Weaknesses in the Key Scheduling Algorithm of RC4

Scott Fluhrer[1], Itsik Mantin[2], and Adi Shamir[2]

[1] Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134
sfluhrer@cisco.com
[2] Computer Science department, The Weizmann Institute, Rehovot 76100, Israel.
{itsik,shamir}@wisdom.weizmann.ac.il

Abstract. In this paper we present several weaknesses in the key scheduling algorithm of RC4, and describe their cryptanalytic significance. We identify a large number of weak keys, in which knowledge of a small number of key bits suffices to determine many state and output bits with non-negligible probability. We use these weak keys to construct new distinguishers for RC4, and to mount related key attacks with practical complexities. Finally, we show that RC4 is completely insecure in a common mode of operation which is used in the widely deployed Wired Equivalent Privacy protocol (WEP, which is part of the 802.11 standard), in which a fixed secret key is concatenated with known IV modifiers in order to encrypt different messages. Our new passive ciphertext-only attack on this mode can recover an arbitrarily long key in a negligible amount of time which grows only linearly with its size, both for 24 and 128 bit IV modifiers.

1 Introduction

RC4 is the most widely used stream cipher in software applications. It was designed by Ron Rivest in 1987 and kept as a trade secret until it leaked out in 1994. RC4 has a secret internal state which is a permutation of all the $N = 2^n$ possible n bits words, along with two indices in it. In practical applications $n = 8$, and thus RC4 has a huge state of $log_2(2^8! \times (2^8)^2) \approx 1700$ bits.

In this paper we analyze the Key Scheduling Algorithm (KSA) which derives the initial state from a variable size key, and describe two significant weaknesses of this process. The first weakness is the existence of large classes of weak keys, in which a small part of the secret key determines a large number of bits of the initial permutation (KSA output). In addition, the Pseudo Random Generation Algorithm (PRGA) translates these patterns in the initial permutation into patterns in the prefix of the output stream, and thus RC4 has the undesirable property that for these weak keys its initial outputs are disproportionally affected by a small number of key bits. These weak keys have length which is divisible by some non-trivial power of two, i.e., $\ell = 2^q m$ for some $q > 0$[1]. When

[1] Here and in the rest of the paper ℓ is the number of words of K, where each word contains n bits.

$RC4_n$ uses such a weak key of ℓ words, fixing $n + q(\ell - 1) + 1$ bits of K (as a particular pattern) determines $\Theta(qN)$ bits of the initial permutation with probability of one half and determines various prefixes of the output stream with various probabilities (depending on their length).

The second weakness is a related key vulnerability, which applies when part of the key presented to the KSA is exposed to the attacker. It consists of the observation that when the same secret part of the key is used with numerous different exposed values, an attacker can rederive the secret part by analyzing the initial word of the keystreams with relatively little work. This concatenation of a long term secret part with an attacker visible part is a commonly used mode of RC4, and in particular it is used in the WEP (Wired Equivalent Privacy) protocol, which protects many wireless networks. Our new attack on this mode is practical for any key size and for any modifier size, including the 24 bit recommended in the original WEP and the 128 bit recommended in the revised version WEP2.

The paper is organized in the following way: In Section 2 we describe RC4 and previous results about its security. In Section 3 we consider a slightly modified variant of the Key Scheduling Algorithm, called KSA*, and prove that a particular pattern of a small number of key bits suffices to completely determine a large number of state bits. Afterwards, we show that this weakness of KSA*, which we denote as the *invariance weakness*, exists (in a weaker form) also in the original KSA. In Section 4 we show that with high probability, the patterns of initial states associated with these weak keys also propagate into the first few outputs, and thus a small number of weak key bits determine a large number of bits in the output stream. In Section 5 we describe several cryptanalytic applications of the invariance weakness, including a new type of distinguisher. In Sections 6 and 7 we describe the second weakness, which we denote as the *IV weakness*, and show that a common method of using RC4 is vulnerable to a practical attack due to this weakness. In Section 8, we show how both these weaknesses can separately be used in a related key attack. In the appendices, we examine how the IV weakness can be used to attack a real system (appendix A), how the invariance weakness can be used to construct a ciphertext-only distinguisher and to prove that RC4 has low sampling resistance (appendices B and C), and how to derive the secret key from an early permutation state (appendix D).

2 RC4 and Its Security

2.1 Description of RC4

RC4 consists of two parts (described in Figure 1): A key scheduling algorithm KSA which turns a random key (whose typical size is 40-256 bits) into an initial permutation S of $\{0, \ldots, N-1\}$, and an output generation part PRGA which uses this permutation to generate a pseudo-random output sequence.

The PRGA initializes two indices i and j to 0, and then loops over four simple operations which increment i as a counter, increment j pseudo randomly, exchange the two values of S pointed to by i and j, and output the value of S pointed to by $S[i]+S[j]$[2]. Note that every entry of S is swapped at least once (possibly with itself) within any N consecutive rounds, and thus the permutation S evolves fairly rapidly during the output generation process.

The KSA consists of N loops that are similar to the PRGA round operation. It initializes S to be the identity permutation and i and j to 0, and applies the PRGA round operation N times, stepping i across S, and updating j by adding $S[i]$ and the next word of the key (in cyclic order). We will call each round of KSA a *step*.

KSA(K)	PRGA(K)
Initialization:	Initialization:
For $i = 0 \ldots N-1$	$i = 0$
$S[i] = i$	$j = 0$
$j = 0$	Generation loop:
Scrambling:	$i = i + 1$
For $i = 0 \ldots N-1$	$j = j + S[i]$
$j = j + S[i] + K[i \bmod \ell]$	Swap$(S[i], S[j])$
$Swap(S[i], S[j])$	Output $z = S[S[i] + S[j]]$

Fig. 1. The Key Scheduling Algorithm and the Pseudo-Random Generation Algorithm

2.2 Previous Attacks on RC4

Due to the huge effective key of RC4, attacking the PRGA seems to be infeasible (the best known attack on this part requires time that exceeds 2^{700}). The only practical results related to the PRGA deal with the construction of distinguishers. Fluhrer and McGrew described in [FM00] how to distinguish RC4 outputs from random strings with 2^{30} data. A better distinguisher which requires 2^8 data was described by Mantin and Shamir in [MS01]. However, this distinguisher could only be used to mount a partial attack on RC4 in broadcast applications.

The fact that the initialization of RC4 is very simple stimulated considerable research on this mechanism of RC4. In particular, Roos discovered in [Roo95] a class of weak keys that reduces their effective size by five bits, and Grosul and Wallach showed in [GW00] that for large keys whose size is close to N words, RC4 is vulnerable to a related key attack.

More analysis of the security of RC4 can be found in [KMP+98], [Gol97] and [MT98].

---

[2] Here and in the rest of the paper all the additions are carried out modulo N

3 The Invariance Weakness

Due to space limitations we prove here the invariance weakness only for a simplified variant of the KSA, which we denote as KSA* and describe in Figure 2. The only difference between them is that KSA* updates i at the *beginning* of the loop, whereas KSA updates i at the *end* of the loop. After formulating and proving the existence of this weakness in KSA*, we describe the modifications required to apply this analysis to the real KSA.

KSA(K)[a]	KSA*(K)

For $i = 0 \ldots N-1$
    $S[i] = i$
$i = 0$
$j = 0$
Repeat N times
    $j = j + S[i] + K[i \bmod \ell]$
    $Swap(S[i], S[j])$
    $i = i + 1$
	For $i = 0 \ldots N-1$
	    $S[i] = i$
	$i = 0$
	$j = 0$
	Repeat N times
	    $i = i + 1$
	    $j = j + S[i] + K[i \bmod \ell]$
	    $Swap(S[i], S[j])$

[a] KSA is rewritten in a way which clarifies the relation to KSA*

Fig. 2. KSA vs. KSA*

3.1 Definitions

Definition 1 *Let S be a permutation of $\{0, \ldots, N-1\}$, t be an index in S and b be some integer. Then if $S[t] \overset{\bmod b}{\equiv} t$, the permutation S is said to* b-conserve *the index t. Otherwise, the permutation S is said to* b-unconserve *the index t.*

Denote the permutation S and the indices i and j *after* round t of KSA* as S_t, i_t and j_t respectively. Denote the number of indices that a permutation b-conserves as $I_b(S)$. For the sake of simplicity, we often write I_t instead of $I_b(S_t)$.

Definition 2 *A permutation S of $\{0, \ldots, N-1\}$ is* b-conserving *if $I_b(S) = N$, and is* almost b-conserving *if $I_b(S) \geq N - 2$.*

Definition 3 *Let b, ℓ be integers, and let K be an ℓ words key. Then K is called a* b-exact *key if for any index t $K[t \bmod \ell] \equiv (1-t) \pmod b$. In case $K[0] = 1$ and $msb(K[1]) = 1$, K is called a* special b-exact *key.*

Notice that for this condition to hold, it is necessary (but not sufficient) that $b \mid \ell$.

3.2 The Weakness

Theorem 1 *Let $q \leq n$ and ℓ be integers and $b \stackrel{def}{=} 2^q$. Suppose that $b \mid \ell$ and let K be a b-exact key of ℓ words. Then the permutation $S = KSA^*(K)$ is b-conserving.*

Before getting to the proof itself, we will prove an auxiliary lemma

Lemma 1 *If $i_{t+1} \equiv j_{t+1} \pmod{b}$, then $I_{t+1} = I_t$.*

Proof: The only operation that might affect S (and maybe I) is the swapping operation. However, when i_{t+1} and j_{t+1} are equivalent $\pmod{b}$, S_{t+1} b-conserves i_{t+1} (j_{t+1}) if and only if S_t b-conserved j_t (i_t). Thus the number of indices S b-conserves remains the same. □

Proof:(of Theorem 1) We will prove by induction on t that for any $1 \leq t \leq N$, it turns out that $I_b(S_t) = N$ and $i_t \equiv j_t \pmod{b}$. This in particular implies that $I_N = N$, which makes the output permutation b-conserving.

For $t = 0$ (before the first round), the claim is trivial because $i_0 = j_0 = 0$ and S_0 is the identity permutation which is b-conserving for every b. Suppose that $j_t \equiv i_t$ and S_t is b-conserving. Then $i_{t+1} = i_t + 1$ and

$$j_{t+1} = j_t + S_t[i_{t+1}] + K[i_{t+1} \bmod \ell] \stackrel{\bmod b}{\equiv} i_t + i_{t+1} + (1 - i_{t+1}) = i_t + 1 = i_{t+1}$$

Thus, $i_{t+1} \equiv j_{t+1} \pmod{b}$ and by applying Lemma 1 we get $I_{t+1} = I_t = N$ and therefore S_{t+1} is b-conserving. □

KSA* thus transforms special patterns in the key into corresponding patterns in the initial permutation. The fraction of determined permutation bits is proportional to the fraction of fixed key bits. For example, applying this result to $\text{RC4}_{n=8,\ell=6}$ and $q = 1$, 6 out of the 48 key bits completely determine 252 out of the 1684 permutation bits.

3.3 Adjustments to KSA

The small difference between KSA* and KSA (see Figure 2) is essential in that KSA, applied to a b-exact key, does not preserve the equivalence $\pmod{b}$ of i and j even after the first round. Analyzing its execution on a b-exact key gives

$$j_1 = j_0 + S_0[i_1] + K[i_1] = 0 + S_0[0] + K[0] = K[0] \stackrel{\bmod b}{\equiv} 1 \stackrel{\bmod b}{\not\equiv} 0 = i_1$$

and thus the structure described in Section 3.2 cannot be preserved by the cyclic use of the words of K. However, the invariance weakness can be adjusted to the real KSA, and the proper modifications are formulated in the following theorem:

Theorem 2 *Let $q \leq n$ and ℓ be integers and $b \stackrel{def}{=} 2^q$. Suppose that $b \mid \ell$ and let K be a special b-exact key of ℓ words. Then*

$$Pr[KSA(K) \text{ is almost } b\text{-conserving}] \geq 2/5$$

when the probability is over the rest of the key bits.

Due to space limitations, the formal proof of this theorem (which is based on a detailed case analysis) will appear only in the full version of this paper. However, we can explain the intuition behind this theorem by concentrating on the differences between Theorems 1 and 2, which deal with KSA* and KSA respectively. During the first round, two deviations from KSA* execution occur. The first one is the non-equivalence of i and j which is expected to cause non-equivalent entries to be swapped during the next rounds, thus ruining the delicate structure that was preserved so well during KSA* execution. The second deviation is that S b-unconserves two of the indices, $i_1 = 0$ and $j_1 = K[0]$. However, we can cancel the ij discrepancy by forcing $K[0]$ (and j_1) to 1. In this case, the discrepancy in $S[j_1]$ ($K[1]$) causes an improper value to be added to j, thus repairing its non-equivalence to i during the second round. At this point there are still two unconserved indices, and this aberration is dragged across the whole execution into the resulting permutation. Although these corrupted entries might interfere with j updates, the pseudo-random j might reach them *before* they are used to update j (i.e., before i reaches them), and send them into a region in S where they cannot affect the next values of j[3]. The probability of this lucky event is amplified by the fact that the corrupted entries are $i_1 = 0$ which is not touched until the termination of the KSA due to its distance from the current location of i, and $j_2 = 1 + K[1] > N/2$ (recall that $msb(K[1]) = 1$), that is far from $i_1 = 2$, which gives j many opportunities to reach it before i does. The probability of $N/2$ pseudo random j's to reach an arbitrary value can be bounded from below by $2/5$, and extensive experimentation indicates that this probability is actually close to one half.

4 Key-Output Correlation

In this section we will analyze the propagation of the weak key patterns into the generated outputs. First we prove Claim 1 which deals with the highly biased behavior of a weakened variant of the PRGA, applied to a b-conserving permutation. Next, we will argue that the prefix of the output of the original PRGA is highly correlated to the prefix of the swapless variant (on the same initial permutation), which implies the existence of biases in the PRGA distribution for these weak keys.

Claim 1 *Let RC4* be a weakened variant of RC4 with no swap operations. Let* $q \le n$, $b \stackrel{def}{=} 2^q$ *and* S_0 *be a b-conserving permutation. Let* $\{X_t\}_{t=1}^{\infty}$ *be the output sequence generated by applying RC4* to* S_0, *and* $x_t \stackrel{def}{=} X_t \bmod b$. *Then the sequence* $\{x_t\}_{t=1}^{\infty}$ *is constant.*

Since there are no swap operation, the permutation does not change and remains b-conserving throughout the generation process. Notice that all the values

[3] if a value is pointed to by j before the swap, it will not be used as $S[i]$ (before the swap) for at least $N-1$ rounds, and in particular it will not affect the values of j during these rounds.

of S are known $\pmod b$, as well as the initial indices $i = j = 0 \equiv 0 \pmod b$, and thus the round operation (and the output values) can be simulated $\pmod b$, independently of S. Consequently the output sequence $\pmod b$ is constant, and deeper analysis implies that it is periodic with period $2b$, as exemplified in Figure 3 for $q = 1$.

i	j	$S[i]$	$S[j]$	$S[i] + S[j]$	Out
0	0	0	0	0	/
1	1	1	1	0	0
0	1	0	1	1	1
1	0	1	0	1	1
0	0	0	0	0	0
1	1	1	1	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Fig. 3. The rounds of RC4*, applied to a 2-conserving permutation

1^{st} word $\boxed{1}\boxed{\cdots}\boxed{1}\boxed{1}\boxed{1}$

2^{nd} word $\boxed{n}\boxed{\cdots}\boxed{3}\boxed{2}\boxed{1}$

3^{th} word $\boxed{n}\boxed{\cdots}\boxed{3}\boxed{2}\boxed{1}$

$\vdots$

ℓ^{th} word $\boxed{n}\boxed{\cdots}\boxed{3}\boxed{2}\boxed{1}$

Fig. 4. The stage in which each one of the bits is exposed during the related key attack

Recall that at each step of the PRGA, S changes in at most two locations, and thus we can expect the prefix of the output stream generated by RC4 from some permutation S_0, to be highly correlated with the stream generated from the same S_0 (or a slightly modified one) by RC4*. In particular the stream generated by RC4 from an almost b-conserving permutation is expected to be highly correlated with the constant substream $\{x_t\}$ from Claim 1. This correlation is demonstrated in Figure 8, where the function $h \longrightarrow Pr[1 \leq \forall t \leq h \; Z_t \equiv x_t \bmod 2^q]$ (for special 2^q-exact keys) is empirically estimated for $n = 8$, $\ell = 16$ and different q's. For example, a special 2-exact key completely determines 20 output bits (the lsb's of the first 20 outputs) with probability $2^{-4.2}$ instead of 2^{-20}, and a special 16-exact key completely determines 40 output bits (4 lsb's from each of the first 10 outputs) with probability $2^{-2.3}$, instead of 2^{-40}.

We have thus demonstrated a strong probabilistic correlation between some bits of the secret key and some bits of the output stream for a large class of weak keys. In the next section we describe how to use this correlation to cryptanalyze RC4.

5 Cryptanalytic Applications of the Invariance Weakness

5.1 Distinguishing RC4 Streams from Randomness

In [MS01] Mantin and Shamir described a significant statistical bias in the second output word of RC4. They used this bias to construct an efficient algorithm which distinguishes between RC4 outputs and truly random sequences by analyzing only one word from $O(N)$ different outputs streams. This is an extremely efficient distinguisher, but it can be easily avoided by discarding the first two words from each output stream. If these two words are discarded, the best known distinguisher requires about 2^{30} output words (see [FM00]). Our new observation yields a significantly better distinguisher for most of the typical key sizes. The new distinguisher is based on the fact that for a significant fraction of keys, a significant number of initial output words contain an easily recognizable pattern. This bias is flattened when the keys are chosen from a uniform distribution, but it does not completely disappear and can be used to construct an efficient distinguisher even when the first two words of each output sequence are discarded.

Notice that the probability of a special 2^q-exact key to be transformed into a 2^q-conserving permutation, does not depend of the key length ℓ (see Theorem 2). However, the number of predetermined bits is linear in ℓ, and consequently the size of this bias (and thus the number of required outputs) also depends on ℓ. In Figure 5 we specify the quantity of data required for a reliable distinguisher, for different key sizes. In particular, for 64 bit keys the new distinguisher requires only 2^{21} data instead of the previously best number of 2^{30} output words.

It is important to notice that the specified output patterns extend over several dozen output words, and thus the quality of the distinguisher is almost unaffected by discarding the first few words. For example, discarding the first two words causes the data required for the distinguisher to grow by a factor of between $2^{0.5}$ and 2^2 (depending on ℓ). Another important observation is that the biases in the lsb's distribution can be combined in a natural way with the biased distribution of the lsb's of English texts into an efficient distinguisher of RC4 streams from randomness in a ciphertext-only attack in which the attacker does not know the actual English plaintext which was encrypted by RC4. This type of distinguishers is discussed in Appendix B.

5.2 RC4 has Low Sampling Resistance

Biryukov, Shamir and Wagner defined in [BSW00] a new security measure of stream ciphers, which they denoted as their *Sampling Resistance*. The strong correlation between classes of RC4 keys and corresponding output patterns can be used to prove that RC4 has relatively low sampling resistance, which improves the efficiency of time/memory/data tradeoff attacks. Further details can be found in Appendix C.

6 RC4 Key Setup and the First Word Output

In this section, we consider related key attacks where the attacker has access to the values of all the bits of certain words of the key. In particular, we consider the case where the key presented to the KSA is made up of a *secret key* concatenated with an attacker visible value (which we will refer to as an Initialization Vector or IV). We will show that if the same secret key is used with numerous different initialization vectors, and the attacker can obtain the first word of RC4 output corresponding to each initialization vector, he can reconstruct the secret key with minimal effort. How often he can do this, the amount of effort and the number of initialization vectors required depends on the order of the concatenation, the size of the IV, and sometimes on the value of the secret key. This observation is especially interesting, as this mode of operation is used by several commercially deployed encryption systems ([Rei01], [LMSon]) and the first word of plaintexts is often an easily guessed constant such as the date, the sender's identity, etc, and thus the attack is practical even in a ciphertext-only mode of attack. However, the weakness does not extend to the Secure Socket Layer protocol that browsers use.

In terms of keystream output, this attack is interested only in the first word of output from any given secret key and IV. Hence, we can simplify our model of the output. The first output word depends only on three specific permutation elements, as shown in the figure below showing the state of the permutation immediately after KSA. When those three words are as shown, the value labeled Z will be output as the first word.

	1				X			$X+Y$	
	X				Y			Z	

In addition, if the key setup reaches a stage where i is greater than or equal to 1, $X = S_i[1]$ and $X + Y = S_i[1] + S_i[S_i[1]]$, then (if we model the remaining swaps in the key setup as random) with probability greater than $e^{-3} \approx 0.05$, none of the elements referenced by these three values will participate in any further swaps, and in that case, the value $S[S[1] + S[S[1]]]$ will be output as the first word. With probability less than $1 - e^{-3} \approx 0.95$, at least one of the three values will participate in a swap, and be set to an effectively random value, which will make the output value effectively random. We will refer to this situation as the *resolved* condition. Our attack involves examining messages with specific IV values such that, at some point, the KSA is in a resolved condition, and where the value of $S[S[1] + S[S[1]]]$ gives us information on the secret key. Then, we observe sufficiently many IV values that the actual value of $S[S[1] + S[S[1]]]$ occurs detectably often.

7 Details of the Known IV Attack

7.1 IV Precedes the Secret Key

First consider the case where the IV is prepended to the secret key. In this circumstance, assuming we have an I word IV, and a secret key $(K[0], K[1], \ldots K[\ell-1])$, we attempt to derive information on a particular word B of the secret key $(K[B])$ by searching for IV values such that, after the first I steps, $S_I[1] < I$ and $S_I[1] + S_I[S_I[1]] = I + B$. Then, with high likelihood (probability $\approx e^{-\frac{2B}{N}}$ if we model the intermediate swaps as random), we will be in a resolved condition after step $I + B$, and then the most probable output value will be

$$Out = S_{I+B-1}[j_{I+B}] = S_{I+B-1}[j_{I+B-1} + K[B] + S_{I+B-1}[I + B]]$$

Or, in other words, if we know the value of j_{I+B-1} and S_{I+B-1}, then given the first word output Out, we can predict the value

$$K[B] = S_{I+B-1}^{-1}[Out] - j_{I+B-1} - S_{I+B-1}[I+B]$$

where $S_t^{-1}[X]$ denotes the location within the permutation S_t where the value X appears. This prediction is accurate more than 5% of the time, and effectively random less than 95% of the time. By collecting sufficiently many values from different IVs, we can reconstruct $K[B]$.

In the simplest scenario (3 word chosen IVs), the attack works as follows[4]: suppose that we know the first A words of the secret key $(K[3], \ldots, K[A+2]$, with $A = 0$ initially), and we want to know the next word $K[A+3]$. We examine a series of IVs of the form $(A+3, N-1, X)$ for approximately 60 different values for X. At the first step, j is advanced by $A+3$, and then $S[i]$ and $S[j]$ are swapped, resulting in the key setup state which is shown schematicly below, where the top array is the combined IV and secret key presented to the KSA, and the bottom array is a portion of the permutation, and where the positions of the i, j variables are indicated.

$A+3$	$N-1$	X	$K[3]$		$K[A+3]$	
0	1	2			$A+3$	
$A+3$	1	2			0	
i_0					j_0	

Then, on the next step, i is advanced, and then the advance on j is computed, which happens to be 0. Then, $S[i]$ and $S[j]$ are swapped, resulting in the below structure:

$A+3$	$N-1$	X	$K[3]$		$K[A+3]$	
0	1	2			$A+3$	
$A+3$	0	2			1	
	i_1				j_1	

Then, on the next step, j is advanced by $X+2$, which implies that each distinct IV assigns a different value to j, and thus beyond this point, each IV acts differently, approximating the randomness assumption made above. Since the attacker knows the value of X and $K[3], \ldots K[A+2]$, he can compute the exact behavior of the key setup until he reaches step $A+3$. At this point, he knows the value of j_{A+2} and the exact values of the permutation S_{A+2}. If the value at $S_{A+2}[0]$ or $S_{A+2}[1]$ has been disturbed, the attacker discards this IV. Otherwise, j is advanced by $S_{A+2}[i] + K[A+3]$, and then the swap is done, resulting in the below structure:

$A+3$	$N-1$	X	$K[3]$		$K[A+3]$	
0	1	2			$A+3$	
$A+3$	0	$S[2]$			$S[j]$	
					i_{A+3}	

[4] This scenario was first published by Wagner in [Wag95]

The attacker knows the permutation S_{A+2} and the value of j_{A+2}. In addition, if he knows the value of $S_{A+3}[A+3]$, he knows its location in S_{A+2}, which is the value of j_{A+3}, and hence he would be able to compute $K[A+3]$. We also note that i_{A+3} has now swept past 1, $S_{A+3}[1]$ and $S_{A+3}[1]+S_{A+3}[S_{A+3}[1]]$, and thus the resolved condition exists, and hence with probability $p > 0.05$, by examining the value of the first word of RC4 output with this IV, the attacker will obtain the correct value of $K[A+3]$. Hence, by examining approximately 60 IVs with the above configuration, the attacker can rederive $K[A]$ with a probability of success greater than 0.5.

By iterating the above process across the secret key, the attacker can rederive ℓ words of secret key using 60ℓ chosen 3 word IVs.

The next thing to note is that the attack works for IVs other than those in the specific $(A+3, N-1, X)$ form. Any I word IV that, after I steps, leaves $S_I[1] < I$ and $S_I[1] + S_I[S_I[1]] = I + B$ will suffice for the above attack. In addition, since the attacker is able to simulate the first I steps of the key setup, he is able to determine which IVs have this property. By examining all IVs that have this property, we can extend this into a known IV attack, without using an excessive number of IVs. The probabilities to find the next word, and the expected number of IVs needed to obtain 60 IVs of the proper form, are given in Figure 6 at the end of this paper.

7.2 IV Follows the Secret Key

In the case that the IV is appended to the secret key, we need to take a different approach. The previous analysis attacked individual key words. When the IV follows the secret key, what we do instead is select IVs that give us the state of the permutation at an early phase of the key setup, such as immediately after the secret key has been used for the first time. Given that only a few swaps have occurred up to that point, it is reasonably straight-forward to reconstruct those swaps from the permutation state, and hence obtain the secret key (see Appendix D for one such method).

To illustrate the attack in the simplest case, suppose we have an A word secret key, and a 2 word IV. Further suppose that the secret key was weak in the sense that, immediately after A steps of KSA, $S_A[1] = X$, $X < A$, and $X + S_A[X] = A$. This is a low probability event ($p \approx 0.00062$ if $A = 13$), but it depends only on the secret key. For such a weak secret key, the attacker can assume the value of $j_{A-1} + S_{A-1}[A]$, and then examine IVs with a first word of $W = Y - (j_{A-1} + S_{A-1}[A])$. With such IVs, the value of j_A will be the preselected value Y. Then, $S[A]$ and $S[Y]$ are swapped, and so $S_A[A] = A_{A-1}[Y]$. Here, assuming Y was neither 1 nor $S_A[1]$, then the resolved condition has been established, and with probability > 0.05, $S_{A-1}[Y]$ will be the first word output. Then, by examining such IVs with the second word being at least 60 different values, we can observe the output a number of times and derive the value of $S_A[Y]$ with good probability. By selecting all possible values of Y, we can directly observe the state of the S_A permutation, from which we can rederive the secret key. We will denote this result as *key recovery*.

If $X + S_A[X] = A + 1$, a similar analysis would appear to apply. By assuming $S_A[A]$, $S_A[A+1]$ and j_A, we can swap $S_{A+1}[Y]$ into $S_{A+2}[A+1]$ for $N-2$ distinct IVs for any particular Y. However, the value of j_{A+2} is always the same for any particular Y, and so the probabilities that a particular IV outputs the value $S[Y]$ is not independently distributed. This effect causes the reading of the permutation state to be 'noisy', that is, for some values of Y, we see $S[Y]$ as the first word far more often than our analysis expected, and for other values of Y, we see it far less often. Because of this, some of the entries $S_{A+1}[Y]$ cannot be reliably recovered. Simulations assuming a 13 word secret key and $n = 8$ have shown that an average of 171 words of the S_A permutation state can be successfully reconstructed, including an average of 8 words of $(S_A[0], \ldots, S_A[12])$, which immediately give you effectively 8 key words. With this information, the key is reduced enough that it can be brute forced. We will denote this result as *key reduction*.

If we have a 3 word IV, then there are more types of weak secret keys. For example, consider a secret key where $S_A[1] = 1$ and $S_A[A] = A$. Then, by assuming j_A, we can examine IV where the first word has a value W so that the new value of j_{A+1} is 1, and so $S_A[1]$ and $S_A[A]$ are swapped, leaving the state after $A+1$ steps to be:

$K[0]$	$K[1]$		$K[A-1]$	W	X	Z
0	1		$A-1$	A	$A+1$	$A+2$
$S_A[0]$	A		$S_A[A-1]$	1	$S_A[A+1]$	$S_A[A+2]$
	j_{A+1}				i_{A+1}	

Then, by assuming $S_A[A+1]$ (which with high probability is $A+1$, and will always be at most $A+1$), we can examine IVs with the second word $X = Y - (1 + S_A[A+1])$, for an arbitrary Y, which will swap the value of $S_A[Y]$ into $S_{A+1}[A+1]$. Assuming Y isn't either 1 or A, then the resolved condition have been set up, and using a number of values for the third IV word Z, we can deduce the value of $S_{A+1}[Y]$ for an arbitrary Y, giving us the permutation after A steps.

There are a number of other types of weak keys that the attacker can take advantage of, summarized in Figure 7 found at the end of this paper.

The last weak secret key listed in Figure 7 is especially interesting, in that the technique that exposes the weakness is rather different than that of the other weak secret keys listed. Immediately after A steps, the state is:

$K[0]$	$K[1]$		$K[X]$		W	Z	
0	1		X		A	$A+1$	
$S_A[0]$	X		$S_A[X]$		Z	$S_A[A+1]$	
					i_A		

The initial IV word causes $S_A[X]$ and $S_A[A]$ to be swapped, leaving the state as:

$K[0]$	$K[1]$		$K[X]$		W	Z	
0	1		X		A	$A+1$	
$S_A[0]$	X		Z		$S_A[X]$	$S_A[A+1]$	
					i_A		

Now, to inquire about the value of $S_{X+Z}[Y+Const]$, we examine numerous IVs with second and third words that all set the value of j_{A+3} to be Y. The KSA will continue for $X+Z-(A+3)$ more steps until i now points to the element $S_{X+Z}[X+Z]$. At this point, since we haven't gone through a great number of steps since we knew the value of j (since $X+Z-(A+3) \leq A-4$), then with high probability, $j_{X+Z+1} = Y + Const$, where $Const$ is a constant term that depends only on the state of the permutation S_{A+1}. If this is true, then $S_{X+Z+1}[X+Z] = S_{X+Z}[Y+const]$, and if the elements $S[1]$ and $S[X]$ have not been disturbed (again, this happens with high probability), the resolved condition has been achieved, and the first output word will be biased towards $S_{X+Z}[Y+const]$. In addition, because the value of $const$ will be the same independent of Y, its value can easily be determined, thus allowing the attacker to observe many of the values of S_{X+Z}. This class of weak keys requires far more known IVs to exploit, but also occurs relatively frequently.

If we have a 4 word[5] IV, then the same general approach as the previous analysis can be used to recover virtually all secret keys, given sufficient IVs. First, we assume j_{A-1}, $S_{A-1}[A]$, $S_{A-1}[A+1]$, $S_{A-1}[A+2]$, $S_{A-1}[A+3]$ [6]. Then, based on this assumption, we search for IVs that, after $A+4$ steps, sets $S_{A+4}[1] = X$ and $S_{A+4}[X] = Z$ for $X, Z < A+4, X+Z \geq A+4$, and we note the value of $j_{A+4} = Y$. Then, we save the value of $X+Z$, the value Y and the value output as the first word for that particular IV. With nontrivial probability, the value of this word will be $S_{X+Z}[Y + const_{X+Z}]$, where $const_{X+Z}$ is a constant term that depends on the secret key, and the value $X+Z$. Since that value is independent of the IV, we can collect numerous possible values of $S_{X+Z}[Y+const_{X+Z}]$ for various values of $X+Z$, and use that to first reconstruct $const_{X+Z}$, and then reconstruct S_{X+Z}.

8 Related-Key Attacks on RC4

In this section, we discuss two related-key attacks based on weaknesses discussed previously in this paper. They work within the following model: the attacker is given a black box that has a randomly chosen RC4 key K inside it, an output button and an input tape of $|K|$ words. In each step the attacker can either press the output button to get the next output word, or write Δ on the tape, which causes the black-box to restart the output generation process with a new key defined as $K' = K \oplus \Delta$. The purpose of the attacker is to find the key K (or some information about it).

[5] This approach generalizes in the obvious way to longer IVs.

[6] Note that $S_{A-1}[x] \leq x$ for $x \geq A$. This limits the size of the search required.

8.1 Related-Key Attack Based on the Invariance Weakness

This attack works when the number of key words, is a power of two. It consists of n stages where in stage q the q^{th} bit of every key word is exposed[7]. The predicate *CheckKey* takes as input an RC4 blackbox and a parameter q (the stage number) and decides whether the key in the box is special 2^q-exact. This purpose can be achieved by randomly sampling key bits that are irrelevant for the 2^q-exactness of the key and estimating the expected length of q-patterned output. For a special 2^q-exact key the expected length will be significantly longer than in a random output (where it is less than 2) and thus *CheckKey* works in time $O(1)$. The procedure *Expand* takes as input an RC4 blackbox and a parameter q (the stage number), assumes that the key in the box is special 2^{q-1}-exact, and makes it special 2^q-exact. The method for doing so is by enumerating all the possibilities for the q^{th} bits ($2^{\ell-1}$ such possibilities) and invoking CheckKey to decide when the key in the box is special 2^q-exact. *Expand* works in a slightly different way for $q = 1$ and $q = n$. For $q = 1$, except for the lsb's, it determines the complete $K[0]$ (by forcing it to 1) and $msb(K[1])$. For $q = n$, there is only one 2^n-exact key and consequently we can calculate the output produced from this key and replace *CheckKey* by simple comparison. The time complexity of this stage is $O(2^{n+\ell})$ for $q = 1$ and $O(2^{\ell-1})$ for any other q.

The total time required for the attack is thus $O(2^{n+\ell}) + (n-1)O(2^{\ell}) = O(2^{n+\ell})$. For typical $\text{RC4}_{n=8}$ key with 32 bytes, the complexity of exhaustive search is completely impractical (2^{256}), whereas the complexity of the new attack is only $O(2^{n+\ell}) = O(2^{40})$.

8.2 Related-Key Attack Based on Known IV Weakness

In this section we use the known IV weaknesses to develop an efficient related key attack on RC4.

The attack consists of 3 stages, where in the first two stages we gain information on the first three words of the secret key, and in the third stage we iterate down the key, and expose each word of the key successively. The stages of the attack are as follows:

Step 1 This step attempts to find values of $K[0]$, $K[1]$ such that $S_1[1] = 1$, and reveal the value of $K[2]$. The procedure is to select random values of (X, Y), and for each such random value, write onto the tape 240 vectors with the initial four words (X, Y, Z, W) for $Z \in \{0, N/4, N/2, 3N/4\}$ and with 60 distinct random values of W, and for each such vector, press the output button. If X and Y are such that $S_1[1] = 1$ (for the modified key), then the output of the first word will be biased towards $3+(K[2] \oplus Z)$, unless that value happens to be 1. Hence, for at least 3 of the selected values of Z, the first word outputs will be biased towards one of $const$, $const + N/4$, $const + N/2$, $const + 3N/4$. This is detectable, and also by examining the value of $const$, the attacker can reconstruct the value of $K[2]$. We expect to try N random values of (X, Y) before finding a pair that is appropriate.

[7] In fact, $K[1]$ is fully revealed during the first stage (see Figure 4)

Step 2 This step attempts to find the values of $K[0]$, $K[1]$. The procedure is to write on the tape 60 vectors with the initial four words (X, Y, Z, W), where X, Y are the values recovered in the previous step, $Z = (N-3) \oplus K[2]$, and with 60 distinct random values of W, and for each such vector, press the output button. This particular initial sequence assures that $S_2[1] = 1$ and $S_2[2] = S_1[0] = K[0]$, and hence the output will be biased towards $K[0]$. Once that has been recovered, $K[1]$ can be computed.

Step 3 This step iteratively recovers individual words of the key. It operates by running a subprocedure that assumes that we have already recovered $(K[0], \ldots, K[A-1])$, and want to learn the value of $K[A]$. The procedure is to write 60 vectors that have the property that, given the known values of $(K[0], \ldots, K[A-1])$, that $S_{A-1}[1] = X < A$ and $X + S_{A-1}[X] = A$. With 60 such vectors, we can use the procedure shown in 7.1 to rederive $K[A]$.

The total time required for the attack is thus (because $2^n \geq \ell$):

$$Step1 + Step2 + (\ell - 3) * Step3 = O(2^{n+8}) + 2^6 + (\ell - 3)2^6 = O(2^{n+8})$$

For a RC4 key with $n = 8$ the time complexity is $O(2^{16})$ and is essentially independent of the key length.

8.3 Comparing the Attacks

Both attacks are able to completely reconstruct the randomly chosen RC4 key[8] with a number of chosen keys and amount of work that is significantly below that of brute force (except for extremely short RC4 keys). The first attack scales upwards as the key grows longer, while the time complexity of the second attack is independent of key length, with a cross-over point at $\ell = 8$.

However, due to the second word weakness, future implementations of RC4 are likely to discard some prefix of the output stream, and in this case the second attack becomes difficult to apply – output word x depends on $2x+1$ permutation elements immediately after KSA, and all the $2x+1$ elements must occur before t for the resolved condition to hold. On the other hand, the first attack extends well, in that the probability of the output words being patterned drops modestly as the number of discarded words increases.

9 Discussion

Section 3 describes an interesting weakness of RC4 which results from the simplicity of its key scheduling algorithm. We recommend to neutralize this weakness by discarding the first N words of each generated stream. After N rounds, every element of S is swapped at least once and the permutation S and the index j are expected to be "independent" of the initialization process.

Section 6 describes a weakness of RC4 in a common mode of operation in which attacker visible IV's are concatenated with a fixed secret key. It is easy

[8] the first attack works only for some key lengths.

to extend the attack to other simple types of combination operators (e.g., when we XOR the IV and the fixed key) with essentially the same complexity. We recommend to neutralize this weakness by avoiding this mode of operation, or by using a secure hash to form the key presented to the KSA from the IV and secret key.

A Applying The Attack to WEP-like Cryptosystems

The Wired Equivalent Privacy (WEP) protocol is designed to provide privacy to packet based wireless networks based on the 802.11 standard (see [LMSon]). It encrypts by taking a secret key and a per-packet 3 byte IV, and using the IV followed by the secret key as the RC4 key. Then, it transmits the IV, and the RC4 encrypted payload. By using the results from Section 7.1, we can show how, by examining enough ciphertext packets, to reconstruct the secret key for a WEP-like cryptosystem. Note that we have not attempted to attack an actual WEP connection, and hence do not claim that WEP is actually vulnerable to this attack.

We assume that the attacker is able to retrieve the first byte of the RC4 output from each packet[9]. By the analysis done in section 7.1, to recover key byte B, the attacker needs to know the previous key bytes, and then search for IVs that sets up the permutation such that

$$X = S_{B+3}[1] < B + 3 \tag{1}$$

$$X + S_{B+3}[X] = B + 3$$

With 60 such IVs, the attacker can rederive the key byte with reasonable probability of success. The number of packets required to obtain that number of IVs depends on the exact IVs that the sender uses. Although the 802.11 standard does not specify how an implementation should generate these IVs, common practice is to use a counter to generate them.

A.1 Analysis of IVs Generated by a Little Endian Counter

If the IVs are generated by a multibyte counter in little endian order (and hence the first byte of the IV increments the fastest), then the attacker can search for IVs of the form $(B, 255, N)$ for $3 \leq B < 8$. If he can collect these for 60 different values of N, then he can derive the secret key with little work. This requires approximately 4,000,000 packets.

[9] Because of the payload format used with 802.11, the attacker typically does know the first byte of each plaintext payload, and hence is able to derive the first byte of RC4 output.

A.2 Analysis of IVs Generated by a Big Endian Counter

If the IVs are generated by a multibyte counter in big endian order (and hence the last byte of the IV increments the fastest), then the attacker can, as above, search for IVs of the form $(B, 255, N)$. This requires approximately 1,000,000 packets to collect the requisite IVs, assuming that the counter starts from zero.

However, if the counter doesn't start from zero, the attacker has an alternative strategy available to him. He can assume the first several bytes of secret key, and then search for IVs that set up the permutation as in Equation 1. If the attacker assumes the first two bytes of secret key, then for each initial IV byte, there are approximately 4 settings of the remaining two bytes that set up the permutation as required to rederive a particular key byte. Hence, with approximately 1,000,000 packets, and an additional 2^{16} work factor, he can still rederive the key.

It is common practice in the industry to extend the length of the WEP secret key (which is specified as 40 bit). Because the above attacks recover each key byte individually, the complexity of the attack grows linearly rather than exponentially with the key length, and thus even an extremely long key is not immune to this attack.

B Ciphertext-Only Distinguishers based on the Invariance Weakness

The distinguishers we presented in Section 5.1, as well as most of the distinguishers mentioned in the literature (for RC4 and other stream ciphers) assume knowledge of the plaintext in order to isolate the XORed key stream.

However, in practice the only information the attacker has is typically some statistical knowledge about the plaintext, e.g., that it contains English text. Combining the non-random behaviors of the plaintext and the key-stream is not always possible, and there are cases where XORing biased streams result with a totally random stream, e.g. when one stream is biased in its even positions and the other stream is biased in its odd positions. We prove here that if the plaintexts are English texts, it is easy to construct a ciphertext-only distinguisher from our biases. The intuition of this construction is that the biases described in Section 5.1 are in the distribution of the lsb's, and consequently they can be combined with the non-random distribution of the lsb's of English texts.

There are many major biases in the distribution of the lsb's of English texts, and they can be combined with biases of the key-stream words in various ways. In Theorem 3, we show how to combine the distribution of the first lsb of the RC4 output stream, with the first order statistics of English texts[10] :

Theorem 3 *Let C be the ciphertext generated by RC4 from a random key and the ASCII representation of plaintexts, distributed according to the first order*

[10] Since the purpose of the theorem is only to demonstrate this approach , we ignore the fact that the distribution of the first characters in an English sentence differs from the distribution of mid-text characters.

statistics of English texts. Let p be the probability of a random key to be special 2-exact. Then C can be distinguished from a random stream by analyzing about $\frac{200}{p^2}$ *output words.*

For example, for $RC4_{n=8}$ with 8 byte keys, $p = 2^{-16}$, which implies a reliable ciphertext-only distinguisher that works with less than 2^{40} data. The proof of Theorem 3 is based on the observation that the lsb of a random English text character is zero with probability of about 55%. The formal proof is omitted due to space limitations.

It is important to note that Theorem 3 does not use all the statistical information which is available in either the key-stream or the plaintext distributions, and consequently does not represent the best possible attack.

C The Sampling Resistance of RC4

Most of the Time/Memory/Data tradeoff attacks on stream ciphers are based on the following paradigm. The attacker keeps a database of [state,output] pairs (sorted by output) and lookups every subsequence of the output stream in this database. When a (sufficiently long) database sequence is located in the output, the attacker can conclude that the actual state is the one stored along with this sequence and predict the rest of the stream.

A drawback of this approach is that the large database must be stored in a hard disk(s) whose random access time is about a million times slower than a computational step. To improve that attack we can keep on disk only states that are guaranteed to produce outputs with some rare but easy recognizable property (e.g., starting with some prefix α). In this case only output sequences that have this property have to be searched in the database, and thus the expected time and the expected number of disk probes is significantly reduced.

In general, producing a pair [state,output] with such a rare property costs much more than producing a random pair. $O(\frac{1}{p})$ *random* states are required to find a single pair, where p is the probability of a random output to have this property. However, if we can efficiently enumerate states that produce such outputs, the number of sampled states decreases dramatically, and this method can be applied without significant additional cost during the preprocessing stage. The sampling resistance of a stream cipher provides a lower bound on the efficiency of such enumeration.

Such an attack can be applied to RC4 in two ways, based on the KSA and PRGA parts. An attack on the generation part constructs a database of pairs [RC4 state, output substring] and analyzes all the substrings along a single output stream. The database construction is very simple since it is easy to enumerate states which produce outputs that have some constant prefix. However, this enumeration seems to be useless due to the huge effective key of this part (1684 bits) which makes such a tradeoff attack completely impractical. A more promising approach is based on the KSA part which uses a key of 40-256 bits and might be vulnerable to tradeoff attacks. In this case, the pairs in the database are [secret key, prefix of the output stream], and the attack requires prefixes from a large number of streams (instead of a single long stream).

The correlation described in Section 4 provides an efficient sampling of keys that are more likely to produce output prefixes of the patterned type specified above (constant $\pmod{b}$).

For example, consider the problem of sampling M keys which are transformed by the KSA into streams whose first five words are fixed $\pmod{16}$. This property of random streams has probability of 2^{-20}, and the expected number of disk probes during the actual attack is reduced by this factor. For stream ciphers with high sampling resistance, such a filter would increase the preprocessing time by a factor of one million, as one would have to sample a million random keys in order to find a single "good" key. For RC4 (due to the invariance weakness), the preprocessing time increases by a factor of less than four, as more than one quarter of the exact special keys produce such streams. Consequently, the preprocessing stage is accelerated by a factor of 2^{18}.

To summarize this section, we proved that RC4 has relatively low *Sampling Resistance*, which greatly improves the efficiency of tradeoff attacks based on its KSA.

D Deriving the Secret Key from an Early Permutation State

Given the values $S_A[0], \ldots, S_A[A-1]$, one method to find all values of $K[0], \ldots, K[A-1]$ that result in such a permutation is:

```
i = 0
S = {0, ..., N − 1}
For i = 0 ... A − 1
    X = S^{-1}[S_A[i]]
    If i < X < A
        Branch over all values of 0 ≤ X < A s.t. X ≥ I or
            S[X] ≠ S_A[X], running the remaining part of this
            algorithm for all such values.
    K[i] = X − j − S[i]
    j = X
    Swap(S[i], S[j])
Verify that {S[0], ..., S[A − 1]} = {S_A[0], ..., S_A[A − 1]}
```

The number of times this algorithm will perform an iteration is bounded by $A^{\lambda+1}$, where λ if the number of values $0 \leq x < A$ where $S_A[x] < A$. Because λ is typically quite small, this algorithm is typically efficient.

An algorithm with a better lower bound on run time could be given by using the values of $S_A[A], \ldots, S_A[N-1]$.

References

[BSW00] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of a5/1 on a pc. In *FSE: Fast Software Encryption*, 2000.

[FM00] Fluhrer and McGrew. Statistical analysis of the alleged rc4 keystream generator. In *FSE: Fast Software Encryption*, 2000.

[Gol97] Golić. Linear statistical weakness of alleged RC4 keystream generator. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1997.

[GW00] A. L. Grosul and D. S. Wallach. a related-key cryptanalysis of rc4. June 2000.

[KMP+98] Knudsen, Meier, Preneel, Rijmen, and Verdoolaege. Analysis methods for (alleged) RC4. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1998.

[LMSon] Wireless lan medium access control (MAC) and physical layer (PHY) specifications. (IEEE Standard 802.11), 1999 Edition. L. M. S. C. of the IEEE Computer Society.

[MS01] I. Mantin and A. Shamir. A practical attack on broadcast RC4. In *FSE: Fast Software Encryption*, 2001.

[MT98] Mister and Tavares. Cryptanalysis of RC4-like ciphers. In *SAC: Annual International Workshop on Selected Areas in Cryptography*. LNCS, 1998.

[Rei01] Arnold Reinhold. The ciphersaber home page. 2001.

[Roo95] A. Roos. A class of weak keys in the *rc4* stream cipher. September 1995.

[Wag95] D. Wagner. Re: Weak keys in rc4. September 1995. Posted to sci.crypt, Archived at http://www.cs.berkeley.edu/ daw/my-posts/my-rc4-weak-keys.

ℓ	q	b	k_1 [a]	k_2 [b]	p [c]	P_{RND} [d]	P_{RC4} [e]	Data
4	1	2	12	15	2^{-3}	2^{-15}	$2 \cdot 2^{-15}$	2^{15}
6	1	2	14	18	2^{-4}	2^{-18}	$2 \cdot 2^{-18}$	2^{18}
8	1	2	16	21	2^{-5}	2^{-21}	$2 \cdot 2^{-21}$	2^{21}
10	1	2	18	24	2^{-6}	2^{-24}	$2 \cdot 2^{-24}$	2^{24}
12	1	2	20	27	2^{-7}	2^{-27}	$2 \cdot 2^{-27}$	2^{27}
14	1	2	22	30	2^{-8}	2^{-30}	$2 \cdot 2^{-30}$	2^{30}
16	1	2	24	34	2^{-10}	2^{-34}	$2 \cdot 2^{-34}$	2^{34}

Fig. 5. Data required for a reliable distinguisher, for different key sizes

[a] number of predetermined bits $(q(\ell - 1) + n + 1)$
[b] number of determined output bits
[c] probability of these k_1 key bits to determine these k_2 output bits (taken from Figure 8)
[d] $= 2^{-k_2}$
[e] $\approx P_{RND} + 2^{-k_1} p$

IV Length	Probability	Expected IVs required
3	4.57×10^{-5}	1310000
4	4.50×10^{-5}	1330000
5	1.65×10^{-4}	364000
6	1.64×10^{-4}	366000
7	2.81×10^{-4}	213000
8	2.80×10^{-4}	214000
9	3.96×10^{-4}	152000
10	3.94×10^{-4}	152000
11	5.08×10^{-4}	118000
12	5.04×10^{-4}	119000
13	6.16×10^{-4}	97500
14	6.12×10^{-4}	98100
15	7.21×10^{-4}	83200
16	7.18×10^{-4}	83600

Fig. 6. For various prepended IV and known secret key prefix lengths, the probability that a random IV will give us information on the next secret key word, and the expected number of IVs required to derive the next secret key word.

Condition	IV Settings			Probability	Result
	First	Second	Third		
$S_A[1] = 1$ $S_A[A] = A$	Swap with 1	Swap with Y	Cycle	0.0037	Key recovery
$S_A[1] = 2$ $S_A[A+1] = A+1$	Swap with 1	Cycle	Swap with Y	0.0070	Key reduction
$S_A[1] = X < A$ $S_A[X] + X = A$	Swap with Y	Cycle	Cycle	0.0007	Key recovery
$S_A[1] = X < A$ $S_A[X] + X = A+1$	Cycle	Swap with Y	Cycle	0.0009	Key recovery
$S_A[1] = X < A$ $S_A[X] + X = A+2$	Cycle	Cycle	Swap with Y	0.0007	Key reduction
$S_A[1] = A$	Swap with $S_A^{-1}[1]$	Swap with Y	Cycle	0.0037	Key recovery
$S_A[1] = A+1$	Swap with Y	Swap with $S_A^{-1}[N-1]$	Cycle	0.0036	Key recovery
$S_A[1] = A+2$	Cycle	Swap with Y	Swap with $S_A^{-1}[N-1]$	0.0038	Key reduction
$S_A[1] = N-2$ $S_A[A+2] = A+2$	Swap with Y	Cycle	Swap with 1	0.0034	Key reduction
$S_A[1] = N-1$ $S_A[A+1] = A+1$	Swap with Y	Swap with 1	Cycle	0.0036	Key recovery
$S_A[1] = X < A$ $S_A[A] = Z$ $X + Z > A+2$	Swap with X	Cycle	Cycle	0.1007	Key reduction

Fig. 7. Weak secret keys with 3 word postfix IVs. Listed are the conditions on the S_A permutation that distinguish them, the IV properties that the attacker searches for to reveal $S[Y]$, the probability that this class of weak key will occur with $n = 8$ and a 16 word secret key, and the result of the attack on the weak key.

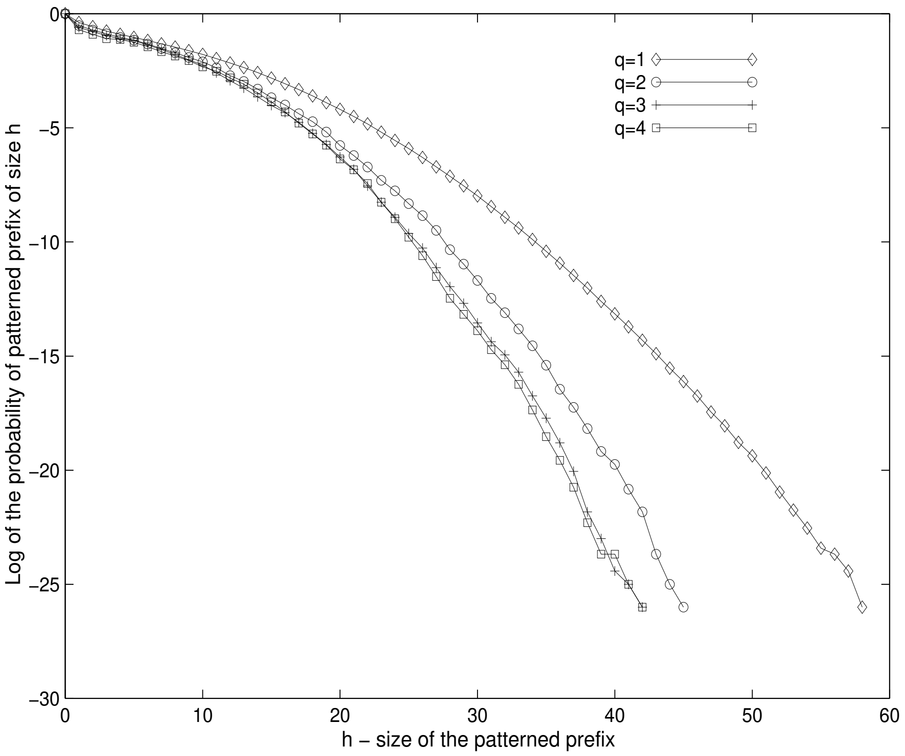

Fig. 8. This graph demonstrates the probabilities of special keys (2^q-exact with $K[0] = 1$, $msb(K[1] = 1)$) of $\mathrm{RC4}_{n=8,\ell=16}$ to produce streams with long patterned prefixes