IEICE TRANS. FUNDAMENTALS, VOL.E95–A, NO.12 DECEMBER 2012

PAPER Special Section on Information Theory and Its Applications

Cryptanalysis of Stream Ciphers from a New Aspect: How to Apply Key Collisions to Key Recovery Attack**

Jiageng CHEN[†a)], ***Nonmember and*** **Atsuko MIYAJI**[†*], ***Member***

SUMMARY In this paper, we propose two new attacks against stream cipher RC4 which can recover the secret key in different length with practical computational amount. However, we have to point out that the proposed attacks are performed under relatively strong related key models. The same as the usual related key models, the adversary can specify the key differentials without knowing the target key information. However, in our attacks, only the relation between two keystream outputs or the two final internal states are required for the attacker. In addition, we discover a statistical bias of RC4 which is the key point to one of the attacks. Besides the inappropriate usage during the WEP environment, RC4 is still considered to be secure with the proper setting, and we believe the result of this paper will add to the understanding of RC4 and how to use it correctly and safely.

key words: *stream cipher, related key model, RC4*

1. Introduction

Cryptanalysis is closely related to the cipher's deployment environment without which the analysis is just pure theoretical work. Basically speaking, the symmetric key cryptography serves as a primitive to the higher level protocols. Thus due to the specific design of the protocol, the primitive can be used in many different ways. In other words, from the adversary's point of view, from the protocols, some information about the cipher may be obtained which is impossible from other protocols. In order to characterize the adversary's ability in those situations, researchers proposed different models such as known plaintext attack, chosen plaintext attack, chosen ciphertext attack, related-key attack, and so on. Examining the ciphers against these well-known models has become a standard rule. However, especially in the world of symmetric key cryptography, the security of the cipher largely depends on the effort of the cryptanalysis, and the design of the cipher can not be guaranteed much more than we can expect. And what's more, the usage of the cipher after being wrapped into some protocol is sometimes totally out of the designer's concern, and even if not, the knowledge gap between the protocol designer and the primitive designer is somewhat huge. This motivates us to analyze the ciphers in somewhat more stronger models that were seldom studied before.

Manuscript received February 10, 2012.
Manuscript revised June 24, 2012.

[†]The authors are with Japan Advanced Institute of Science and Technology, Nomi-shi, 923-1292 Japan.

[*]This study is partly supported by Grant-in-Aid for Scientific Research (B), 20300032.
[**]The preliminary versions were presented at INSCRYPT 2010 [1].

a) E-mail: jg-chen@jaist.ac.jp

DOI: 10.1587/transfun.E95.A.2148

In this paper we focus on analyzing stream cipher RC4, which is one of the most famous ciphers widely deployed in the real world applications such as Microsoft Office, Secure Socket Layer (SSL), Wired Equivalent Privacy (WEP), and so on. Due to its big influence and simple structure, it has become a hot cryptanalysis target since its specification was made public on the Internet in 1994 [8]. More than 20 years study on RC4 has revealed a lot of weaknesses and a lot attacks have been proposed since then. However, except the inappropriate usage in the WEP environment, it has not been broken yet. In the following section, we will briefly review the attacks against RC4 under different models along with the two models we proposed here. These models do not only apply to RC4 but also to other stream ciphers as well. We mainly demonstrate the attacks against RC4 to make these models more clearly. In Sect. 5 we give the attack to recover short and long secret keys of RC4 in Modified RKA model. In Sect. 6, we show how to recover even short keys in Related-Key KFISA model by using the statistical weaknesses described in Appendix. The comparison among the different key recovery attacks are summarized in Sect. 7 and finally conclusion is given in Sect. 8.

2. Notations

We describe the notations that will be used in this paper to address the different attacking models as well as the cryptanalysis of RC4.

Notations for Attacking models in Sect. 3

- z_i: The i-th keystream output generated by $PRBG$.
- IV: Initialization vector.
- S_F: The internal state at the end of KSA stage.
- $KSA(K, IV)$: Key scheduling algorithm with secret key and initial vector as input, and output the internal state ready to be used for keystream output.
- $PRBG(S_F, m)$: Pseudorandom number generator with input S_F, m, and output keystream sequence z_i, where $0 \le i \le m-1$.
- ΔK: The key differential which is chosen by the adversary. It can be treated just as a normal secret key.
- ΔIV: The IV differential which is chosen by the adversary. It can be treated just as a normal IV.
- $f(z_i, z_i^{'})$: A function with two keystream outputs as inputs, outputs 1 if $z_i = z_i^{'}$, and 0 otherwise.
- $g(S_F, S_F^{'})$: A function with two internal states as inputs, outputs the number of elements in S_F that differ

from the ones in S'_F.

Notations for RC4 cryptanalysis

- K_1, K_2: a secret key pair with some differences between them.
- $S_{1,i}, S_{2,i}$: S-Boxes corresponding to the secret key pair at time i before the swap operation.
- $i, j_{1,i}, j_{2,i}$: internal states of RC4. When $j_{1,i} = j_{2,i}$, we use j_i to denote.
- k: the lengths (bytes) of the secret keys.
- h: Hamming distances between the two keys (number of different positions where two keys differ from each other).
- d: the first index of the key differences.
- Γ: the set of indices at which two keys differ from each other, $|\Gamma| = h$, $\Gamma = \{\gamma_0, ..., \gamma_{h-1}\}$ and $d = \gamma_0$.
- n_i: the number of times the key difference γ_i appears during the KSA. $n_i = \lfloor \frac{256+k-1-\gamma_i}{k} \rfloor$ for $i = 0, ..., h-1$.

3. Key Recovery Attacks against Stream Ciphers

From now on, we assume that stream cipher is deployed in some environment and one master key K is used for the encryption and decryption along with the publicly known IV for a period of time before the K is updated. The adversary's target is to recover the master key (target key) K better than brute force searching, and the faster the better. Given the same target, the resources that the adversary can obtain differ from each other in different attacks. In this section, we summarize some of the previous known attacks against stream cipher along with two newly proposed ones. For the details of the different attacks, we mainly demonstrated by using RC4.

Known Final Internal State Attack (KFISA) As we know that almost all the stream ciphers are composed by two algorithms, namely, KSA and PRGA. In the KSA stage, the internal state is scrambled by using the IV and the K. After that, the scrambled internal state will be used in PRGA to generate the keystream. The final internal state here refers to the internal state after the KSA algorithm. In other words, the adversary has in possession of this information and try to recover the key. This attack put more focus on the KSA algorithm itself and it is a nice way to evaluate the strength of the key scheduling part of the stream cipher. [19], [20] and [21] are some theoretical representatives of such attack. Figure 1 shows the procedure of the attack.

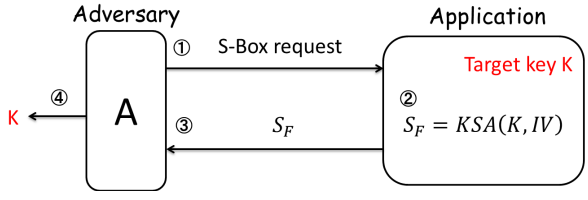


Fig. 1 Known final internal state attack.

This model first reflects the security of the key scheduling algorithm, which has already been studied in the case of RC4. One significant point is that if part of the initial state is leaked due to some reasons for instance side channel attack, then the key could be recovered if there is an efficient attack in KFISA. This will cause big trouble in the IV based construction since the master secret key will be used for many sessions while only the IV is changed. Other than side channel attack, to recover the initial state given the keysteram output is also address in [7] although it's result is only of theoretical interests.

Besides the passive attacks described above, the following three attacks can be seen as active attacks since the adversary is allowed to query the application service (Oracle) for the information he wants.

Related Key Attack In the related key attack, the adversary can obtain many different keystream outputs, which are generated by different IVs and secret keys. The adversary has the ability to query the IV and key differentials but does not know the secret key or internal state information. The Oracle will return the corresponding keystream[†]. The attack against WEP environment is one of the most successful practical attack examples under this model such as [17] and [18]. Figure 2 shows the attack.

Modified Related Key Attack (Modified-RKA) This is our newly proposed modified related key attack. We call it modified version of RKA because the adversary can also query the IV and key differentials, however, the Oracle will return the relation between the two keystreams instead of the keystream itself. The relation can be treated as the simple exclusive or or modular addition operations. One of the translation of this model is to first find key collisions like in [3], then by making use of this special property (identical keystream outputs) to launch the attack. This attack is demonstrated in Fig. 3.

Notice that the function f is specified to output one bit information to indicate whether the two input streams are the same or not. As we will demonstrate later that even without knowing the keystream itself, secret key can still be recovered in this model, not even mention that we further strenthen the power of the function f.

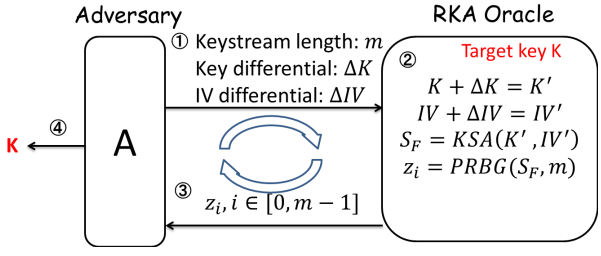


Fig. 2 Related Key Attack (RKA).

[†]The attack which involves only the IV such as Chosen IV attack [2] can be seen as a special case of this model.

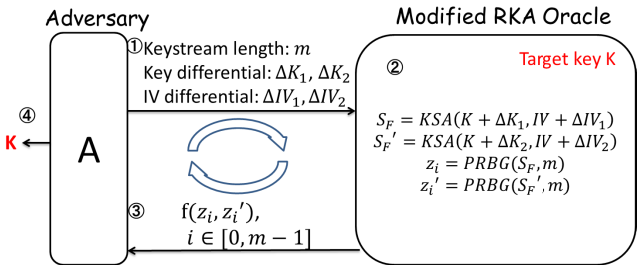


Fig. 3 Modified related key attack.

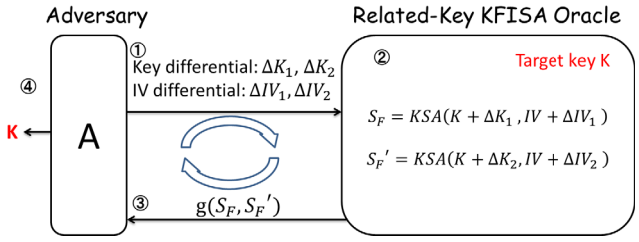


Fig. 4 Related-key KFISA.

Related-Key Known Final Internal State Attack (Related-Key KFISA) This attack can be seem as the related key version of the KFISA. Instead of getting the randomized S-Box by KSA directly, the Oracle will return the relation between the two S-Boxes, which are generated according to the key and IV differentials queried by the attacker. The attack is demonstrated in Fig. 4.

The way to obtain the differentials could be as trivial as checking if the key collision happens, namely, observing if two keystreams are exactly the same or not. This can be further extended to observe the near collision, namely, in case of RC4, two similar initial state will tend to output the similar keystream for the first few bytes. Also, as in the KFISA model, side channel attack or due to some flawed protocol design, the differentials of the initial state are available to the adversary. For the future research, if we can find a relation between the keystream differentials and the initial state differentials from a more general point of view, then we may unify this model with related key model. For now, we investigate them separately. The details will be addressed in Sect. 6 and Appendix.

Except the models we discussed here, we have to mention that there still exists other models which characterize the different aspects of the stream cipher. When the adversary is given only one keystream output, it can be seen as a known plaintext attack. Different from block ciphers, the adversary here can only obtain one keystream output since the secret key is used only once. The adversary's power is so weak that all the practical stream ciphers should be secure against this attack. In another example in [6], the model focuses on the differences of the two related internal states after the key scheduling algorithm, and tries to recover the internal state or key itself. However, we won't do further discussion about them in the following parts.

In the following sections, we choose the stream cipher RC4 as our target and evaluate how it behaves under our proposed Modified Related Key Attack and Related-Key KFISA.

4. Description of RC4 and Previous Key Recovery Results

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 [8].

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 Pseudorandom 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.

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**
  $j \leftarrow j + S[i] + K[i \bmod k]$
  swap$(S[i], S[j])$
end for

Algorithm 2. PRGA (PRBG)

$i \leftarrow 0$
$j \leftarrow 0$
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

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, and almost all the attacks can be described by using the previous defined security model. The most practical one is the distinguishing attack, which is under the **KPA** model. The attacker tries to distinguish between an output stream generated by PRGA and a random stream [10]–[12]. Other various general weaknesses of RC4 have been discovered in the previous works [9], [13], [14], etc. However, the targets of those papers are not the key re-

covery, or at least the weaknesses exploited are not sufficient to recover the secret key. Thus RC4 can still be consider to be secure under **KPA** model.

The attacks under **RKA** model have also been studied, among which the ones against WEP environment can be seen as successful real world attack. In this environment, 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 [15]. 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 [16] and [17]. 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 [18]. However, one important thing to notice is that these attacks depend heavily on the property of IV (Weakness of IV, or at least the existence of IV settings), thus it can be viewed as a stronger version of **RKA** since **RKA** does not add any conditions on IV. In other words, if IV setting is not available, these attacks will not be successful.

Attacks that only observing the final S-Box after KSA algorithm [19]–[21] belong to the **KFISA** model. 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.

5. Modified Related Key Attack (Modified RKA)

Our first proposed key recovery attack works under the Modified RKA, and it is based on the fact that RC4 can generate a large amount of colliding key pairs. The fact that RC4 can generate colliding keys was first discovered in [3], and later generalized by [4], [5]. We call a key pair K_1 and K_2 a colliding key pair if the two corresponding S-Boxes are equal to each after KSA algorithm, namely they have the same effect on the encryption. Previous researches have showed that collisions can be achieved under some specific key pattern with some probability. Since the pattern to achieve long key collisions is different from the ones for short keys, we need to use different techniques to recover long keys and short keys. This is so because the attack heavily depends on the key collision patterns themselves which we separately introduce in Sects. 5.1 and 5.2. And also we will see that the technique to recover the long keys can not be adapted to recover the short ones and vice versa.

5.1 Random Full Length 256-Byte Key Recovery

[5] has showed that it is very easy for large keys to achieve collisions under some specific patterns. Here we describe one collision pattern to use for our attack.

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

In order for two keys in the above pattern to achieve collisions, the following necessary conditions have to be satisfied.

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 $j_{1,d+1} = d+1$ ($j_{2,d+1} = d$).

One example is given in Table 1 where $d = 0$. Now we are ready for the key recovery procedure.

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 Fig. 5. The attacker first queries two differentials $\Delta K_1^1[0]$ and $\Delta K_1^1[1]$ to the 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

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	
0	0	0	0	1	2	$K_2[0] = K_1[0] + 1$
	1	1	1	0	2	$j_{2,0} = j_{1,0} + 1, S_1 \neq S_2$
1	0	1	0	1	2	$K_2[1] = K_1[1] - 1$
	255	0	0	1	2	$j_{2,1} = j_{1,1} - 1, S_1 = S_2$
2	X	$X+3$	0	1	$S[X+3]$	$K_2[2] = K_1[2] + 1$
	$X+1$	$X+3$	0	1	$S[X+3]$	$j_{1,2} = j_{2,2}, S_1 = S_2$

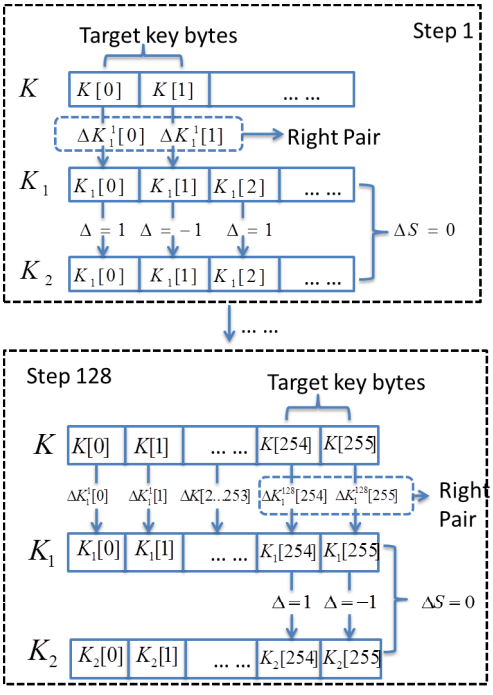


Fig. 5 Querying differentials for recovering 256-byte keys.

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		

$i \neq 0, 1$ 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 by observing the two keystream output differentials. Recall the previous collision requirements which 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 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 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 $128 \times 256^2 = 2^{23}$ with probability 1.

5.2 Random Short Key Recovery

The previous pattern used in recovering the full length key can not be used to recover short keys. For short keys, we need to find a pattern that first it can lead to collisions, and second it has a relatively low complexity. We decide to choose the following pattern which is first discovered in [3].

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 $d = 0, k = 128$.

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 guarantee 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 Fig. 6. It is an example of 64-byte key and difference is at index 0.

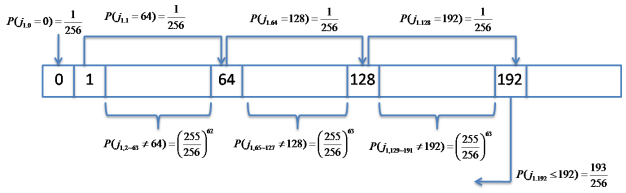


Fig. 6 Determine the number of differentials to query.

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: $(\frac{1}{256})^4(\frac{255}{256})^{62+63+63}\frac{193}{256} \approx 8.5\times10^{-11}$. 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:

$$m = \log_2\left(\left(\frac{1}{256}\right)^n \times \left(\frac{255}{256}\right)^{k-2+(k-1)(n-2)} \times \frac{(n-1)k+d}{256}\right)^{-1}/8$$
$$\approx \frac{\sum_{d=0}^{k-1}\left(\log_2\left(\left(\frac{1}{256}\right)^n \times \left(\frac{255}{256}\right)^{k-2+(k-1)(n-2)} \times \frac{(n-1)k+d}{256}\right)^{-1}/8\right)}{k}$$

Here, k is the key length, $n = \lfloor\frac{256+k-1-d}{k}\rfloor$ and d denotes the first key difference index. Since d does not play the dominant role in the equation, and what we care is the relation between m and k, we can approximate m for $d \in [0, k-1]$, which leaves us the relation between m and k.

Proof 1: Since we have computed the probability for 64-byte key, we can derive the general case based on it very easily. The near collision is composed by three parts. The first one is the probability at the locations where key difference exists, which is denoted by $(\frac{1}{256})^n$, where n is the time key difference is repeated during KSA ($n = 4$ in case of 64-byte key). Second part is the probability that i does not touch any of the S-Box differences when it is between the two difference indices. Given a general parameter key length k and n, it can be computed as $(\frac{255}{256})^{k-2+(k-1)(n-2)}$. Finally, in order to achieve near collision, we would like S-Box difference to be swap to the small indices when i touch the last key difference index, which is addressed by the probability $\frac{(n-1)k+d}{256}$. Thus taking the logarithm of the probability and divide by 8 will give the number bytes that are required to be queried before observing a collision.

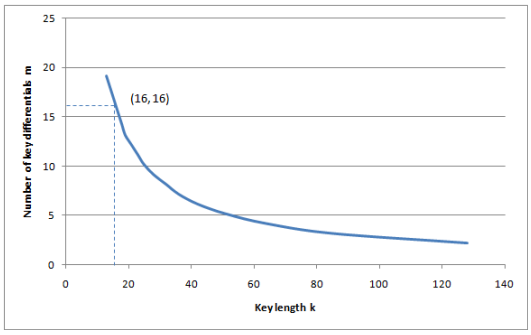


Fig. 7 Relations between m and k.

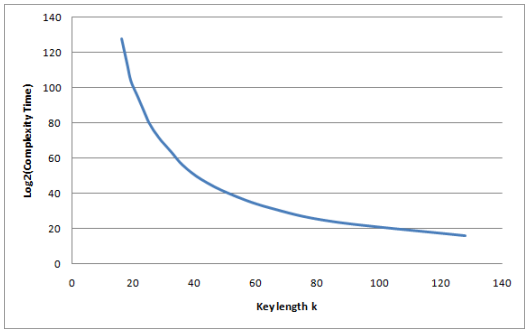


Fig. 8 Complexity for recovering two key bytes.

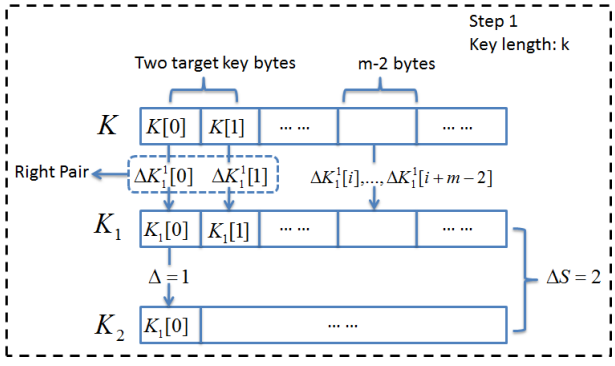


Fig. 9 Querying differentials for recovering short keys.

Figure 7 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. This also matches with the result in [3], which says that for key length less than 17 bytes, collisions can not be achieved. We give the theoretical bound of the complexity time for the attack in Fig. 8.

After the attacker knows how many bytes he should query, he is ready to launch the attack. The querying differential procedure for short keys is illustrated in Fig. 9.

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

Table 3 Experimental Results of recovering short keys under **Modified RKA**.

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)

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 Oracle will return the differential information of two keystream output Δz_i under K_1 and K_2 which differs from K_1 at index 0 by value 1. By observing the Δz, he can confirm whether a collision (near collision) has happened ($\Delta z_i = 0$ for all i, or for a relatively large i, which can be easily tested). According to the short key collision pattern, $j_{1,0} = 0$ ($j_{2,0} = 1$) and $j_{1,1} = j_{2,1} = k+1$. Thus, $K_1[0] = 0 - S_{1,0}[0] = 0 (K_2[0] = 1)$, and $K_1[1] = K_2[1] = k + 1 - 1 = k$. 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.

Table 3 shows the experiment results on recovering 256-byte, 128-byte and 86-byte keys, and results indicate that they can all be recovered in practical time. However, there is a very big limitation of this technique, namely, it's complexity depends on the key collisions, which in turn are impractical when to recover short keys, let's say the lengths less than 32 bytes. Actually, it has been shown in [3] that there exists no colliding key pairs for 16-byte key, which is the practical size deployed in most of the real world environments. Thus we are simply not able to recover those keys. The following sections will describe how to overcome those difficulties.

6. Related-Key Known Final Internal State Attack

In this section, we show how to break RC4 (recovering the secret key) in Related-Key Known Final Internal State Attack. This attack is heavily based on the new statistical weaknesses that output differentials of the initial state is far from uniform distribution which can leak some information about the secret key. The statistical weaknesses are described in Appendix and we focus on recovering the key in this Section.

Adversary's target: A random secret key K with length k. Only the Oracle (Server) knows K.

The whole procedure can be divided into pre-attacking phase and the real time attacking phase. In the pre-attacking phase, the adversary will build a database without the interaction with the Oracle. And in the real time attacking phase, the adversary tries to recover the K by the prebuilt database and the interactive results with the Oracle. For the simplicity, no IV setting is involved.

Off-line Attacking Phase

The adversary runs the RC4 simulation by himself and store the most valuable ΔS which connects to the appearance of the special j value. The smaller the ΔS is, the easier for the special j value to appear, however, the adversary also has to take the complexity of the interactive stage with Oracle into consideration. As a result, the adversary will store a table with the following parameters: the expected ΔS, the corresponding complexity for obtain the expected ΔS, the differential index d, and the possible special j_x indices. Table A·3 in Appendix shows the precomputed table for $k = 16$, which is the length for the real world application.

On-line Attacking Phase

1. The adversary randomly chooses $\Delta K = \Delta K[0], ..., \Delta K[k-1]$ and $\Delta K'$ where $\Delta K'[d] = \Delta K[d] + 1$, and $0 \le d < k-1$. Submit them to the Oracle.

2. Upon receiving the key differentials, the Oracle will first calculate $S_F \leftarrow KSA(K + \Delta K)$ and $S'_F \leftarrow KSA(K + \Delta K')$, where $K + \Delta K = K[0] + \Delta K[0], ..., K[k-1] + \Delta K[k-1]$ and $K + \Delta K' = K[0] + \Delta K'[0], ..., K[k-1] + \Delta K'[k-1]$. Return the $\Delta S = g(S_F, S'_F)$, where ΔS is the number of S-Box bytes that are differ from each other.

3. The adversary will check the ΔS in the precomputed table to see if it is the expected one. If it is the expected one, launch the **Locating x** and **Locating $\mathbf{j_x}$** experiments (refer to Appendix) to recover one j value. Let's denote the corresponding key differentials candidates ΔK_E and $\Delta K'_E$ and store them somewhere.

3-1. For the **Locating x** experiment, modify the candidate key differentials ΔK_E and $\Delta K'_E$ according to the step 2 of the **Locating x** experiment, and submit to the Oracle. After the experiment, the adversary should know the index x of the special j_x for the candidate key differentials.

3-2. For the **Locating $\mathbf{j_x}$** experiment, the adversary modifies the candidate key differentials ΔK_E and $\Delta K'_E$ according to the step 1 of the experiment, and submit to the Oracle. After the experiment, the adversary recovers the special j_x for the secret key $K + \Delta K_E$ ($K + \Delta K'_E$).

4. In this step, we have obtained some special j_x values, which are the j values corresponding to some secret keys for example $K + \Delta K_E$. In order to obtain the j^t values for the target secret key, we need to map them back. For some obtained special j_x, we have

$$j_x = \sum_{i=0}^{x}(K[i] + \Delta K_E[i]) + \sum_{i=0}^{x} S_{i-1}[i]$$

and for the target j^t, we have

$$j_x^t = \sum_{i=0}^{x} K[i] + \sum_{i=0}^{x} S_{i-1}^t[i]$$

Target key K can be canceled and we can obtain

$$j_x^t = j_x - \sum_{i=0}^{x} \Delta K_E[i] + \sum_{i=0}^{x} S_{i-1}^t[i] - \sum_{i=0}^{x} S_{i-1}[i]$$

If we recover j_x^t in order from j_0^t, then the adversary knows $\sum_{i=0}^{x} S_{i-1}^t[i]$ for sure. The only remaining part is $\sum_{i=0}^{x} S_{i-1}[i]$ which is not known to the adversary. However, we have

$$\sum_{i=0}^{x} S_{i-1}[i] \overset{P_s(x)}{=} \sum_{i=0}^{x} i$$

where $P_s(x) = \frac{256-x}{256} \times \frac{256-x+1}{256} \times \cdots \times 1 = \prod_{i=1}^{x} \frac{256-i}{256}$

P_s indicates that before i touches some specific index, j should not touch it anywhere before. By using the above equation, given some j_x, the adversary is able to recover the j_x^t in the probabilistic way. Good news is that P_s is significantly large even for large x, thus the adversary can try many different j_x to vote the target j_x^t. The correct j_x^t always gets the most votes.

5. According to the precomputed table, select other d values which can be used to efficiently recover other j^t. Go to step 1 to repeat the procedure to recover the next j values until all the $j_0^t, ..., j_{k-1}^t$ are recovered. The secret key K can be recovered in the following straightforward manner:

$$K[i] = j_i^t - j_{i-1}^t - S_{i-1}[i]$$

Starting from i=0 until K[k-1] is recovered.

The complexity of the attack mainly comes from observing the expected ΔS and the procedure of voting on the candidate j_x^t values. The complexity for the voting can be obtained from the probability P_s. In case of the 16 bytes key, the smallest $P_j(x)$ is when $x = 15$, and $P_j(15) = 0.6197$. Namely, given a j_{15}, the resulting j_{15}^t is correct with probability 0.6197, and equals to any other values with probability $\frac{1}{256}$. Thus voting 10 times should be enough to distinguish the correct one from the other wrong candidates. Let's do 10 times voting for each of the previous j which will result the worst case complexity. And the complexity for recovering each special j_x value is the complexity to successfully observe one, which could be obtained from the precomputed table, as a result the total complexity for recovering the whole 16 bytes random key is

$$10\times(\frac{2^{15.4}}{0.193}+\frac{2^{15.4}}{0.166}+\frac{2^{15.4}}{0.138}+\frac{2^{15.4}}{0.115}+\frac{2^{14.8}}{0.173}+\frac{2^{14.8}}{0.158}+\frac{2^{14.8}}{0.119}+\frac{2^{14.8}}{0.107}+\frac{2^{14.1}}{0.166}+\frac{2^{14.1}}{0.168}+\frac{2^{14.1}}{0.126}+\frac{2^{14.1}}{0.116}+\frac{2^{13.3}}{0.154}+\frac{2^{13.3}}{0.149}+\frac{2^{13.3}}{0.129}+\frac{2^{13.3}}{0.118}) \approx 2^{24.75}$$

7. Comparison and Other Applications

We summarize all the key recovery attacks against RC4 in Table 4. As we can see, no practical attack is available under the weakest **KPA** model. And RC4 is not secure under any of the other models. The attacks under the special case of the **RKA** can be viewed as a successful break of RC4, but it has its limitations. First, it heavily depends on the IV settings, without which the attack becomes impossible. Second, in order to prevent from the attack, it is the usual case that the implementation will discard the first hundreds output bytes of the keystream, which make the attacks such as by taking advantage of the weak IVs impossible, while our proposed attack does not take advantage of the IV settings. And in the model **Modified RKA** if collision is achieved, discarding the first hundreds output keystream bytes will not affect the observation of the output differences. Also, compared with the attacks under model **KFISA**, our proposed attack under **Related-Key KFISA** has a better effect of recovering the practical 16-byte key deterministically with less computation complexity. Notice that in the Modified RKA, keys with length less than 16 bytes cannot be recovered due to the fact that no known key collision is available in this case.

Table 4 Comparison of the key recovery attacks.

Paper	Models	Key Length (bytes)	Complexity	Probability
[15]–[18]	**RKA**	k = 13(WEP)	2^{20}	1
		k = 256	Not given	Not Given
[19]–[20]	**KFISA**	k = 16	2^{64}	0.005
		k = 256	$> 2^{80}$	-
[21]	**KFISA**	k = 16	2^{35}	0.075
		k = 256	$> 2^{80}$	-
Ours	**Modified RKA**	k = 16	$> 2^{80}$	-
		k = 256	2^{23}	1
	Related-Key KFISA	k = 16	2^{25}	1
		k = 256	$< 2^{25}$	1

Now we clarify that our proposed attack can be a real world threat. Recall that the design of stream cipher can be divided into two kinds, which are IV-dependent and IV-less. Let's consider the scenario where an IV-less stream cipher is deployed such as RC4. Suppose the server setups up a master secret key K_{master} and negotiate it with the client. In the following communications, session keys are derived from the master key and are used to encrypt the session conversations. Since no IV is involved, let's assume that the server and the client have agreed on a rule to update the key per session. One of the reasonable way to implement is to assume part of the master key K_{master} is not changed, while update the other part of the secret key according to the rule for each session. Our proposed **Modified RKA** model catches this IV-less scenario, since the training game between the adversary and the Oracle can be replaced by passively observing the differentials.

Our specific attacks are based on the fact of the key collision of RC4. There are many related key patterns of RC4 that have been discovered so far such as in [10], [11] and [12]. Still there are even more unknown patterns out there which could lead to the key collisions or near collisions, and also we cannot rule out the existence of key collision for any other stream ciphers. This indicates that with the development of the key collision techniques in the future, the attacks under the Modified RKA and Related-Key KFISA could become more powerful, which deserves our attention.

8. Conclusion

In this paper, we summarize and propose some attacks against stream ciphers including **RKA, KFISA, Modified RKA** and **Related-Key KFISA**. For the **KFISA** and our

newly proposed Related-Key version, although initial state seems to be difficult to obtain in the real attacks, the models investigate the security margin of the KSA algorithm, and besides we should also consider the case that the initial state is leaked due to the side channel attack. And as we have mentioned previously, in the IV-based construction, master key will be used for many sessions and if there is an efficient algorithm to predict some part of the initial state, by combining the attacks here in **KFISA** or **Related-Key KFISA** models, master key can be recovered. Our contributions also indicate that large key size does not always guarantee a better security margin than the small key size. Thus we suggest that all stream ciphers should be carefully examined under relatively stronger models described in the paper to gain confidence in the security margin itself as well as the usage in any unpredictable environments.

References

[1] J. Chen and A. Miyaji, "A new practical key recovery attack on the stream cipher RC4 under related-key model," in eds. X. Lai, M. Yung, and D. Lin, Inscrypt 2010, LNCS, vol.6584, pp.62–76, Springer, Heidelberg, 2011.
[2] H. Wu and B. Preneel, "Key recovery attack on Py and Pypy with chosen IVs," eSTREAM, http://www.ecrypt.eu.org/stream/papersdir/2006/052.pdf
[3] M. Matsui, "Key collisions of the RC4 stream cipher," in ed. O. Dunkelman, FSE 2009, LNCS, vol.5665, pp.38–50, Springer, Heidelberg, 2009.
[4] J. Chen and A. Miyaji, "A new class of RC4 colliding key pairs with greater hamming distance," in eds. J. Kwak, R. Deng, Y. Won, and G. Wang, ISPEC 2010, LNCS, vol.6047, pp.30–44, Springer, Heidelberg, 2010.
[5] J. Chen and A. Miyaji, "Generalized RC4 key collisions and hash collisions," in eds. J. Garay and R. De Prisco, SCN 2010, LNCS, vol.6280, pp.73–87, Springer, Heidelberg, 2010.
[6] A. Miyaji and M. Sukegawa, "New analysis based on correlations of RC4 PRGA with nonzero-bit differences," in eds. C. Boyd and J. Gonzlez Nieto, ACISP 2009, LNCS, vol.5594, pp.134–152, Springer, Heidelberg, 2009.
[7] A. Maximov and D. Khovratovich, "New state recovery attack on RC4," in ed. D. Wagner, CRYPTO 2008, LNCS, vol.5157, pp.297–316. Springer, Heidelberg, 2008.
[8] Anonymous: RC4 Source Code, CypherPunks mailing list (Sept. 9, 1994), http://cypherpunks.venona.com/date/1994/09/msg00304.html, http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0
[9] A. Roos, A Class of Weak Keys in the RC4 Stream Cipher, 1995, http://marcel.wanda.ch/Archive/WeakKeys
[10] S. Fluhrer and D. McGrew, "Statistical analysis of the alleged RC4 keystream generator," in eds. G. Goos, J.Hartmanis, J. van Leeuwen, and B. Schneier, FSE2000, LNCS, vol.1978, pp.66–71, Springer, Heidelberg, 2001.
[11] J. Golic, "Linear statistical weakness of alleged RC4 keystream generator," in ed. W. Fumy, EUROCRYPT 1997, LNCS, vol.1233, pp.226–238, Springer, Heidelberg, 1997.
[12] I. Mantin, "Predicting and distinguishing attacks on RC4 keystream generator," in ed. R. Cramer, EUROCRYPT 2005, LNCS, vol.3494, pp.491–506, Springer, Heidelberg, 2005.
[13] I. Mantin and A. Shamir, "A practical attack on broadcast RC4," in ed. M. Matsui, FSE 2001, LNCS, vol.2355, pp.87–104, Springer, Heidelberg, 2001.
[14] S. Paul and B. Preneel, "A new weakness in the RC4 keystream generator and an approach to improve security of the cipher," in eds. B. Roy and W. Meier, FSE 2004, LNCS, vol.3017, pp.245–259, Springer, Heidelberg, 2004.
[15] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," in eds. S. Vaudenay and A. Youssef, SAC 2001, LNCS, vol.2259, pp.1–24, Springer, Heidelberg, 2001.
[16] K. Andreas, "Attacks on the RC4 stream cipher," Designs, Codes and Cryptography, vol.48, no.3, pp.269–286, 2008.
[17] E. Tews, R.-P. Weinmann, and A. Pyshkin, "Breaking 104 bit WEP in less than 60 seconds," in eds. S. Kim, M. Yung, and H.-W. Lee, WISA 2007, LNCS, vol.4867, pp.188–202, Springer, Heidelberg, 2007.
[18] S. Vaudenay and M. Vuagnoux, "Passive-only key recovery attacks on RC4," in eds. C. Adams, A. Miri, and M. Wiener, SAC 2007, LNCS, vol.4876, pp.344–359, Springer, Heidelberg, 2007.
[19] G. Paul and S. Maitra, "Permutation after RC4 key scheduling reveals the secret key," in eds. C. Adams, A. Miri, and M. Wiener, LNCS, vol.4876, pp.360–377, Springer, Heidelberg, 2007.
[20] E. Biham and Y. Carmeli, "Efficient reconstruction of RC4 keys from internal states," in ed. K. Nyberg, LNCS, vol.5086, pp.270–288, Springer, Heidelberg, 2008.
[21] M. Akgun, P. Kavak, and H. Demirci, "New results on the key scheduling algorithm of RC4," in eds. D. Chowdhury, V. Rijmen, and A. Das, LNCS, vol.5365, pp.40–52, Springer, Heidelberg, 2008.

Appendix: New Statistical Weakness of RC4

Many statistical weaknesses of RC4 have been exploited during the last 20 years research. Weaknesses such as [15] and [16] played a very important role in recovering the key in the WEP environment. Here we describe a new statistical weaknesses of RC4 which is universal to all the secret keys with different length, and then show how it can be used to recover the key.

We know that when $\Delta S = 0$, then collision happens. However, usually the ΔS that the adversary gets has a large value greater than 250 due to the j differences which are introduced by the key differences. By the following observation, we point out that there is a relationship between the j behaviour and the ΔS. Let's denote $+\Delta S$ be the number of changed different S-Boxes between two consecutive steps. For example, assume ΔS_i and ΔS_{i+1} denote the number of different S-Boxes at step i and i+1, then $+\Delta S = |\Delta S_i - \Delta S_{i+1}|$. Intuitively, if the related j values are different, three S-Box elements will be affected which could lead to $+\Delta S$ as large as 3. Also if the corresponding three S-Box elements differ from each other, then by choosing special S-Box values, the differentials will disappear and $+\Delta S$ could be as small as -3. Since there are only limited situations, it is easy to enumerate all the possible $+\Delta S$ and the corresponding internal states. Let's consider the internal states at step $i = \alpha$ before the swap operation. The two corresponding j values are $j_{1,\alpha}, j_{2,\alpha}$. α, β and γ ($\alpha < \beta < \gamma$) denote the S-Box indices, and a, b, c, d, e, f denote the S-Box values such that $a \neq b \neq c \neq d \neq e \neq f$. Then the internal states before the swap operation for each of the possible $+\Delta S$ are list as follows.

$+\mathbf{\Delta S} = \mathbf{3}$:

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \quad \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = a \end{cases} \quad \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = b \end{cases} \quad \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = c \end{cases}$$

$+\Delta S = 2$:

$$\begin{cases} j_{1,\alpha} = \alpha \\ j_{2,\alpha} = \beta \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = a \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = b \end{cases} \text{ or}$$

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = a \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = b \end{cases} \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = d \end{cases}$$

$+\Delta S = 1$:

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = a \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = d \end{cases} \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = e \end{cases}$$

$+\Delta S = 0$:

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \beta \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = c \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = d \end{cases} \text{ or}$$

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \beta \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = a \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = c \end{cases} \text{ or}$$

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = d \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = e \end{cases} \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = f \end{cases} \text{ or}$$

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = a \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = c \end{cases} \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = b \end{cases}$$

$+\Delta S = -1$:

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = d \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = e \end{cases} \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = b \end{cases}$$

$+\Delta S = -2$:

$$\begin{cases} j_{1,\alpha} = \alpha \\ j_{2,\alpha} = \beta \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = b \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = a \end{cases} \text{ or}$$

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = d \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = a \end{cases} \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = b \end{cases}$$

$+\Delta S = -3$:

$$\begin{cases} j_{1,\alpha} = \beta \\ j_{2,\alpha} = \gamma \end{cases} \begin{cases} S_{1,\alpha}[\alpha] = a \\ S_{2,\alpha}[\alpha] = c \end{cases} \begin{cases} S_{1,\alpha}[\beta] = b \\ S_{2,\alpha}[\beta] = a \end{cases} \begin{cases} S_{1,\alpha}[\gamma] = c \\ S_{2,\alpha}[\gamma] = b \end{cases}$$

Figure A·1 summarizes all the cases we list previously along with the two related S-Boxes before the swap operation and after the swap operation, so that we can see directly from the figure that how the $+\Delta S$ is achieved.

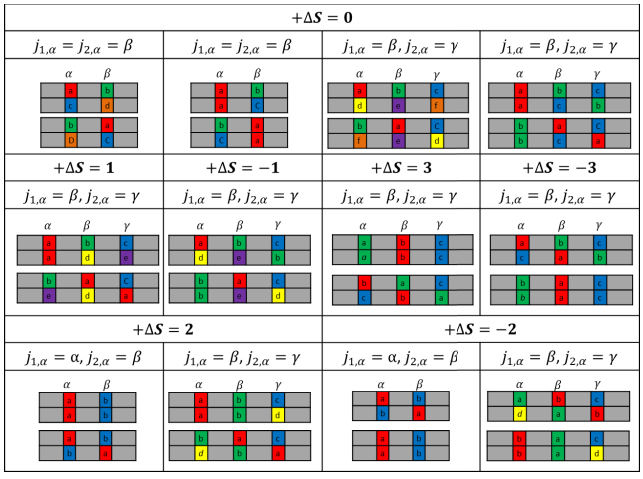


Fig. A· 1 The number of changed different S-Boxes between two consecutive steps during KSA ($+\Delta S$).

Let's consider a key pair with only one index differs from each other, namely, $K_2[d] = K_1[d] + 1$. For $i < d$, $\Delta S = 0$ since no differences are introduced yet. When $i = d$, a difference is introduced by the key and causes $+\Delta S = 3$. If everything behaves randomly, we would expect the ΔS to keep increasing, and at the end of KSA we would have a very large ΔS close to 255. However, if we have a relatively long period of steps with $\Delta S \leq 0$, then we would expect to get a relatively small ΔS at the end. Here we focus on the beginning of KSA where most of the S-boxes have the property $S[i] = i$, namely, they haven't been scrambled yet. This means S_1 and S_2 are very likely equal to each other at the beginning of KSA. Under this situation, we find that the only way to maintain the $+\Delta S \leq 0$ for a relatively long period is to let $+\Delta S = 0$ with $j_1 = j_2$. Otherwise even $+\Delta S < 0$ happens, the j difference will soon make the ΔS to increase. Then the only remaining question is whether it is possible to achieve $j_1 = j_2$ and how hard it is.

In order to answer this question, let's assume $j_{1,d} = m$ and $j_{2,d} = m + 1$ and $m \in [d, d + k - 1]$. Then when $i = m - 1$ after the swap, with high probability $((\frac{254}{256})^{m-d-1})$, we have $S_{1,m-1}[m] = d, S_{2,m-1}[m] = m, S_{1,m-1}[m + 1] = m+1, S_{2,m-1}[m+1] = d$. According to the j update equation $j_{i+1} = j_i + K[i + 1] + S_i[i + 1]$, $\Delta j = j_{2,i} - j_{1,i} = 1$ for $i \in [d, m - 1]$ as long as $S_i[i + 1] = i + 1$. Then

$$j_{1,m} = j_{1,m-1} + S_{1,m-1}[m] + K_1[m] = j_{1,m-1} + K_1[m] + d$$
$$j_{2,m} = j_{2,m-1} + S_{2,m-1}[m] + K_2[m] = j_{2,m-1} + K_2[m] + m$$

Thus

$$\Delta j_m = j_{2,m} - j_{1,m} = m - d + 1$$

When $i = m + 1$, we have

$$j_{1,m+1} = j_{1,m} + S_{1,m}[m+1] + K_1[m+1] = j_{1,m} + K_1[m+1] + m + 1$$
$$j_{2,m+1} = j_{2,m} + S_{2,m}[m+1] + K_2[m+1] = j_{2,m} + K_2[m+1] + d$$

Then we have $\Delta j_{m+1} = j_{2,m+1} - j_{1,m+1} = 0$
And it is very likely that $+\Delta S = 0$ will last till index $d + k$ as long as $S_i[i + 1] = i + 1$ for $i \in [m + 1, d + k]$. Generally speaking, the smaller the value $m-d$ is, the bigger the chance is for ΔS to deviate from an average level.

In other words, the previous analysis indicates that for a given relatively small ΔS, we expect that the j difference after generated by the key difference at d, will be absorbed quickly at some index not far from d. From now on, describing one key behavior is enough, thus if we don't mention specifically, j_i refers to the first key value $j_{1,i}$. More formally, let's define A_i^u to be the event that $j_{d+i} \in [d+i, d+i+u]$, and windows L_i^u denotes the interval $[d+i, d+i+u]$. If RC4 is ideal which means everything is uniformly distributed, then $P(A_i^u) = P(A_{theo}^u) = \frac{u}{256}$. However, the actual case is that event A depends heavily on the ΔS and is severely biased when ΔS is small. We run the following experiment to verify our analysis. For 16, 32, 64 and 128 bytes random key pairs with $K_2[0] = K_1[0] + 1$, we run the KSA algorithm under 5 window with j value window length to be $u = 5$.

Table A· 1 Probabilities of A_i^u and A_{theo}^u given u=5, d=0.

Key	ΔS	$P(A_0^5)$	$P(A_1^5)$	$P(A_2^5)$	$P(A_3^5)$	$P(A_4^5)$	$P(A_{theo}^5)$
16	240	8.3%	8.1%	6.5%	8.3%	7.6%	1.95%
16	245	4.9%	8.0%	6.8%	5.2%	5.7%	1.95%
32	230	17.0%	13.3%	9.7%	10.0%	7.9%	1.95%
32	240	12.1%	10.3%	9.6%	6.0%	5.5%	1.95%
64	210	32.1%	26.3%	15.9%	8.6%	6.6%	1.95%
64	220	24.2%	19.5%	16.2%	8.8%	7.1%	1.95%
128	195	47.1%	23.8%	13.3%	7.2%	3.1%	1.95%
128	210	42.6%	24.2%	12.8%	7.5%	3.6%	1.95%

Table A· 1 shows the frequency that the possible j values from the j window locates at the corresponding S-Box window. For example, for a 16-byte key given $\Delta S = 240$, the probability $P(j_0 \in [0,5]) = 8.3\%$, $P(j_1 \in [1,6]) = 8.1\%$, $P(j_2 \in [2,7]) = 6.5\%$, $P(j_3 \in [3,8]) = 8.3\%$ and $P(j_4 \in [4,9]) = 7.6\%$.

For each key length, we choose two different ΔS to show that as ΔS decreases, the probability of Event A_i^u increases, which confirms our previous analysis. And also for the theoretical evaluation, we have $Prob(A_{theo}^5) = \frac{5}{256} = 1.95\%$ which differs greatly from the experimental data list in the table. The greater the difference differs from the theoretical value, the more efficiently the attack works, which will be covered shortly. Thus, from the table, we can see that the probability of Event A_i^u is more biased given larger key size (128-byte) than the key with smaller size (16-byte). This is reasonable because once the j difference is absorbed, it is not likely to differ with each other until the next key difference, which takes longer to meet for larger keys than smaller keys. Another thing to notice is that in the real attack, the adversary can choose much smaller ΔS than the ones given in Table A· 1 to gain advantages. The only reason that we choose the above ΔS is for data gathering, because the smaller the ΔS is, the longer it will take for the Oracle to generate it. The above ΔS is chosen to be relatively large so that we can repeat the experiment for 10000 times to get the average values in several minutes time.

In short, what we have discovered indicates that given a relatively small ΔS, with very high probability, some special j_x will appear in rounds x which is bigger and close to d, and the value j_x is bigger and close to x. We have narrowed down the range of the special j_x and its index x. If we stop here, we can only guess the values in the probabilistic manner, which is of course much better than the exhaustively search. However, we can do the following test to help locate the exact values. Index x can be located in the following way:

Locating x

1. Given a relatively small ΔS, let's assume $K_1[d], ..., K_1[l-1]$ $(K_2[d], ..., K_2[l-1])$ are the secret key bytes used during S-Box window, namely, during rounds d to $d+l-1$.

2. Modify $K_1[i](K_2[i])$ for $i \in [d, d+l-1]$ to some random value while remaining all the other key bytes unchanged.

3. For each key byte modification, run the KSA under the new key pair. This will generate a ΔS sequence, $\Delta S_d, ..., \Delta S_{d+l-1}$. For the statistical accuracy, each ΔS can be derived by modifying the corresponding key byte many times and take the average value of all the ΔS.

4. Make a differential ΔS sequence $\mathrm{Diff}\Delta S_d, ..., \mathrm{Diff}\Delta S_{d+l-1}$ by computing $\mathrm{Diff}\Delta S_i = \Delta S_i - \Delta S_{i+1}$ for $i \in [d, d+l-2]$.

5. $x = i$ where $\mathrm{Diff}\Delta S_i$ is the biggest value among the differential ΔS sequence, and also it is significantly larger than any of the other values.

Table A· 2 should give you the idea. It demonstrates the differential ΔS for some random keys with different length. Again we demonstrate by assuming $K_2[0] = K_1[0] + 1$, namely, $d = 0$ and with S-Box window size $l = 13$. In the real attack, we could choose even larger l and smaller ΔS to gain efficiency. The x always locates at the place where $\mathrm{Diff}\Delta S_x$ is the biggest one. This is because when modifying the key bytes before x, the special j_x will disappear and all the ΔS will tend to be the same and large. However, when the key bytes after index x are modified, the special j_x will still exist and the ΔS value will tend to be small. This phenomenon can be observed by making the subtraction of the consecutive ΔS and find the largest one in the sequence.

Once we have located x, locating j_x is only one step away. By using the following techniques, we can achieve the goal.

Locating $\mathbf{j_x}$:

1. Assume that $j_x - x \leq r$. Fix the key values $K_1[0], ..., K_1[x-1](K_2[0], ..., K_2[x-1])$ unchanged.

2. For each $i \in [x, k-1]$, reduce $K_1[x](K_2[x])$ as $K_1[x] = K_1[x] - i(K_2[x] = K_2[x] - i)$, and randomly modify other key bytes $K_1[x+1], ..., K_1[k-1](K_2[x+1], ..., K_2[k-1])$ to generate ΔS. Repeat the random modification part to get the average value.

3. After step 2, we have a sequence of average ΔS values, and j_x equals to the index of the smallest ΔS in the sequence.

If $j_x = x$, sub figure (a) of Fig. A· 2 shows the case. Sub figure (b) shows the case where $x < j_x < r$. The explanation is rather straightforward. By decreasing the key value $K_1[x](K_2[x])$ one by one, the expected j_x will get closer and closer to the index x, and it will achieve the smallest ΔS only when $j_x = x$. And once $j_x < x$, due to the previous statistical weakness, ΔS will increase dramatically compared with the previous ΔS because the special j disappears and the j value difference will not be absorbed immediately, which will lead to the jump of the ΔS value.

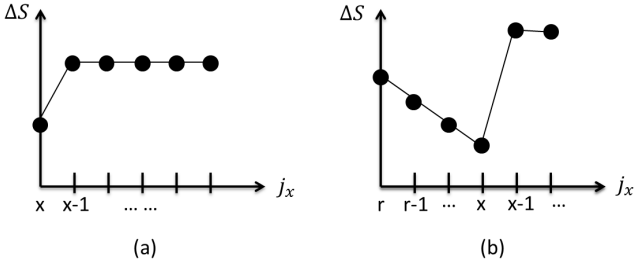


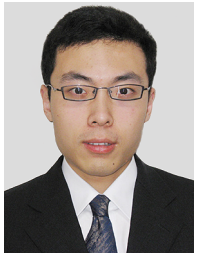
Fig. A· 2 Locating j_x.

Table A· 2 Locating x.

	S_{win}	0	1	2	3	4	5	6	7	8	9	10	11	12	x
k=16	j	131	33	46	**6**	197	163	140	241	215	130	53	175	36	3
	DiffΔS	0.543	−0.279	−0.029	**1.691**	0.130	−0.142	−0.205	0.285	0.215	0.027	−0.156	−0.361	–	–
ΔS = 225	j	145	34	77	137	125	**7**	159	43	215	57	228	162	115	7
	DiffΔS	0.162	−0.398	0.205	−0.103	−0.092	**1.265**	0.545	−0.537	0.162	0.093	−0.494	0.252	–	–
k=32	j	124	247	**3**	169	208	191	194	135	53	97	190	2	242	2
	DiffΔS	−0.194	0.458	**2.465**	0.054	1.437	−0.063	0.062	−0.331	0.659	−0.261	0.389	−0.312	–	–
ΔS = 210	j	252	193	223	37	91	**7**	191	160	154	47	130	126	143	5
	DiffΔS	−0.442	−0.703	0.507	−0.292	0.109	**4.460**	−0.353	0.164	0.464	1.262	−0.479	−0.384	–	–
k=64	j	203	**1**	244	217	251	183	38	14	221	133	123	116	114	1
	DiffΔS	−0.005	**14.192**	0.891	0.599	−0.244	0.602	1.250	−2.963	0.729	1.423	−0.461	−0.009	–	–
ΔS = 190	j	**0**	175	27	216	67	226	179	154	185	169	150	217	167	0
	DiffΔS	**13.889**	3.165	−2.788	−0.563	1.329	0.125	−0.295	−0.277	0.849	0.420	−0.503	0.120	–	–
k=128	j	255	148	163	235	**5**	171	129	123	164	32	115	101	121	4
	DiffΔS	−0.023	−0.203	0.014	0.558	**23.369**	6.829	10.056	0.466	−0.370	−0.031	0.422	−0.352	–	–
ΔS = 180	j	173	159	221	**3**	27	196	240	246	28	99	27	17	108	3
	DiffΔS	0.220	−0.327	0.917	**41.600**	−36.895	−0.648	0.655	0.082	1.680	1.376	34.937	−0.017	–	–

Table A· 3 Precomputed table for recovering 16-byte key.

ΔS	Complexity	d	Probabilities of x to be in the following indexes															
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
235	$2^{15.4}$	0	**0.193**	**0.166**	**0.138**	**0.115**	0.084	0.077	0.047	0.049	0.039	0.018	0.023	0.009	0.007	0.010	0.004	0.001
	$2^{14.8}$	4	-	-	-	-	**0.173**	**0.158**	**0.119**	**0.107**	0.077	0.087	0.066	0.043	0.037	0.029	0.025	0.019
	$2^{14.1}$	8	-	-	-	-	-	-	-	-	**0.166**	**0.168**	**0.126**	**0.116**	0.083	0.072	0.052	0.069
	$2^{13.3}$	12	-	-	-	-	-	-	-	-	-	-	-	-	**0.154**	**0.149**	**0.129**	**0.118**



Jiageng Chen received the B.Sc. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China in 2004. He received the M.Sc. and Dr.Sci. degrees in information science from Japan Advanced Institute of Science and Technology (JAIST), Nomi, Japan in 2007. He was a postdoctral researcher at school of information science, JAIST from 2012.4.1 to 2012.5.31. He received the Best Ph.D graduate student Award in 2012. He is now an Assitant Professor of School of Information Science, Japan Advanced Institute of Science and Technology (JAIST). His research interests include cryptography and infromation security.

Atsuko Miyaji received the B.Sc., the M.Sc., and the Dr. Sci. degrees in mathematics from Osaka University, Osaka, Japan in 1988, 1990, and 1997 respectively. She joined Panasonic Co., LTD from 1990 to 1998 and engaged in research and development for secure communication. She was an associate professor at the Japan Advanced Institute of Science and Technology (JAIST) in 1998. She has joined the computer science department of the University of California, Davis since 2002. She has been a professor at the Japan Advanced Institute of Science and Technology (JAIST) since 2007 and the director of Library of JAIST since 2008. Her research interests include the application of number theory into cryptography and information security. She received Young Paper Award of SCIS'93 in 1993, Notable Invention Award of the Science and Technology Agency in 1997, the IPSJ Sakai Special Researcher Award in 2002, the Standardization Contribution Award in 2003, Engineering Sciences Society: Certificate of Appreciation in 2005, the AWARD for the contribution to CULTURE of SECURITY in 2007, IPSJ/ITSCJ Project Editor Award in 2007, 2008, 2009, and 2010, the Director-General of Industrial Science and Technology Policy and Environment Bureau Award in 2007, Editorial Committee of Engineering Sciences Society: Certificate of Appreciation in 2007, DoCoMo Mobile Science Awards in 2008, Advanced Data Mining and Applications (ADMA 2010) Best Paper Award, and The chief of air staff: Letter of Appreciation Award. She is a member of the International Association for Cryptologic Research, the Information Processing Society of Japan, and the Mathematical Society of Japan.