Novel Strategies for Searching RC4 Key Collisions[☆]

Jiageng Chen[1]

School of Information Science, Japan Advanced Institute of Science and Technology, 1-1, Asahidai, Nomi, Ishikawa, 923-1292, Japan

Atsuko Miyaji[2]

School of Information Science, Japan Advanced Institute of Science and Technology, 1-1, Asahidai, Nomi, Ishikawa, 923-1292, Japan

Abstract

The property that the stream cipher RC4 can generate the same keystream outputs under two different secret keys has been discovered recently. The principle that how the two different keys can achieve a collision has been studied in the previous researches by investigating the key scheduling algorithm of RC4. However, how to find those colliding key pairs is a different story, which has been largely remained unexploited. Previous researches have demonstrated that finding colliding key pairs becomes more difficult as the key size decreases, and also finding key collisions can be related to key recovery attacks and hash collisions. In this paper, we propose novel searching techniques which can be used to find short colliding key pairs that belong to two different kind of colliding key patterns efficiently. The colliding key pairs we find in both patterns are by far the shortest ones ever found.

Keywords: RC4, Key Collision, Searching Algorithm, Transitional Pattern, Self-Absorbing Pattern.

1. Introduction

The stream cipher RC4 is one of the oldest and most wildly used stream ciphers in the world. It has been deployed to innumerable real world appli-

[☆]The preliminary version will be presented at ISC2011 [3].
[1]The author is supported by GRP program, JAIST.
[2]This study is partly supported by Grant-in-Aid for Scientific Research (B), 20300032.

cations including Microsoft Office, Secure Socket Layer (SSL), Wired Equivalent Privacy (WEP), etc. Since its debut in 1994 [1], many cryptanalysis works have been done on it, and many weaknesses have been exploited, such as [8] [9] and [10]. However, if RC4 is used in a proper way, it is still considered to be secure. Thus it is still considered to be a high valuable cryptanalysis target both in the industrial and academic world.

In this paper, we focus on exploiting the weakness that RC4 can generate colliding key pairs, namely, two different keys will result in the same keystream output. This weakness was first discovered by [2] and later generalized by [5]. The key collisions itself can cause trouble if the adversary is well aware of the pattern and he can target the secret key generation algorithm so that although different keys are generated, they have the same encryption and decryption effects. The existence of the colliding key pairs has been known but how to find those special keys are to some degree remained unexploited. To efficiently find colliding key pairs is of both theoretical and practical interests. It has been demonstrated that for short key length 16 to 32 bytes which is also the length used in most of the real world applications, the number of the colliding key pairs is so small compared with the key space so that it is impossible to find them by just brute force searching given the current computing resources. Also if we can find key collision efficiently, in other words, we can identify which key has a related one to form a collision, then when perform brute force searching, we do not need to check that related one. This can help speeding up the brute force searching for the correct secret key. Another motivation comes from the key recovery attack described in [7], whose complexity depends heavily on how fast we can find those colliding key pairs. In [12], a hash function based on RC4 was proposed. The general idea is that the key scheduling part (KSA algorithm) of RC4 has a similar structure to a hash function (secret key is replaced by message) and by extending with some necessary parts, hash function can be achieved. Finding key collisions efficiently in other words demonstrates a general attack on hash functions based on the RC4 structure, which is different from the specific attack proposed in [13]. All the above points explain our motivation to investigate how to efficiently search the RC4 colliding key pairs.

Our main contribution is proposing a searching algorithm that can find short colliding key pairs efficiently for both of the known RC4 colliding key patterns, namely, Transitional pattern and Self-Absorbing pattern [4]. Although it has been demonstrated that colliding keys of Transitional pattern is relatively easier to find than the ones of Self-Absorbing pattern, to find short colliding key pairs (under 64-bytes) by brute force searching is almost impossible. In [2], 24-byte colliding key pair of Transitional pattern were found in about ten days time by using multi-cores. In this paper with our new proposed methods, even shorter 22-byte colliding key pair of Transitional pattern can be found within only 5 hours on a normal desktop PC, and also 39-byte colliding key pairs of Self-Absorbing pattern are found within only 5 seconds time.

We organize the paper as follows. In Section 2, we give a short introduction on RC4 algorithm and the details on the key collisions in both patterns. In Section 3, new proposed algorithm for searching key collisions in Transitional pattern is covered, along with the previous results from [2]. We also show the complexity evaluations of the two algorithms to compare the efficiency. Section 4 then covers the searching techniques for finding key collisions in Self-Absorbing pattern. Finally the Conclusion is given in Section 5.

2. RC4 algorithm and Key Collisions

First we shortly describe the RC4 algorithm. The internal state of RC4 consists of a permutation S of the numbers $0, ..., N-1$ and two indices $i, j \in \{0, ..., N-1\}$. The index i is determined and known to the public, while j and permutation S remain secret. RC4 consists of two algorithms: The Key Scheduling Algorithm (KSA) and the Pseudo Random Generator Algorithm (PRGA). The KSA generates an initial state from a random key K of k bytes as described in Algorithm 1. It starts with an array $\{0, 1, ..., N-1\}$ where $N = 256$ by default. At the end, we obtain the initial state S_{N-1}.
Once the initial state is created, it is used by PRGA. The purpose of PRGA is to generate a keystream of bytes which will be XORed with the plaintext to generate the ciphertext. PRGA is described in Algorithm 2. Since key collision is only related to KSA algorithm, we will ignore PRGA in the rest of the paper.

Algorithm 1. KSA

```
1: for $i = 0$ to $N - 1$ do
2:     $S[i] \leftarrow i$
3: end for
4: $j \leftarrow 0$
5: for $i = 0$ to $N - 1$ do
6:     $j \leftarrow j + S[i] + K[i \bmod l]$
7:     swap($S[i], S[j]$)
8: end for
```

Algorithm 2. PRGA

```
1: $i \leftarrow 0$
2: $j \leftarrow 0$
3: loop
4:     $i \leftarrow i + 1$
5:     $j \leftarrow j + S[i]$
6:     swap($S[i], S[j]$)
7: Output $z_i = S[S[i] + S[j]]$
8: end loop
```

The weakness of key collision was first discovered by [2], and then different patterns have been discovered by the following researches such as [5], [7] and [4]. Especially [4] generalized the key collision and concluded into two patterns, namely, Transitional Pattern and Self-Absorbing Pattern, and the key collisions discovered so far can be covered by either of the patterns. One of the most important theorem from [4] states that the colliding key pairs become more difficult to find as the key length becomes shorter, and the hamming distance (the number of locations that two keys differ from each other) becomes larger in both patterns. Thus we only focus on finding the colliding key pairs with the smallest hamming distances, and as short as possible to cover the real world applications.

Brute force search is always the most trivial way in cryptanalysis. Here we give the brute force attack only to give an intuition on how difficult to find key collisions. The attacker simply generates a random secret key K with length k, and runs the KSA to test its random variable j's behavior. If the trial fails, then repeat the procedure until one colliding key pair is found. For Transitional Pattern in [2], it has been demonstrated that for each trial, the successful probability is around $(\frac{1}{256})^{n+2}$. Thus the complexity for the brute force searching is as high as 2^{104} for 22-byte keys. For Self-Absorbing Pattern, it has been shown that it is even harder to find, thus the complexity would be even higher.

A searching algorithm was proposed by Matsui, and 24-byte is the shortest colliding key pair found. However, no complexity evaluation was given by the author, and it is not trivial to see. In section 3.6, we give the complexity evaluation on both our algorithm and Matsui's algorithm to compare the efficiency. Here we make a short introduction on his searching technique which is described in Table 1. It defines a search function with two related keys as input, and output a colliding key pair or fail. When some trial fails to find the colliding key pair, the algorithm does not restart by trying another random related key pair, instead, it modifies the keys as $K_1[x] = K_1[x] + y, K_1[x+1] = K_1[x+1] - y$ for every x and y. Since $j_x = j_x + y$ and $j_{x+1} = j_x + S_x[x+1] + K_1[x]$, thus j_{x+1} after the modification will not be changed. This means that by modifying in this way, the next trial will have a relatively close relation with the previous trial, in other words, if the previous trial before the modification tends to achieve a collision, then the next trial after the modification will also have the tendency. The algorithm recursively calls the function Search(K_1, K_2) until it return a colliding key or fail. We will take advantage of this idea to combine with our proposed method to achieve the best searching efficiency, which will be covered in detail in section 3.5.

3. How to find key collisions in Transitional Pattern

3.1. Transitional Pattern

In [2], it clearly described how two keys K_1 and K_2 with the only one difference $K_2[d] = K_1[d] + 1$ can achieve a collision. It traced two KSA procedure and two S-Box states generated by the two keys, and pointed out how two S-Box states become equal to each other at the end of the KSA. Actually, the essence of the key collisions is only related to some j values at some specific locations. If these conditions once satisfied, a collision is expected. Thus we prefer to use another way to explain the collision by listing all the j conditions. In this way, we only need to exam the behavior of one key, since once the j values generated by this key satisfy all the conditions, then deterministically, there exists another key that they form a colliding key pair. To simplify, we check whether a given key K_1 has a related key K_2 such that $K_2[d] = K_1[d] + 1$ and K_1 and K_2 can achieve a collision. Then all we need is to confirm whether K_1's j behaviors satisfy the conditions in Table 1.

The Round column presents the round number in the KSA steps in the Round Interval column. There are $n = \lfloor \frac{256+k-1-d}{k} \rfloor$ rounds, which is also the times that the key difference repeats during KSA. We separate the conditions into two categories, Classes 1 and 2. From Table 1, you see that the conditions in Class 1 column are computational dominant compared with Class 2. This is because for j at some time to be some exact value, probability will only be 2^{-8} assuming random distribution, while not equal to some exact value in Class 2 has a relatively much higher probability. Also the main point for

Table 1: Matsui's Algorithm [2]

```
Input: Key length $k$, $d = k - 1$
Output: colliding key pair $K_1$ and $K_2$ such that $K_2[d] = K_1[d] + 1$,
$K_1[i] = K_2[i]$ if $i \neq d$, $KSA(K_1) = KSA(K_2)$.
------------------------------------------------------------
1: Generate a random key pair $K_1$ and $K_2$ which differs at position $d$ by one.
Set $K_1[d+1] = K_2[d+1] = k - d - 1$.
2: Call function Search$(K_1, K_2)$, if Search$(K_1, K_2)$=1, collision is found, else goto 1.
------------------------------------------------------------
Search$(K_1, K_2)$ :
    $s = MaxColStep(K_1, K_2)$
    If $s == 255$, then return 1.
    $MaxS = max_{x,y} MaxColStep(K_1\langle x, y\rangle, K_2\langle x, y\rangle)$
    If $Maxs \leq s$, then return 0.
    C=0
    For all $x$ and $y$, do:
        If $MaxColStep(K_1\langle x, y\rangle, K_2\langle x, y\rangle) = MaxS$, call $Search(K_1, K_2)$
        $C = C + 1$
        If $C == MaxC$, then return 0.
------------------------------------------------------------
Notations:
$MaxColStep(K_1, K_2)$: The maximal steps at which the number of $S$-Box that $S_1$ differs
                        from $S_2$ is at most two.
$K\langle x, y\rangle : K[x] = K[x] + y, K[x+1] = K[x+1] - y$, $K[i] = K[i]$ if $i \neq x, x+1$.
```

finding a colliding key pair is how to meet those low probability conditions in Class 1 column. In the rest of the paper, we focus on the Class 1 conditions. When we say a KSA procedure (a trial) under some key K passes round i and fails at round $i+1$, we indicate that all the Class 1 j conditions are satisfied in the previous i rounds and fails at the $i+1$-th round.

3.2. Bypassing the first round deterministically

Our first observation is that we can pass the first round. Recall that in the first round, there are two j conditions in Class 1 that we need to satisfy,

Table 2: j conditions for Transitional Pattern

Round	Round Interval	Class 1	Class 2
1	$[0, d+1]$	$j_d = d$, $j_{d+1} = d+k$	$j_{0\sim d-1} \neq d, d+1$
2	$[d+2, d+k]$	$j_{d+k} = d+2k$	$j_{d+2\sim d+k-1} \neq d+k$
… …	… …	… …	… …
t	$[d+(t-2)k+1,$ $d+(t-1)k]$	$j_{d+(t-1)k} = d+tk$	$j_{d+(t-2)k+1\sim d+(t-1)k-1} \neq$ $d+(t-1)k$
… …	… …	… …	… …
$n-1$	$[d+(n-3)k+1,$ $d+(n-2)k]$	$j_{d+(n-2)k} = (d-1)+(n-1)k$	$j_{d+(n-3)k+1\sim d+(n-2)k-1} \neq$ $d+(n-2)k$
n	$[d+(n-2)k+1,$ $d+(n-1)k-1]$	$j_{d+(n-1)k-2} = S^{-1}_{d+(n-1)k-3}[d]$, $j_{d+(n-1)k-1} = d+(n-1)k-1$	$j_{d+(n-2)k+1\sim d+(n-1)k-3} \neq$ $d+(n-1)k-1$

namely

$$j_d = d \quad \text{and} \quad j_{d+1} = k+d$$

As in [2], the setting of $K[d+1] = k-d-1$ always meets the condition $j_{d+1} = k+d$ since we have $j_{d+1} = j_d + d + 1 + K[d+1]$. But still we have another condition $j_d = d$ left in the first round. Thus we add the following key modification:

$$K[d] = 256 - j_{d-1}$$

at the time when KSA is proceeded at index $d-1$ after the swap. As a result, since $j_d = j_{d-1} + d + K[d] = d$, and by modifying $K[d]$ dynamically when the previous value j_{d-1} is known, $j_d = d$ will always be satisfied. Then we can bypass the first round and reduce the necessary number of rounds to $n-1$.

3.3. Bypassing the second round with high probability

If we choose the differential key index carefully, we find that the second round can also be skipped with very high probability compared with the uniform distribution. Generally speaking, we would like to choose $d = k-1$ so that in the KSA procedure, the key differential index will be repeated as a few times as possible. Actually choosing the d at the indices close to $k-1$ will have the same affect as the last index $k-1$. For example, for key with length 20-24 bytes, setting the key differential at indices $k-1, k-2, k-3, k-4$

will cause the key differential index to be repeated the same times during the KSA. Thus instead of setting $d = k - 1$, let's set

$$d = k - 3$$

so that after d, we have another two key bytes. For the first and second round, the following two j conditions are necessary to meet:

$$j_{d+1} = j_{k-2} = 2k - 3$$

$$j_{d+k} = j_{2k-3} = 3k - 3$$

and we have

$$j_{2k-3} = j_{k-2} + K[k-1] + \sum_{i=0}^{k-3} K[i] + \sum_{i=k-1}^{2k-3} S_{i-1}[i] \quad (1)$$

$$\overset{P_{2nd}}{=} j_{k-2} + K[k-1] + \sum_{i=0}^{k-3} K[i] + \sum_{i=k-1}^{2k-3} S_{k-2}[i] \quad (2)$$

Thus by modifying

$$K[d+2] = K[k-1] = j_{2k-3} - j_{k-2} - \sum_{i=0}^{k-3} K[i] - \sum_{i=k-1}^{2k-3} S_{k-2}[i]$$

at the time $i = k - 2$ after the swap, with probability

$$P_{2nd} = \frac{256 - (k-2)}{256} \times \frac{256 - (k-3)}{256} \times \cdots \times \frac{256 - 1}{256} = \prod_{i=1}^{k-2} \frac{256 - i}{256}$$

we can pass the second round.

This can be explained as follows. For two fixed j values j_{k-2} in the first round, and j_{2k-3} in the second round, we have equation (1). At the time $i = k - 2$ after the swap, we don't know $\sum_{i=k-1}^{2k-3} S_{i-1}[i]$, but we can approximate it by using $\sum_{i=k-1}^{2k-3} S_{k-2}[i]$. The conditions on this approximation is that for $i \in [k-1, 2k-4]$, j does not touch any indices $[i+1, 2k-3]$, which gives us the probability P_{2nd}. Then if we set the $K[d+2]$ as before, with P_{2nd} we can pass the second round. Notice that the reason why we can modify $K[d+2]$ is related to the choice of d. When modifying $K[d+2]$, we don't wish the modification will affect the previous execution, which has been successfully passed. When modifying $K[d+2]$ trying to meet the second round condition, this key byte is used for the first time during KSA, thus we won't have the previous concern. For short keys such as $k = 24$, $P_{2nd} = 0.36$, and for $k = 22$, $P_{2nd} = 0.43$. The successful probability is thus much bigger compared with the uniform probability $2^{-8} = 0.0039$.

3.4. Reducing the complexity in the last round

In the last round, there are two j conditions need to be satisfied, namely,

$$j_{(n-1)k+d-2} = r \text{ such that } S_{(n-1)k+d-3}[r] = d$$

$$j_{(n-1)k+d-1} = d + (n-1)k - 1$$

And from $j_{(n-1)k+d-1} = j_{(n-1)k+d-2} + S_{(n-1)k+d-2}[(n-1)k+d-1] + K[d-1]$, $K[d-1]$ can be decided if $j_{(n-1)k+d-2}$ is fixed to some value. During the KSA procedure, $j_{(n-1)k+d-2}$ could be touching any indices, but with overwhelming probability, it will touch index d. This is because after step $i = d$, one of the two S-Box differentials will be staying at index d till step $i = (n-1)k+d-2$ unless it is touched by any j during the steps $[d+1, (n-1)k+d-3]$. Thus we can assume that

$$j_{(n-1)k+d-2} = d$$

and we can thus modify $K[d-1]$ at step $i = d-1$ before the swap as follows:

$$K[d-1] = j_{(n-1)k+d-1} - j_{(n-1)k+d-2} - S_{(n-1)k+d-2}[(n-1)k+d-1] = (n-1)k+d-1-d-(d+1) = (n-1)k-d-2$$

This modification indicates that if some trial meets the $j_{(n-1)k+d-2} = d$ condition in the last round, then with probability 1, the other condition in this round on $j_{(n-1)k+d-1}$ will be satisfied. Simply speaking, 2^{16} computation cost is required to pass the final round, while we reduce it to

$$P_{last} = 2^8 \times (\frac{255}{256})^{-((n-1)k-3)}$$

For a 24-byte key, the computation cost can be reduced to around $2^{9.2}$, which is a significant improvement. The overall cost will be covered in the next section, here we just demonstrate to give a intuition.

3.5. Multi-key modification

In the area of finding hash collisions, multi-message modification is a widely used technique that first proposed by [11]. MD5 and some other hash functions are broken by using this technique. The idea is that when modifying the message block at some later round i to satisfy the i-th round conditions, leaving the previous rounds conditions satisfied (in hash functions, a message block is usually processed for many rounds in different orders). Compared with finding hash collisions, the essence of the Algorithm is actually very similar to the multi-message modification, thus we call it multi-key modification. This is because for the round $i > 1$, if we want to modify the key to satisfy the i-th round conditions, almost for sure we will violate the previous round conditions. In Matsui's algorithm, by adding to the values of the consecutive two indexes with $+y$ and $-y$ difference, the previous satisfied conditions with still hold with high probability. Since this technique does not conflict with our previous mentioned ones, we can take advantage of this multi-key modification to enhance our searching efficiency. One thing to take care is that we restrict x to the range $[0, d-3]$ so that the key modification will not violate our previous modifications. Then we derive our algorithm for searching colliding key pairs in the Transitional Pattern, which is described in Table 3.

3.6. Complexity evaluation

We will see from a theoretical point of view, how efficiently our proposed algorithm can perform. Since we combine Matsui's algorithm, the analysis we give here can also be adapted to [2], in which no complexity was given. We start by giving the following theorem which is important to compute the complexity, and show the proof.

Theorem 1. *Define $Pr_{t,(x,y)}$ be the probability for a trial that passes round t ($t \geq 2$) by modifying the secret key as $K[x] = K[x] + y, K[x+1] = K[x+1] - y$ according to the multi-key modification given the previous trial fails to pass the t-th round. And assuming the probability to satisfy the Class 1 j conditions in the t-th round is R_t, then*

Table 3: Our Key Collision Searching Algorithm for Transitional Pattern

```
Input: Key length $k$, different index $d = k - 3$, $n = \lfloor \frac{256+k-1-d}{k} \rfloor$
Output: $K_1$ and $K_2$ such that $K_2[d] = K_1[d] + 1$, $K_1[i] = K_2[i]$ if $i \neq d$,
        $KSA(K_1) = KSA(K_2)$

1: Store the following $j^*$ values in the table, which are the conditions needed to be
   satisfied. $j^*_d = d, j^*_{d+1} = k + d, j^*_i = i + k$ for $i \in \{d + k, ..., d + k(n - 2)\}$,
   $j^*_{d-2+k(n-1)} = d$, $j^*_{d-1+k(n-1)} = d - 1 + k(n - 1)$. (Class 1 $j$ conditions)
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 $K_1[d] = 256 - j_{d-1}$, and
   $K_2[d] = K_1[d] + 1$.
4: Keep running the KSA until $i = d + 1$ after the swap. Modify
   $K_1[d + 2] = j^*_{2k-3} - j^*_{k-2} - \sum_{i=0}^{k-3} K_1[i] - \sum_{i=k-1}^{2k-3} S_{1,k-2}[i]$
   $K_2[d + 2] = j^*_{2k-3} - j^*_{k-2} - \sum_{i=0}^{k-3} K_2[i] - \sum_{i=k-1}^{2k-3} S_{2,k-2}[i]$
5: Set the recursive depth variable $R = 0$.
6: If TranSearch($K_1$, $K_2$)==$n$
       Colliding key pair found. Output $K_1$ and $K_2$.
   Else goto 2.

TranSearch($K_1, K_2$):
If ($MaxR = Round(K_1, K_2)$) == $n$, then return $n$.
For $x$ from 0 to $d - 3$ do
    For $y$ from 0 to 255 do
        $K_1[x] = K_1[x] + y, K_1[x + 1] = K_1[x + 1] - y$
        $K_2[x] = K_2[x] + y, K_2[x + 1] = K_2[x + 1] - y$
        If $Round(K_1, K_2) \leq MaxR$ or $R = n$
            Return $Round(K_1, K_2)$
        Else $R = R + 1$, TranSearch($K_1, K_2$)

Notation
$Round(K_1, K_2)$ : The number of rounds that a key pair $K_1, K_2$ can pass. In other
                    words, key pair $K_1$ and $K_2$ satisfy all the $j$ conditions in the first
                    $Round(K_1, K_2)$ rounds.
```

$$Pr_{t,(x,y)} \approx \Bigg((\frac{256-(n-2)k-d+x}{256})^4 - \sum_{i=2}^{t-1} \Bigg((1-(\frac{256-(t-i-1)k-d+x}{256})^4) \cdot \prod_{j=1}^{i-1} (\frac{256-(t-j-1)k-d+x}{256})^4 \Bigg) \Bigg) \cdot R_t$$

Proof 1. *Now let's consider some trial that passes all the first $t-1$ rounds and fails to pass the t-th round. Then we modify the secret key at indices x and $x+1$ with value difference y so that $K[x] = K[x]+y, K[x+1] = K[x+1]-y$. Let's denote $j^{'}_{s,x}, j^{'}_{s,x+1}$ and $j_{s,x}, j_{s,x+1}$ be the j values for the current trial and the trial after the key modification at the modified key indices at round s. It is easy to see that for each such key modification, the change of the 4 j values at each round will cause 4 S-Box values to be changed.*

For the trial before the key modification, the successful pass of the first $t-1$ rounds indicates the correct S-Box sum for some fixed key sum $\sum_{i=0}^{k-1} K[i]$. Since our modification doesn't change the key sum, thus, after the key modification, the previous correct S-Box sum should still be satisfied in order to have a chance to pass the t-th round. Otherwise, the key modification will only cause a failure at an rather early round. For example if the previous trial passes the first $t-1$ rounds, for the key modification in round $1 \le s \le t-1$ (assuming this key modification passes all the previous s-1 rounds), the S-Box sum $\sum_{i=x+(s-1)k}^{(t-1)k-1} S_{i-1}[i]$ should not be violated by the 4 changed j values $j^{'}_{s,x}, j^{'}_{s,x+1}$ and $j_{s,x}, j_{s,x+1}$.

First let's consider the probability that due to the key modification that the previous correct S-Box sum is violated. The modification is processed in the same order as the KSA procedure. And notice that in each round, due to the key modification, we have 4 changed j values, and they are checked in the sequence $j^{'}_{s,x}, j_{s,x}, j^{'}_{s,x+1}, j_{s,x+1}$ whether the failure conditions are satisfied. Notice that due to the use of the multi-key modification technique, the S-Box sum in the t-th round can not be touched since we have already precomputed the sum and are expecting the corresponding swap. The following events define the S-Box intervals that once touched, the previous correct S-Box sums will be violated due to the modification in round s.

- $A_s : j^{'}_{s,x} \in [x+(s-1)k, (t-2)k+d]$ *(the original $j^{'}_{s,x}$ violates the S-Box sum $\sum_{i=x+(s-1)k}^{(t-2)k+d} S_{i-1}[i]$)*

- $B_s : j_{s,x} \in [x+(s-1)k, (t-2)k+d]$ *(the newly modified* $j_{s,x}$ *violates the S-Box sum* $\sum_{i=x+(s-1)k}^{(t-2)k+d} S_{i-1}[i]$*))*
- $C_s : j'_{s,x+1} \in [x+(s-1)k+1, (t-2)k+d]$ *(the original* $j'_{s,x+1}$ *violates the S-Box sum* $\sum_{i=x+(s-1)k+1}^{(t-2)k+d} S_{i-1}[i]$*)*
- $D_s : j_{s,x+1} \in [x+(s-1)k+1, (t-2)k+d]$ *(the newly modified* $j_{s,x+1}$ *violates the S-Box sum* $\sum_{i=x+(s-1)k+1}^{(t-2)k+d} S_{i-1}[i]$*)*

Denote $Pr(S_s)$ *to be the probability that the modification in round* s *will not break the Class 1* j *conditions that have been satisfied in the previous trial.*

$$\begin{aligned} Pr(S_s) &= (1-Pr(A_s))\cdot(1-Pr(B_s))\cdot(1-Pr(C_s))\cdot(1-Pr(D_s)) \\ &= Pr(\bar{A_s})\cdot Pr(\bar{B_s})\cdot Pr(\bar{C_s})\cdot Pr(\bar{D_s}) \end{aligned}$$

Denote $Pr(F_s)$ *to be the probability that the modification in round* s *will break the Class 1* j *conditions that have been satisfied in the previous trial so that the current trial fails to pass round* t.

$$Pr(F_s) = 1 - Pr(S_s)$$

The exact values for the four events can be computed as follows for $s \geq 1$*:*

$$Pr(A_s) = Pr(B_s) = \frac{(t-s-1)\times k + d - x + 1}{256}$$
$$Pr(C_s) = Pr(D_s) = \frac{(t-s-1)\times k + d - x}{256}$$

Then the total probability that after the key modification the trial fails to pass the $t-1$*-th round can be computed as follows:*

$$Pr(F) \;=\; Pr(F_1) + Pr(S_1)\cdot Pr(F_2) + \cdots + \prod_{i=1}^{t-2} Pr(S_i)\cdot Pr(F_{t-1})$$

Thus the probability that for some key modification succeeds to pass the $t-1$*-th round while these* $t-1$ *rounds are also cleared before the modification is*

$$Pr_{t-1,(x,y)} = 1 - Pr(F)$$

Given the probability R_t to satisfy the Class 1 j conditions in the t-th round, the probability to pass the first t rounds is thus

$$Pr_{t,(x,y)} = Pr_{t-1,(x,y)} \times R_t$$

After replacing with detailed parameters we complete our proof.

Based on Theorem 1, we can compute the complexity of finding key collisions in the Transitional Pattern by using our algorithm and the Matsui's algorithm. The main difference between ours and Matsui's algorithm is that we are able to bypass the first and second round almost deterministically and reducing the last round complexity. In other words, the total number of rounds can be approximately reduced to $n-2$ in our case (namely, $n-2$ is the last round). And $R_{n-2} \approx 2^{-9.2}$ for our algorithm while $R_n \approx 2^{-16}$ for Matsui's algorithm due to the reducing of the last round complexity (only one j condition is required instead of two in [2]'s case). Thus we are ready to have the following two theorems.

Theorem 2. *The complexity to find a colliding key pair with key length k using our algorithm is*

$$C_{TranNew} \approx (Pr_{n-2,(\bar{x},\bar{y})})^{-1}$$

where $Pr_{n-2,(\bar{x},\bar{y})}$ is the average case on all possible x and y, and $R_{n-2} \approx 2^{-9.2}$, $n = \lfloor \frac{256+k-1-d}{k} \rfloor$, $d = k-3$.

Theorem 3. *The complexity to find a colliding key pair with key length k using Matsui's Algorithm is*

$$C_{TranMatsui} \approx (Pr_{n,(\bar{x},\bar{y})})^{-1}$$

where $Pr_{n,(\bar{x},\bar{y})}$ is the average case on all possible x and y, and $R_n \approx 2^{-16}$ $n = \lfloor \frac{256+k-1-d}{k} \rfloor$, $d = k-1$.

We run the experiment under our proposed algorithm and successfully find by far the shortest 22-byte colliding key pair in only 5 hours computational time by using normal desktop PC (Intel i7 2.80 GHz). In case of Matsui's Algorithm, around 10 days computational time and multiple cores were used (the detailed information was not published) to find a 24-byte colliding key pair. Also, our proposed algorithm has a better efficiency searching for other short colliding keys which seems difficult to find by using the Matsui's algorithm. Here is the concrete 22-byte colliding key pair found by us in hexadecimal form:

$\mathbf{K_1}(\mathbf{K_2})$: A2 27 43 A7 03 94 2F 17 75 BB A7 27 8F DD 3E 7B C6 A1 C7 **81(82)** 02 5A

We also summarize the complexity cost given by the previous theorems in the following figure to show the trend of the complexity to find the colliding key pairs and the key length of both algorithms. Note that the complexity for finding 22-byte colliding key pairs by using our algorithm is around $2^{30.3}$ which gives a nice explanation of the 5 hours execution time. Also the complexity for finding 24-byte colliding key pairs by using Matsui's Algorithm is around $2^{48.6}$, which is also a reasonable evaluation given it's running time.

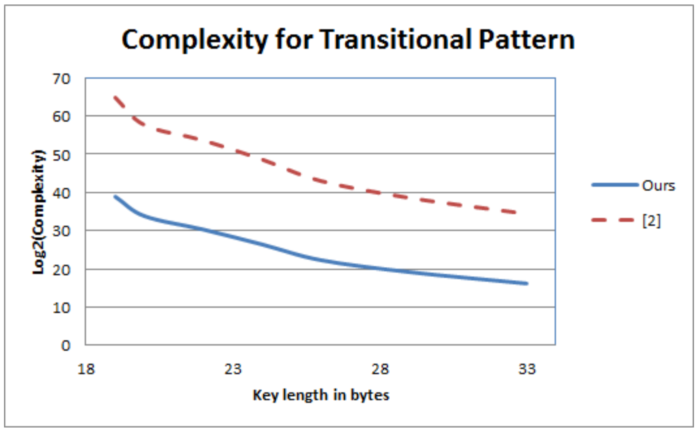

Figure 1: Complexity for finding colliding key pairs for Transitional Pattern

4. How to find colliding key pairs in Self-Absorbing Pattern

In this section, we will demonstrate how to find colliding key pairs in Self-Absorbing pattern. This pattern has a greater hamming distance (the number of key byte that differ from each other) than Transitional pattern, and it has been shown that it is more difficult to find those colliding key pairs than in this pattern (due to the increasing of the j conditions). However, we find out that the increasing of the hamming distance on the other hand will have a positive effect on searching those colliding key pairs. In general, we discover that the j conditions will actually form some relations that affect

Table 4: j&S-Box conditions for Self-Absorbing Pattern

Round	Round Interval	j conditions	S-Box conditions
1	$[0, d+t]$	$j_d = d$, $j_{d+t} = d+t$	$S_{d-1}[d] = S_{d-1}[d+t] - t$
2	$[d+t+1, d+t+k]$	$j_{d+k} = d+k$, $j_{d+k+t} = d+k+t$	$S_{d+k-1}[d+k] =$ $S_{d+k-1}[d+k+t] - t$
... ...	... ...	... ...	... ...
n	$[d+(n-1)k+t+1,$ $d+(n-1)k+t+k]$	$j_{d+(n-1)k} = d+(n-1)k$, $j_{d+(n-1)k+t} = d+(n-1)k+t$	$S_{d+(n-1)k-1}[d+(n-1)k] =$ $S_{d+(n-1)k-1}[d+(n-1)k+t] - t$

each other, and by satisfying those relations, the number of j conditions can be reduced. Although the shortest one we find is still in the Transitional pattern, this novel technique gives us a potential option which should be taken into consideration during the collision search, and it also shows that the increasing of the hamming distance will not always grantee a hard searching time.

4.1. Self-Absorbing Pattern

The existence of colliding key pairs in Self-Absorbing pattern was first theoretically and experimentally demonstrated in [6], and later generalized by [4]. Different from Transitional patterns, the internal state differences are generated and absorbed within one key appearance, which will not be propagated to the later phase. Again, we still consider the simplest case where the two keys differ from each other at three locations (this is also the smallest hamming distances required by this pattern). Two keys K_1 and K_2 are related in the following way: $K_2[d] = K_1[d] + t, K_2[d+1] = K_1[d+1] - t, K_2[d+t+1] = K_1[d+t+1] + t$, where d is the first index that differs from each other, and t is the value difference. To characterize the collision in this pattern, both j and S-Box conditions are required. We summarize them as follows:

The j conditions are much more difficult to satisfy than the S-Box conditions. This is because as long as the two corresponding S-Box elements are not touched before i touches them, then the S-Box conditions will hold, and this will occur with very high probability than j conditions.

4.2. Some special properties behind the Self-Absorbing Pattern

Besides the parameter d, we have the freedom to choose another parameter t in the Self-Absorbing Pattern. In the previous section, we have already seen how the selecting of d will affect the searching efficiency. Similarly, we find that the selection of t will also affect the searching process. Let's first demonstrate the first two rounds of KSA. In the first round, we have following conditions to satisfy:

$$j_d = d, j_{d+t} = d + t \text{ and } S_{d-1}[d] = S_{d-1}[d+t] - t$$

Since in the first round, with very high probability that $S_{d-1}[d] = d$ and $S_{d-1}[d+t] = d+t$, then from $j_{d+t} = j_d + \sum_{i=d+1}^{d+t} K[i] + \sum_{i=d+1}^{d+t} S_{i-1}[i]$ we can derive

$$\sum_{i=d+1}^{d+t} K[i] = t - \sum_{i=d+1}^{d+t} S_{i-1}[i] \tag{3}$$

Then let's look at the situation in the second round. In the second round, the condition we need to satisfy is as follows:

$$j_{d+k} = d + k, j_{d+k+t} = d + k + t \text{ and } S_{d+k-1}[d+k] = S_{d+k-1}[d+k+t] - t$$

Then from $j_{d+k+t} = j_{d+k} + \sum_{i=d+1}^{d+t} K[i] + \sum_{i=d+k+1}^{d+k+t} S_{i-1}[i]$ and equation (6), we can derive the following relation:

$$\sum_{i=d+1}^{d+t} S_{i-1}[i] = \sum_{i=d+k+1}^{d+k+t} S_{i-1}[i] \tag{4}$$

And this can be easily generalized to get the following relations by analyzing in the same way:

$$\sum_{i=d+1}^{d+t} S_{i-1}[i] = \sum_{i=d+k+1}^{d+k+t} S_{i-1}[i] = \cdots = \sum_{i=d+(n-1)k+1}^{d+(n-1)k+t} S_{i-1}[i] \tag{5}$$

This means that in each round, the sum of some S-Box values should equal to each other. The most important aspect is that we have transformed one of the j conditions in each round to the S-Box conditions in (8). Namely, if we can satisfy one of the j conditions and the S-Box conditions, then the dominant round conditions can be cleared. Now it is the time to come back to the t parameter. If t is big, then it seems that we can do very little to the S-Box sum conditions. However, when t becomes small, the number of S-Box elements involved in the sum is thus limited, and we can to some degree predict the exact values of those S-Box elements.

4.3. Novel searching techniques

Now let's see how to take advantage of the previous properties. The Self-Absorbing pattern tells us that t can be as small as 2, so let's first fix $t = 2$. Then after the first round, the following conditions must be satisfied:

$$S_{d-1}[d] = d, S_{d-1}[d+1] = d+1, S_{d-1}[d+2] = d+2 \tag{6}$$

$$K[d+1] + K[d+2] = 256 - 2d - 1 \tag{7}$$

For (9), it is not 100% the case, but since it is in the first round, the probability that j does not touch the three elements until i touches them is very high, thus we make the above assumption. The first round actually fixes some parameters such as the S-Box values and key values. Then in the later rounds, according to the previous analysis, we have:

$$\sum_{i=d+1}^{d+2} S_{i-1}[i] = \sum_{i=d+k+1}^{d+k+2} S_{i-1}[i] = \cdots = \sum_{i=d+(n-1)k+1}^{d+(n-1)k+2} S_{i-1}[i] = 3d+3 \tag{8}$$

$$S_{d+ik-1}[d+ik+2] = S_{d+ik-1}[d+ik] + 2, i \in [0, n-1] \tag{9}$$

(11) and (12) tells us that in each round, the sum of the three elements are fixed to 3d+3, and the value difference between two elements is 2. Now before running the KSA, we can compute the sum of $\Delta = \sum_{i=d+1}^{d+2} S_{i-1}[i] = d+1+d+2 = 2d+3$. Then we partition Δ into two different number within the range $[0, k-1]$, namely $S_{i-1}[i]$ and $S_{i-1}[i+1]$ where $i = d + jk + 1$ for $j \in [0, n-1]$. Then compute $S_{i-2}[i-1] = \Delta - (S_{i-1}[i] + S_{i-1}[i+1])$. Now we have computed three S-Box candidates, and all these three candidates are within the range of the first key appearance. Next is to modify the key in the first round to swap the candidates to the corresponding locations in the later rounds. For each of the later rounds, we need three S-Box candidates.

Fig.2 illustrates how to satisfy the S-Box conditions in the first 4 rounds so that the j conditions in each of the rounds can be reduced to one. We do the first round key modification by first choosing the $d, d+1, d+2$ to be in the middle locations within the range $[0, k-1]$. This is because we need both the values before and after the location d to be swapped to the corresponding locations in the later rounds. For the second round, we swap $d+3, d-2, d+5$ to the corresponding indexes $S[d+k], S[d+k+1], S[d+k+2]$. For the third round, we swap $d+4, d-3, d+6$ to the corresponding indexes $S[d+2k], S[d+2k+1], S[d+2k+2]$, and $d+7, d-6, d+9$ to the indexes

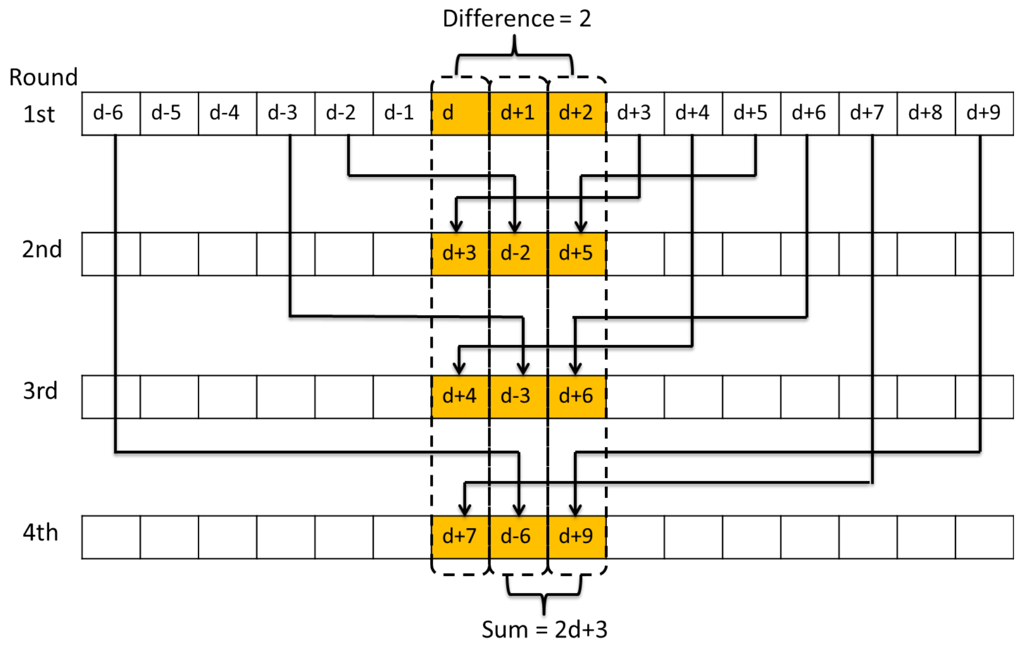

Figure 2: Modifying the secret key in the first round

$S[d+3k], S[d+3k+1], S[d+4k+2]$ for the fourth round. Thus after the key modification in the first round, the S-Box conditions are satisfied. This can be seen from Fig.2 that the difference between the first column and third column in the same line is equal to 2, and the sum of the second column and third column in the same line are equal to $2d+3$. After we adjust the S-Box values in the first round, we only hope that these conditions are not violated (touched by j) before i touches them.

Also, we can make use some of the techniques previous introduced to search key collisions in the Transitional Pattern. First round can also be passed deterministically since we have the total control of the key at this time. Also second round can be passed easily. This is because we have already reduced the number of second round j conditions from 2 to 1, the remaining one j condition can be satisfied by using the similar technique described in 3.3. In order to satisfy j condition $j_{d+k} = d+k$, we first proceed the KSA until $i = k-1$ before the swap. The S-Box sum $\sum_{i=d+3}^{d+k} S_{i-1}[i]$ is determined by the two j values j_{d+2} and j_{d+k}. Since we proceed until $i = k-1$, we obtain the correct S-Box sum $\sum_{i=d+3}^{k-1} S_{i-1}[i]$. Then, at step

$i = k-1$, we assume that $\sum_{i=k}^{d+k} S_{i-1}[i] = \sum_{i=k}^{d+k} S_{k-1}[i]$, thus we can modify $K[k-1]$ at step $i = k-1$ to satisfy the condition $j_{d+k} = d+k$.

However, we have to point out that multi-key modification is not suitable for Self-Absorbing pattern. This is because we will modify almost all the key indices in the first round especially when key is short, thus any key modifications in the later rounds will violate the modifications made during the first round, and thus will not help increasing the success probability.

39-byte colliding key pair can be found within only 5 seconds time on a normal desktop PC, here is one concrete example. Obviously, this result has a large space to be further improved.

$\mathbf{K_1}(\mathbf{K_2})$: C2 30 B3 54 07 D8 A5 D4 DF 25 C7 5B 1B 59 27 2F C9 75 77 B8 C5 5E **4F(51) C2(C0)** 11 **0C(0E)** 0D C0 0B 08 09 BC 07 04 D2 EB E1 C8 D1

5. Conclusion

In this paper, we investigate how to find RC4 colliding key pairs efficiently in both Transitional and Self-Absorbing patterns. The techniques we proposed to bypass the first and second round can be used by both patterns. For the Transitional pattern, we propose how to reduce the last round complexity and by combining the multi-key modification techniques, we are able to reduce the searching complexity dramatically compared with [2]. For Self-Absorbing pattern, novel techniques are proposed which can help reducing the j conditions in certain rounds from two to one, and thus the complexity can also be reduced significantly. 22-byte colliding key pair is the shortest one found so far, and the experimental results confirm that our algorithm does work efficiently as expected.

References

[1] 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

[2] Matsui, M.: Key Collisions of the RC4 Stream Cipher. In: Dunkelman, O., Preneel, B. (eds.) FSE 2009. LNCS, vol. 5665, pp. 1.24. Springer, Heidelberg (2009)

[3] J. Chen and A. Miyaji. How to Find Short RC4 Colliding Key Pairs, The 14th Information Security Conference, ISC 2011, Lecture Notes in Computer Science, 7001 (2011), Springer-Verlag, 32-46.

[4] J. Chen and A. Miyaji. Generalized Analysis on Key Collisions of Stream Cipher RC4, IEICE Trans., Fundamentals. Vol.E94-A,No.11,pp.-,Nov. 2011. To appear.

[5] Chen, J., Miyaji, A.: Generalized RC4 Key Collisions and Hash Collisions. In: J.A.Garay., R.De Prisco (eds.): SCN 2010. LNCS, vol. 6280, pp.73-87, Springer, Heidelberg (2010)

[6] Chen, J., Miyaji, A.: A New Class of RC4 Colliding Key Pairs With Greater Hamming Distance. In: et al. (eds.): ISPEC 2010, LNCS, vol. 6047, pp.30-44, Springer, Heidelberg (2010).

[7] Chen, J., Miyaji, A.: A New Practical Key Recovery Attack on the Stream Cipher RC4 Under Related-Key Model. In: et al. (eds.): Inscrypt 2010, LNCS, vol. 6584, pp.62-76, Springer, Heidelberg (2011).

[8] Sepehrdad, P., Vaudenay, S., Vuagnoux, M: Statistical Attack on RC4. In: Paterson, K. (eds.): Eurocrypt 2011. LNCS, vol. 6632, pp.343-363, Springer, Heidelberg (2011)

[9] Sepehrdad, P., Vaudenay, S., Vuagnoux, M: Discovery and Exploitation of New Biases in RC4. In: Biryukov, A., Gong, G., Stinson, D.(eds.): SAC2010. LNCS, vol. 6544, pp.74-91, Springer, Heidelberg (2011)

[10] Subhamoy , M., Goutam , P., Sourav , S: Attack on Broadcast RC4 Revisited . In: .(eds.): FSE2011. LNCS, vol. 6733, pp. 199-217, Springer, Heidelberg (2011).

[11] Wang, X., and Yu, H. How to break MD5 and other hash functions. In: Advances in Cryptology - EUROCRYPT 2005, LNCS, vol.3494, pp. 19-35, Springer, Heidelberg (2005)

[12] Chang, D., Gupta, K.C., Nandi, M.: RC4-Hash: A New Hash Function Based on RC4. In: Progress in Cryptology - INDOCRYPT 2006, LNCS, vol.4329, pp. 80-94, Springer, Heidelberg (2006)

[13] Indesteege, S., Preneel, B.: Collision for RC4-Hash. In: 11th International Conference on Information Security. LNCS, vol. 5222, pp. 355-366. Springer, Heidelberg (2008)