing key size. Note that the complexity estimates are computed assuming independence of multiple events that are not independent in practice. Hence, these estimates should be considered as lower bounds and the actual complexities are much higher. The comparison between the complexity estimates of the three algorithms is still valid, because the independence is assumed for the same class of events.

Note that, in Table 4, the search complexities for the key sizes with same number of round intervals are in decreasing order. This is because their last round intervals are different, that means the keys with the same round intervals have the property that with decreasing key length, the corresponding class 1 condition indices ($= b_n - 1 = d + (n-1)k - 1$) for the last round interval also

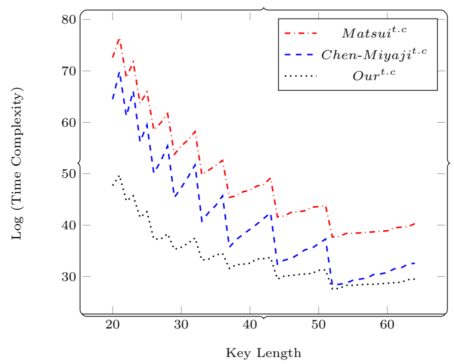

Fig. 3. Theoretical time complexity for $k \in [20, 64]$.

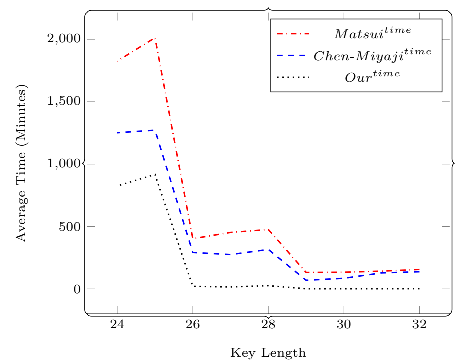

Fig. 4. Experimental average time for $k \in [20, 32]$.

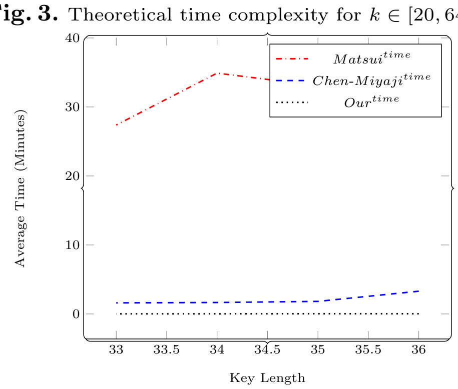

Fig. 5. Experimental average time for $k \in [33, 36]$.

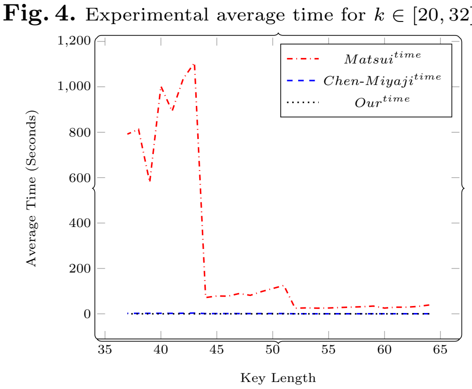

Fig. 6. Experimental average time for $k \in [37, 64]$.

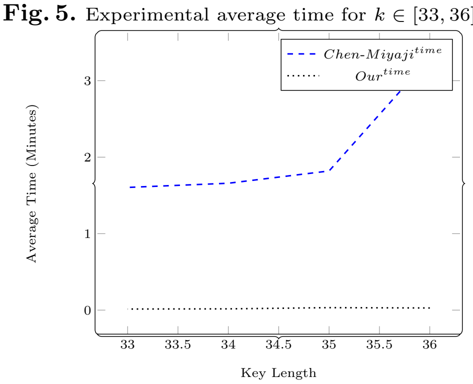

Fig. 7. Experimental average time (magnified plot, excluding Matsui's) for $k \in [33, 36]$.

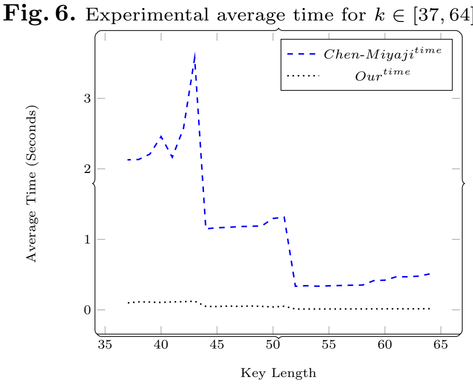

Fig. 8. Experimental average time (magnified plot, excluding Matsui's) for $k \in [37, 64]$.

decrease and consequently the search complexities decrease. For example, with $n = 9, d = k - 3$,

$$d + (n-1)k - 1 = \begin{cases} 251, \text{ if } k = 28; \\ 242, \text{ if } k = 27; \\ 233, \text{ if } k = 26. \end{cases}$$

We show this opposite characteristics of the keys within same number of round intervals in Fig. 3, which is the graphical representation of the theoretical time complexity (vertical axis) with the key length (horizontal axis).

We give an experimental comparison of Matsui, Chen-Miyaji and our algorithm in terms of the average time[1] to get a colliding pair of different sizes in Table 5. The full experimental time table for $k \in [24, 64]$ is given in Appendix B (Table 7).

Fig. 4 to Fig. 8 are the graphical representations of the experimental time (vertical axis) with the key length (horizontal axis). Fig. 3 to Fig. 8 show that our algorithm has a better efficiency searching for short colliding keys compare to Matsui and Chen-Miyaji's proposed search algorithms. $Matsui^{time}$, $Chen\text{-}Miyaji^{time}$, Our^{time} denote the average times taken by Matsui, Chen-Miyaji and our algorithm respectively.

Note that in Fig. 5 and 6, the difference between Chen-Miyaji's and our performance timings are not visible due to the scale of the vertical axis. To show the difference, we remove Matsui's curve from these figures and magnify the scale in Fig. 7 and Fig. 8, which clearly demonstrate that our algorithm performs better than Chen-Miyaji's as well. The source code of these algorithms are available in [19].

6 Conclusion

In this paper, we show some limitations of both Matsui's and Chen-Miyaji's search algorithms for finding colliding key-pairs of RC4 stream cipher and introduce a new second round interval passing technique. We also take care of all these limitations and propose a new algorithm that has less search complexity and discovers a colliding key-pair faster compared to Matsui's and Chen-Miyaji's algorithms. We have reported three new 22-byte colliding key pairs and 12 new 20-byte near-colliding key pairs. Whether there exists a colliding key pair of 21-byte size or less remains an interesting open question.

References

1. N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, J. C. N. Schuldt. On the Security of RC4 in TLS. In USENIX Security Symposium 2013, pp. 305–320.
2. Anonymous. RC4 Source Code. Cypherpunks mailing list(September 9, 1994), `http://cypherpunks.venona.com/date/1994/09/msg00304.html,http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0`.
3. E. Biham and O. Dunkelman. Differential Cryptanalysis in Stream Ciphers. In IACR Cryptology ePrint Archive 2007: 218.
4. J. Chen and A. Miyaji. How to Find Short RC4 Colliding Key Pairs. In ISC 2011, vol. 7001 of Lecture Notes in Computer Science, Springer, pp. 32–46.
5. J. Chen and A. Miyaji. Cryptanalysis of Stream Ciphers from a New Aspect: How to Apply Key Collisions to Key Recovery Attack. In IEICE Trans., Fundamentals, 2012, vol. 95-A(12), pp. 2148–2159.

[1] Processor time is calculated by the difference of the start time and end time of getting a colliding key pair divided by CLOCKS-PER-SEC, by using the clock() function in C language.

6. J. Chen and A. Miyaji. Novel Strategies for Searching RC4 Key Collisions. In Computers and Mathematics with Applications, vol. 66(1), 2013, pp. 81–90, http://dx.doi.org/10.1016/j.camwa.2012.09.013.
7. C. Garman, K. G. Paterson and T. Van der Merwe. Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS. In USENIX Security Symposium 2015, pp. 113–128.
8. T. Isobe, T. Ohigashi, Y. Watanabe and M. Morii. Full Plaintext Recovery Attack on Broadcast RC4. In FSE 2013, vol. 8424 of Lecture Notes in Computer Science, Springer, pp. 179–202.
9. S. Maitra, G. Paul, S. Sen Gupta. Attack on Broadcast RC4 Revisited. In: Joux, A. (ed.) FSE 2011, vol. 6733 of Lecture Notes in Computer Science, Springer, pp. 199–217.
10. S. Maitra, G. Paul, S. Sarkar, M. Lehmann, W. Meier. New Results on Generalization of Roos-Type Biases and Related Keystreams of RC4. In AFRICACRYPT 2013, vol. 7918 of Lecture Notes in Computer Science, Springer, pp. 222–239.
11. I. Mantin and A. Shamir A practical attack on broadcast RC4. In FSE 2001, M. Matsui, Ed., vol. 2355 of Lecture Notes in Computer Science, Springer, pp. 152–164.
12. I. Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. In EUROCRYPT 2005, vol. 3494 of the series Lecture Notes in Computer Science, Springer, pp. 491–506.
13. M. Matsui. Key Collisions of the RC4 Stream Cipher. In FSE 2009, vol. 5665 of the series Lecture Notes in Computer Science, Springer, pp. 38-50.
14. A. Maximov and D. Khovratovich. New State Recovery Attack on RC4. In CRYPTO 2008, vol. 5157 of Lecture Notes in Computer Science, Springer, pp. 297–316.
15. K. G. Paterson, B. Poettering, J. C. N. Schuldt. Plaintext Recovery Attacks Against WPA/TKIP. In FSE 2014, vol. 8540 of Lecture Notes in Computer Science, Springer, pp. 325–349.
16. R. L. Rivest and J. C. Schuldt. Spritz–A spongy RC4-like stream cipher and hash function. In CRYPTO 2014 Rump Session, 2014.
17. P. Sepehrdad, P. Susil, S. Vaudenay, and M. Vuagnoux. Tornado attack on RC4 with applications to WEP and WPA. In IACR Cryptology ePrint Archive, 2015: 254.
18. M. Vanhoef and F. Piessens. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In USENIX Security Symposium 2015, pp. 97–112.
19. Unoptimized implementations of RC4 collision search algorithms, `https://github.com/janaamit001/Our_algo`,`https://github.com/janaamit001/Chen_Miyaji`,`https://github.com/janaamit001/Matsui_algo`.

A Table for Theoretical Time Complexity

Key size (bytes)	Number of round intervals	Matsui's algorithm		Chen-Miyaji's algorithm		Our algorithm	
		Search complexity	Time complexity	Search complexity	Time complexity	Search complexity	Time complexity
20	12	$2^{56.7}$	$2^{72.6}$	$2^{51.6}$	$2^{64.5}$	$2^{33.7}$	$2^{47.7}$
21	12	$2^{60.6}$	$2^{76.5}$	$2^{56.9}$	$2^{69.9}$	$2^{35.7}$	$2^{49.7}$
22	11	$2^{52.7}$	$2^{68.7}$	$2^{47.7}$	$2^{61.2}$	$2^{30.7}$	$2^{44.7}$
23	11	$2^{55.7}$	$2^{71.7}$	$2^{52.6}$	$2^{66.1}$	$2^{32.5}$	$2^{46.5}$
24	10	$2^{47.8}$	$2^{63.7}$	$2^{42.6}$	$2^{56.2}$	$2^{27.5}$	$2^{41.5}$
25	10	$2^{49.9}$	$2^{65.9}$	$2^{45.9}$	$2^{59.5}$	$2^{29.6}$	$2^{42.6}$
26	9	$2^{42.6}$	$2^{58.5}$	$2^{36.5}$	$2^{50.1}$	$2^{23.6}$	$2^{37.6}$
27	9	$2^{43.9}$	$2^{59.9}$	$2^{38.8}$	$2^{52.4}$	$2^{24.6}$	$2^{38.6}$
28	9	$2^{45.8}$	$2^{61.7}$	$2^{41.7}$	$2^{55.4}$	$2^{25.6}$	$2^{39.6}$
29	8	$2^{37.9}$	$2^{53.8}$	$2^{31.6}$	$2^{45.4}$	$2^{21.5}$	$2^{35.5}$
30	8	$2^{39.5}$	$2^{55.4}$	$2^{33.6}$	$2^{47.3}$	$2^{21.9}$	$2^{35.9}$
31	8	$2^{40.6}$	$2^{56.6}$	$2^{35.6}$	$2^{49.4}$	$2^{22.7}$	$2^{36.7}$
32	8	$2^{41.7}$	$2^{58.2}$	$2^{37.8}$	$2^{51.7}$	$2^{23.6}$	$2^{37.6}$
33	7	$2^{33.9}$	$2^{49.8}$	$2^{27.0}$	$2^{40.8}$	$2^{19.5}$	$2^{33.5}$
34	7	$2^{34.8}$	$2^{50.7}$	$2^{28.6}$	$2^{42.5}$	$2^{19.7}$	$2^{33.7}$
35	7	$2^{35.7}$	$2^{51.6}$	$2^{29.9}$	$2^{43.8}$	$2^{20.5}$	$2^{34.5}$
36	7	$2^{36.6}$	$2^{52.6}$	$2^{31.7}$	$2^{45.7}$	$2^{20.7}$	$2^{34.7}$
37	6	$2^{29.6}$	$2^{45.4}$	$2^{21.8}$	$2^{35.7}$	$2^{16.8}$	$2^{30.8}$
38	6	$2^{29.8}$	$2^{45.7}$	$2^{22.7}$	$2^{37.2}$	$2^{17.0}$	$2^{31.0}$
39	6	$2^{30.6}$	$2^{46.5}$	$2^{23.7}$	$2^{38.2}$	$2^{17.6}$	$2^{31.6}$
40	6	$2^{30.8}$	$2^{46.8}$	$2^{24.7}$	$2^{39.2}$	$2^{17.8}$	$2^{31.8}$
41	6	$2^{31.6}$	$2^{47.6}$	$2^{25.7}$	$2^{40.2}$	$2^{18.0}$	$2^{32.0}$
42	6	$2^{31.9}$	$2^{47.9}$	$2^{26.7}$	$2^{41.3}$	$2^{18.6}$	$2^{32.6}$
43	6	$2^{32.7}$	$2^{49.2}$	$2^{27.9}$	$2^{42.4}$	$2^{18.8}$	$2^{32.8}$
44	5	$2^{25.7}$	$2^{41.6}$	$2^{17.8}$	$2^{32.4}$	$2^{14.8}$	$2^{28.8}$
45	5	$2^{25.9}$	$2^{41.8}$	$2^{18.6}$	$2^{33.2}$	$2^{15.0}$	$2^{29.0}$
46	5	$2^{26.6}$	$2^{42.5}$	$2^{18.9}$	$2^{33.5}$	$2^{15.5}$	$2^{29.5}$
47	5	$2^{26.7}$	$2^{42.6}$	$2^{19.7}$	$2^{34.3}$	$2^{15.6}$	$2^{29.6}$
48	5	$2^{26.8}$	$2^{42.8}$	$2^{20.5}$	$2^{35.2}$	$2^{15.7}$	$2^{29.7}$
49	5	$2^{27.5}$	$2^{43.5}$	$2^{20.8}$	$2^{35.5}$	$2^{15.8}$	$2^{29.8}$
50	5	$2^{27.7}$	$2^{43.6}$	$2^{21.7}$	$2^{36.4}$	$2^{16.0}$	$2^{30.0}$
51	5	$2^{27.8}$	$2^{43.8}$	$2^{22.6}$	$2^{37.3}$	$2^{16.6}$	$2^{30.6}$
52	4	$2^{21.9}$	$2^{37.7}$	$2^{13.6}$	$2^{28.3}$	$2^{12.8}$	$2^{26.8}$
53	4	$2^{22.0}$	$2^{37.8}$	$2^{13.7}$	$2^{28.5}$	$2^{12.8}$	$2^{26.8}$
54	4	$2^{22.5}$	$2^{38.4}$	$2^{13.9}$	$2^{28.7}$	$2^{12.9}$	$2^{26.9}$
55	4	$2^{22.6}$	$2^{38.4}$	$2^{14.6}$	$2^{29.3}$	$2^{13.0}$	$2^{27.0}$
56	4	$2^{22.6}$	$2^{38.5}$	$2^{14.7}$	$2^{29.5}$	$2^{13.5}$	$2^{27.5}$
57	4	$2^{22.7}$	$2^{38.6}$	$2^{14.9}$	$2^{29.7}$	$2^{13.6}$	$2^{27.6}$
58	4	$2^{22.8}$	$2^{38.7}$	$2^{15.6}$	$2^{30.4}$	$2^{13.6}$	$2^{27.6}$
59	4	$2^{22.9}$	$2^{38.8}$	$2^{15.7}$	$2^{30.5}$	$2^{13.7}$	$2^{27.7}$
60	4	$2^{23.0}$	$2^{38.9}$	$2^{15.9}$	$2^{30.8}$	$2^{13.7}$	$2^{27.7}$
61	4	$2^{23.5}$	$2^{39.5}$	$2^{16.6}$	$2^{31.5}$	$2^{13.8}$	$2^{27.8}$
62	4	$2^{23.6}$	$2^{39.6}$	$2^{16.8}$	$2^{31.7}$	$2^{13.8}$	$2^{27.8}$
63	4	$2^{23.7}$	$2^{39.7}$	$2^{17.5}$	$2^{32.4}$	$2^{13.9}$	$2^{27.9}$
64	4	$2^{23.8}$	$2^{40.3}$	$2^{17.7}$	$2^{32.6}$	$2^{14.0}$	$2^{28.0}$

Table 6. Complexity comparison of Matsui's, Chen-Miyaji's and Our collision search algorithms

B Table for Experimental Average Time

Key size (bytes)	Matsui's algorithm (seconds)	Chen-Miyaji's algorithm (seconds)	Our algorithm (seconds)
64	39.07751	0.20979	0.01706
63	32.69903	0.17822	0.01510
62	29.40904	0.16928	0.01525
61	29.35850	0.16883	0.01512
60	25.33800	0.12122	0.01471
59	34.26300	0.11481	0.01385
58	31.27700	0.19179	0.01368
57	29.65300	0.14958	0.01348
56	27.88500	0.14565	0.01293
55	25.72200	0.14136	0.01271
54	25.05300	0.13583	0.01263
53	26.17500	0.14257	0.01227
52	24.86400	0.13489	0.01194
51	125.00286	0.31536	0.05149
50	111.63613	0.30634	0.04128
49	98.04527	0.19151	0.04951
48	81.63158	0.18493	0.05546
47	89.43752	0.18153	0.04902
46	77.98217	0.17043	0.05442
45	75.64383	0.16466	0.04744
44	71.76262	0.14948	0.04952
43	1105.51430	3.57249	0.12318
42	1035.87532	2.57249	0.11635
41	892.73275	2.16465	0.11378
40	1002.64386	2.45864	0.10520
39	579.56563	2.20962	0.11068
38	812.75401	2.13297	0.11332
37	791.61340	2.12798	0.09890
36	2184.34910	197.81812	1.65500
35	1984.84735	109.16570	1.92418
34	2094.10821	99.52875	0.97058
33	1643.07930	96.25031	0.82837
32	9356.34319	8259.52852	68.34430
31	8550.38677	7707.62430	38.60300
30	8025.36729	5110.37252	29.48690
29	7922.56746	4146.97494	26.25940
28	28551.76842	18970.05472	1563.82900
27	27155.22835	16511.23852	1141.33700
26	24141.91946	17518.71410	1235.80500
25	120770.73947	76316.31213	55014.73840
24	109520.08072	75079.41634	49519.62140

Table 7. Average processor time taken to get a single collision for different key sizes in Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2 core, by running a single instance 2^{20} times for the keys with $4, 5, 6, 7$ round intervals, 2^{10} times for the keys with 8 round intervals, 2^{5} times for the keys with 9 round intervals and 5 times for the keys with 10 round intervals