A Related-Key Cryptanalysis of RC4*

Alexander L. Grosul and Dan S. Wallach
Department of Computer Science
Rice University

June 6, 2000

Abstract

In this paper we present analysis of the RC4 stream cipher and show that for each 2048-bit key there exists a family of related keys, differing in one of the byte positions. The keystreams generated by RC4 for a key and its related keys are substantially similar in the initial hundred bytes before diverging. RC4 is most commonly used with a 128-bit key repeated 16 times; this variant does not suffer from the weaknesses we describe. We recommend that applications of RC4 with keys longer than 128 bits (and particularly those using the full 2048-bit keys) discard the initial 256 bytes of the keystream output.

1 Introduction

RC4, a fast output-feedback cipher, is one of the most widely used cryptosystems on the Internet, commonly used as the default cipher for SSL/TLS connections [DA99]. Ron Rivest designed RC4 in 1987 for RSA Data Security, Inc., which only made it available under nondisclosure. In September, 1994, the algorithm was posted anonymously to the Internet and is now available for public analysis [Sch96, chapter 17]. RC4 is currently being standardized by the IETF under the name "Arcfour" [KT99]. Our analysis is based on Schneier's description of RC4.

*This work was partially supported by DARPA through USAFRL grant F30602-97-2-298.

Given RC4's popularity and simplicity of design, we felt it would be valuable to study how susceptible RC4 is to attacks based on related keys [KSW96]. Two keys are "related" if they differ in some small number of bits from each other, where the attacker can choose these differences. If related keys result in similar keystreams, this would imply a weakness in the cipher. Section 2 describes how RC4 works. Section 3 presents our analysis of the algorithm followed by experimental results in section 4. Finally, section 5 presents some related work and section 6 presents our conclusions.

2 Description of RC4

RC4 is an output-feedback mode cipher. Its keys are 2048 bits long, and its internal state consists of two counters i and j (each within $[0, 255]$) plus an array of 256 8-bit bytes, called the S-box. If a cryptography system wishes to use a shorter key, the key is repeated as many times as necessary to fill the 2048-bit key. Once the S-box is initialized with the key, the RC4 algorithm is a loop that updates the internal state of the S-box and returns a byte of keystream. Normally, this keystream is XOR-ed with the plaintext message to produce the ciphertext. As with any output-feedback cipher, RC4 only protects the secrecy of a message, not its integrity. Other measures, such as the use of cryptographic checksums, are commonly used along with RC4.

The S-box is initialized using the key K as follows:

```
for i = 0 to 255
  S(i) = i
  j = 0

for i = 0 to 255
  j = (j + S(i) + K(i)) mod 256
  swap S(i) and S(j)

i = 0
j = 0
```

Each next byte b of the keystream produced using:

```
i = (i + 1) mod 256
j = (j + S(i)) mod 256
```

```
swap S(i) and S(j)
b = S(S(i) + S(j)) mod 256
```

3 Analysis of RC4

Our analysis focuses on how the S-box initialization responds to a single-byte difference in its input. Assume $K'(i) = K(i)$ except when $i = t$, where $K(t) \neq K'(t)$. In this case, $j = (j + S(i) + K(i))$ *mod* 256 will result in different values of j, and the initialization's state will be different thereafter. If t is close to zero, the resulting S-boxes will be completely different. If t is close to 255, however, the S-boxes will be substantially similar because the first $t-1$ iterations through the initialization loop performed exactly the same work. For clarity, we will use j_i to signify the value of j after the i'th iteration of the initialization loop with K and j'_i to refer to j after the i'th iteration with K'. Likewise, S_i and S'_i refer to the S-boxes after the i'th iteration. We can now restate the RC4 initialization as follows:

$$
\begin{aligned}
j_i &= (j_{i-1} + S_{i-1}(i) + K(i)) \; mod \; 256 \\
\forall n \in [0,255] : S_i(n) &= \begin{cases} S_{i-1}(j_i) & \text{if } n = i \\ S_{i-1}(i) & \text{if } n = j_i \\ S_{i-1}(n) & \textit{otherwise} \end{cases}
\end{aligned}
$$

We observe that we can make two complementary changes in the key with the goal of not disturbing the S-box initialization. If $K'(t) = K(t) + \delta$ and $K'(t+1) = K(t+1) - \delta$, then $j'_t = j_t + \delta$ and j'_{t+1} is likely to be the same as j_{t+1}. The $+\delta$ and $-\delta$ added to K cancel out each other's effects, with the exception that the t iteration will perform a different swap for K' than it did with K. When we derive K' from K in this fashion, we refer to K' as being *twiddled* from K at position t, and we say K and K' are *related keys*.

Of course, it's possible the swap on the t iteration could effect the value of j'_t on any subsequent iteration if $j'_t > t$. The probability of this occurring is $(256-t)/256$. Immediately following the twiddle, unless either $j_t = t+1$ or $j'_t = t+1$, $j'_{t+1} = j_{t+1}$. We call this a *good twiddle*, meaning only three members of S'_t will be different from S_t (which will occur with probability $(255/256)^2$).

Following a good twiddle, the RC4 initialization will proceed in lock step for both K and K' until $S_{i-1}(i) \neq S'_{i-1}(i)$. At this point, $j_i \neq j'_i$ and the initialization will diverge.

At the end of initialization both i and j are set to 0. As with initialization, the output will be identical between two runs until $S(i) \neq S'(i)$. Following our earlier example, we restate the output generation of RC4 as follows:

$$
\begin{aligned}
j_i &= (j_{i-1} + S_{i-1}(i)) \; mod \; 256 \\
\forall n \in [0,255] : S_i(n) &= \begin{cases} S_{i-1}(j_i) & \text{if } n = i \\ S_{i-1}(i) & \text{if } n = j_i \\ S_{i-1}(n) & otherwise \end{cases} \\
b_i &= S_i((S_i(i) + S_i(j)) \; mod \; 256)
\end{aligned}
$$

As with the initialization phase, the RC4 output for the original key and the related key will proceed in lock step (*i.e.*, $b'_i = b_i$) until $S'_{i-1}(i) \neq S_{i-1}(i)$. This results in $j'_i \neq j_i$. At this point, we say the two RC4 systems have *derailed*. The number of identical bytes produced from the two related keys before the derailment is called the *derailment length*.

We will also use the term *similarity* to refer to the number of output bytes where $b'_i = b_i$.

4 Experiments

RC4 is normally used in a mode where 16-byte (128-bit) keys are repeated sixteen times. In our experiments we use full 256-byte keys. For a given key, the total number of keys differing in at most one twiddle is $255 * 256 = 65280$. We will consider these keys first.

4.1 Observable effects of single twiddles

As an example, we chose a random key and encrypted the first 256 characters of Schneier's description of RC4. We then decrypted the message with the same key except we performed a twiddle with $t = 254, \delta = 1$. The result is:

> RC4*is a variable-key-size stream cipher developed in 1987 by*Ron Ri*est for RSA Data Security, Inc. For seven years it was*proprietary, and *etails of the alg*rithm were only available after signing a*no*disclosure agreement. In September, 1994 som*******

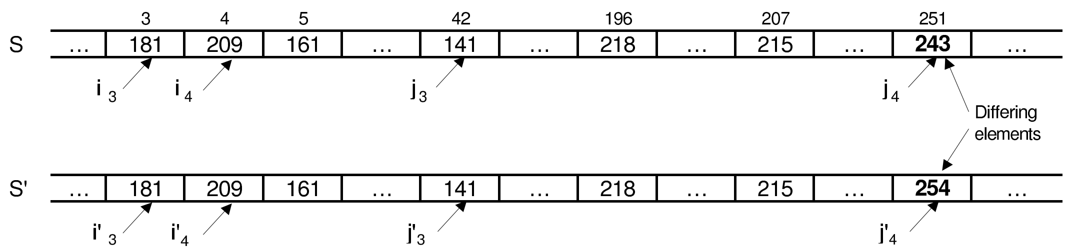

Figure 1: The state of S-boxes S and S' before hitting a bump.

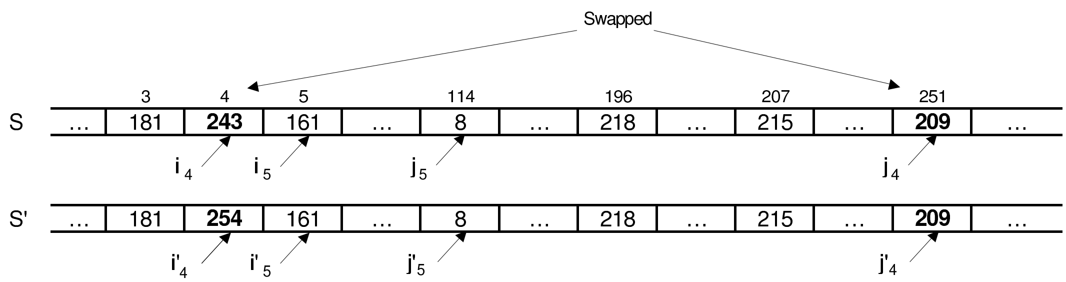

Figure 2: The state of S-boxes S and S' after hitting a bump.

We use the "*" symbol for characters which were decoded incorrectly. There are eight stand-alone and seven consecutive undecoded characters. Close examination of the states of S-boxes helps us understand appearance of "*"s.

The derailment length is 249. At this point $S(i_{249}) \neq S'(i'_{249})$, so $j_{249} \neq j'_{249}$, and the RC4 streams derail. This is exactly the reason for consecutive "*"s at the end of our example output above.

It is interesting to note that there are some "*"s in the text before the derailment at the 249th character. We call these locations where $b_i \neq b'_i$ *bumps*, since the two RC4 keystreams continue together after what could have been a derailment. To explain bumps, we need to look at how each keystream byte is generated. b_i is dependent on the values of $S_i(i)$, $S_i(j)$, and S_i indexed by their sum. the next keystream byte is different and a bump or derailment will occur.

An an example of a bump, figure 1 shows the state of S-boxes S and S' just before a bump occurs. Then the following steps are performed:

- $i_3 = 3$, $i'_3 = 3$;

- $i_4 = i_3 + 1 = 4$, $i'_4 = i'_3 + 1 = 4$;
- $j_4 = j_3 + S(i_4) = 251$, $j'_4 = j'_3 + S'(i'_4) = 251$;
- swap $S(i_4)$ and $S(j_4)$, swap $S'(i'_4)$ and $S'(j'_4)$ – see figure 2;
- for S return $S(S(i_4) + S(j_4)) = S((243 + 209)\, mod\, 256) = S(196) = 218$;
- for S' return $S'(S'(i'_4) + S'(j'_4)) = S'((254 + 209)\, mod\, 256) = S(207) = 215$ – a bump.

At this point, despite the bump, we see most things stayed the same:

- $i_4 = i'_4 = 4$
- $j_4 = j'_4 = 251$
- $S(j_4) = S(j'_4) = 209$

The only noticeable difference is that $S(i_4) \neq S'(i_4)$. This difference has no effect on the next byte of output because i is incremented. This example represents the most common effect from applying twiddles to an RC4 key. We can see that RC4 is resilient to small errors in the S-box, so long as $S(i) = S'(i')$. Otherwise, the resulting values of j will differ, and the RC4 keystreams derail from each other.

4.2 Measured derailment lengths

Our next experiment will measure the expected derailment length for RC4 between S-boxes initialized with keys related by a single twiddle.

A keystream should have the property that any value $n \in [0, 255]$ is equally likely to appear in each position b_i of the keystream, that is $P(b_i = n) = 1/256$. For two S-boxes S and S' initialized with randomly chosen keys the probability that two bytes b_i and b'_i produced at any step will be the same is

$$P(b_i = b'_i) = \sum_{n=0}^{255} P(b_i = n) * P(b'_i = n) = 256 * \frac{1}{256} * \frac{1}{256} = \frac{1}{256} \tag{1}$$

Thus, the probability[1] that the derailment length will be d is

$$P(\text{derailment} = d) = (\prod_{n=1}^{d} P(b_n = b'_n)) * P(b_{n+1} \neq b'_{n+1}) = \frac{255}{256^{d+1}} \tag{2}$$

[1] We expect that for $i \neq j$, events $b_i = b'_i$ and $b_j = b'_j$ are independent.

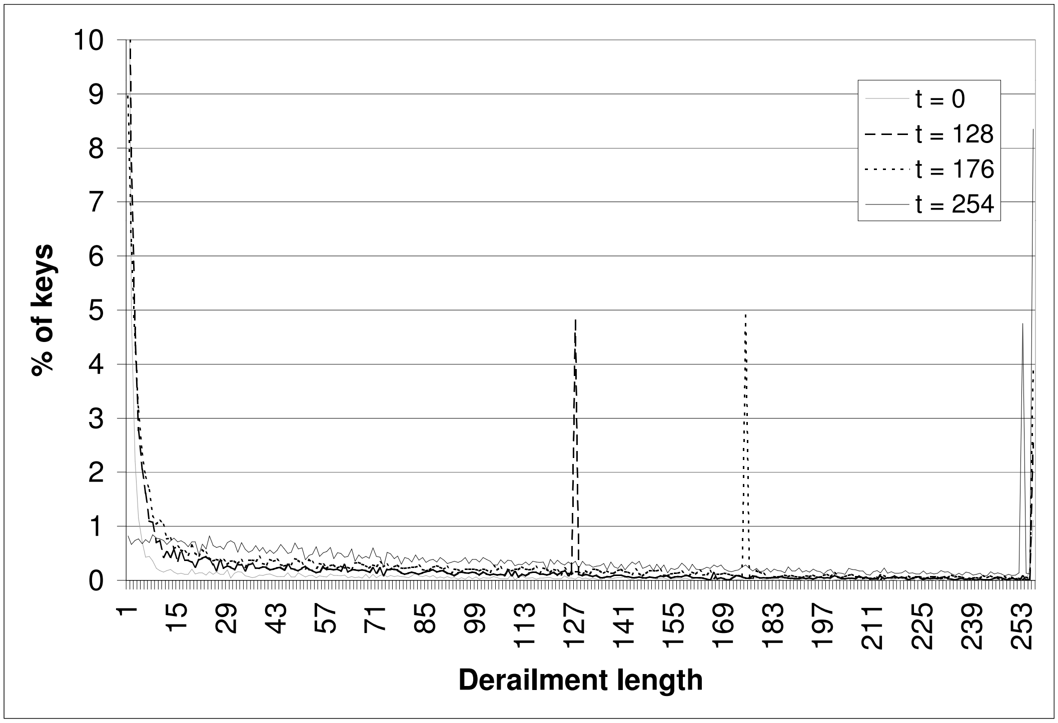

Figure 3: Distribution of keys with respect to derailment length for some values of t, the location of the twiddle.

In a simple experiment we measure the derailment length for 10000 pairs of randomly generated 256-byte keys. 9961 key pairs produce streams that derail immediately; for the other 39 pairs the derailment length is 1. This exactly supports the formula (2).

However, the derailment length for S-boxes initialized with related keys is different. To show this we measure the percentage of related keys, which generate keystreams with a given derailment length. For each value of $t \in [0, 254]$ we inspect 10000 pairs consisting of a randomly generated key and a corresponding related key generated by a twiddle at t, where $\delta \in [0, 255]$ is chosen randomly. The results for $t \in 0, 128, 176, 254$ are shown on Figure 3. The results for other values of t are similar.

The vertical axis shows the percentage of cases which had the derailment length as shown on the horizontal axis. For any t there are three spikes in the graph:

- In the area 0–3. For $t = 0$ the height of this spike is approximately 60%. As t increases, the spike decreases to about 8% for $t = 254$. This spike appears because the smaller the value of t, the better chance there is that S-boxes will derail quickly.

- Exactly at the points where derailment is equal to $t - 1$. This spike appears because a number of derailments happen while generating tth byte of keystream. This results from the twiddle, which changed $S(t)$.

- The last spike shows the percentage of tests which have derailment length greater than 255. For $t = 0$ it is around 1.1%. As t increases, so increases this spike, reaching 8.3% for $t = 254$. We attribute such 'runaway' cases to the fact mentioned above: each time S-boxes hit a bump, the differing elements of S-boxes are moved to another location. Thus, even though each entry in the S-box is used during keystream generation, sometimes the differing elements themselves escape being used as input to advancing the state of the S-box.

To estimate the number of related keys producing keystreams which derail no earlier than the t'th output byte, we add and normalize the results of the previous experiment. The result is shown on figure 4. The vertical axis shows the percentage of cases which have at least the derailment length represented on the horizontal axis. For example, the graph shows that for a random key, 18.6% of keys from the corresponding related key space (resulting from single twiddles) result in the derailment length of at least 100.

In addition to measuring the derailment length, we also wanted to measure the similarity in the keystreams, that is we wanted to measure the number of identical bytes. This will help us understand the influence of bumps and different values of t on derailment. On Figure 5 we show the result.

The horizontal axis shows the similarity between two keystreams, and the vertical axis shows the percentage of cases that produced keystreams with at least that similarity. To exhibit dependence on values of δ we show results for $\delta = 255$ (the top line), and $\delta = 127$ (the bottom line). The maximal distance between these lines is 16.9%, and the average distance is 8.8%. The results for all other values of $\delta \in [1, 255]$ lay between these two lines.

The line in the middle shows the results for random values of δ. This line is the most precise estimation of the percentage of related keys which decrypt

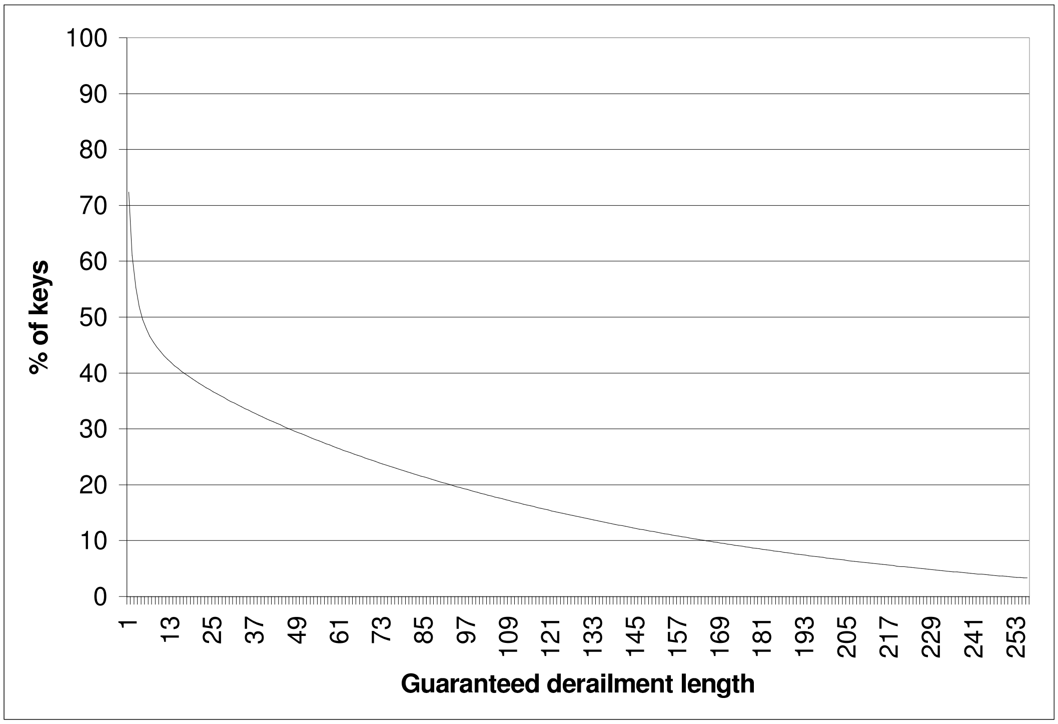

Figure 4: Percentage of keys producing keystreams with at least given derailment length

the ciphertext with guaranteed similarity. To see how bumps affect similarity, we compare this line to figure 4. Except for values on horizontal axis less than 3 and greater than 246, when similarity changes sharply, it stays within 1% of the values in figure 4. Thus, we conclude that the influence of bumps on the similarity must be insignificant.

In addition, as in the example above, for a random key, 18.6% of related keys generated by one twiddle result in a derailment length of at least 100. This means that the effective key length is $log_2(65280 * 0.186) = 13.5$ bits shorter. In figure 6, we plot how the effective length of a key (the vertical axis) depends on the required similarity (the horizontal axis).

These losses in key strength are in addition to losses from keys which generate identical S-boxes. The S-box has the property that it always describes a one-to-one mapping from the integers $[0,255]$ to $[0,255]$. There are 256! such permutations

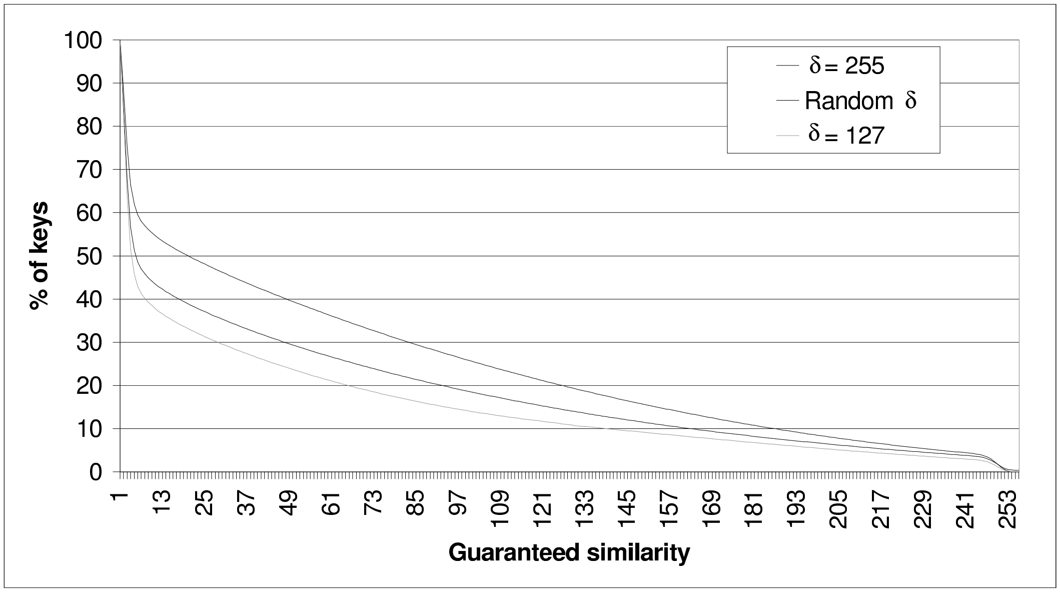

Figure 5: Percentage of related keys for a randomly chosen key having at least a given amount of similarity in the output keystreams.

(approximately 2^{1684}). By the pigeon-hole principle, a perfect initialization of the S-box would still translate $2^{(2048-1684)} = 2^{364}$ different input keys to precisely the same S-box state.

4.3 Limitations

Our method works well only for long keys which are not repeated during the initialization of S-box. In systems using RC4 commercially, much shorter keys are used, commonly 16 bytes. For such a key even a one-bit change will cause 16 changes in the key schedule. This does not mean, however, that related keys do not exist for short RC4 keys, merely that we do not know how to construct such keys.

An additional interesting question for future study is what information is revealed by the location of the bumps, and whether some form of bumps might occur with shorter keys.

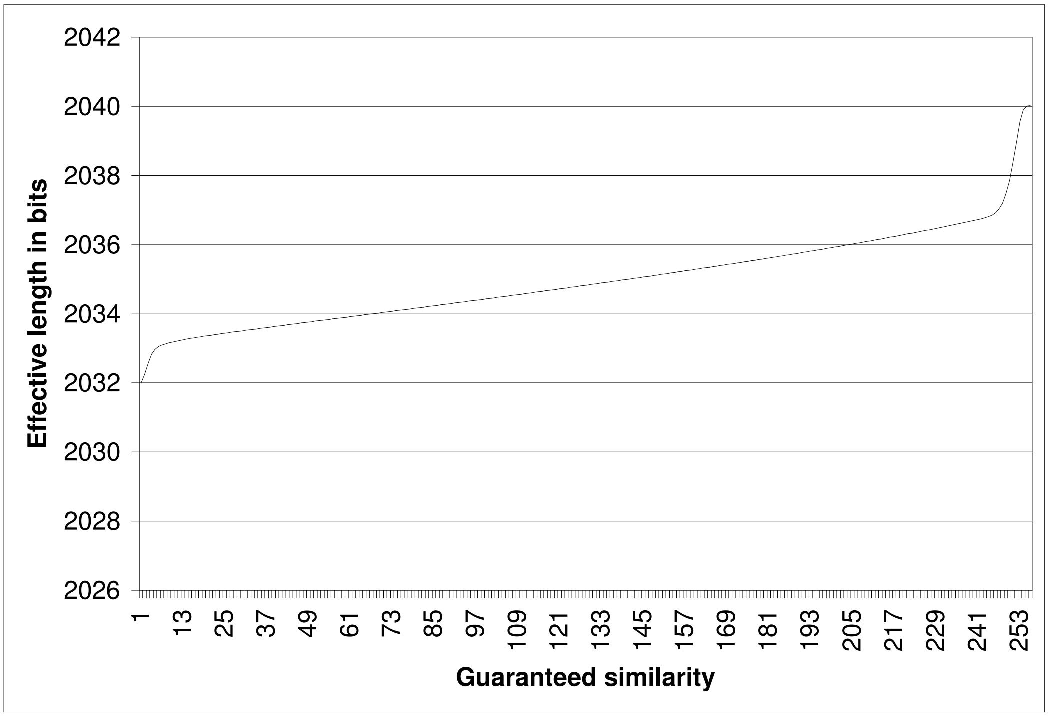

Figure 6: The effective length of 256-byte keys in respect to required similarity

5 Related Work

Kelsey [KSW96] describes related-key analysis techniques for various ciphers. Roos [Roo95] shows that RC4 has a class of detectable keys, each of which can reveal with high probability the first 2 bytes of the key. One of the conclusions in Roos is to "discard a number of bytes"; this agrees with our own conclusion. Golić [Gol97] analyzes the relations between bits of an RC4 keystream using the linear model approach and shows that RC4 can be distinguished from other keystream ciphers and the word size can be recovered (in our model it is equal to 8 bits).

6 Conclusions

We have shown, that for the RC4 stream cipher, every key has a family of related keys which result in a substantially similar keystream. The strength of the RC4 key does not grow linearly with the increase in the key length. However, the weakness we have shown affects only the beginning of the keystream and only manifests itself when extremely long (*i.e.*, non-repeating) keys are used. If RC4 is deployed using keys longer than the customary 128 bits, we advise discarding the first 256 bytes of the keystream.

Acknowledgments

This paper began as a discussion with David Wagner about how RC4 might be useful for a class project to study encryption. All the students in the spring 1999 computer security class at Rice implemented a simpler version of the work described here. The authors also wish to thank Sameer Siruguri, who contributed to the original RC4 test harness used in this paper.

References

[DA99] Tim Dierks and Christopher Allen. *The TLS Protocol, Version 1.0*. Internet Engineering Task Force, January 1999. RFC-2246, `ftp://ftp.isi.edu/in-notes/rfc2246.txt`.

[Gol97] Jovan Dj. Golić. Linear statistical weakness of alleged RC4 keystream generator. In *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238, Konstanz, Germany, May 1997. Springer-Verlag.

[KMP+98] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege. Analysis methods of (alleged) RC4. In *Advances in Cryptology – ASIACRYPT 98*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–341, Beijing, China, October 1998. Springer-Verlag.

[KSW96] John Kelsey, Bruce Schneier, and David Wagner. Key-schedule cryptanalysis of IDEA, G-DES, GHOST, SAFER, and Triple-DES. In *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes* *in Computer Science*, pages 237–251, Santa Barbara, California, August 1996. Springer-Verlag.

[KT99] Kalle Kaukonen and Rodney Thayer. *A Stream Cipher Encryption Algorithm "Arcfour"*. Internet Engineering Task Force, July 1999. Internet draft, `http://search.ietf.org/internet-drafts/draft-kaukonen-cipher-arcfour-03.txt`.

[Rob] Matthew J. B. Robshaw. Security of RC4. Technical Report TR-401, RSA Laboratories. Unpublished.

[Roo95] Andrew Roos. A class of weak keys in the RC4 stream cipher. September 1995. `sci.crypt.research` posting, `http://www.deja.com/getdoc.xp?AN=110586157&fmt=text`.

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, New York, New York, 2nd edition, 1996.