Near Collisions in the RC4 Stream Cipher

Anindya Shankar Bhandari
Indian Institute of Technology, Kharagpur
Email : anindyaorasb@gmail.com

Abstract

In this paper we explore the intriguing factors involved in the non one-one nature of the RC4, and explore new techniques and present interesting findings regarding the same. The first part of this paper studies near colliding keys of the RC4, and discusses how these keys are localized into clusters in the key-space. The second part of this paper proposes a new collision search algorithm specifically for 16-byte keys. It is generally the practice to choose the byte that differs between two keys to be near the end of the key. However, this is not necessary for 16-byte keys, and the second part of this paper discusses how this may be used to grant us an additional degree of control.

Keywords

RC4; near collisions; collisions; stream cipher;

1 Introduction

The RC4 stream cipher is one of the world's most widely used cryptographic systems. The most interesting aspects of it is its simplicity of its code, and it is also very fast in implementation. However, several statistical weaknesses have been found in the RC4 cipher [1, 10], but even today, in 2015, it remains the most widely used stream cipher by volume of internet transactions.
One year after the specification of the RC4 was made publicly available, Andrew Roos [18] made an observation that the initial few bytes of the output stream of the RC4 had a significant correlation with the key being used. This observation was exploited by many researchers in subsequent years. Mantin and Shamir [13] mounted a distinguishing attack of the RC4 which showed that an attacker can obtain many short-size output streams using randomized keys. Paul and Preneel [17] later demonstrated that a distinguisher requires a total of 225 output bytes (2 bytes per key, 224 keys in total). Further works were conducted on this, such as by Mantin in 2005 [12].
The fact that the KSA permutation was not random had been observed by Paul, Maitra and Srivastava in 2007 [16], which showed significant biases of each permutation byte that are independent of the secret key. This led to follow-up work in 2014 by Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul and Santanu Sarkar where they thoroughly analysed these biases [19].

The RC4 was suspected not to be one-one a long time ago [8], in fact, in 2008, Sebastiaan Indesteege and Bart Preneel [9] modelled the RC4 as a hash function and looked for collisions. It was not until 2009 that the key-lengths were brought down to feasible limits, when Mitsuru Matsui's work [14] was published, where he located a colliding 24-byte key, and a near colliding 20 byte key. Mitsuru Matsui proposed a key collision algorithm for the RC4 stream cipher after theorizing that the KSA algorithm was not one-one. This algorithm described a particular pair of keys type, and he described a recursive algorithm to find those near colliding keys.

The pair of keys Matsui chose differed in their last byte, with a hamming distance of 1, which is to say, if one key had in its last byte a value of x, the other key would have x+1.

This recursive algorithm worked on choosing the best direction in which to probe further, say a pair of key has been input into the search algorithm. The algorithm first looks for +x/-x difference versions of the same keys at different locations along the keys in order to find a near colliding key closeby. The KSA is tuned to return the iteration number at which it is evident that a particular pair of keys shall not (nearly) collide. Only the maximum returned values for neighboring keys are investigated further, by recursively introducing these keys into the same algorithm. This algorithm had fruitful results, and it resulted in finding a 24 byte collision and a 20 byte near collision.

Jiageng Chen and Atsuko Miyaji have contributed significantly to the area of RC4 key collisions as well, with their works in [3, 2] demonstrating that for keys to collide, they need not have a hamming distance of one, but can collide with various other hamming distances as well. In 2011, they proposed an alternative algorithm [5, 4, 7] in which the difference was kept on the third last byte of the key. This was done in order to be able to probabilistically bypass another round of the KSA. They also suggested several modifications of the search algorithm specifically tuned to finding collsions. The results of this was the discovery of a 22 byte near colliding key.

Another type of near collision from the ones discussed above are those that occur in the final keystream of the stream cipher. These have been discussed well in [15].

Relations between keystreams of different keys due to either types of collisions led to the mounting of several other attacks on the RC4 [6, 11], based on related keystreams.

While the results of the papers on key collisions of the RC4 stream cipher that were discussed above were all extremely interesting discoveries, they are a far cry from the (near) collisions that actually matter, that is, for 16 byte keys, which is the key-size used in practise by the RC4.

In this paper, we shall first discuss the distribution of the near colliding keys in the keyspace, which we discovered during our tests. These discoveries were further tested by applying them to a smaller scale RC4 to see whether similar patterns are followed.

In the second part of the paper, we propose an algorithm of our own. This particularly tackles 16 byte keys, as it is the only kind that matters. The algorithm

discussed shows improvements over prior algorithms, and may be used fruitfully in the future to obtain collisions/near collisions for 16 byte keys.

2 Clustering of Near Colliding Keys

As stated previously, the RC4 stream cipher has been found to be not one-one for keys of up to length 22, and near collisions (defined here as keys that generate initial states that differ in not more than 2 positions) have been located for up to length 20 bytes. Over here, we discuss the relations between near colliding keys in the RC4 stream cipher.
Mitsuru Matsui proposed a collision search algorithm which helped him locate one 24-byte colliding key and one 20 byte near colliding key. However, Matsuis algorithm for collision search can be heavily modified to give faster results. Chen-Miyaji in their 2011 paper applied several of those modifications, resulting in a faster collision search algorithm, that which resulted in finding a 22-byte key collision.
On using a modified version of Matsuis algorithm, we located several near-collisions of 24 bytes. These collisions seem rather interesting, as they all form in clusters. The clusters consist of key pairs that differ slightly from one another, **with the same key sum**.
We found several near-colliding keys that occur in large clusters. The reason for this is that modifying key bytes at two subsequent positions by +x and -x yield related initial states, and is discussed in detail in the next section.
However the exact distribution of these clusters is of importance. This has also been tested in this paper, in a later section, with a miniature RC4.

2.1 Clusters Observed

The clusters that were observed in searching the key space for 24 byte keys shall be displayed in this section. Each key byte is represented in decimal, and not in hexadecimal. This is for ease of reading.
The following table consists of a set of a few representative keys for each cluster that we observed.

Cluster	Key
1	0 220 185 236 240 98 223 61 227 188 157 75 72 185 209 237 43 229 154 55 89 155 208 45(46)
	0 220 185 236 240 98 223 61 227 188 157 75 72 185 209 237 43 229 154 55 89 157 206 45(46)
	0 220 185 236 240 98 223 61 227 188 157 75 72 185 209 237 43 229 154 55 89 158 205 45(46)
	0 220 185 236 240 98 223 61 227 188 157 75 72 185 209 237 43 229 154 55 89 159 204 45(46)
2	0 225 212 184 215 64 244 111 72 42 66 123 214 117 46 234 137 79 241 136 48 44 91 130 (131)
	0 225 212 184 215 64 244 111 72 42 66 123 214 117 46 234 137 79 241 136 48 30 105 130 (131)
3	0 225 12 100 229 115 244 48 129 32 3 40 114 86 58 232 127 79 185 241 66 235 91 128 (129)
	0 225 12 100 229 115 244 48 129 32 49 250 114 86 58 241 118 79 185 241 66 151 175 128 (129)
	0 225 12 100 229 115 244 48 129 32 11 32 114 86 58 246 113 79 185 241 66 150 176 128 (129)
	0 225 12 100 229 115 244 48 129 32 11 32 114 86 58 246 113 79 185 241 66 174 152 128 (129)

Cluster 1 consists of roughly 58 keys that nearly collide, of which only a handful has been shown here. Modifying the 2nd last and third last bytes of the key

will give you many more near colliding keys. Also, some of these keys generate initial states that match in almost 240 positions with the keys of a different near colliding pair.
Cluster 2 is a sparse cluster, consisting of just 2 near colliding keys (more may be found using a deeper search).
Cluster 3 is slightly denser than the previous one, and comprises of around 4 keys that have been located (more may be present).
These clusters are evident of certain facts. It is known that a change of +x/-x in a key gives rise to related initial states. In fact, it was Matsuis initial idea that key collisions would be found in making a key differ from another by +x/-x on two different bytes of the key. But from the clusters that we have found, we speculate that the key near-collisions, and perhaps even collisions, occur in groups, owing to the related initial states, and the above observations.

2.2 Search Algorithm

The algorithm used to locate these clusters is described as follows:
1) Initialize a key, making sure that $K[L-1] = 256-j$, and $K_2[L-1] = 257-j$. L denotes the length of the key. The algorithm is so designed so that $K[L-1]$ is not swapped out before $i = L-1$. Although this requires the first L steps of KSA to be performed, it is not done in the KSA. *
2) Search (K, K_2)
3) y-for loop running from 0-255
4) nested within y loop, x-for loop running from 1 to $L-2$
5) Alter $K[x]$ as $K[x] = K[x] + y$. Do the same for $K_2[x]$.
6) Alter $K[x+1]$ as $K[x+1] = K[x+1] - y$. Do the same for $K_2[x+1]$.
7) Run KSA on K, K_2**
8) From return value of KSA, set a MaxS(as according to Matsuis algorithm).
9) end for loops
10) y-loop running from 0-255
11) nested within y loop, x-loop running from 1 to $L-2$
12) Alter $K[x]$ as $K[x] = K[x] + y$. Do the same for $K_2[x]$.
13) Alter $K[x+1]$ as $K[x+1] = K[x+1] - y$. Do the same for K_2[x+1].
14) Run KSA on K, K_2**
15) If the S-Value of the K, K_2 pair is equal to MaxS, Recursively call Search(K, K_2). Recursive depth is set to 10.
16) end for loops

*If the $K[L-1] = 256-j$ and $K_2[L-1] = 257-j$ step is done in the KSA, we risk ruining the cluster, as any changes in the Key in the KSA function would be reflected back to the Key in the search algorithm. This is because K[L-1] needs to be fixed in a specific cluster.
**The KSA algorithm used here checks the number of steps of KSA that can run before the initial states of the 2 keys differ by more than 2 positions, as according to Matsuis algorithm. This can be made faster by a worst case of $O(2^L)$ probability improvement, by setting specific conditions on j, as according

to Chen-Miyajis 2011 paper. The function returns the value $i-1$, and stops KSA at the point where more than 2 initial states differ. If it survives all the steps, it is printed as a collision / near collision, with the string distance.

2.3 Views

These clusters are evident of certain facts. It is known that a change of $+x/-x$ in a key gives rise to related initial states. In fact, it was Matsuis initial idea that key collisions would be found in making a key differ from another by $+x/-x$ on two different bytes of the key. But from the clusters that we have found, we speculate that the key near-collisions, and perhaps even collisions, occur in groups, owing to the related initial states, and the above observations.
Hence, it raises the question on whether one will be able to construct more collisions/near-collisions from the collisions/near-collisions already located, or *develop a search algorithm that specifically looks through clusters when a certain number of rounds of the KSA has been passed successfully.*
In the following section, we shall attempt to test the clustering hypothesis on key-spaces we can exhaustively search through, using a small-scale RC4.

2.4 Testing hypothesis using a smaller scale RC4

To test the clustering hypothesis which was formulated on the basis of the observations noticed for 24 byte keys, we ran a modified smaller-scale RC4 on Matsuis key type. This was done for 2 key lengths, 7 word and 8 word, and keys with word lengths of 32 bits.
As such, for this miniaturized RC4, the state array for the RC4 permutation was also of length 32, and the KSA algorithm for this RC4 ran for 32 iterations per call.
The $256-j$ and $257-j$ modification of the last word of the key during KSA for Matsui's algorithm that was suggested in Chen-Miyaji's paper was trimmed from the algorithm as the goal was an exhaustive search throughout the keyspace in accordance with Matsui's key type, and all such near collisions were enumerated.
The columns of the following tables enumerate the number of near collisions for a specific last word of the second key, ie, the key word that has been modified as $x, x+1$ in the two keys used to compare for a near collision. The word depicted by the columns is the $x+1$ word. The rows are all the collisions for a specific key sum.

For 7 word keys :

Key Sum	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Sum 18	52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sum 21	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sum 22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sum 50	552	0	17	15	12	11	8	7	5	1	1	0	0	0	1	0	0	23	7	0	0	0	0	0	0	33	0	0	0	0	0	1
Sum 51	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sum 52	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	11	0	0	0	52	0	0	0	0	0	0	2	0
Sum 53	7	0	5	3	5	5	5	5	5	4	4	5	5	4	2	3	0	5	10	1	0	0	0	0	0	0	0	0	0	1	0	10
Sum 54	0	0	7	6	7	7	6	8	7	5	5	7	6	11	25	12	1	16	2	5	1	1	0	0	0	45	0	0	0	0	0	1
Sum 55	3	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0
Sum 56	0	0	1	1	1	1	1	1	1	1	1	2	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Sum 57	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0
Sum 82	728	0	178	175	172	166	162	150	142	153	141	121	94	99	68	81	81	138	88	39	43	53	1	0	10	1611	0	0	39	190	116	96
Sum 83	22	0	23	24	24	24	24	23	22	21	16	14	19	11	9	5	18	20	16	18	13	0	0	0	200	32	0	0	1	27	44	5
Sum 84	10	0	2	2	2	2	2	1	1	2	2	2	1	2	11	1	8	9	0	7	1	0	7	176	8	89	0	0	36	23	56	7
Sum 85	0	0	0	0	0	0	0	0	0	0	0	0	0	1	14	18	37	52	34	35	5	0	61	0	18	21	0	0	3	21	15	29
Sum 86	0	0	1	1	1	1	1	1	1	1	1	0	1	1	38	26	18	21	10	21	5	2	0	0	4	80	0	0	3	28	2	16
Sum 87	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	2	2	3	0	0	0	4	1	2	0	0	3	2	4	6
Sum 88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	4	0	0	0	0	0	1	1	0
Sum 89	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0
Sum 114	167	0	108	115	121	126	131	141	134	146	142	109	106	189	227	275	194	383	335	167	197	97	78	22	26	1450	0	0	257	423	347	187
Sum 115	1	0	3	3	3	3	3	2	3	3	3	5	4	8	16	2	23	27	14	16	11	9	2	3	234	109	0	0	51	32	121	64
Sum 116	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	70	1	35	0	0	13	13	29	9
Sum 117	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	15	0	5	40	0	0	2	3	4	7
Sum 118	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3	3	58	0	0	0	18	3	0
Sum 119	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0
Sum 120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Sum 121	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
Sum 146	0	0	0	0	0	0	0	0	0	0	4	4	9	18	36	42	10	68	72	25	67	29	37	4	0	305	0	0	44	43	150	76
Sum 147	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	9	4	1	0
Sum 149	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 1

For 8 word keys :

Key Sum	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Sum 14	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sum 15	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sum 16	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sum 44	250	5	444	399	353	296	302	249	213	200	186	172	114	89	70	98	34	13	7	44	59	6	0	13	3	0	0	0	0	0	0	0
Sum 45	70	0	177	161	147	137	104	112	105	62	102	175	28	28	20	4	10	9	9	52	26	0	1	5	6	0	0	1	0	0	0	0
Sum 46	80	5	144	131	124	121	120	86	83	53	58	86	22	28	22	8	4	0	2	35	29	0	7	0	0	0	0	0	0	1	8	0
Sum 47	127	0	116	120	105	97	94	88	83	79	62	68	10	34	11	4	1	0	1	4	2	10	1	2	0	0	0	4	0	1	7	6
Sum 48	87	0	38	43	42	42	43	28	15	26	25	68	10	34	11	4	1	0	1	4	2	10	1	2	0	0	0	4	0	1	7	6
Sum 49	24	0	13	13	11	11	10	9	5	6	4	54	20	3	38	18	12	18	2	22	1	8	7	6	34	0	0	1	7	16	0	0
Sum 50	8	1	2	2	2	2	2	2	1	1	1	0	1	0	1	2	0	0	1	3	0	0	0	2	0	0	0	0	0	0	0	0
Sum 51	13	0	3	3	3	3	3	2	3	3	3	0	2	1	2	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Sum 76	30063	36	4209	4101	3912	3820	3761	3354	3103	3087	3120	3862	2485	1932	1627	1317	785	954	1442	2839	5774	2634	1839	1348	2082	0	0	794	636	375	588	318
Sum 77	636	22	1127	1156	1139	1139	1073	1011	966	855	1014	1085	1017	562	530	271	417	267	496	1211	1005	669	288	231	126	0	0	102	86	46	62	132
Sum 78	495	6	591	596	581	563	546	557	407	557	551	749	561	341	329	335	391	182	262	547	1032	437	394	79	157	0	0	85	162	122	71	65
Sum 79	444	3	691	683	671	685	645	563	521	551	559	811	393	339	267	338	139	417	324	528	227	325	349	320	88	0	0	144	108	97	68	169
Sum 80	351	12	176	172	166	166	154	161	146	164	161	530	134	217	77	181	128	75	42	180	100	263	216	214	95	0	0	66	39	172	68	111
Sum 81	59	1	58	53	54	61	60	60	56	67	59	169	161	95	194	84	75	47	42	540	156	267	163	171	371	0	0	11	59	25	19	35
Sum 82	34	0	8	8	8	9	10	8	7	8	1	12	43	61	7	23	6	10	4	22	21	22	31	6	22	0	0	8	3	0	1	0
Sum 83	28	0	1	1	1	1	1	1	0	1	1	17	1	4	17	3	5	9	4	33	8	11	40	2	18	0	0	14	1	2	15	8
Sum 84	0	0	10	9	9	9	6	8	8	6	6	6	4	3	2	0	0	0	2	5	0	0	2	1	2	0	0	0	0	0	0	0
Sum 85	0	0	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	0	1	5	2	3	0	0	0	0	0	0	0
Sum 86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
Sum 108	4325	38	5652	5675	5621	5909	5696	5963	5799	5967	6388	10298	8073	5837	5632	4500	2994	3327	5519	14178	16645	9500	6957	8664	6791	0	0	4468	3422	2384	3542	2176
Sum 109	453	10	934	952	945	945	1030	1079	1093	1033	1125	1617	1309	932	590	423	683	538	1254	2856	2818	1348	877	991	600	0	0	432	477	387	270	459
Sum 110	340	10	533	539	552	550	594	571	468	598	573	969	722	391	391	399	799	405	739	1516	2847	1629	1416	468	594	0	0	426	367	511	708	361
Sum 111	268	1	492	492	516	518	515	614	641	715	658	579	513	246	418	879	623	514	821	1113	670	1272	786	878	500	0	0	434	493	404	475	555
Sum 112	302	5	113	118	128	133	129	142	138	156	146	831	109	283	128	217	213	435	203	759	367	1077	592	633	331	0	0	496	232	466	208	287
Sum 113	24	0	19	19	19	17	20	18	19	19	20	135	165	21	49	47	40	25	44	339	388	412	594	279	235	0	0	77	211	89	95	56
Sum 114	0	0	7	6	6	6	7	6	5	5	1	6	6	5	17	25	3	2	8	35	32	50	51	40	38	0	0	12	9	10	5	20
Sum 115	4	0	0	1	1	1	1	1	1	1	1	1	0	1	14	1	3	1	1	20	8	19	20	6	4	0	0	7	3	5	9	12
Sum 116	1	0	2	3	4	5	5	1	8	9	6	9	8	9	1	0	0	0	10	2	7	12	4	2	6	0	0	0	12	0	0	0
Sum 117	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	0	0	0	0	1	0
Sum 140	755	7	743	809	855	917	956	1143	1344	1423	1532	3060	3846	2063	2120	1890	1961	2365	3811	8846	9451	4579	3422	6157	3964	0	0	3844	4271	2715	4227	2730
Sum 141	21	0	62	71	79	59	81	95	117	104	123	134	361	96	127	146	159	161	562	750	559	353	463	683	471	0	0	270	325	320	411	297
Sum 142	5	0	15	18	21	26	33	60	64	75	76	164	188	52	30	51	178	163	284	380	549	415	505	166	200	0	0	292	343	282	390	217
Sum 143	5	0	2	2	4	5	7	8	11	9	16	250	46	31	31	172	185	109	131	589	296	491	161	251	85	0	0	288	176	96	190	233
Sum 144	1	0	0	0	0	0	0	0	0	0	0	3	18	1	19	14	17	28	52	183	235	173	196	199	98	0	0	183	75	106	86	167
Sum 145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	6	43	14	65	61	44	0	0	8	16	19	17	14
Sum 146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	1	3	7	8	14	1	0	0	3	6	6	7	5
Sum 147	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	3	1	1	0	0	2	1	3	1	4
Sum 148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	3	3	1	0
Sum 149	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
Sum 172	0	0	0	0	0	0	0	0	0	0	0	4	3	14	69	53	165	197	204	533	563	289	117	207	255	0	0	271	274	296	613	456
Sum 173	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	5	22	35	30
Sum 174	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	3	3	1	10	2
Sum 175	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	2	0	0	2	2	0	1	3
Sum 176	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	0	0	0	2	0

Figure 2

The distributions are reminiscent of the distribution we had noticed for 24 byte keys. Some relatively large clusters (of a fixed key sum, which implies small changes to the individual words) interspersed with sparser clusters, while certain regions are devoid of any near collisions.

2.5 Results of the Small Scale simulation

Once again, it is evident that the keys clearly appear in clusters. Among the large number (256 for the 8 word case and 224 for the 7 word case) of possible key sums, only a fraction of these are occupied by near colliding keys (47 for the 8 word case and 30 for the 7 word case). Moreover, the ones that are occupied usually create large clusters around a specific last byte (the byte that has the d/d+1 difference). These results are a clear depiction of the cluster distribution, with many large clusters with thousands of near colliding keys, and various sparse clusters, with only a few such near colliding keys. The distribution is similar to what we had observed with the regular RC4 on 24 byte keys.
Another important observation was that no complete collisions were observed. This may give us a reason to believe that Matsui's approach isn't the correct way to look for collisions.

3 Faster Collision Search for 16-byte Keys

In all the search algorithms described so far, people have dealt with key sizes>16 bytes, and hence chosen to create the difference only in the last few bytes of the key.
In Chen-Miyajis 2011 paper, they have devised a way to probabilistically skip the second round of KSA, by modifying K[d+2], where d=L-3. However, it is possible to almost deterministically skip the second round of KSA, by using the difference in Key 1 and Key 2 on the 1st byte of the key. But this is viable for only 16-byte keys, as for bigger key lengths, the first byte enters KSA an additional round.
It is shown as follows :
$K[0]$=0, $K_2[0]$=1
$K[1] = L - d - 1 = L - 1$
$K[L-1] = 2 \times L - j_{i=L-2} - s[L-1] - s[L] - K[0]$ $(mod\ 256)$
By making this modification, the key almost deterministically (with a probability of $(\frac{255}{256})^{L-2}$) passes the 2nd round.
All of the conditions on j stated in Chen-Miyaji's 2011 paper can also be modified in accordance to the difference being in the first byte, and imported to this idea.
We are limited by today's computer hardware in searching for key collisions of 16-bytes, and we hope this may be applicable in the future.

3.1 Search Algorithm

The proposed algorithm is given as follows :
1) The random key generated is first pre-constructed, by setting $K[L-1]$ to $2 \times L - j_{i=L-2} - S[L-1] - S[L] - K[0] \pmod{256}$, during a mock L iterations of KSA.
2) This key is then run through the same search algorithm proposed in the previous section.
3) The KSA modifies the L-1$^{\text{th}}$ byte of the key according to $K[L-1] = 2 \times L$ -$j_{i=L-2}$ -$S[L-1] - S[L] - K[0] \pmod{256}$. This has to be done again as, during the search through various modifications of the originally pre-constructed key, this condition might not be maintained.

3.2 Results and Discussions

For the time, with the computation efficiency that is available to us, the algorithm was run for 16-byte keys over 144 rounds of KSA (instead of 256). The following results were found (within a few seconds).
0(1) 15 250 44 152 103 197 60 59 84 97 203 136 140 223 214
0(1) 15 41 205 255 35 142 97 160 194 154 15 207 20 218 219
0(1) 15 250 170 29 44 42 184 49 211 191 201 233 211 181 226
Many more such keys were located, many forming localized clusters, as was seen in the previous section of this paper.

3.3 Comparison to previous algorithms

It is clear, that in comparison to the algorithm where the last byte of the second key has the 1 bit difference, this algorithm runs **1 less round of the KSA** (where a round consists of L iterations, and L is the length of the key).
This is because by modifying the last byte of the key during the first round of KSA, we **almost deterministically** (with a probability of $(\frac{255}{256})^{L-2}$, similar to the probability of the deterministic bypass claimed by Chen-Miyaji with the modification 255-j) pass the second round of KSA (the first round is passed by setting L-d-1 to the d+1$^{\text{th}}$ byte of the key, that is L-1 to position 1. In comparison, Matsui's approach **only passes the first round deterministically**. Passing an additional round in the KSA has a significant impact in the time taken to find a solution. It is also clear that this algorithm runs better than the last byte modification algorithm only when i%L==0, where i is the total number of iterations of the KSA, as for other key lengths the first byte has to undergo an extra round as it is. However, it seems completely pointless to find near-collisions of any size apart from 16 bytes, as that is the size intended for the RC4 stream cipher.
As 32 byte near-collisions are found too fast in both algorithms to have a basis for comparison, we try 28 byte keys on a 252-iteration KSA. On an i5-3337u laptop processor, near collisions were found after over 10 seconds for the last byte modification algorithm (Matsui's, with a few tweaks taken from Chen-Miyaji's

proposed algorithm) and **under half a second for the algorithm proposed here**.

In comparison to Chen-Miyaji's probabilistic bypassing of the second round, this algorithm performs better, as the probability for them to bypass the second round is : $\prod_{i=1}^{L-2} \frac{256-i}{256}$
This is equal to 0.61 for a 16 byte key.
For our algorithm, the probability of passing this round is equal to $(\frac{255}{256})^{L-2}$, which is equal to 0.9467 for a 16 byte key, thereby making the likelihood 1.55 times more.
However, the probabilistic bypass of Chen-Miyaji's algorithm could also be applied to our algorithm, as there are several available key bytes.

3.3.1 Applying Chen-Miyaji's modification to our algorithm

If the last byte modification of our scheme is shifted to the second last byte, and the modification of Chen-Miyaji's paper, $K[L-1] = j_{2L-3}$ - j_{L-2} - $\sum_{i=0}^{L-3} K[i]$ - $\sum_{i=L-1}^{2L-3} S_{L-2}[i]$ (Section 4.2 of [5] discusses this modification in detail) applied to our algorithm, we obtain the following equations that need to be satisfied (k is the length of the key).
(i)$j_{L-3}+K[L-2]+S_{L-2}[L-2]+K[L-1]+S_{L-2}[L-1]+K[0]+S[k] = 2 \times L$
(ii)$j_{L-3}+S_{L-2}[L-2]+S_{L-2}[L-1]+2(K[L-2]+K[L-1])+ \sum_{i=0}^{L-3} K[i] + \sum_{i=L}^{2L} S_{L-2}[i] + K[0] = 3 \times L$
Which basically boils down to satisfying the following equations :
(i)$j_{L-3}+K[L-2]+S[L-2]+K[L-1]+S[L-1]+K[0]+S[L] = 2 \times L$
(iii)$\sum_{i=1}^{L-1} K[i] + \sum_{i=L+1}^{2L} S_{L-2}[i] = L$
These need to be satisfied after the swap at $i = L-3$, if we succeed, we may probabilistically bypass another round, thereby leading to an improvement nearly of the order of 2^{-8} over Chen-Miyaji's algorithm.
The probability of successfully bypassing the second and third round combined is $\prod_{i=1}^{k+2} \frac{256-i}{256}$, Which comes to 0.504 for 16 byte keys, which is an improvement of more than 211 times over Chen Miyaji's algorithm. However there are additional difficulties in making this work.
For this to work, we need to engineer enough swaps in the S-box within $k+1$ to $2k$ using the key-bytes available to us, such that both the above equations, i.e. (i) and (iii), are satisfied. However, this is not always possible, and may require re-initiation of keys till a satisfying pair is found.

3.4 Complexity Analysis

When we modify the last byte of a key and look for a near-collision, we need a swap to occur at an exact position in the initial state. This means that K[15] needs to swap with K[31] for the near collision to propagate forward. For a 16 byte key, round 1 is bypassed automatically. Thereafter K[31] needs to swap with K[47], and there are 2^8 other possibilities. 14 such rounds need to be passed, making the complexity of finding a near collision 2^{112} for 16-byte keys. However, for the first byte modification algorithm proposed here, the number

of rounds that need to be passed is 13, which gives rise to a probability of 2^{104} for 16-byte keys, an improvement in the order of 2^8. Also, 3 of the 16 key bytes are fixed now, as compared to 2 in the last byte modification algorithm, which means the number of bytes that are to be randomly generated is reduced, thereby reducing the search space.
On applying Chen-Miyaji's modification to the algorithm, we probabilistically bypass another round of the KSA, which leads to a probability of $2^{96.28}$. This is the best result for finding near colliding keys of length 16 bytes.

3.5 Other observations

The small-scale RC4 was also used to test our proposed search algorithm.
The first byte difference algorithm was also tested in comparison to Matsui's algorithm. The search was exhaustive, and the probabilistic steps of $K[L-1]$ = 256-$j_{i=L-2}$ (mod 256) for Matsui's algorithm and K[L-1]= $2 \times L - j_{i=L-2} - s[L-1] - s[L] - K[0]$ (mod 256) for the algorithm proposed in this paper were removed. The results gave us around 500000 near colliding keys among 2^{35} candidate pairs for Matsui's algorithm, and nearly 300000 near colliding keys among 2^{30} candidate pairs for our algorithm, which gives us a 19.2 times better probability of near collision for our algorithm.

4 Conclusion

From the facts discussed in this paper, we notice two primary things.
1) Near Collisions occur in clusters. This is most likely due to the nature of generating related permutations when a key is modified at two locations with $+x$ and $-x$.
2) The first byte of a key offers us control over one extra round of the KSA. This may be properly utilized in the future to generate collisions and near collisions of 16-bytes, which is the key-length that actually matters, and has eluded us so far.

5 Acknowledgements

The author would like to thank Sourav Sen Gupta for his guidance regarding the experiments performed in this paper.

References

[1] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C.N. Schuldt. "on the security of rc4 in tls and wpa". 2013. "USENIX Security Symposium 2013".

[2] Jiageng Chen and Atsuko Miyaji. Generalized rc4 key collisions and hash collisions. In *Proceedings of the 7th International Conference on Security*

and Cryptography for Networks, SCN'10, pages 73–87, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Jiageng Chen and Atsuko Miyaji. *A New Class of RC4 Colliding Key Pairs with Greater Hamming Distance*, pages 30–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[4] Jiageng Chen and Atsuko Miyaji. Generalized analysis on key collisions of stream cipher rc4. pages 2194–2206, 2011.

[5] Jiageng Chen and Atsuko Miyaji. *How to Find Short RC4 Colliding Key Pairs*, pages 32–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6] Jiageng Chen and Atsuko Miyaji. *A New Practical Key Recovery Attack on the Stream Cipher RC4 under Related-Key Model*, pages 62–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[7] Jiageng Chen and Atsuko Miyaji. Novel strategies for searching {RC4} key collisions. *Computers & Mathematics with Applications*, 66(1):81 – 90, 2013.

[8] A.L. Grosul and D.S. Wallach. A related-key cryptanalysis of rc4. 2000.

[9] Sebastiaan Indesteege and Bart Preneel. *Collisions for RC4-Hash*, pages 355–366. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[10] Poonam Jindal and Brahmjit Singh. Rc4 encryption-a literature survey. *Procedia Computer Science*, 46:697 – 705, 2015.

[11] Subhamoy Maitra, Goutam Paul, Santanu Sarkar, Michael Lehmann, and Willi Meier. *New Results on Generalization of Roos-Type Biases and Related Keystreams of RC4*, pages 222–239. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[12] Itsik Mantin. Predicting and distinguishing attacks on rc4 keystream generator. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 491–506, Berlin, Heidelberg, 2005. Springer-Verlag.

[13] Itsik Mantin and Adi Shamir. *A Practical Attack on Broadcast RC4*, pages 152–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[14] Mitsuru Matsui. *Key Collisions of the RC4 Stream Cipher*, pages 38–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[15] Willi Meier. Near-colliding keys in rc4. 2013. ESC 2013.

[16] Goutam Paul, Subhamoy Maitra, and Rohit Srivastava. *On Non-randomness of the Permutation After RC4 Key Scheduling*, pages 100–109. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[17] Souradyuti Paul and Bart Preneel. *A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher*, pages 245–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[18] Andrew Roos. A class of weak keys in the rc4 stream cipher. 1995.

[19] Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. (non-)random sequences from (non-)random permutations—analysis of rc4 stream cipher. *Journal of Cryptology*, 27(1):67–108, 2014.