Key Collisions of the RC4 Stream Cipher

Mitsuru Matsui

Information Technology R&D Center
Mitsubishi Electric Corporation
5-1-1 Ofuna Kamakura Kanagawa 247-8501, Japan
`Matsui.Mitsuru@ab.MitsubishiElectric.co.jp`

Abstract. This paper studies "colliding keys" of RC4 that create the same initial state and hence generate the same pseudo-random byte stream. It is easy to see that RC4 has colliding keys when its key size is very large, but it was unknown whether such key collisions exist for shorter key sizes. We present a new state transition sequence of the key scheduling algorithm for a related key pair of an arbitrary fixed length that can lead to key collisions and show as an example a 24-byte colliding key pair. We also demonstrate that it is very likely that RC4 has a colliding key pair even if its key size is less than 20 bytes. This result is remarkable in that the number of possible initial states of RC4 reaches $256! \approx 2^{1684}$. In addition we present a 20-byte near-colliding key pair whose 256-byte initial state arrays differ at only two byte positions.

1 Introduction

The RC4 stream cipher is one of the most widely-used real-world cryptosystems. Since its development in 1980's, RC4 has been used in various software applications and standard protocols such as Microsoft Office, Secure Socket Layer (SSL), Wired Equivalent Privacy (WEP). The architecture of RC4 is extremely simple and its encryption speed is remarkably fast. It has been suitable for not only PC applications but also embedding environments.

RC4 accepts a secret key whose length is 1 to 256 bytes, where a typical key size is 5 bytes (due to an old export regulation), 13 bytes (in the WEP encryption) or 16 bytes. It consists of two parts; the key scheduling algorithm (KSA) and the pseudo-random generating algorithm (PRGA). The KSA creates the 256-byte initial state from the secret key, and the PRGA generates pseudo-random byte stream from the initial state. Either of the algorithms can be described unambiguously in only a few lines. Due to this simplicity and wide applicability, a vast amount of efforts have been made for cryptanalysing RC4 since its specification was made public on Internet in 1994 [1].

As early as one year after the specification of RC4 was publicly available, it was pointed out by Roos [10] that the initial few bytes of its output stream are strongly correlated with the key. This observation was later extended in various ways. Mantin and Shamir [8] mounted a distinguishing attack of RC4 under the "strong distinguisher scenario" that an attacker can obtain many short output streams using random and unrelated keys. Paul and Preneel [9] successfully demonstrated a distinguisher that requires a total of 2^{25} output bytes (2 bytes/key $\times$ 2^{24} keys).

For cryptanalysis of a real-world application using RC4, Fluhrer, Mantin and Shamir [3] showed an attack of WEP for the first time in 2001. They found that partial information of the key of RC4 gives non-negligible information of its output bytes. This fact was efficiently used for mounting a passive attack, since the initial vector of WEP is packet-variable and embedded in the key of RC4. It was recently improved by Klein [6], and finally in 2007 Tews, Weinmann and Pyskin [11] and Vaudenay and Vuagnoux [12] independently demonstrated a very practical key recovery attack of the 104-bit WEP.

For another attacks, Fluhrer and McGrew [4] presented a distinguishing attack of RC4 using $2^{30.6}$ output bytes generated by a single key, and Mantin [7] successfully reduced the output stream size required for the successful attack down to $2^{26.5}$ bytes and also showed how to predict output bits. More recently Biham and Carmeli [2] concentrated on the key scheduling algorithm of RC4 and discussed how to recover a secret key from a given internal state.

This paper studies another type of weakness of the key scheduling algorithm of RC4; that is, existence of secret keys that create the same initial state and hence generate the same pseudo-random byte stream, which we call "colliding keys". It had been already pointed out by Grosul and Wallach [5] in 2000 that RC4 has related-key key pairs that generate substantially similar hundred output bytes when the key size is close to the full 256 bytes. In this paper we explore much stronger key collisions in a shorter key size.

Since the total number of possible initial states of RC4 reaches $256! \approx 2^{1684}$, RC4 must have colliding keys if its key size exceeds $\lfloor 1684/8 \rfloor = 210$ bytes. Moreover, due to the birthday paradox, it is not surprising that RC4 has colliding keys when its key size is $\lfloor (1684/2)/8 \rfloor = 105$ (or more) bytes. However, it was unknown whether colliding keys exist in a shorter key size. The contribution of this paper is to give a positive answer to this problem.

In this paper, we begin by demonstrating a specific example of a colliding 64-byte key pair, whose internal 256-byte state arrays differ at most two byte positions in any step $\mathtt{i}$ ($0 \le \mathtt{i} \le 255$) of the key scheduling algorithm. Then by generalizing this example, we show a state transition pattern that is applicable to a key pair of an arbitrary fixed length and estimate the probability that such pattern actually takes place for randomly given key pairs with a fixed difference.

We have confirmed that our probability estimation mostly agrees with computer experimental results when the key size is around 32 bytes or more. We also demonstrate that it is very likely that RC4 has a colliding key pair even when its key size is much shorter, say 20 bytes, while the minimal key size that has colliding keys is still an open problem.

We further extend our observation to a near-colliding key pair; that is, a key pair whose initial states differ at exactly two positions. In the same way as the key collision case, we show a state transition pattern of a key pair of an arbitrary length that can lead to a near-collision and analyze the probability that the pattern actually takes place. Finally we illustrate our (near-)colliding key pair search algorithm, which has successfully found a 24-byte colliding key pair and a 20-byte near-colliding key pair.

2 The RC4 Stream Cipher

RC4 is a stream cipher with a secret key whose length is 1 to 256 bytes. We define `k` and `K` as the target key size in byte and the `k`-byte secret key, respectively, and `K[0]...K[k-1]` denote `k` key bytes. For arbitrary `i`, `K[i]` means `K[i mod k]`. RC4 consists of the key scheduling algorithm (KSA), which creates the 256-byte initial state array `S[0]...S[255]` from the secret key, and the pseudo-random generating algorithm (PRGA), which generates byte sequence `Z[0]...Z[L-1]` of arbitrary length `L` from the initial state.

This paper discusses colliding key pairs of RC4 that create the same initial state. Hence only the key scheduling algorithm, described below in the syntax of C language, is relevant. When necessary, we use notations `S1/S2` and `j1/j2` for the state array and the state index for the first and second key `K1/K2`, respectively. The goal of this paper is to find, or show a strong evidence of existence of, key pair `K1` and `K2` such that the corresponding state arrays `S1` and `S2` are completely same at the end of the key scheduling algorithm.

We define "the distance of a key pair at step `i`" as the number of distinct bytes between `S1` and `S2` at the bottom (i.e. after the swap) of the `i`-th iteration in the state randomization loop. If the distance of key pair `K1` and `K2` is 0 at step 255, then they are a colliding key pair. This paper will deal with key pairs whose distance is at most 2 at any step `i` (0≤`i`≤255).

```
[The Key Scheduling Algorithm of RC4]
    /* State Initialization */
    for(i=0; i<256; i++){
        S[i] = i;
    }
    /* State Randomization */
    j=0;  /* Index j */
    for(i=0; i<256; i++){
        /* Step i */
        j = (j + S[i] + K[i % k]) & 0xff;
        SWAP(S[i], S[j]);
    }

[The Pseudo-Random Generator Algorithm of RC4]
    i = 0;
    j = 0;
    for(n=0; n<L; n++){
        i = (i + 1) & 0xff;
        j = (j + S[i]) & 0xff;
        SWAP(S[i], S[j]);
        Z[n] = S[(S[i] + S[j]) & 0xff];
    }
```

3 An Example: How It Works

In this section we demonstrate a specific example of a colliding key pair and explain how its collision is created in a step-by-step fashion. In fact, this simple example contains all tricks that we need for finding colliding key pairs of an arbitrary length in later sections. The following is a 64-byte key pair, written in hexadecimal form, which differs at only one byte position, K1[2]≠K2[2]. These two keys create the same initial state, and hence they are cryptographically indistinguishable.

```
K1 = 45 3d 7d 3d c9 45 57 12 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

K2 = 45 3d 7e 3d c9 45 57 12 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Table 1 shows internal values in the state randomization loop for each key, where “@XX” denotes an address of the state array. For example, at the end of step i=02 (i.e. after the swap), j1=02, j2=03 and the state arrays differ at exactly two positions 02 and 03. At these positions, S1[02]=02, S2[02]=03, S1[03]=03 and S2[03]=02.

i	K1 K2	j1 j2	differences between S1 and S2
00	45 45	45 45	S1=S2
01	3d 3d	83 83	S1=S2
02	7d 7e	02 03	@02(S1=02 S2=03) @03(S1=03 S2=02)
03	3d 3d	42 42	@02(S1=02 S2=03) @42(S1=03 S2=02)
04-40			@02(S1=02 S2=03) @42(S1=03 S2=02)
41	3d 3d	02 02	@41(S1=02 S2=03) @42(S1=03 S2=02)
42	7d 7e	82 82	@41(S1=02 S2=03) @82(S1=03 S2=02)
43-80			@41(S1=02 S2=03) @82(S1=03 S2=02)
81	3d 3d	41 41	@81(S1=02 S2=03) @82(S1=03 S2=02)
82	7d 7e	c1 c1	@81(S1=02 S2=03) @c1(S1=03 S2=02)
83-bf			@81(S1=02 S2=03) @c1(S1=03 S2=02)
c0	45 45	81 81	@c0(S1=02 S2=03) @c1(S1=03 S2=02)
c1	3d 3d	c1 c0	S1=S2
c2	7d 7e	54 54	S1=S2
c3-ff			S1=S2

Table 1: The State Transition Pattern of the 64-byte Key Pair.

Now let us take a look at this transition sequence more closely to see why/how the distance of this key pair remains 0 or 2 at any step.

i	
00,01	K1[i]=K2[i]. Hence S1=S2 at these steps.
02	K1[i]=K2[i]-1 (1st time). Hence j1=j2-1.
	Also since i=j1, S1 and S2 differ at j1=02 and j2=03 only.
	(If i≠j1, S1 and S2 differ at three positions.)
03	K1[i]=K2[i] and S1[i]=S2[i]+1. Hence j1=j2 again.
	Also since j1=j2=42, S1/S2[i] is swapped with S1/S2[42].
	Now S1 and S2 differ at 02 and 42. (Note that K1[42]=K2[42]-1.)
04-40	K1[i]=K2[i]. Hence j1=j2.
	Also since j1(=j2)≠02 and j1(=j2)≠42, S1 and S2 differ at 02 and 42.
41	Since j1=j2=02, S1/S2[i] is swapped with S1/S2[02].
	Now S1 and S2 differ at 41 and 42.
42	K1[i]=K2[i]-1 (2nd time). Since S1[i]=S2[i]+1, j1=j2.
	Also since j1=j2=82, S1/S2[i] is swapped with S1/S2[82].
	Now S1 and S2 differ at 41 and 82. (Note that K1[82]=K2[82]-1.)
43-80	K1[i]=K2[i]. Hence j1=j2.
	Also since j1(=j2)≠41 and j1(=j2)≠82, S1 and S2 differ at 41 and 82.
81	Since j1=j2=41, S1/S2[i] is swapped with S1/S2[41].
	Now S1 and S2 differ at 81 and 82.
82	K1[i]=K2[i]-1 (3nd time). Since S1[i]=S2[i]+1, j1=j2.
	Also since j1=j2=c1, S1/S2[i] is swapped with S1/S2[c1].
	Now S1 and S2 differ at 81 and c1.
	(Note that this time the swapped address is c1, not c2).
83-bf	K1[i]=K2[i]. Hence j1=j2.
	Also since j1(=j2)≠81 and j1(=j2)≠c1, S1 and S2 differ at 81 and c1.
c0	Since j1=81, S1/S2[i] is swapped with S1/S2[81].
	Now S1 and S2 differ at c0 and c1.
c1	Since j1=c1 and j2=c0, the differences between S1/S2 disappear.
	Now S1 is the same as S2.
c2	K1[i]=K2[i]-1 (4th time). Since j1=j2+1 at the previous step,
	now j1=j2. Hence S1=S2.
c3-ff	K1[i]=K2[i]. Hence j1=j2 and S1=S2.

Table 2: The State Transition Details of the 64-bit Key Pair.

We have K1[i]=K2[i]-1 four times. For i=02, this difference is absorbed at the next step because it causes j1=j2-1 while S1[3]=S2[3]+1. Note that the relation K1[2]=K2[2]-1 is essential for this example. For i=42 and i=82, j1 remains the same as j2 because S1[42]=S2[42]+1 and S1[82]=S2[82]+1, respectively.

The last time i=c2 is a bit tricky; at two steps before i=c0, S1 and S2 differ at c0 and c1. Moreover at i=c1, j1=c0 and j2=c1 and hence the differences of S1 and S2 disappear at this step. The remaining difference is between j1 and j2, which is finally cancelled at i=c2 due to the relation K1[c2]=K2[c2]-1.

4 General Collision Sequence

In this section we extend the sequence shown in the previous section to a key pair of an arbitrary length and give a general transition pattern that can lead to the same initial state. We also estimate the probability that the key collision actually occurs for randomly given key pairs with a fixed difference.

We here consider `k`-byte key pair `K1` and `K2` such that `K1[i]=K2[i]-1` if `i=d` ($0 \leq d < k$) and `K1[i]=K2[i]` otherwise. We also define `n` as $\lfloor$`(256+k-1-d)/k`$\rfloor$. Then `K1[i]`$\neq$`K2[i]` takes place exactly `n` times in the state randomization loop.

Table 3 illustrates details of the state transition pattern for given `k` and `d` with estimated probability, where "-" allows any address (a "don't-care" value) and "x" at step `i=d+(n-1)k-2` is the value such that `S1[x]=d` and `S2[x]=d+1` at step `i=d+(n-1)k-3`.

Step	Internal State Values	Estimated Prob.
(a) `i=0...d-1`	`j1(=j2)`$\neq$ `d, j1(=j2)`$\neq$`d+1`	$(254/256)^{d}$ or
	`S1=S2`	$(254/256)^{d-1}$
(b) `i=d`	`j1=d, j2=d+1`	1/256
	`@d(S1=d S2=d+1)`	
	`@d+1(S1=d+1 S2=d)`	
(c) `i=d+1`	`j1=j2=d+k`	1/256
	`@d(S1=d S2=d+1)`	
	`@d+k(S1=d+1 S2=d)`	
(d) `i=d+2...d+k-1`	`j1(=j2)`$\neq$`d+k`	$(255/256)^{k-2}$
	`@-(S1=d S2=d+1)`	
	`@d+k(S1=d+1 S2=d)`	
Repeat steps (e) and (f) for `m=1...n-3`.		
(e) `i=d+mk`	`j1=j2=d+(m+1)k`	1/256
	`@-(S1=d S2=d+1)`	
	`@d+(m+1)k(S1=d+1 S2=d)`	
(f) `i=d+mk+1...`	`j1(=j2)`$\neq$`d+(m+1)k`	$(255/256)^{k-2}$
`d+(m+1)k-1`	`@-(S1=d S2=d+1)`	
	`@d+(m+1)k(S1=d+1 S2=d)`	
(g) `i=d+(n-2)k`	`j1=j2=d+(n-1)k-1`	1/256
	`@-(S1=d S2=d+1)`	
	`@d+(n-1)k-1(S1=d+1 S2=d)`	
(h) `i=d+(n-2)k+1...`	`j1(=j2)`$\neq$`d+(n-1)k-1`	$(255/256)^{k-4}$
`d+(n-1)k-3`	`@-(S1=d S2=d+1)`	
	`@d+(n-1)k-1(S1=d+1 S2=d)`	
(i) `i=d+(n-1)k-2`	`j1=j2=x`	1/256
	`@d+(n-1)k-2(S1=d S2=d+1)`	
	`@d+(n-1)k-1(S1=d+1 S2=d)`	
(j) `i=d+(n-1)k-1`	`j1=d+(n-1)k-1, j2=d+(n-1)k-2`	1/256
(k) `i=d+(n-1)k...255`	`S1=S2`	1

Table 3: The State Transition Pattern of a `k`-byte Colliding Key Pair.

The probability that event (a) takes place is $(254/256)^{d-1}$ only if `d=k-1`, and $(254/256)^d$ otherwise, because `K1[0]=K2[0]=0` when `d=k-1` (also see below). Now assuming that the state index `j` is uniformly random at all steps, the probability that this transition sequence actually takes place for randomly given key pairs with the fixed difference is

$$ColProb(k,d) = \begin{cases} (254/256)^d (255/256)^{(n-1)(k-2)-2} (1/256)^{n+2} & (d \neq k-1) \\ (254/256)^{d-1} (255/256)^{(n-1)(k-2)-2} (1/256)^{n+2} & (d = k-1) \end{cases}.$$

This probability is actually very close to $1/e(1/256)^{n+2}$, which depends on n only. Of course `j`'s are not uniformly random in practice, and therefore this probability estimation is not necessarily correct. However this (very intuitive) assumption mostly agrees with our computer experimental results when the length of the key is around 32 bytes or more.

The following is one of the 43-byte colliding key pairs that we found (43 is the minimal `k` such that `⌊256/k⌋=5`), and its experimentally observed conditional probability of each event. These keys differ at the last byte, hence `d=42`. Note that successful events (a) and (b) effectively determine one-byte information of the key. In other words, `K[0]...K[d-1]` uniquely determines `K[d]`. Moreover for meeting events (b) and (c), we must have `K1[0]=K2[0]=0`, more generally `K1[d+1]=K2[d+1]=k-d-1`. We hence do not have to "wait for" events (b) and (c) in searching for key collisions.

Table 4 shows that our experimental results, where we obtained two 43-byte colliding key pairs from $2^{41.5}$ candidate pairs, perfectly agree with our probability estimation.

```
K1 = 00 6d 41 8b 95 46 07 a4 87 8d 69 d7 bc bc c4 70
     4a 3b ed 94 34 50 04 68 4d 4f 2e 30 c1 6e 20 a8
     bf 80 b6 ae df ae 43 56 0a 80 e7

K2 = 00 6d 41 8b 95 46 07 a4 87 8d 69 d7 bc bc c4 70
     4a 3b ed 94 34 50 04 68 4d 4f 2e 30 c1 6e 20 a8
     bf 80 b6 ae df ae 43 56 0a 80 e8
```

Event	Estimated Prob.	Measured Prob.
(a)	$0.725010=(254/256)^{41}$	0.725010 = 2272363208729/3134252384256
(d)	$0.851743=(255/256)^{41}$	0.851638 = 1935231636873/2272363208729
(e-1)	0.003906=1/256	0.004120 = 7973306038/1935231636873
(f-1)	$0.851743=(255/256)^{41}$	0.851706 = 6790914484/7973306038
(e-2)	0.003906=1/256	0.003933 = 26707884/6790914484
(f-2)	$0.851743=(255/256)^{41}$	0.851605 = 22744564/26707884
(g)	0.003906=1/256	0.003891 = 88498/22744564
(h)	$0.858437=(255/256)^{39}$	0.858336 = 75961/88498
(i)	0.003906=1/256	0.003976 = 302/75961
(j)	0.003906=1/256	0.006623 = 2/302

Table 4: Experimental Results of Finding 43-byte Colliding Key Pairs.

Also, assuming again that $ColProb(k,d)$ is correct for any `k` and `d`, the expected number of `k`-byte colliding key pairs out of a total of 2^{8k} keys is

$$ColPairs(k) = 2^{8k} \times \sum_{d=0}^{k-1} ColProb(k,d).$$

Table 5 is a list of $log_2(ColPairs(k))$ for `k=17...64`. This table clearly shows that key collisions can exist in much shorter key length. On the other hand, $ColProb(k,d)$ does not always agree with our experimental results when `k` is small, say 30 or less, because probabilistic dependency of `j`'s cannot be ignored in such range. It seems that more detailed probability analysis is needed for accurately estimating the density of colliding key pairs in a small key size.

`k`	Pairs	`k`	Pairs	`k`	Pairs	`k`	Pairs	`k`	Pairs
15	-	25	106.9	35	211.2	45	306.4	55	394.9
16	-	26	120.7	36	219.6	46	314.9	56	403.2
17	2.7	27	130.5	37	232.3	47	323.3	57	411.5
18	18.5	28	139.2	38	242.0	48	331.6	58	419.7
19	34.0	29	153.0	39	250.7	49	339.9	59	427.9
20	48.7	30	162.5	40	259.2	50	348.1	60	436.1
21	58.8	31	171.2	41	267.6	51	356.3	61	444.2
22	73.7	32	179.7	42	275.9	52	368.7	62	452.4
23	83.0	33	193.7	43	287.7	53	377.8	63	460.5
24	97.7	34	202.6	44	297.6	54	386.4	64	468.6

Table 5: List of $log_2(ColPairs(k))$ for `k=17...64`.

5 Near-Collision Sequence

This section gives another extension of section 2, a near-colliding key pair whose initial state arrays `S1` and `S2` differ at exactly two positions. We use the same notations as in the previous section.

Table 6 shows the details of our state transition pattern that can lead to a near-collision, which is the same as the collision case except the last part. "-" allows any address and "x" is any value equal to or less than `d+(n-1)k`. Note that if `x` exceeds `d+(n-1)k`, the distance between `S1` and `S2` exceeds two in (h).

Step	Internal State Values	Approx. Prob.

(a) `i=0...d-1`	`j1(=j2)≠d, j1(=j2)≠d+1`
	`S1=S2`
		$(254/256)^d$ or
		$(254/256)^{d-1}$

(b) `i=d`	`j1=d, j2=d+1`
	`@d(S1=d S2=d+1)`
	`@d+1(S1=d+1 S2=d)`
		1/256

(c) `i=d+1`	`j1=j2=d+k`
	`@d(S1=d S2=d+1)`
	`@d+k(S1=d+1 S2=d)`
		1/256

(d) `i=d+2...d+k-1`	`j1(=j2)≠d+k`
	`@-(S1=d S2=d+1)`
	`@d+k(S1=d+1 S2=d)`
		$(255/256)^{k-2}$

Repeat steps (e) and (f) for `m=1..n-2`		.

(e) `i=d+mk`	`j1=j2=d+(m+1)k`
	`@-(S1=d S2=d+1)`
	`@d+(m+1)k(S1=d+1 S2=d)`
		1/256

(f) `i=d+mk+1...`
`d+(m+1)k-1`
	`j1(=j2)`$\neq$`d+(m+1)k`
	`@-(S1=d S2=d+1)`
	`@d+(m+1)k(S1=d+1 S2=d)`
		$(255/256)^{k-2}$

(g) `i=d+(n-1)k`	`j1=x, j2=x`
	`@-(S1=d S2=d+1)`
	`@x(S1=d+1 S2=d)`
		`(d+(n-1)k+1)`/256

(h) `i=d+(n-1)k+1...255`	`@-(S1=d S2=d+1)`
	`@-(S1=d+1 S2=d)`
		1

Table 6: The State Transition Pattern of a `k`-byte Near-Colliding Key Pair.

The expected probability that this transition pattern actually takes place is

$$NearColProb(k,d) = \begin{cases} (254/256)^{d}(255/256)^{(n-1)(k-2)}(1/256)^{n}(d+(n-1)k+1)/256 & (d \neq k-1) \\ (254/256)^{d-1}(255/256)^{(n-1)(k-2)}(1/256)^{n}(d+(n-1)k+1)/256 & (d = k-1) \end{cases}.$$

This probability is actually very close to $1/e(1/256)^n$, which depends on n only, and hence roughly it holds that $NearColProb(k,d) = 2^{16}ColProb(k,d)$. The following is a 33-byte near-colliding key pair that we found (33 is the minimal `k` such that $\lfloor$`256/k`$\rfloor$`=7`), and its experimentally observed conditional probability of each event. These two keys differ at the last byte, hence `d=32`.

```
K1 = 00 3d 3f 08 4f cd d8 f1 11 8c 83 80 1e 7f 5b c3
     d9 60 e2 c8 22 88 3c bc 56 2c 22 d2 b3 d9 ab d9 41

K2 = 00 3d 3f 08 4f cd d8 f1 11 8c 83 80 1e 7f 5b c3
     d9 60 e2 c8 22 88 3c bc 56 2c 22 d2 b3 d9 ab d9 42
```

Event	Estimated Prob.	Measured Prob.
(a)	$0.784163{=}(254/256)^{31}$	0.784163=15852958662942/20216411062272
(d)	$0.885741{=}(255/256)^{31}$	0.885471=14037333620475/15852958662942
(e-1)	0.003906=1/256	0.001946=27312181761/14037333620475
(f-1)	$0.885741{=}(255/256)^{31}$	0.885555=24186440984/27312181761
(e-2)	0.003906=1/256	0.003954=95628579/24186440984
(f-2)	$0.885741{=}(255/256)^{31}$	0.885499=84679053/95628579
(e-3)	0.003906=1/256	0.003930=332809/84679053
(f-3)	$0.885741{=}(255/256)^{31}$	0.886160=885741/332809
(e-4)	0.003906=1/256	0.003774=1113/294922
(f-4)	$0.882281{=}(255/256)^{31}$	0.881402=981/1113
(e-5)	0.003906=1/256	0.005097=5/981
(f-5)	$0.882281{=}(255/256)^{31}$	0.800000=4/5
(g)	0.902344=231/256	1.000000=4/4

Table 7: Experimental Results of Finding 33-byte Near-colliding Key Pairs.

Table 7 shows that our experimental results, where we obtained four 33-byte near-colliding key pairs from $2^{44.2}$ candidates, mostly agree with our probability estimation. Now the expected number of `k`-byte near-colliding key pairs out of a total of 2^{8k} keys is

$$NearColPairs(k) = 2^{8k} \times \sum_{d=0}^{k-1} NearColProb(k, d).$$

Table 8 is a list of $log_2(NearColPairs(k))$ for `k=16...64`. This table clearly shows that near-key collisions can also exist in much shorter key length. However, again, more detailed probabilistic analysis is needed for accurately estimating the density of near-colliding key pairs in a small key size.

k	Pairs	k	Pairs	k	Pairs	k	Pairs	k	Pairs
15	-	25	122.8	35	227.1	45	322.2	55	410.6
16	2.7	26	136.5	36	235.5	46	330.7	56	418.9
17	18.7	27	146.3	37	248.1	47	339.1	57	427.2
18	34.5	28	155.1	38	257.8	48	347.4	58	435.4
19	49.9	29	168.8	39	266.5	49	355.7	59	443.6
20	64.6	30	178.3	40	275.0	50	363.9	60	451.8
21	74.7	31	187.1	41	283.4	51	372.1	61	460.0
22	89.6	32	195.5	42	291.7	52	384.4	62	468.1
23	98.9	33	209.5	43	303.5	53	393.5	63	476.3
24	113.6	34	218.5	44	313.4	54	402.1	64	484.4

Table 8: List of $log_2(NearColPairs(k))$ for `k=16...64`.

6 Faster Collision Search

In previous sections we searched for colliding and near-colliding key pairs in a very simple fashion, — checking transition patterns step-by-step and if fails, restarting the search with another random candidate —, whose primary purpose was to confirm theoretical claims experimentally. In this section we explore a faster method for finding (near-)colliding key pairs for smaller key sizes.

We here try to find (near-)colliding key pairs by checking distance between two keys at each step but not checking our transition patters. We now define `MaxColStep(K1,K2)` as maximal step `i` such that distance between `K1` and `K2` is at most two at all steps up to step `i`. For a colliding or near-colliding key pair, `MaxColStep(K1,K2)=255`. Also for given key `K` and 0 ≤`x,y`<256, we define a slightly modified key `K<x,y>` as

```
K<x,y>[x]   = K[x]  +y
K<x,y>[x+1] = K[x+1]-y
K<x,y>[i]   = K[i]      if i is not x or x+1.
```

It is naturally expected that the given key `K` and its modified keys `K<x,y>` are likely to create similar initial states. Hence when the check fails for `K1` and `K2`, we can try `K1<x,y>` and `K2<x,y>`, which are likely to be a better pair, instead of rewinding and restarting the search with another random candidate. Which pair is better (i.e. closer to an actual (near-)collision) can be measured by the `MaxColStep` function. These observations lead to the following simple recursive search algorithm,

Collision Search Algorithm: Generate a random key pair `K1` and `K2` which differ at position `d` by one. Set `K1[d+1]=K2[d+1]=k-d-1` (see section 4) and call `Search(K1,K2)`. Repeat this until a (near-)colliding key pair is found:

```
Search(K1,K2)
S = MaxColStep(K1,K2)
if S = 255 then stop (found a (near-)collision!) or return (to find more)
MaxS = $max_{x,y}$ MaxColStep(K1<x,y>,K2<x,y>)
if MaxS $\leq$ S then return
C = 0
For all x and y, do the following:
if MaxColStep(K1<x,y>,K2<x,y>) = MaxS
call Search(K1<x,y>,K2<x,y>)
C = C + 1
if C = MaxC then return
endif
return
```

where `x` runs from 0 to 255 except `d` and `d+1` (for not changing `K1[d+1]/K2[d+1]`), and `y` runs from 1 to 255. This algorithm finds mostly near-colliding key pairs, but may also find colliding key pairs (if we are lucky enough or patient enough). In fact, using this algorithm we reached a 24-byte colliding key pair and a 20-byte near-colliding key pair as follows:

```
[24-byte Colliding Key Pair]

K1 = 00 42 CE D3 DF DD B6 9D 41 3D BD 3A B1 16 5A 33
     ED A2 CD 1F E2 8C 01 76

K2 = 00 42 CE D3 DF DD B6 9D 41 3D BD 3A B1 16 5A 33
     ED A2 CD 1F E2 8C 01 77

[20-byte Near Colliding Key Pair]

K1 = 00 73 2F 6A 01 37 89 C5 15 49 9A 55 98 54 D7 53 4E F6 4F DC

K2 = 00 73 2F 6A 01 37 89 C5 15 49 9A 55 98 54 D7 53 4E F6 4F DD
```

`MaxC` is a pre-defined value. In our experiments, the search worked efficiently when `MaxC` is around 10, and the maximal depth of recursive calls was less than

20. Obviously this algorithm has a room for improvement. For instance, it can pick up the same near-colliding key pair twice or more; that is, the search contains some redundancy. Also another evaluation function, instead of `MaxColStep`, or another key modification is a possibility. Studying a better collision search method seems an interesting future topic.

7 Concluding Remarks

This paper explored key collisions of the RC4 stream cipher. We presented a 24-byte colliding key pair and a 20-byte near-colliding key pair, and demonstrated that our probabilistic analysis well agrees with experimental results when the key size is around 32 bytes. It seems now very likely that RC4 has colliding keys in even smaller key length, say less than 20 bytes.

While tables 5 and 8 suggest an existence of 17-byte colliding key pairs and 16-byte near-colliding key pairs, respectively, further research is needed for more accurately estimating $ColProb(k, d)$ and $NearColProb(k, d)$ in such small k. In fact, we have already seen that the observed probability of event (e-1) in table 7 was much smaller than the expected probability 1/256. This kind of phenomenons (much larger or smaller than our estimation) frequently appears when k is small.

It might not be very easy to derive a simple formula of $ColProb(k, d)$ and $NearColProb(k, d)$ applicable to all k and d. As far as we know, this is the first paper that went deep into key collisions of RC4. We hope that our observation will lead to further study of this direction.

8 Acknowledgements

The author would like to thank an anonymous reviewer for suggesting the fast search algorithm, which has been further improved by the author and included in section 6. Another anonymous reviewer has pointed out that $ColProb(k, d)$ and $NearColProb(k, d)$ can approximate to $1/e(1/256)^{n+2}$ and $1/e(1/256)^{n}$, respectively, which makes things much clearer. The author is grateful to all reviewers for their valuable and constructive comments.

References

[1] Anonymous: RC4 Source Code. CypherPunks mailing list (September 9, 1994). Available at `http://cypherpunks.venona.com/date/1994/09/msg00304.html`. Also at `http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0`.

[2] Biham, E., Carmeli, Y.: Efficient Reconstruction of RC4 Keys from Internal States. Proceedings of Fast Software Encryption 2008, LNCS 5086, pp. 270-288, Springer-Verlag (2008).

[3] Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. Proceedings of Selected Areas in Cryptography 2001, LNCS 2259, pp. 1-24, Springer-Verlag (2001).

[4] Fluhrer, S., McGrew, D.: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of Fast Software Encryption 2000, LNCS 1978, pp. 19-30, Springer-Verlag (2000).

[5] Grosul, A.L, Wallach, D.S.: A Related-Key Cryptanalysis of RC4. Technical Report TR-00-358, Department of Computer Science, Rice University, (2000). Available at `http://cohesion.rice.edu/engineering/computerscience/tr/TR_Download.cfm?SDID=126`.

[6] Klein, A.: Attacks on the RC4 Stream Cipher. Designs, Codes and Cryptography, Vol. 48-3, pp.269-286, Springer-Verlag (2008).

[7] Mantin, I.: Predicting and Distinguishing Attacks on RC4 Keystream Generator. Advances in Cryptology, Proceedings of Eurocrypt 2005, LNCS 3494, pp. 491-506, Springer-Verlag (2005).

[8] Mantin, I., Shamir, A.: A Practical Attack on Broadcast RC4. Proceedings of Fast Software Encryption 2001, LNCS 2355, pp. 152-164, Springer-Verlag (2001).

[9] Paul, S., Preneel, B.: A New Weakness in the RC4 Keystream Generator and an Approach to Improve Security of the Cipher. Proceedings of Fast Software Encryption 2004, LNCS 3017, pp. 245-259, Springer-Verlag (2004).

[10] Roos, A.: A Class of Weak Keys in the RC4 Stream Cipher (1995). Available at `http://marcel.wanda.ch/Archive/WeakKeys`.

[11] Tews, E., Weinmann, R.P, Pyshkin, A.: Breaking 104 Bit WEP in Less than 60 Seconds, Proceedings of Workshop on Information Security Applications 2007, LNCS 4867, pp. 188-202, Springer-Verlag (2007)

[12] Vaudenay, S., Vuagnoux, M.: Passive-Only Key Recovery Attacks on RC4. Proceedings of Selected Areas in Cryptography 2007, LNCS 4876, pp. 344-359, Springer-Verlag (2007).