A New Practical Key Recovery Attack on the Stream Cipher RC4 Under Related-Key Model

Jiageng Chen * and Atsuko Miyaji**

School of Information Science,
Japan Advanced Institute of Science and Technology,
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
{jg-chen, miyaji}@jaist.ac.jp

Abstract. A new key recovery attack under related-key model on RC4 is presented in this paper. This novel attack is based on the property that RC4 can generate a large amount of colliding key pairs. By making use of this property, we are able to recover any random key in practical time when the length of the key is large under a new proposed related key model. Differing from the attack against WEP, neither the knowledge of the IVs nor the keystream outputs are required. Also compared with some recent key recovery attacks, which assume that the attacker knows the S-Box after KSA algorithm and can only recover very short keys (5 bytes) efficiently, our attack works very well for keys with larger size. We give the theoretical proof for the complexity of our attack which matches with the experimental result very well. An 86-byte random secret key can be recovered in about 21.2 hours time by using a standard desktop PC. This novel attack provides us with another theoretical approach to attack WPA and WEP. Remark that our model can be used for more efficient key recovering if any new key collisions can be further discovered in the future.

Key words: RC4, KSA, Related Keys, Key Collisions, Key Recovery.

1 Introduction

The stream cipher RC4 is one of the most famous ciphers widely used in real world applications such as Microsoft Office, Secure Socket Layer (SSL), Wired Equivalent Privacy (WEP), etc. Due to its popularity and simplicity, RC4 has become a hot cryptanalysis target since its specification was made public on the Internet in 1994 [5]. More than twenty-year study on RC4 has revealed a lot of weaknesses of this cipher and a lot different attacks have been proposed since then. Generally speaking, all these attacks can be categorized into two kinds, namely, distinguishing attack and key recovery attack. This paper focuses on key recovery attack. In a distinguishing attack, the attacker tries to distinguish

* This author is supported by the Graduate Research Program.
** This work is supported by Grant-in-Aid for Scientific Research (B), 20300003.

between an output stream generated by PRGA and a random stream [7–9]. Other various general weaknesses of RC4 have been discovered in the previous works [6, 10, 11], etc. We first briefly summarize the previous key recovery attack against RC4.

The first class of this kind of attack applies to the WEP environment, where RC4 is used with a session key which is derived from a shared secret key and an Initial Value (IV). The secret key is concatenated after the IV which is transmitted unencrypted. First chosen IV attack was shown in [12]. By observing the first many keystream outputs, they recovered the secret key with high probability. Another statistical bias between the keystream output and the value of $S[j]$ was discovered in [13] and [14]. By using this bias, they can also recover the entire key in practical time, which was then improved by reducing the dependency when recovering the key bytes later in [15]. The above attacks assume that the attacker has the knowledge of the IVs and the keystream outputs.

Another kind of key recovery approach is just by observing the final S-Box after KSA algorithm [17–19]. The basic idea is that the first few bytes of the S-Box is obviously biased, which indicates a connection to the secret key. By creating equations which hold with certain probability, they try to recover the whole keys. This kind of attack works only when the key has a very small size (5 byte), and the successful probability will drop dramatically to impractical level when the key size is larger than 16 bytes.

We propose yet another approach to launch a practical key recovery attack against RC4. Our attack is based on the property that RC4 has a large amount of colliding key pairs [2, 4] especially when the key size is very large. Since the colliding key pairs of RC4 follow some specific patterns, some key information will leak if the attacker knows under which pattern the two unknown keys can achieve a collision. In our attack, the attacker is allowed to query key differentials to the KSA Oracle, which will return the S-Box differences to the attacker. If KSA Oracle returns with a (near) collision, then the attacker is able to recover some information of this tweaked key pair according to the key collision properties. And since the attacker knows the key differentials he submitted to the Oracle, he thus can trace back to recover the key based on the key differentials and the leaked key information from the tweaked key pair. Compared with the attacks against WEP, neither the knowledge of the IVs nor the keystream outputs are required. If the first hundreds keystream output bytes are discarded, which is the usual way to fix this weak point, the first kind of attack will not work while our attack will still be available. Compared with the attacks that require the knowledge of the final S-Box, our attack works efficiently for any random keys having large size with probability one, which can be seen as a complement to the second kind of previous attack.

Structure of the paper. In Section 2, we briefly describe the RC4 algorithm followed by the key collision techniques which are needed for the attack in Section 3. Section 4 describes the detailed attack starting with the description of the related key model, and then followed by the detailed techniques to recover full length keys as well as the short keys. Section 5 gives the comparison between our

attack and some of the previous attacks in complexity and probability, and also the experimental results are shown in this section. Finally, we give the conclusion in Section 6.

2 The RC4 Stream Cipher and Notations

2.1 RC4

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. In this paper, we mainly focus on KSA.

Algorithm 1. KSA

for $i = 0$ **to** $N-1$ **do**
    $S[i] \leftarrow i$
end for
$j \leftarrow 0$
for $i = 0$ **to** $N-1$ **do**
    $j \leftarrow j + S[i] + K[i \bmod l]$
    swap$(S[i], S[j])$
end for

Algorithm 2. PRGA

$i \leftarrow 0$
$j \leftarrow 0$
loop
    $i \leftarrow i + 1$
    $j \leftarrow j + S[i]$
    swap$(S[i], S[j])$
keystream byte $z_i = S[S[i] + S[j]]$
end loop

2.2 Notations

The following are the notations used in this paper.

- K: target random secret key.
- K_1, K_2: two secret keys related in some pattern, which will be described in the next section.
- $\Delta K_1^t[i](\Delta K_2^t[i])$: Differential between target key K and K_1 (K_2) at index i at attacking step t.
- $j_{1,i}(j_{2,i})$: internal state j at step i for $K_{1,t}$ ($K_{2,t}$) respectively.
- $S_{1,i}(S_{2,i})$: the S-Box at step i for $K_{1,t}$ ($K_{2,t}$) before the swap operation.
- d: the first key difference index.
- k: the lengths (bytes) of the secret keys.
- n: the number of times the differences of the keys appear during KSA. $n = \lfloor \frac{256+k-1-d}{k} \rfloor$.

3 Key Collisions of RC4

This novel attack is based on the fact that RC4 can generate a large amount of (near) colliding key pairs. Before going into detail of the attack, we briefly describe the key collision techniques here.

We call a key pair K_1 and K_2 ($K_1 \neq K_2$) a colliding key pair if after KSA, the two corresponding S-Boxes are equal to each other ($S_{1,255} = S_{2,255}$). A near colliding key pair is only different at that the final two S-Boxes need not to be totally the same. The corresponding procedures are called key collisions and near key collisions. The previous researches showed that key collisions or near key collisions can be achieved under some specific key pattern with some probability.

The key collision of RC4 has been first studied back in 2000 in [1]. They pointed out the existence of the near collisions for large size keys. First key collisions with the pattern that two keys differ at one position from each other were found in [2], where we say that those key pairs have hamming distance one. In [3], key collisions with another pattern which has hamming distance three was confirmed, and later in [4], formalized key collisions have been studied and generalized RC4 collision patterns were demonstrated in the paper. Our attack is divided into two categories based on the key collision properties, namely, key collision for full length 256-byte keys, and short keys. We describe the two key collision techniques below.

3.1 Key Collisions for Full Length 256-byte Keys

The previous researches have demonstrated that the probability for a key pair following some certain pattern to form a colliding key pair will drop as the key size gets shorter and the hamming distance gets larger. Thus for a full length key pair with length 256 bytes, it is very easy to achieve a collision under some specific pattern. We describe one here that will be used in the later attack.

Key Pattern: $K_2[d] = K_1[d]+1, K_2[d+1] = K_1[d+1]-1, K_2[d+2] = K_1[d+2]+1$

The following extra conditions during KSA are necessary for two keys with the above pattern to achieve a collision.

1. When i touches index d, we require $S_{1,d}[d+1] = S_{1,d}[d] + 1$ ($S_{2,d}[d+1] = S_{2,d}[d] + 1$).
2. At step $i = d$ after the swap, we require $j_{1,d} = d$ ($j_{2,d} = d + 1$).
3. At step $i = d + 1$ after the swap, we require $j_{1,d+1} = d + 1$ ($j_{2,d+1} = d$).

Table 1 illustrates how it works when $d = 0$. The S-Box part of the internal state shown in Table 1 demonstrates the S-Box state after the swap operation at each step of KSA.

Table 1 shows that for a key pair, which follows the previous pattern (differing from each other at indices 0, 1 and 2 in this example), will achieve a collision after round 2 (The internal states j and S-Box become the same again after round 2).

Table 1: Collisions for 256-byte full length keys when $d = 0$

Internal State								Difference
i	$K_1[i]/K_2[i]$	$j_{1,i}/j_{2,i}$	0	1	2	3	4	
0	0	0	0	1	2	3	4	$K_2[0] = K_1[0] + 1$
	1	1	1	0	2	3	4	$j_{2,0} = j_{1,0} + 1, S_1 \neq S_2$
1	0	1	0	1	2	3	4	$K_2[1] = K_1[1] - 1$
	255	0	0	1	2	3	4	$j_{2,1} = j_{1,1} - 1, S_1 = S_2$
2	X	$X+3$	0	1	$S[X+3]$	3	4	$K_2[2] = K_1[2] + 1$
	$X+1$	$X+3$	0	1	$S[X+3]$	3	4	$j_{1,2} = j_{2,2}, S_1 = S_2$

3.2 (Near) Key Collisions for Short Keys

To recover random keys with shorter size using the above collision pattern will result in a relatively high complexity time. Here we introduce another collision pattern which was first discovered in [2] and later generalized in [4].

In this pattern, two keys can differ from each other at h places, which is called to have hamming distance h. We will only use $h = 1$ in our attack. The general idea is that when i touches the different index d, two consecutive S-Box differences are expected to be generated, and one of them will be swapped to the later key difference indices. And the differences will be absorbed when i touches the last key difference in the KSA.

Key Pattern: $K_2[d] = K_1[d] + 1$

The following extra conditions during KSA are necessary for two keys with the above relations to achieve a collision.

1. When i touches index d, we require $S_{1,d}[d+1] = S_{1,d}[d] + 1$ ($S_{2,d}[d+1] = S_{2,d}[d] + 1$).
2. At step $i = d$ after the swap, we require $j_{1,d} = d$ ($j_{2,d} = d + 1$).
3. At step $i = d + 1$, we require $j_{1,d+1} = j_{2,d+1} = d + k$.
4. During steps $i = d + 2$ to $i = d + k$, we require $j_{1,i} \neq d + k$.
5. At step $i = d + p \times k$, $p = 1, ..., n - 2$, we require $j_{1,i} = j_{2,i} = i + k$.
6. During steps $i = d + p \times k + 1$ to $i = d + (p + 1) \times k$, we require $j_{1,i} \neq i + k$.
7. At step $i = d + (n - 1) \times k - 2$, we require the two S-Box differences to be at indices $d + (n - 1) \times k - 2$ and $d + (n - 1) \times k - 1$.
8. At step $i = d + (n - 1) \times k - 1$, we require $j_{1,i} = i - 1 (j_{2,i} = i)$.

Table 2 illustrates how it works when $d = 0, k = 128$.

Notice that we can achieve a near collision if we replace the conditions 7 and 8 with the condition: At step $i = d + (n - 1)k$, we require $j_{1,i} \leq i (j_{2,i} < i)$. Namely, we will have two S-Boxes with three differences between them since i will never touch those differences again.

Table 2: Collisions for short keys when $d = 0, k = 128$

Internal State				Difference
i	$K_1[i]/K_2[i]$	$j_{1,i}/j_{2,i}$	0 1 2 3 ... 126 127 128	
0	$K_1[0] = 0$	0	0 1	j, S-Box
	$K_2[0] = K_1[0] + 1 = 1$	1	1 0	
1	$K_1[1] = 127$	128	0 1	S-Box
	$K_2[1] = 127$	128	1 0	
126	$K_1[126]$	0	0 1	S-Box
	$K_2[126] = K_1[126]$	0	1 0	
127	$K_1[127]$	127	0 1	j, S-BoX
	$K_2[127] = K_1[127]$	126	0 1	
128	$K_1[0] = 0$	$S_{1,127}[128] + 127$	0 1	Same
	$K_2[0] = K_1[0] + 1 = 1$	$S_{2,127}[128] + 127$	0 1	

4 New Key Recovery Attacks

4.1 Related-Key Model

First, we define the related-key model under which the attack is carried. In this model, the attacker's goal is to recover a random secret key K and the attacker has no prior information about it. We assume that there is a KSA Oracle service which can be queried by the attacker. The attacker has the power to do the following things. He can specify key differential sets $\Delta K_1[i]$ and $\Delta K_2[i]$ for $i \in [0, 255]$ and send them to the KSA Oracle. Then the KSA Oracle will run the KSA algorithm under the new key pair $K_1 = K + \Delta K_1[i]$ and $K_2 = K_1 + \Delta K_2[i]$ and send back the S-Box differential information, namely, whether $\Delta S = 0$ to the attacker (key relations is shown in Figure 1). Repeat the previous procedure many times. The attacker tries to recover the target key K from the submitted key differentials and the corresponding S-Box differentials. Key recovery attack under the related-key model is described in Figure 2. Although recovering full size 256-byte key is different from recovering short size key, the procedure of the algorithm is the same. There are three blocks we need to explain specifically, namely the querying differentials block, observing the S-Box differentials block and recovering two key bytes block.

Querying Differentials Block. In this block, the attacker tries to query the KSA Oracle a key differential set ΔK and expecting the KSA Oracle to send back the two S-Box differentials. The attacker tries to recover two consecutive key bytes at one time and starts from the beginning of the key to the end in turn. For two target consecutive key bytes, the attacker will submit the corresponding designed key differentials.

Observing the S-Box differentials Block. In this block, the KSA Oracle will return the corresponding S-Box differentials (ΔS) to the attacker. Here the ΔS specifically means how many indices the two S-Boxes differ from each other. The attacker needs not to know the detailed values of the S-Boxes nor the

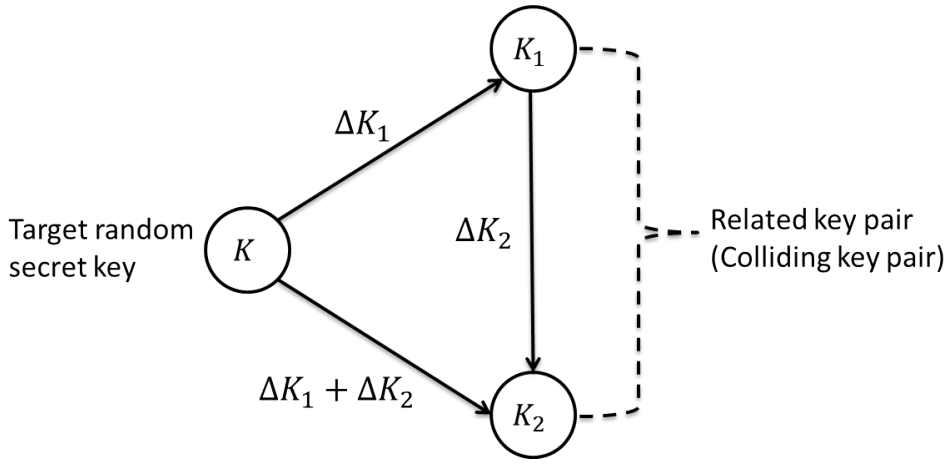


Fig. 1: Related Key Model

differentials. The attacker will decide whether to submit new key differentials to the KSA Oracle again depending on the ΔS.

Recovering two key bytes Block. In this block, the attacker has already gotten the expected ΔS after querying the key differentials several times. He will check the corresponding key collision patterns described previously to search for clues of the target key bytes. After he recovered the two key bytes successfully, go to the querying differentials block to repeat the previous procedures until the key is fully recovered.

4.2 Recovering the Full Length Random 256-byte Key

The querying differentials block for recovering 256-byte key is shown in Figure 3.

The attacker's goal is to recover a 256-byte secret key K in turn, namely, from $K[0]$ to $K[255]$, two consecutive key bytes can be recovered at one time as illustrated in Figure 1. The attacker first queries two differentials $\Delta K_1^1[0]$ and $\Delta K_1^1[1]$ to the KSA Oracle, and ask it to run the KSA algorithm under two keys K_1 and K_2 which satisfy $K_1[0] = K[0] + \Delta K_1^1[0], K_1[1] = K[1] + \Delta K_1^1[1], K_1[i] = K[i]$ for $i \neq 0, 1$ and $K_2[0] = K_1[0] + 1, K_2[1] = K_1[1] - 1, K_2[2] = K_1[2] + 1, K_2[i] = K_1[i]$ for $i \neq 0, 1, 2$ respectively. If he is lucky enough, he will observe a collision. Recall the previous collision requirements that a collision means that $j_{1,0} = i = 0$ ($j_{2,0} = i + 1 = 1$) and $j_{1,1} = i = 1$ ($j_{1,2} = i - 1 = 0$). This gives $K_1[0] = 0 - S_{1,0}[0] = 0$ ($K_2[0] = 1$) and $K_1[1] = 1 - 0 - S_{1,1}[1] = 0$ ($K_2[1] = 255$). Then the attacker can easily recover $K[0]$ and $K[1]$ by computing $K[0] = K_1[0] - \Delta K_1^1[0]$ and $K[1] = K_{1,1}[1] - \Delta K_1^1[1]$ with the two known differentials. We call this two differentials ($\Delta K_1^1[0]$, $\Delta K_1^1[1]$) a right differential pair, and ($K_{1,1}$, $K_{1,2}$) a right related key pair at step one. The worst case to

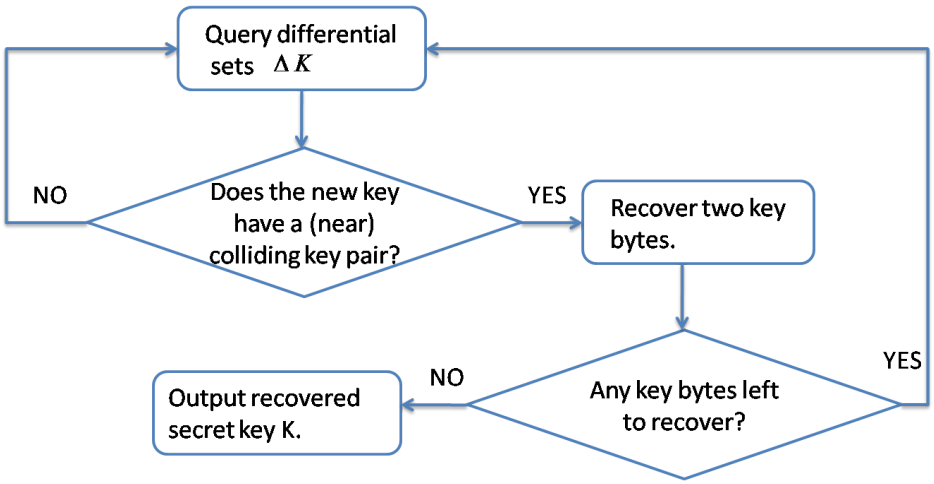


Fig. 2: New Key Recovery Algorithm

the attacker is that he won't be able to get the right differential pair until he queries all the 256 possible values for one differential, namely, 256^2 time queries in total.

Now the attacker has successfully recovered $K[0]$ and $K[1]$. To recover the next two bytes $K[2]$ and $K[3]$, the attacker tries to query two differentials $\Delta K_1^2[2]$ and $\Delta K_2^2[3]$, hoping it to be a right differential pair, also along with the previous right pair $(\Delta K_1^1[0], \Delta K_2^1[1])$. The KSA Oracle will run the KSA algorithm under two keys K_1 and K_2 which satisfy $K_1[0] = K[0] + \Delta K_1^1[0], K_1[1] = K[1] + \Delta K_1^1[1], K_{1,2}[2] = K[2] + \Delta K_1^2[2], K_{1,2}[3] = K[3] + \Delta K_1^2[3], K_{1,2}[i] = K[i]$ for $i \neq 0, 1, 2, 3$ and $K_2[2] = K_1[2] + 1, K_2[3] = K_1[3] - 1, K_2[4] = K_1[4] + 1, K_2[i] = K_1[i]$ for $i \neq 2, 3, 4$ respectively, and sends back the information whether a collision happens or not. If he is unlucky, query differentials $\Delta K_1^2[2]$ and $\Delta K_2^2[3]$ again until it is a right pair (collision happens). Notice that the differential pair $(\Delta K_1^1[0], \Delta K_1^1[1])$ need not be changed since it is the right pair results from the previous stage and we need it there to satisfy the first collision condition for the second stage attack.

Each time when the attacker successfully recovers two key bytes $K[i]$ and $K[i+1]$, he has the knowledge of the right differential pair, and with all the previous known right differential pairs, he is able to recover the future key bytes. The complexity of the attack can be computed from the worst case in which the attacker has to try 256^2 times before he can find the right pair to recover two bytes key. Thus the total complexity time in the worst case is $O(128 \times 256^2) = O(2^{23})$ with probability 1.

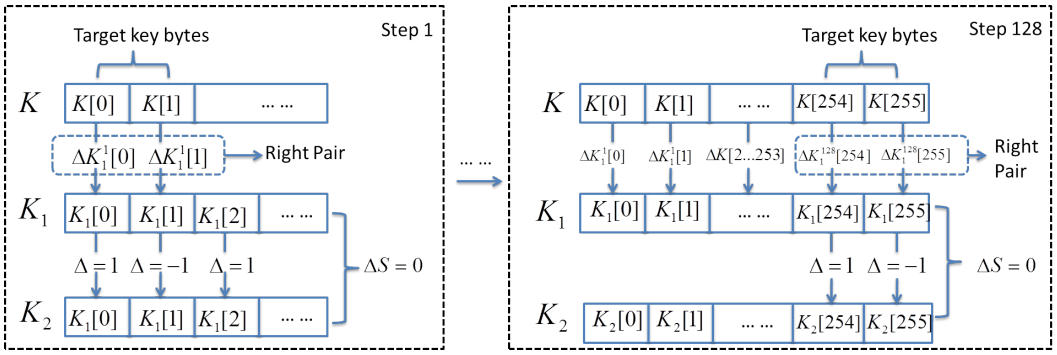


Fig. 3: Querying Differentials Block for recovering 256-byte keys

4.3 Recovering the Random Short Keys

When recovering the full size 256-byte keys, the attacker at each step only need to query key differentials at the two target key indices. Because he knows that due to the collision pattern, he will observe a collision no later than the worst case. However, in case of short keys, only changing the target key bytes will not grantee a collision even in the worst case. It is straightforward because the worst case in querying key differentials at two target key bytes involves 256^2 operations, since the probability for the collision may be smaller than $\frac{1}{256^2}$, in other words, we may need to query also some other key bytes to ensure a (near) collision observation. We illustrate how many key bytes differentials we need to query by computing the collision probability in Figure 4. It is an example of 64-byte key and difference is at index 0.

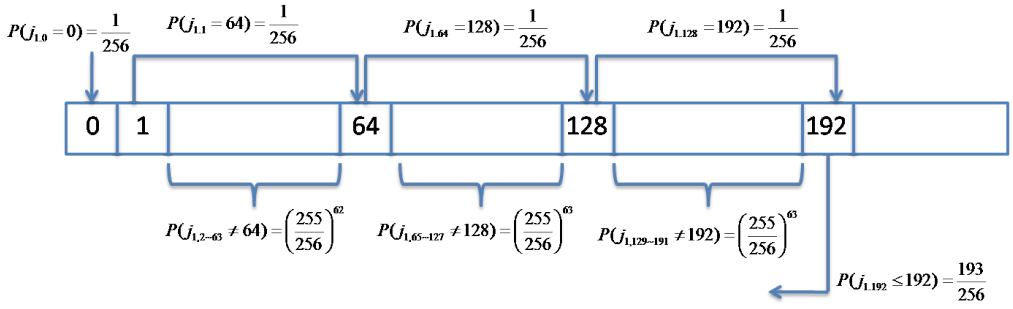


Fig. 4: Determine the number of differentials to query

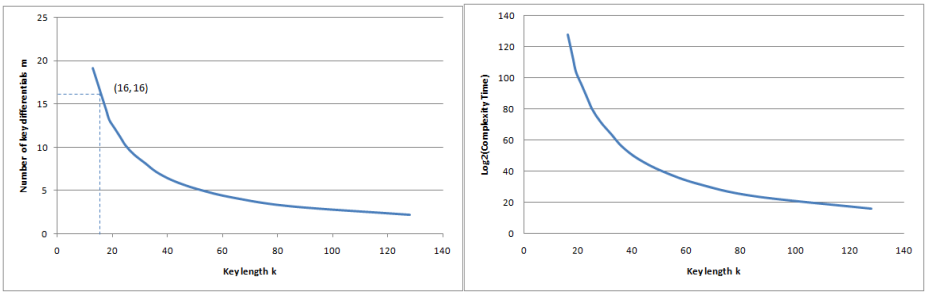
Recall that for a key pair to achieve a collision in short key pattern, we need $j_{1,0} = 0$ and j touches 64, 128 and 192 when i touches 1, 64 and 128 respectively.

And when i is between $[1, 63]$, $[65, 127]$ or $[129, 191]$, j is not allowed to touch the later bound. Finally, when i touches 192, if j is less or equal than 192, it will result in a near collision with two different indices. We assume the internal variable j behaves randomly, which is a reasonable assumption in most of the cases, we get the probability for the collision of a 64-byte key: $(\frac{1}{256})^4(\frac{255}{256})^{62+63+63}\frac{193}{256} \approx 8.5 \times 10^{-11}$. In other words, we'll need to query $\frac{log_2(8.5\times10^{-11})^{-1}}{8} \approx 4$ bytes each time, two extra indices except the two target key bytes. By generalizing this analysis, we can get the following theorem.

Theorem 1. *To construct a related key K_1 which has a colliding key pair under the short key collision pattern from the target key K, the attacker has to query m key differential bytes at two target key bytes indices and $m-2$ other indices. m is given below:*

$$m = log_2\left(\left(\frac{1}{256}\right)^n \times \left(\frac{255}{256}\right)^{k-2+(k-1)(n-2)} \times \frac{(n-1)k+d}{256}\right)^{-1}/8$$

Here, k is the key length, n is the number of times the differences of the keys appear during KSA, and d denotes the first key difference index.



(a) Relations between m and k (b) Complexity for recovering two key bytes

Fig. 5: Theorem 1

Figure 5(a) is the direct visualization of the Theorem 1. From the figure, we know that for key size smaller than 16 bytes, the attacker has to query more than 16 bytes in order to observe a collision, thus makes the attack impossible. We give the theoretical bound of the complexity time for the attack in Figure 5(b). We can conclude that for keys with length larger than 40 bytes, they will fall into our practical attack area.

After the attacker knows how many bytes he should query, he is ready to launch the attack. The querying differentials block for short keys is illustrated in Figure 6.

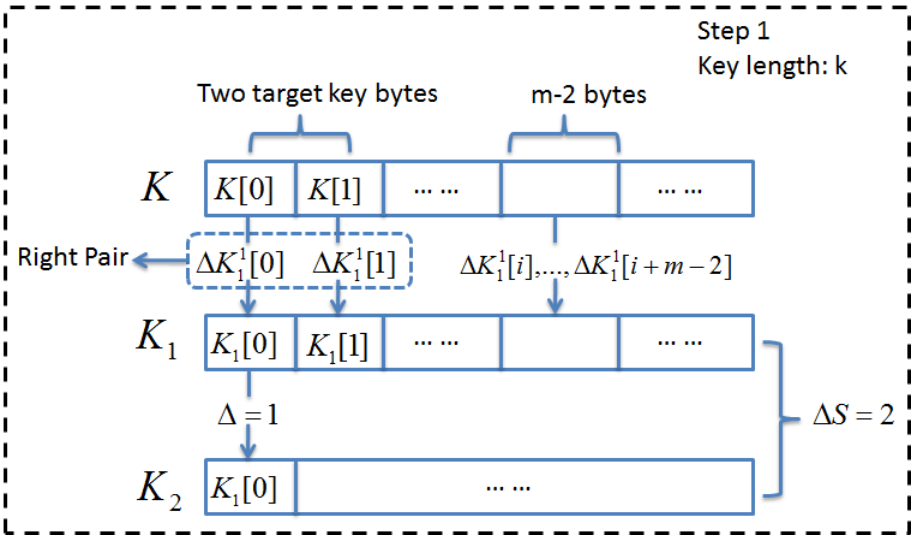


Fig. 6: Querying Differentials Block for Recovering Short keys

Suppose the target key has length k and will repeat n times during KSA. Again the attacker tries to recover the key one by one starting from the beginning. His first two target key bytes is $K[0]$ and $K[1]$. According to Theorem 1, he can compute m, thus besides indices 0 and 1, he can randomly choose other $m-2$ indices and submit the differential queries $\Delta K_1^1[0], \Delta K_1^1[1], \Delta K_1^1[i], ..., \Delta K_1^1[i+m-2]$. To ease the explanation, we assume the $m-2$ indices to be the consecutive ones. The KSA Oracle will return the differential information of two S-Boxes under K_1 and K_2 which differs from K_1 at index 0 by value 1. If he observes that $\Delta S = 2$, then he confirms that a near collision has happened. According to the short key collision pattern, $j_{1,0} = i = 0$ ($j_{2,0} = i+1 = 1$) and $j_{1,1} = j_{2,1} = i+k = k+1$. Thus, $K_1[0] = 0 - S_{1,0}[0] = 0 (K_2[0] = 1)$, and $K_1[1] = K_2 = k+1-1 = k$. Then the attacker can recover $K[0]$ and $K[1]$ by computing $K[0] = K_1[0] - \Delta K_1^1[0]$ and $K[1] = K_1[1] - \Delta K_1^1[1]$. Store the right differential pair $\Delta K_1^1[0]$ and $\Delta K_1^1[1]$ to recover the next two target key bytes, repeat the procedure until the whole key is recovered.

5 Experiment and Comparison

5.1 Implementation

The following table illustrates the implementation details we used to recover the short keys. Each time two consecutive bytes of key are recovered once a near collision ($\Delta S = 2$) is found.

Table 3 illustrates the experimental results of key recovery by using our new algorithm. We show the data of recovering three random secret key with length

Short Key Recovery Implementation

Input:

1. Target key length (bytes) of target secret key K: k.

2. S-Box difference: ΔS.

3. Key differentials: ΔK_1(randomly chosen) and $\Delta K_2 = 1$ (fixed).

Output:

Target secret key K.

Procedures:

1. Set target key index byte $d = 0$.

   1-1. Set the number of query bytes $m = 2$.

   1-2. Prepare the differentials $\Delta K_1 = (\Delta K_1[d], \Delta K[d+1], ..., \Delta K[d+m-1])$ and $\Delta K_1 + \Delta K_2 = (\Delta K_1[d] + 1, \Delta K[d+1], ..., \Delta K[d+m-1])$, and run the KSA under the related keys $K_1 = K + \Delta K_1$ and $K_2 = K + \Delta K_1 + \Delta K_2$.

   1-3. If $\Delta S \neq 2$ after running KSA with 256^m related key pairs, $m = m + 1$, goto 1-2.

   1-4. If $\Delta S \neq 2$ before running KSA with 256^m related key pairs, goto 1-2.

   1-5. If $\Delta S = 2$, then recover two bytes $K[d]$ and $K[d+1]$, $d = d + 2$, goto 1-1.

256, 128 and 86 bytes. Actually, this three random key represents three classes of keys with parameter n equals to 1, 2 and 3, respectively. The experiment is performed by randomly choosing many target keys and the average data is shown in the table. For key length 256 and 128, 100 times were performed, but due to the long time period, only 3 times were done for the key length 86. We list the number of bytes (m) we need to query in order to recovery two key bytes. For 256-byte full length key, we only need to query two key bytes, and can be recovered in a very short time (0.038 seconds). For 128-byte key, 55% of the key bytes need querying two bytes in order to be recovered, and rest 45% of the key need to query three key bytes in order to be recovered. The number of query bytes increased to four when we try to recover 86-byte key. About 63% of the key requires to query four bytes in order to be recovered. The experimental value of m is very close to the theoretical value which can be calculated from Theorem 1. The experiment is done on an i7 CPU desktop PC with Windows XP system (parallel computing is not involved).

5.2 Comparison and Other Applications

We summarize all the key recovery attacks on RC4 in Table 4. Compared with the previous works, our main contribution is that we can attack long keys within practical complexity time. And also, due to various biases discovered at the beginning of PRGA, it is the usual case that the implementation will discard

Table 3: Experimental Results of the New Key Recovery Algorithm

n	**Key Length**	**Number of Query Bytes**					**Time**
		$m = 2$	$m = 3$	$m = 4$	**Exp Value**	**Theo Value**	
1	256	128(100%)	0	0	2	2	0.038s
2	128	35(55%)	29(45%)	0	2.42	2.21	287s
3	86	0	16(37%)	27(63%)	3.63	3.19	76323s (21.2h)

the first hundreds output bytes of the keystream, which make the attacks such as by taking advantage of the weak IVs impossible. Our attack can survive this kind of remedies because discarding the first hundreds output keystream bytes will not affect the observation of the output differences, especially when the total key collisions are achieved. Early in [15], the authors proposed a new way to attack WEP and pointed out that their passive attack can be theoretically adapted to WPA, although no further detailed explanation was given. Here we give another theoretical way to attack WPA. Recall the pattern we use to recover the full 256-byte key, where two keys differ from each other at three indices. Here treat the first one as IV, by changing its value so that let the second and third key differences step through the whole key. Once a collision or a near collision is observed, the attacker can recover the key at the second and third difference indices. Similarly, we can also use the transitional pattern to recover. The main difficulty is that all the currently known key collision patterns have a high complexity when the key is short, and in WPA as well as in WEP settings, only 16-byte key is used. This makes our attack impractical in such environments. However, we cannot rule out the existence of other kinds of colliding key pairs with short key size and low complexity. If that is possible, than the attack is not theoretical any more.

6 Conclusion

In this paper, we presented a new approach to recover secret keys of RC4 in practical time in a related-key model by making use the property that RC4 can generate a large mount of colliding key pairs. Our main contribution is that our attack can recover large keys efficiently with probability 1 while some the previous researches can only recover very short keys with small probability. Although the attack against WEP can efficiently recover the secret keys within practical time, it requires the knowledge of IVs and the keystream outputs. It will not work if the first hundreds keystream outputs are discarded, while our attack is not affected by this remedy. Thus our method shows another way to attack the applications which use the IV setting, especially when the key size is

Table 4: Comparison of the Key Recovery Attacks

Paper	**Resources available to the attacker**	**Key Length**	**Complexity**	**Probability**
[12–15]	IV, Keystream output	All keys	Practical	1
[17]	S-Box	$k = 5$	2^{20}	0.86
[18]		$k = 16$	2^{64}	0.005
		$k \gg 16$	Impractical	Impractical
[19]	S-Box	$k = 5$	2^{24}	0.998
		$k = 16$	2^{35}	0.075
		$k \gg 16$	Impractical	Impractical
Ours	$\Delta K, \Delta S$	$k = 256$	2^{23}	1
		$k > 40$	$< 2^{48}$	1
		$k < 16$	Impractical	Impractical

large. Theoretically speaking, our method can also be adapted to attack WPA if colliding key pairs with shorter key size (lower complexity) can be found.

References

1. Grosul, A.L., Wallach, D.S.: A Related-Key Cryptanalysis of RC4. Technical Report TR-00-358, Department of Computer Science, Rice University (2000), http://cohesion.rice.edu/engineering/computerscience/tr/ TR Download.cfm?SDID=126
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. Chen, J., Miyaji, A.: A New Class of RC4 Colliding Key Pairs With Greater Hamming Distance. In: J.Kwak et al. (eds.): ISPEC 2010, LNCS, vol. 6047, pp. 30-44, Springer, Heidelberg (2010)
4. 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)
5. 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
6. Roos, A.: A Class of Weak Keys in the RC4 Stream Cipher (1995), http://marcel.wanda.ch/Archive/WeakKeys
7. Fluhrer, S.R., McGrew, D.A.: Statistical Analysis of the Alleged RC4 Keystream Generator. In: Schneier, B. (ed.) FSE2000. LNCS, vol.1978, pp.19-30. Springer, Heidelberg (2001)
8. Golic, J.D.: Linear Statistical Weakness of Alleged RC4 Keystream Generator. In: Fumy, W.(ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp.226-238. Springer, Heidelberg (1997)

9. Mantin, I.: Predicting and Distinguishing Attacks on RC4 Keystream Generator. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 491-506. Springer, Heidelberg (2005)
10. Mantin, I., Shamir, A.: A Practical Attack on Broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152.164. Springer, Heidelberg (2001)
11. Paul, S., Preneel, B.: A New Weakness in the RC4 Keystream Generator and an Approach to Improve Security of the Cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 245.259. Springer, Heidelberg (2004)
12. Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 1.24. Springer, Heidelberg (2001)
13. Klein, A.: Attacks on the RC4 Stream Cipher. Designs, Codes and Cryptography 48(3), 269.286 (2008)
14. Tews, E., Weinmann, R.P., Pyshkin, A.: Breaking 104 Bit WEP in Less than 60 Seconds. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 188.202. Springer, Heidelberg (2007)
15. Vaudenay, S., Vuagnoux, M.: Passive-Only Key Recovery Attacks on RC4. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 344.359. Springer, Heidelberg (2007)
16. Finney, H.: An RC4 cycle that can't happen. Newsgroup post in sci.crypt, September 1994.
17. Paul, G., Maitra, S.: Permutation After RC4 Key Scheduling Reveals the Secret Key. In: Adams, C., Miri, A., Wiener, M. (eds.) LNCS, vol.4876, pp. 260-377. Springer, Heidelberg (2007).
18. Biham, E., Carmeli, Y.: Efficient Reconstruction of RC4 Keys from Internal States. In: Nyberg, K. (eds.) LNCS, vol.5086, pp. 270-288. Springer, Heidelberg (2008).
19. Akgun, M., Kavak, P., Demirci, H.: New Results on the Key Scheduling Algorithm of RC4. In: Vincent, R., Abhijit, D., Dipanwita Roy, C. (Eds.) LNCS, vol.5365, pp. 40-52. Springer, Heidelberg (2008).