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

Amit Jana and Goutam Paul

Indian Statistical Institute, Kolkata 700 108, India.
janaamit001@gmail.com, goutam.paul@isical.ac.in

Abstract. If two different secret keys of a stream cipher yield the same internal state after the key scheduling algorithm (KSA) and hence generates 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 length (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] claimed to design 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 [ISC 2011]. We additionally give 12 new 20-byte near-colliding key pairs different from Matsui's [FSE 2009]. These results are significant, considering the argument by the experts [Biham and Dunkelman, 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.

1 Introduction

RC4 has been one of the most widely-used real-world stream cipher. It has been used in various practical software applications and network protocols such as Secure Socket Layer (SSL), 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 II) 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]. Apart from being a replacement for RC4 encryption, Spritz can also be used for other cryptographic tasks based on *sponge-like* constructive functions. Moreover, theoretically, RC4 serves as a good model for 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**
  $j = (j + S[i] + K[i \mod k]) \mod N$;
  swap$(S[i], S[j])$;

Algorithm 2: PRGA

Input: Internal state S, generated by KSA
Output: keystream X
$i = 0$;
$j = 0$;
for *each new message byte* **do**
  $i = (i + 1) \mod N$;
  $j = (j + S[i]) \mod N$;
  swap$(S[i], S[j])$;
  $X = S[(S[i] + S[j]) \mod N]$;
  output X;

1.1 RC4 key collision and why it is hard to find

In this paper, we revisit the methods to find 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 ($\geq$ 106) bytes, with high probability there exists a colliding key pair. But to find 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 to find 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. Till now, this is the shortest known colliding key pair of RC4.

It is interesting to note that Biham and Dunkelman, in their differential cryptanalysis in stream ciphers [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 different from the one reported in [4] (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 of two keys K_1, K_2 in which they differ.
- $n = \lceil \frac{256+k-1-d}{k} \rceil$ denotes the total number of rounds.
- S_1, S_2 denote the two states produced by KSA starting with the keys K_1, K_2 separately.
- $S_{1,x}[y], S_{2,x}[y]$ denote the value of states $S_1[y], S_2[y]$ at $i = x$ computation in KSA corresponding to K_1, K_2 respectively. In particular, when $x \leq y-1$, we use the notations $S_1[y]$, $S_2[y]$ to mean $S_{1,x}[y]$, $S_{2,x}[y]$ respectively and $S_1 = S_2$ means $\forall i \in [0, 255]$, $S_1[i] = S_2[i]$.
- $J_{1,x}, J_{2,x}$ denote the j-values after running KSA at $i = x$ corresponding to K_1, K_2 respectively.
- $K\langle x, y\rangle$ denotes the key modification on $K[x]$ by y and is defined as $K_1[x] = K_1[x] + y$, $K_2[x] = K_2[x] + y$, $K_1[x+1] = K_1[x+1] - y$, $K_2[x+1] = K_2[x+1] - y$, where $0 \leq x < k$, $0 \leq y < 256$.
- $K^{'}\langle x, y\rangle$ is defined as: $K_1[x] = K_1[x] + y$, $K_2[x] = K_2[x] + y$.
- we sometime use a key modification on $\langle x, y\rangle$ means $K\langle x, y\rangle$.
- Other primed variables denote their updated values.
- The closed interval notation $[a, b]$ means the set of integers x such that $a \leq x \leq b$. The open interval notation (a, b) means the set of integers x such that $a < x < b$. Half-open or half-closed intervals are defined accordingly.
- Sometimes in short we write j-values at index $i \in [0, 255]$ such that $J_{1,i} = J_{2,i}$ corresponding to K_1, K_2 as J_i, instead of writing $J_{1,i}, J_{2,i}$ separately. We also use, state $S_{i-1}[i]$ in which $S_{1,i-1}[i] = S_{2,i-1}[i]$ corresponding to K_1, K_2 instead of $S_{1,i-1}[i], S_{2,i-1}[i]$ separately and key $K[i]$ in which $K_1[i] = K_2[i]$.
- $J_{[a,b]}$ means all j-values in which $J_{1,m} = J_{2,m}, \forall m \in [a, b]$, used in Table 1 and in the round r, $J_x^r, J_x^{'r}$ denotes the j-values before and after the key modifications at the index x respectively.
- We use the t-th round as $[a_t, b_t]$, where $a_t = d + (t-2)k + 1$, $b_t = d + (t-1)k$.
- For any round $r \geq 2$, 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 $c_{1,x} = x$.
- All operations are done modulo 256 except operations with the key indices for which modulo k is used.
- $MaxS$ is the maximal index i such that the distance between S_1 and S_2 is at most two.
- MaxColStep(K_1, K_2) function uses the KSA procedure for both the keys and return that state index i in which state differ first.

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 using. 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 of 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 position, $K_1[2] \neq K_2[2]$. 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 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 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 $i = 2$ (i.e., after the swap operation), $J_1 = 2, J_2 = 3$, causing the states to differ at exactly two positions 2 and 3 (i.e., $S_1[2] = 2, S_1[3] = 3, S_2[2] = 3, S_2[3] = 2$). At the next step $i = 3$, previous difference (i.e., $J_1 = J_2 - 1$) must be absorbed due to state difference $S_1[3] = S_2[3]+1$. For $i \in [4, 65]$, $J_{1,i} = J_{2,i} \neq 66$ but at the state index $i = 66$, where $S_1[66] = 3, S_2[66] = 2$, we swap the values between 66 and the next state index (130) in which the keys differ. Similarly we need, $\forall i \in [67, 129]$, $J_1 = J_2 \neq 130$. Proceeding in this way until we go to the second last state index (130) in which the keys differ, i.e., at $i = 130$, we swap the values between i and $194 - 1$ (i.e., a state index just before the last state index in which the key differs). Then $\forall i \in [131, 191]$, $J_1 = J_2 \neq 193$ and at $i = 192$, we have $J_1 = J_2 = r$ $(= 129)$ such that $S_1[r] = 2, S_2[r] = 3$. At $i = 193$, $J_{1,i} = i, J_{2,i} = i - 1$, 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. The equality conditions on j-values are called Class 1 conditions and the inequality patterns are called Class 2 conditions. Throughout this paper, we make frequent references to this table.

2.1 Matsui's work [13]

In [13], it is clearly described how two keys K_1 and K_2 with 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 stop, otherwise it tries for another key pair. If value of the MaxS variable (the MaxS notation is defined in Section 1.2) 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. 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

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

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

Round	Round Interval	Class 1 conditions	Class 2 conditions
1	$[0, d+1]$	$J_{1,d} = d, J_{2,d} = d+1$ $J_{d+1} = d+k$	$J_{[0,d-1]} \neq d, d+1$
2	$[d+2, d+k]$	$J_{d+k} = d+2k$	$J_{[d+2,d+k-1]} \neq d+k$
3	$[d+k+1, d+2k]$	$J_{d+2k} = d+3k$	$J_{[d+k+1,d+2k-1]} \neq d+2k$
...	...	...	...
t	$[a_t, b_t]$	$J_{b_t} = b_t + k$	$J_{[a_t,b_t-1]} \neq b_t$
...	...	...	....
$n-1$	$[a_{n-1}, b_{n-1}]$	$J_{b_{n-1}} = b_{n-1} + k - 1$	$J_{[a_{n-1}+1,b_{n-1}-1]} \neq b_{n-1}$
n	$[a_n, b_n]$	$S_{1,b_n-3}[x] = d, S_{1,b_n-3}[x] = d+1,$ $J_{b_n-2} = x, J_{1,b_n-1} = b_n - 1, J_{2,b_n-1} = b_n - 2$	$J_{[a_n+1,b_n-3]} \neq b_n - 1$
	$[b_n + 1, 255]$		$J_{[b_n+1,255]} \in [0, 255]$

Table 2. Transition pattern involving j-conditions, where $a_t = d + (t-2)k + 1, b_t = d + (t-1)k, t \geq 2$

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 = 00 42 CE D3 DF DD B6 9D 41 3D BD 3A B1 16 5A 33 ED A2 CD 1F E2 8C 01 76,

K_2 = 00 42 CE D3 DF DD B6 9D 41 3D BD 3A B1 16 5A 33 ED A2 CD 1F E2 8C 01 77.

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

K_1 = 00 73 2F 6A 01 37 89 C5 15 49 9A 55 98 54 D7 53 4E F6 4F DC,

K_2 = 00 73 2F 6A 01 37 89 C5 15 49 9A 55 98 54 D7 53 4E F6 4F DD.

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 deterministically. To bypass the first round, i.e., to satisfy 1st round class 1 condition, the following three steps are needed.
(a) 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.
(b) Modify the keys as $K_1[d] = 256 - J_{1,d-1}$, $K_2[d] = K_1[d]+1 \implies J_{1,d} = J_{1,d-1}+K_1[d]+S_{1,d-1}[d]$ $= J_{1,d-1} + 256 - J_{1,d-1} + d = d$, & $J_{2,d} = J_{2,d-1} + K_2[d] + S_{2,d-1}[d] = J_{2,d-1} + 256 - J_{2,d-1} + 1 + d$ $= d + 1$.
(c) Modify $K_1[d+1] = K_2[d+1] = k - d - 1 \implies J_{1,d+1} = J_{1,d} + K_1[d+1] + S_{1,d}[d+1]$ $= d + (k-d-1) + (d+1) = d+k$, & $J_{2,d+1} = J_{2,d} + K_2[d+1] + S_{2,d}[d+1] = (d+1) + (k-d-1) + d$ $= d + k$.

2.2.2 Bypassing the second Round with high probability. By choosing $d = k - 3$ and assuming all second round class 2 conditions are satisfied, we need to satisfy second round class 1 condition, i.e., $J_{d+k} = d + 2k$. Now we can write,

$$J_{d+k} = ((...(..((J_{d+1} + K[d+2] + S_{d+1}[d+2]) + K_{d+3} + S_{d+2}[d+3]) + ..)...) + K[d+k] + S_{d+k-1}[d+k])$$

$$= J_{d+1} + K[d+2] + \sum_{i=0}^{i=d} K[i] + \sum_{i=d+2}^{i=d+k} S_{i-1}[i]$$

$$=^{\text{with prob } p_2} J_{d+1} + K[d+2] + \sum_{i=0}^{i=d} K[i] + \sum_{i=d+2}^{i=d+k} S_{d+1}[i].$$

If $\forall l \in [d+2, d+k]$, $J_l \notin [l+1, d+k]$ holds, then the 2nd round will be passing probabilistically by setting the value of $K[d+2]$ using the above equation i.e., it is always possible to get a unique value δ $(= d + 2k - J_{d+k} (\neq d + 2k))$ and then modify the key as $K^{'}\langle d+2, \delta\rangle$, will be satisfying the 2nd round class 1 condition. Thus the probability of passing the second round 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 passing technique to reduce complexity. For the last round class 1 conditions, we have to satisfy: $J_{b_n-2} = x$ so that $S_{1,b_n-3}[x] = d, S_{2,b_n-3}[x] = d+1$ and $J_{1,b_n-1} = b_n - 1, J_{2,b_n-1} = b_n - 2$. As we know,

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

.

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

. 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 index x is almost uniform. So if $\forall i \in [d+2, b_n-3]$, $J_i \neq d$ holds and assuming $J_{b_n-2} = d$, then we can modify $K[d-1]$ (at $i = b_n - 1$ in KSA) before the swap, as:

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

This modification indicates that if some trial meets $J_{b_n-2} = d$ with $J_i \neq d$, $\forall i \in [d+2, b_n-3]$, 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 multi-key modification technique to pass the the class 1 condition for all rounds t, $3 \leq t < n$. 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 $t-1$ rounds but fails to pass the t-th round $[a_t, b_t]$, i.e., $J_{b_t} \neq b_{t+1}$. One can write

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

.

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

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

For each i, if $\Delta_i \leq d + (t-2)k$, then it will pass the t-th round by modifying the key as

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

where J_{Δ_i} is the j-value at the Δ_i-th index before the key modification.

Using the bypassing techniques and the multi-key modification, Chen-Miyaji proposed an algorithm in [4, Table 3] that produced the following 22-byte colliding key.
$K_1(K_2)$: A2 27 43 A7 03 94 2F 17 75 BB A7 27 8F DD 3E 7B C6 A1 C7 81(82) 02 5A.

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 class 2 j-condition on some index x in the t-th round. At this point, the previous algorithms try to adjust the key value at the 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, class 1 j-condition is 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 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 $K\langle x_1, y_1\rangle$ such that $K\langle x_1, y_1\rangle \equiv K\langle x_1 \mod k, y_1\rangle$ 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 $z \equiv x_1 \mod k$ and take rounds $l(< n)$ such that $z + (l-1)k < MaxS$. Then the new key modification $K\langle x_1 \mod k, y\rangle$, $y > y_1$ will work only if for each such l, any one of the following cases hold:

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

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

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

2 After adding δ to $K[d+2]$, we need to satisfy that $\forall l \in [d+2, d+k-1], J'_l (= J_l + \delta) \notin [l+1, d+k]$.

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

1. If $S_{\Delta_i-1}[\Delta_i] \neq \Delta_i$ for each $i \in [a_t, b_t]$, then one can pass the t-th round if $\exists$ an index $q \in [a_t, b_t-1]$ such that $J_q^{'} = J_q + \Delta$, where $J_{b_t}^{'} = J_{b_t} + \Delta = d + tk$, with the required condition that $J_r^{'} \notin [r+1, b_t], \forall r \in [J_q^{'}+1, b_t]$.
2. If $S_{\Delta_i-1}[\Delta_i] = \Delta_i$ 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. So one needs to check that all the $J_{\Delta_i-mk}^{'}$ values, for each $m = 1, 2, 3, \ldots$ such that $\Delta_i - mk > 0$, corresponding to the key modification on Δ_i index. To ensure that the previous rounds will not affect due to these key modifications, we have to take care of the following conditions.
   (a) If $J_{\Delta_i-mk} \in (\Delta_i - mk, b_t]$, then one has to find that $\exists$ an index $r \in (\Delta_i - mk, J_{\Delta_i-mk}]$ such that $J_r = J_{\Delta_i-mk}$ and so $S_r[J_{\Delta_i-mk}] = S_{J_{\Delta_i-mk}-1}[J_{\Delta_i-mk}]$ and also one needs to check that $J_{\Delta_i-mk}^{'} \notin (\Delta_i - mk, b_t]$; or if $J_{\Delta_i-mk}^{'} \in (\Delta_i - mk, b_t]$, then to find that $\exists$ an index $r^{'} \in (\Delta_i - mk, J_{\Delta_i-mk}^{'}]$ so that $J_{r^{'}} = J_{\Delta_i-mk}^{'}$, shown in Fig 1.
   (b) If $J_{\Delta_i-mk} \notin (\Delta_i - mk, b_t]$ with $J_{\Delta_i-mk}^{'} \notin (\Delta_i - mk, b_t]$, then it creates no problem on that round. But if $J_{\Delta_i-mk}^{'} \in (\Delta_i - mk, b_t]$ then one needs to check that $\exists$ an index $r^{'} \in (\Delta_i - mk, b_t]$ so that $J_{r^{'}} = J_{\Delta_i-mk}^{'}$, shown in Fig 2.

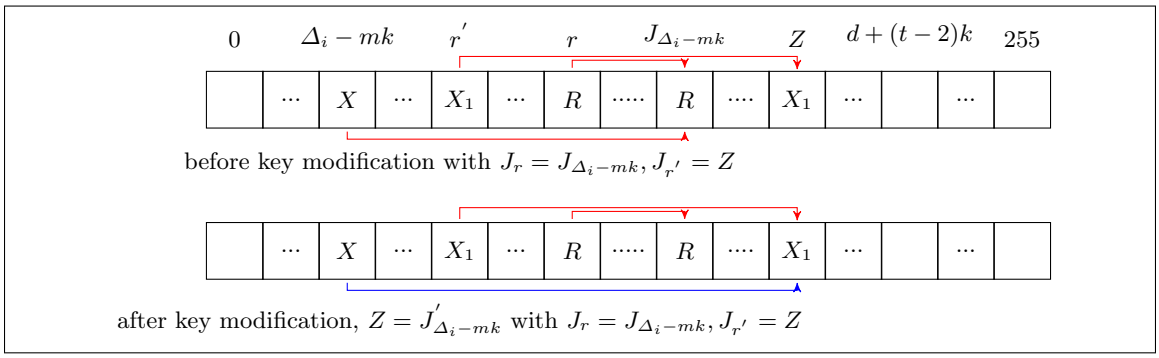

Fig. 1.

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 $S_{1,m}[b_t] = (d+1), S_{2,m}[b_t] = d, \forall m \in [a_t, b_t - 1]$ according to Matsui's transition pattern. It is obvious from our discussion that if the round 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 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 and get the same 22-byte colliding key pair in

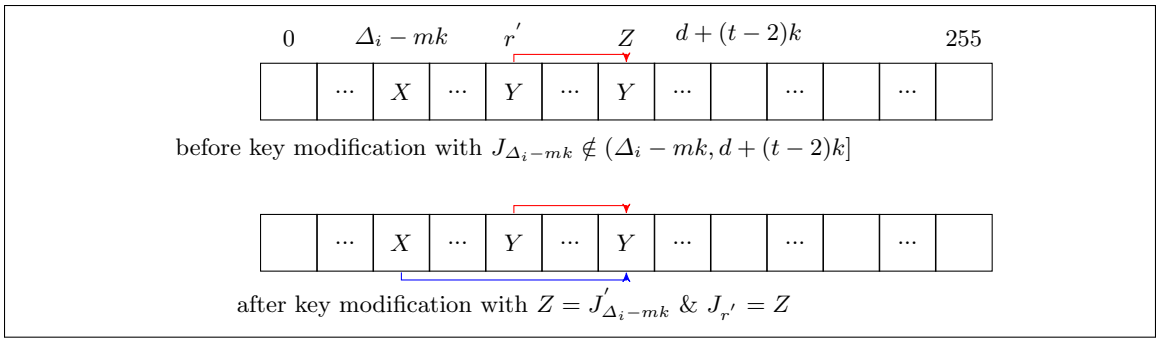

Fig. 2.

five hours. For any round t (except the first round), there are $(k-1)$-positions in which we can change state values (differ from its previous value) such that it would help to pass this round. In [4, Table 3], Chen-Miyaji use all key modification 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 passing techniques, but we don't use their second round 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 deterministically and the class 2 conditions of the second round.
- Modify the key as $K'\langle k-1, y\rangle$, $\forall y \in [1, 255]$.

Key size (bytes)	Matsui's algorithm		Chen-Miyaji's algorithm		Last key index increasing technique		Our algorithm	
	Exp number	Exp prob	Exp number	Exp prob	Exp number	Exp prob	Exp number	Exp prob
30	461635584	0.43	175951052	0.16	431069593	0.40	695992320	0.65
28	431646310	0.40	264922726	0.25	468765900	0.44	697460326	0.64
25	338165760	0.31	258578841	0.24	446745804	0.42	635804057	0.59
24	324586700	0.30	371877478	0.35	504784486	0.47	669410918	0.62
23	308333772	0.29	429024870	0.40	437728051	0.41	688862003	0.64
22	292814848	0.27	460587008	0.42	564920320	0.53	697722470	0.65
20	246153216	0.23	534773760	0.49	632291328	0.59	724199014	0.67

Table 3. Second round passing comparison with 2^{30} samples ("Exp number" refers to the total number of keys that pass second round out of 2^{30} keys and "Exp prob" is the "Exp number" divided by 2^{30})

In our algorithm, we use this technique along with the key modification $K\langle x, y\rangle$, $0 \leq x < k, 1 \leq y \leq 255$ except at $x \neq d-2, d-1, d, d+1$, 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 as compared to the previous existing techniques.

Algorithm 3: New Collision Search Algorithm

Input: key K_1 with key length k and fix d ($< k$)
Output: key K_1, which passes the first round
while *True* **do**
  Generate one 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 $Temp < d$ **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;
  Modify $K_1[d]$ as:$K_1[d] = 256 - J_{1,d-1}$;
  for $i = 0$ *to* $k-1$ **do**
    $A[i] = 1$;
  if ***NewSearch***$(K_1) =$ *True* **then**
    stop(found a collision) or continue(to find more);
  else
    return;
return;

Algorithm 4: **NewMaxColStep**(K_1)

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

We present our search strategy in Algorithm 3, Algorithm 4 and Algorithm 5. 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 3 generates a random key and passes the first round deterministically, then pass it into Algorithm 5, which in turn invokes Algorithm 4 to check all class 1 and class 2 conditions in Table 2 and returns the corresponding failing state index, whenever j-values break either of class 1 or class 2 conditions according to Table 2.

Algorithm 5: **NewSearch**(K_1)

```
Input: key K_1
Output: colliding key pair
P = NewMaxColStep(K_1);
if P = 255 then
    return True;
Set MaxS = P;
Flag = False;
for x = 0 to k − 1 do
    copy K_1 into K_3;
    if A[x] = 1 then
        if x = d − 2 or x = d − 1 or x = d or x = d + 1 then
            break;
        else if x < k − 1 then
            for y = A[x] to 255 do
                Modify K_3⟨x, 1⟩ and Temp = NewMaxColStep(K_3);
                if Temp = 255 then
                    return True;
                if Temp > MaxS then
                    u = x, v = y − A[x] + 1, MaxS = Temp and Flag = True;
        else   /* only for first time in NewSearch(K_1) call, then skip this else
        part */
            for y = A[x] to 255 do
                Modify K'_3⟨x, 1⟩;
                Temp = NewMaxColStep(K_3);
                if Temp = 255 then
                    return True;
                if Temp > MaxS then
                    u = x, v = y − A[x] + 1, MaxS = Temp and Flag = True;
        if Flag = True then
            if x = k − 1 then
                Modify K'_1⟨u, v⟩ and A[x] = v;
            else
                Modify K_1⟨u, v⟩ and A[x] = v;
            Flag = False;
if MaxS ≤ P then
    return False
temp1 = NewSearch(K_1);
if temp1 = True then
    return True;
else
    return False;
```

To get near-colliding pair, we will do a slight modification in Algorithm 4 (**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 $J_{b_{n-1}} = b_n$.
– For $i \in [b_{n-1}+1, b_n-2]$, the j-conditions remain the same as in Table 1 and $J_i \neq i+1$, $i = b_n-1$.
– At $i = b_n$, we need to satisfy the condition $J_{b_n} \leq b_n$.

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 index (state) is greater than the current $MaxS$ value. 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 5). 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 index $temp$ (i.e., $temp > MaxS$) by modifying key on index x, $\forall y \in [1, 255]$, further key modifications on that key index might not help to pass the next rounds. So to overcome this situation, we introduce a new search algorithm in Algorithm 5 (named as **NewSearch**(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.

4.1 Average Number of Key Modifications in Our and Previous Algorithms

We see from our algorithm that the number of key modifications is $(k-3) \cdot 2^8 + 2^8$ (2^8 for the last key index increasing technique) on an average to pass first and second round. Also to pass the remaining rounds, our algorithm performs on an average $(k-3) \cdot 2^8$ number of key modifications. So on an average our algorithm does total $(k-3) \cdot 2^8 + (k-3) \cdot 2^8 + 2^8$ 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 first and second round, Chen-Miyaji's algorithm needs $(k-3) \cdot 2^8$ number of key modifications on an average. But for the remaining rounds their algorithm uses $(n-2) \cdot (2^8 + k)$ number of key modifications. So on an average Chen-Miyaji's algorithm needs $(n-2) \cdot (2^8 + k) + (k-3) \cdot 2^8$ number of key modifications and Matsui's algorithm needs $\frac{n}{2} \cdot k \cdot 2^8$ number of key modifications, because Matsui's algorithm tries all possible key modifications on each key indexes to pass the rounds.

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 Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2 core machine, we 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: 1C 50 22 84 5B 74 EC B6 62 F5 95 7E 04 5F 5A 08 4A 6B C7 2C (2D) 02 95,
2nd pair: 58 76 1D 95 DC F6 E8 13 00 47 4E D2 41 C1 01 4A 57 7D C7 BD (BE) 02 68,
3rd pair: 8F E2 82 5D 51 74 8A BF 28 1D E1 23 0A 8E 3C FE 98 5C C7 21 (22) 02 6A.

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. 6D BC E3 0D BD 7C 6D 8F 63 82 F4 F3 72 09 FA A6 80 D0 (D1) 02 AA,
2. 2B E9 1F F9 AF 37 8A 2F E4 D0 15 00 02 A3 12 C9 01 63 (64) 02 CB,
3. E8 D4 B0 BE F6 70 02 AF A6 D4 C1 A8 CE A3 D3 E1 C7 6B (6C) 02 98,

4. 40 DE DF 44 4E A1 47 18 7C B9 35 81 1B 44 98 BD 07 43 (44) 02 B9,
5. 9A 94 C8 F8 09 23 2F 0B 5D AD 1C 7C 59 4D 08 66 3A 34 (35) 02 9B,
6. DE 55 B2 92 BD BB DC 1D D0 5B A5 7A B6 62 DE 6C 7D 67 (68) 02 9B,
7. 4C EA 23 34 68 36 F2 04 8E A2 62 14 72 5F 3E 6B E7 50 (51) 02 9B,
8. 2B 1E 53 0A AF 4F 7C F4 13 B8 7B 1D A7 7B FB 9D AF 9B (9C) 02 98,
9. D7 47 4B 4C 73 D2 AD B9 F2 2F DC D5 C1 AB F4 80 FF 6B (6C) 02 B5,
10. 52 92 AE 7B 15 F6 DA 45 F4 3F 37 E2 02 11 AA 16 E2 41 (42) 02 AA,
11. 40 DE DF 44 4E A1 47 18 7C B9 35 81 1B 44 98 BD 07 43 (44) 02 B9,
12. 9A 94 C8 F8 09 23 2F 0B 5D AD 1C 7C 59 4D 08 66 3A 34 (35) 02 9B.

5 Complexity Estimates and Comparison with Previous Works

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* $\Pr_{t,(x,y)}$ *defines the probability that a trial passes the* t*-th* $(2 < t < n-1)$ *round after modifying the secret key as* $K[x] = K[x] + y$, $K[x+1] = K[x+1] - y$ *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* $p_{1,i} = \frac{256-(c_{t,x}-c_{i,x})}{256}$, $p_{2,i} = \frac{c_{t,x}-c_{i+1,x}}{256}$, $q_i = p_{1,i}^2 + 2\cdot p_{1,i}\cdot(1-p_{1,i})\cdot p_{2,i}^2 + (1-p_{1,i})^2\cdot p_{2,i}^2$, $p_3 = (\frac{255}{256})^k \cdot \frac{1}{256}$.

Proof. Let us consider that the trial passes the first r $(< t)$ rounds, but fails to pass the t-th round. 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 $J_{x,r}$, $J_{x+1,r}$ denote the j-values before the key modification on the key index x at round r and $J^{'}_{x,r}$, $J^{'}_{x+1,r}$ denote the modified (new) j-values after the key modification. Due to the key modification, we must have $J_{x+1,r} = J^{'}_{x+1,r}$. So these two values $J_{x+1,r}, J^{'}_{x+1,r}$ don't affect the state. But the remaining two j-values $J_{x,r}$, $J^{'}_{x,r}$ causes difference in the state S for some $y \in [1,255]$. We see that after the key modification in a trial, the key sum $\sum_{x=0}^{k-1} K[i]$ always remains the same, but the state may be change for any round 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 affect the current and the previous rounds and the corresponding restrictions are as follows.

Suppose we are at the round r and perform a key modification as $K\langle x,y\rangle$, then the previous rounds will not affect if $\forall$ rounds $r < t$, any of the following four cases hold.

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

For any t-th round, we define the following events. $\mathcal{A}_r$: $J_{x,r} \in [c_{r,x}+1, c_{t,x}]$.
$\mathcal{B}_r$: $J^{'}_{x,r} \in [c_{r,x}+1, c_{t,x}]$.
$\mathcal{C}_r$: $\exists m^{'} \in [c_{r,x}+1, J^{'}_{x,r})$ such that $J_{m'} = J^{'}_{x,r}$.

$\mathcal{D}_r$: $\exists m \in [c_{r,x}+1, J_{x,r})$ such that $J_m = J_{x,r}$.
$\mathcal{E}_r$: the event that the key modification in round r will not break the class 1 and class 2 j-conditions, that have already been satisfied in the previous rounds.
$\mathcal{F}$: the event that our algorithm will help to pass the t-th round.
$\mathcal{G}$: the trial fails to pass the t-th round.

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

$$\Pr(\mathcal{E}_r) = \Pr(\mathcal{A}_r^c, \mathcal{B}_r^c) + \Pr(\mathcal{A}_r^c, \mathcal{B}_r, \mathcal{C}_r) + \Pr(\mathcal{A}_r, \mathcal{B}_r^c, \mathcal{D}_r) + \Pr(\mathcal{A}_r, \mathcal{B}_r, \mathcal{C}_r, \mathcal{D}_r), \tag{1}$$

where $\Pr(\mathcal{A}_r^c, \mathcal{B}_r^c) = \Pr(\mathcal{A}_r^c) \cdot \Pr(\mathcal{B}_r^c) = \frac{256-(c_{t,x}-c_{r,x})}{256} \cdot \frac{256-(c_{t,x}-c_{r,x})}{256}$, and
$\Pr(\mathcal{A}_r^c, \mathcal{B}_r, \mathcal{C}_r) = \Pr(\mathcal{A}_r^c) \cdot \Pr(\mathcal{B}_r) \cdot \Pr(\mathcal{C}_r) = \frac{256-(c_{t,x}-c_{r,x})}{256} \cdot \frac{c_{t,x}-c_{r,x}}{256} \cdot \frac{c_{t,x}-c_{r,x}+1}{256^2}$, because $\mathcal{C}_r \equiv \exists m^{'} \in [c_{r,x}+1, c_{t,x}-1)$ such that $J_{m'} =$ a state index.

Similarly,

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

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

Plugging in these values in Equation (1), we get the value of $\Pr(\mathcal{E}_r)$.

Thus the probability that the key modification in round r will break the class 1 and class 2 j-conditions, that have already been satisfied in the previous rounds is denoted and defined as $\Pr(\mathcal{E}_r^c) = 1 - \Pr(\mathcal{E}_r)$.

Also, $\Pr(\mathcal{F}) = \left(\frac{255}{256}\right)^k \cdot \frac{1}{256}$ and hence $\Pr(\mathcal{F}^c) = 1 - \Pr(\mathcal{F})$.

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

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

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

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

In our algorithm, we use last round passing technique. Here the last round indicates the interval as $[d+(n-2)k, d+(n-1)k]$. If the key size k is small, then we expect that the last round will be passed by uniform probability in the worst case, because in this case the key modification does not help much. We denote $\mathcal{X}_n$ as the probability to pass n-th round. So in worst case, our algorithm has $\mathcal{X}_n = \frac{1}{256}$ and hence $Pr_{n,(x,y)} = Pr_{n-1,(x,y)} \cdot \mathcal{X}_n$. 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 $n = \lceil \frac{256+k-1-d}{k} \rceil$.*

The search complexity of Matsui's algorithm can be found in [4, Theorem 3] and that of Chen-Miyaji's algorithm can be found in [4, Theorem 2]. 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. It is easy to see that if $\mathcal{COMP}^{our}$, $\mathcal{COMP}^{Mat}$ and $\mathcal{COMP}^{CM}$ 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 $\mathcal{COMP}^{our} \cdot \left[(k-3) \cdot 2^8 + (d-1) \cdot 2^8 + 2^8\right]$, $\mathcal{COMP}^{Mat} \cdot k \cdot \frac{n}{2} \cdot 2^8$ and $\mathcal{COMP}^{CM} \cdot \left[(n-2) \cdot (2^8+k) + (k-3) \cdot 2^8\right]$ respectively. In Table 4, we provide

a numerical comparison of the search complexities of the three algorithms for different key lengths. 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.

Key size (bytes)	Matsui's algorithm		Chen-Miyaji's algorithm		Our algorithm	
	Search Complexity	Time Complexity	Search Complexity	Time Complexity	Search Complexity	Time Complexity
20	2^{60}	2^{75}	2^{53}	2^{65}	2^{38}	2^{51}
22	2^{53}	2^{68}	2^{45}	2^{57}	2^{35}	2^{48}
24	2^{46}	2^{61}	2^{40}	2^{52}	2^{32}	2^{45}
27	2^{40}	2^{55}	2^{34}	2^{46}	2^{27}	2^{40}
30	2^{35}	2^{50}	2^{30}	2^{42}	2^{23}	2^{36}

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

We give an experimental comparison of Matsui, Chen-Miyaji and our algorithm in terms of the average time to get a colliding pair of different sizes in Table 5.

Key size (bytes)	Matsui's algorithm	Chen-Miyaji's algorithm	Our algorithm
64	0.50 sec	0.45 sec	0.07 sec
43	80 sec	95 sec	0.8 sec
30	392 sec	1200 sec	90 sec
28	1.8 hrs	4.8 hrs	35 mins
26	4 hrs	-	2 hrs
25	23 hrs	-	12 hrs
24	23 hrs	-	12 hrs

Table 5. Average 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 for 2^{20} times ('-' means that we get no colliding pair by running one instance for 3 days)

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 the RC4 stream cipher and introduce a new second round 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.
6. J. Chen and A. Miyaji. Novel Strategies for Searching RC4 Key Collisions. In Computers & 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 & 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.