Improved distinguishers for HC-128

Stankovski, Paul; Ruj, Sushmita; Hell, Martin; Johansson, Thomas

Published in:
Designs, Codes and Cryptography

DOI:
10.1007/s10623-011-9550-9

2012

Improved Distinguishers for HC-128*

Paul Stankovski[1], Sushmita Ruj[2], Martin Hell[1], and Thomas Johansson[1]

[1] Deptartment of Electrical and Information Technology,
Lund University, P.O. Box 118, 221 00 Lund, Sweden
[2] School of Electrical Engineering and Computer Science,
University of Ottawa, Ottawa Ontario K1N6N5, Canada

Abstract. HC-128 is an eSTREAM final portfolio stream cipher. Several authors have investigated its security and, in particular, distinguishing attacks have been considered. Still, no one has been able to provide a distinguisher stronger than the one presented by Wu in the original HC-128 paper. In this paper we first argue that the keystream requirement in Wu's original attack is underestimated by a factor of almost 2^8. Our revised analysis shows that the keystream complexity of Wu's original attack is $2^{160.471}$ 32-bit keystream blocks. We then go on to investigate two new types of distinguishers on HC-128. One of them, a distinguisher counting the number of zeros in created blocks of bits, gives a biased distribution that requires $2^{143.537}$ such constructed block samples ($2^{152.537}$ 32-bit keystream blocks). For fairness, the same metric is used to compare our attack to Wu's, and our improvement is significant compared to Wu's original result. Furthermore, the vector-based methodology used is general and can be applied to any cryptographic primitive that reveals a suitable probability distribution.

Keywords: stream cipher, HC-128, cryptanalysis, distinguisher

1 Introduction

HC-128 [13] is a stream cipher selected for the eSTREAM [5] final portfolio [4]. Being in the Profile 1 category it is suitable for fast encryption in software. In fact, in most reported results HC-128 is the fastest stream cipher in the eSTREAM final portfolio. As one example, on a Pentium M processor, the speed of HC-128 reaches 3.05 cycles/byte [13]. This makes HC-128 a very interesting target for cryptanalysis.

HC-128 was proposed in 2006 and there have been very few cryptanalytic results on the cipher. In the design paper [13], Wu constructed a distinguisher based on the least significant bit of the 32-bit keystream words. According to our revised analysis, $2^{160.471}$ keystream words are required to distinguish the output of the cipher from a random sequence.

* The final publication is available at springerlink.com

Subsequent results should be compared to this figure. To the best of our knowledge there is still no distinguisher presenting better performance than the one originally proposed by Wu. Maitra et al. [9] showed that the "least significant bit"-based distinguisher constructed by Wu can be generalized to work for any one bit of the complete 32-bit word. However, the bias of the other bits is smaller than that of the least significant bit. Thus, they present several new distinguishers, all of which are weaker than Wu's distinguisher.

Dunkelman [3] observed that keystream bits would leak information of the secret state. However, this observation has not yet been exploited in designing a distinguisher, and it was argued by Wu [12] that it cannot be used for cryptanalysis at all. The initialization step was analyzed by Liu and Qui in [8]. They showed that the key can be recovered if the internal state of the cipher is known. In [10], Paul et al. showed that it was possible to construct one state array with knowledge of the other array and 2048 keystream words. The time complexity for this reconstruction is 2^{42}. While not saying much about the security of HC-128, those results give more insight into the algorithm itself, possibly providing a foundation for future attacks.

Differential fault analysis of HC-128 was performed in [7]. It was shown that by injecting faults into the state, without control over the location or value of the fault, it is possible to recover the internal state.

1.1 Contributions of This Paper

In this paper, we consider distinguishers for the HC-128 keystream. First we argue that the result given by Wu is not correct. The amount of keystream needed is almost a factor of 2^8 higher than originally claimed. Then we give two new distinguishing attacks, both improving the attack given by Wu. First we generalize Wu's original distinguisher, which only considered the least significant bit of the keystream words, to consider several bits simultaneously. This gives a very slight improvement, about a factor of 2. A more significant improvement is achieved by looking at long vectors of the least significant bit of the keystream words. It is shown that this attack succeeds with high probability if $2^{152.537}$ keystream words are known.

While the attack complexities of the distinguishing attacks in this paper exceeds that of an exhaustive key search, we still believe that this paper presents a significant contribution. As HC-128 is in the final eSTREAM portfolio, additional understanding of the security of this stream cipher is important. It is clear that considering words instead of a single

bit in Wu's original distinguisher does not worsen the attack, but the exact improvement has previously been an open question. This question is resolved in our paper. Moreover, by treating the information from the keystream differently, we are able to significantly improve the keystream complexity of the attack. Our new attack is the first attack requiring less keystream than Wu's original attack, presented in 2006. It is possible that these new ideas can lead to even better attacks, perhaps reducing the keystream complexity to a number below 2^{128}.

The paper is organized as follows. In Section 2 we describe distinguishing attacks and hypothesis testing. This provides the necessary tools to determine the complexity of our attack. In Section 3 we give an overview of HC-128 and then, in Section 4, we present Wu's original distinguishing attack and argue that the keystream complexity is underestimated by a factor of almost 2^8 due to dependencies. Then in Section 5 we give our first new distinguisher based on the idea of considering words instead of bits as was done in the original attack. This distinguisher generalize the efforts of Maitra et al. [9], showing that it *is* possible to construct a (slightly) more efficient distinguisher in this way. In Section 6 we give our second distinguisher based on the idea of considering a large vector of the least significant bit, thereby extracting more information from the keystream. The paper is concluded in Section 7.

2 Preliminaries

In this paper we will consider distinguishing attacks on the keystream generator HC-128. A distinguisher in this case is an algorithm that takes a sequence of n samples as input and outputs either "HC-128" or "RANDOM". The algorithm uses the n input samples to perform a hypothesis test. Knowing the two distributions corresponding to HC-128 and random, the algorithm determines which distribution is most likely, given the n samples. The optimal hypothesis test is given by the Neyman-Pearson lemma.

Lemma 1 (Neyman-Pearson). *Let $X_1, X_2, \ldots, X_n$ be drawn iid (independent identically distributed) according to mass function P_X. Consider the decision problem corresponding to the hypotheses $P_X = P_0$ vs. $P_X = P_1$. For $T \geq 0$ define a region*

$$\mathcal{A}_n(T) = \left\{ \frac{P_0[x_1, x_2, \ldots, x_n]}{P_1[x_1, x_2, \ldots, x_n]} > T \right\}. \tag{1}$$

Let $\alpha = P_0^n[\mathcal{A}_n^c(T)]$, where c denotes complement, and $\beta = P_1^n[\mathcal{A}_n(T)]$ be

the error probabilities corresponding to the decision region $\mathcal{A}_n$. Let $\mathcal{B}_n$ be any other decision region with associated error probabilities α^* and β^*. If $\alpha^* \leq \alpha$, then $\beta^* \geq \beta$.

Assuming that all samples are independent, (1) is equivalent to

$$\mathcal{A}_n(T) = \left\{ \sum_{i=1}^{n} \log \left(\frac{P_0[x_i]}{P_1[x_i]} \right) > \log T \right\}.$$

The most important measurement of the power of a distinguisher in this setting is the number of samples that is required for it to make a correct decision with probability significantly larger than 1/2. We start by defining the relative entropy between two distributions.

Definition 1. *The relative entropy between two probability distributions P_0 and P_1 over the same domain $\mathcal{X}$ is defined as*

$$D\left(P_0 \| P_1\right) = \sum_{x \in \mathcal{X}} P_0[x] \log \frac{P_0[x]}{P_1[x]}. \tag{2}$$

The relative entropy is sometimes also referred to as information divergence or Kullback-Leibler distance.

In a hypothesis test we observe a collection of independent and identically distributed samples drawn from a distribution P_X. This will give us two possible hypotheses, the null hypothesis H_0 and the alternate hypothesis H_1:

$$H_0 : P_X = P_0,$$
$$H_1 : P_X = P_1.$$

There are two types of errors associated with our hypotheses. We can reject the null hypothesis when it is in fact true, a type I error. The probability of this error is denoted α. The other alternative is that we accept the null hypothesis when the alternate hypothesis is true, a type II error. The probability of this error is denoted β.

No general expression for the error probabilities α and β exists. Thus, we know how to perform the optimal test using the Neyman-Pearson lemma, but we do not know the performance of the test (in the general case). However, there exist *asymptotic* expressions for the error probabilities.

The relative entropy is related to the asymptotic error probabilities through Stein's lemma. Stein's lemma states that if we fix the error probability α then β decreases so that

$$\lim_{n \to \infty} \frac{\log \beta}{n} = -D(P_0 \| P_1). \tag{3}$$

The value of α does not affect the exponential rate at which β decreases. According to (3) we can asymptotically write

$$\beta \approx 2^{-nD(P_0\|P_1)}. \tag{4}$$

When the number of samples used in the hypothesis test approaches

$$n \approx \frac{1}{D(P_0\|P_1)}, \tag{5}$$

we reach the point at which the error probability starts to decrease exponentially. In this paper we will use (5) as a measure of the number of samples that are needed in our distinguishing attack. However, we emphasize that in a practical scenario we would actually need a small multiple of this number. Still, as we are comparing our attack to previous attacks, if we use (5) for all attacks, the comparison is indeed fair. Other ways of estimating the number of samples needed in the distinguisher exist, and we refer to [1, 2, 6] for a more detailed treatment.

3 Brief Description of HC-128

In this section we give a brief description of HC-128. The size of the key K is 128 bits and the initialization vector IV used is also 128 bits. According to the proposal, a keystream of length up to 2^{64} bits per key/IV pair can be generated.

3.1 Notations

We use the same notations as in [13]. Let x and y be 32-bit integers. Then,

- $+$: $x + y$ means $x + y \bmod 2^{32}$,
- $\boxminus$: $x \boxminus y$ means $x - y \bmod 512$,
- $\oplus$: bit-wise exclusive OR,
- $||$: concatenation,
- $\gg$: Right shift, $x \gg n$ means x right shifted by n bits,
- $\ll$: Left shift, $x \ll n$ means x left shifted by n bits,
- $\ggg$: Right shift with rotation, $x \ggg n$ means $((x \gg n) \oplus (x \ll (32 - n))$, where $0 \leq n < 32$,
- $\lll$: Left shift with rotation, $x \lll n$ means $((x \ll n) \oplus (x \gg (32 - n))$, where $0 \leq n < 32$.

Two tables, denoted P and Q, make up the internal state of HC-128. Each of the tables contain 512 32-bit elements, or words. The keystream is denoted by s and the 32-bit keystream word generated at the i^{th} step is denoted s_i. Thus $s = s_0||s_1||s_2||\ldots$. The following six functions are used in HC-128.

$$\begin{aligned}
f_1(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3) \\
f_2(x) &= (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10) \\
g_1(x,y,z) &= ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8) \\
g_2(x,y,z) &= ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8) \\
h_1(x) &= Q[x_0] + Q[256 + x_2] \\
h_2(x) &= P[x_0] + P[256 + x_2]
\end{aligned}$$

where $x = x_3||x_2||x_1||x_0$, x is a 32-bit word and the x_i represent a byte each, x_0 being the least significant byte of the word and x_3 being the most significant byte. The functions $f_1(x)$ and $f_2(x)$ are only used in the initialization of the cipher. As our analysis is independent of the initialization it will not be detailed here. Instead we refer to [13] for a description.

3.2 The Keystream Generation Algorithm

The keystream generation algorithm is the subject of our analysis. At each step, one element of a table is updated and one 32-bit keystream word is generated. One table is used to generate 512 consecutive keystream words. That table is then updated during the following 512 steps. The keystream generation algorithm of HC-128 is given in Fig. 1.

4 Original Distinguishing Attack by Wu

In the HC-128 design paper, a distinguishing attack was given based on the least significant bit of the keystream word s_i. At the i^{th} step, if $(i \bmod 1024) < 512$, the table P is updated as

$$P[i \bmod 512] \mathrel{+}= g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511]).$$

Also, $s_i = h_1(P[i \boxminus 12]) \oplus P[i \bmod 512]$. For $10 \leq (i \bmod 1024) < 511$, this feedback function can be written alternatively as

$$\begin{aligned}
s_i \oplus h_1(z_i) = (s_{i-1024} \oplus h_1'(z_{i-1024})) + \\
g_1(s_{i-3} \oplus h_1(z_{i-3}), s_{i-10} \oplus h_1(z_{i-10}), s_{i-1023} \oplus h_1'(z_{i-1023})).
\end{aligned} \tag{6}$$

Algorithm 1 – HC-128 Keystream Generation

Input: initialized tables P and Q, each containing 512 32-bit words.
Output: 32-bit keystream words s_i for $i = 0, 1, \ldots$.

```
i = 0;
repeat (until enough keystream bits are generated) {
    j = i mod 512;
    if ((i mod 1024) < 512) {
        P[j] += g1(P[j ⊟ 3], P[j ⊟ 10], P[j ⊟ 511]);
        s_i = h1(P[j ⊟ 12]) ⊕ P[j];
    } else {
        Q[j] += g2(Q[j ⊟ 3], Q[j ⊟ 10], Q[j ⊟ 511]);
        s_i = h2(Q[j ⊟ 12]) ⊕ Q[j];
    }
    i += 1;
}
```

Fig. 1. HC-128 keystream generation.

$h_1(x)$ and $h_1'(x)$ indicate two different functions because they refer to different S-boxes; z_j denotes the $P[j \boxminus 12]$ at the j^{th} step.

As shown in [13], for the least significant bit, this equation reduces to

$$\begin{aligned}&[s_i]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23} \oplus [s_{i-1024}]^0 \\ &= [h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h_1'(z_{i-1023})]^{23} \oplus [h_1'(z_{i-1024})]^0,\end{aligned}$$

where $[a]^i$ represents the i^{th} least significant bit of a. Looking at two *different* time instances i and j where $1024 \times \gamma + 10 \le i, j < 1024 \times \gamma + 511$ we can write

$$\begin{aligned}&[s_i]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23} \oplus [s_{i-1024}]^0 \\ =&[s_j]^0 \oplus [s_{j-3}]^{10} \oplus [s_{j-10}]^8 \oplus [s_{j-1023}]^{23} \oplus [s_{j-1024}]^0,\end{aligned} \tag{7}$$

which holds if and only if

$$\begin{aligned}&[h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h_1'(z_{i-1023})]^{23} \oplus [h_1'(z_{i-1024})]^0 \\ =\ &[h_1(z_j)]^0 \oplus [h_1(z_{j-3})]^{10} \oplus [h_1(z_{j-10})]^8 \oplus [h_1'(z_{j-1023})]^{23} \oplus [h_1'(z_{j-1024})]^0.\end{aligned} \tag{8}$$

We call these expressions t_i and t_j, respectively, so that

$$t_i = [h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h_1'(z_{i-1023})]^{23} \oplus [h_1'(z_{i-1024})]^0.$$

Equation (8) can be approximated as

$$H(a_1) = H(a_2), \tag{9}$$

where H denotes a random secret 80-to-1-bit S-box, a_1 and a_2 are two 80-bit random inputs,

$$a_1 = \overline{z}_i||\overline{z}_{i-3}||\overline{z}_{i-10}||\overline{z}_{i-1023}||\overline{z}_{i-1024},$$
$$a_2 = \overline{z}_j||\overline{z}_{j-3}||\overline{z}_{j-10}||\overline{z}_{j-1023}||\overline{z}_{j-1024},$$

where $\overline{z}$ indicates the concatenation of the least significant byte and the second most significant byte of z, i.e., $\overline{z} = x_0||x_2$. The following theorem, from [13], shows the bias of (9).

Theorem 1. *Let H be an m-to-n-bit S-box and all those n-bit elements are randomly generated, where $m \geq n$. Let a_1 and a_2 be two m-bit random inputs to H. Then $H(a_1) = H(a_2)$ with probability $2^{-m} + 2^{-n} - 2^{-m-n}$.*

Thus, (9) holds with probability $\frac{1}{2} + \varepsilon = \frac{1}{2} + 2^{-81}$. Approximating the number of samples needed in a distinguisher by $4\varepsilon^{-2}$, Wu concludes that 2^{164} such equations are needed. Since it is possible to obtain $\binom{501}{2} \approx 2^{17}$ pairs for each 512-word keystream chunk, the number of keystream words needed are concluded to be 2^{156}, a factor of $\frac{\binom{501}{2}}{501} \approx 2^8$ less. This is however not correct. Let us explain why. The distinguisher considers two different cases, one when the output is from the cipher (HC-128) and the other one when the output is truly random and bits are independent. When we consider the random case, a block contains 512 truly random bits. These give rise to 501 truly random values t_i, where $10 \leq i < 511$. But when we then create 2^{17} samples as $t_i \oplus t_j$, $10 \leq i, j < 511$, these are no longer independent, and counting these samples will have a larger variance compared to independent samples. So in order to keep the samples independent, we can only use 501 samples and the number of required keystream words is thus still 2^{164} in Wu's attack.

Wu claims that the keystream complexity of his distinguisher can be lowered by a factor of almost 2^8 using his dependent sampling technique. Our simulations[3] suggest otherwise. They clearly show that this kind of dependent sampling can be much less efficient. The consequence of this is that more samples need to be used with Wu's sampling method. In this sense, the dependent samples are not "high quality", they contain less information than the independent ones.

[3] The keystream requirement of the 2-vector distinguisher defined in Section 6 was measured for both independent and dependent samples using Eq. (1) with $T = 1$ and a false negative rate of 0.01. The main difference to Wu's attack is that an 8–to–1 S-box was sampled instead of an 80–to–1 S-box. Dependent sampling lowered the keystream requirement by a factor of about 2^3, as opposed to the 2^8 suggested by Wu. It is not immediately clear how these observations translate to the 80–to–1 S-box case.

When it comes to assessing the efficiency level of Wu's dependent sampling, this is not a trivial task. It is unclear how much the keystream complexity can be lowered using this technique. Wu himself provides neither analytical nor simulation results to back his assumption of perfect efficiency. His analysis remains correct for the standard setting of linear sampling, however, so this is what we will assume henceforth.

This being said, our simulations do show that *some* improvement gain in keystream requirement can result from using dependent sampling. Our simulations further suggest that the vector approach detailed in Section 6 improves the result of Wu's distinguisher in this regard as well. Sampling all $\binom{501}{3}$ triplets for the 3-vector distinguisher is more keystream-efficient than sampling all $\binom{501}{2}$ pairs for the 2-vector case. The approach is trivially generalized into sampling all $\binom{501}{250}$ 250-sets for the 250-vector distinguisher.

To conclude, we continue the analysis and comparison of our attacks to Wu's using only *independent* samples for both attacks. As we are using the relative entropy measure in this paper, we give the corresponding figure using this metric also for Wu's attack. This allows us to make a fair comparison between our results and Wu's attack. With $p = \frac{1}{2} + 2^{-81}$ we get $D(P_0||P_1) = 2^{-160.471}$ giving a distinguisher requiring $2^{160.471}$ samples. According to the arguments given above this also corresponds to $2^{160.471}$ keystream words. For comparison, our best distinguisher (in Section 6) will be shown to have both a computational and keystream complexity of $2^{152.537}$.

5 A New Word-Based Distinguisher

Wu's distinguisher considered the least significant bit of each keystream word. We now construct a word-based distinguisher, in which we consider the w least significant bits together. Since the least significant bit is included in the w least significant bits, we know that the attack can never get worse by including more bits. The main question that we aim to answer here is exactly *how* much better the word-based distinguisher is.

We use $[s_i]^{<w>}$ to denote the w least significant bits of $[s_i]$ and $[s_i]^{<w+b>}$ to denote bits $b, b+1, \ldots, b+w-1$ of $[s_i]$. For the least significant

w bits $0, 1, 2, \ldots, w-1$, equation (6) can be written as

$$\begin{aligned} & [s_i]^{<w>} \oplus [s_{i-3}]^{<w+10>} \oplus [s_{i-10}]^{<w+8>} \oplus \\ & [s_{i-1023}]^{<w+23>} \oplus [s_{i-1024}]^{<w>} \\ = & [h_1(z_i)]^{<w>} \oplus [h_1(z_{i-3})]^{<w+10>} \oplus [h_1(z_{i-10})]^{<w+8>} \oplus \\ & [h_1'(z_{i-1023})]^{<w+23>} \oplus [h_1'(z_i - 1024)]^{<w>} + \text{carry}. \end{aligned} \tag{10}$$

We write this as $[t_i]^{<w>} = H_w(a_i) + \text{carry}$. The carry contribution comes from the approximation of '+' by $\oplus$, and this part will be discussed in Section 5.1 below. The main goal is to find the distribution of $[t_i]^{<w>} \oplus [t_j]^{<w>}$, where i and j are chosen as in (7). Once this is known we can use the relative entropy to determine the number of samples needed in the distinguisher.

Approximate $H_w(a_i)$ as a 80-to-w-bit S-box and use Theorem 1 to show that

$$\begin{aligned} & \Pr(H_w(a_i) \oplus H_w(a_j) = 0) = \\ & \qquad 2^{-80} + 2^{-w} - 2^{-(80+w)} = \frac{1}{2^w}(1 + \frac{2^w - 1}{2^{80}}), \\ & \Pr(H_w(a_i) \oplus H_w(a_j) = u | u \neq 0) = \\ & \qquad \frac{1}{2^w - 1}(1 - (2^{-80} + 2^{-w} - 2^{-(80+w)})) = \frac{1}{2^w}(1 - \frac{1}{2^{80}}). \end{aligned}$$

Let

$$p_l = \sum_{u,v \in \{0,1\}^n} \Pr([t_i]^{<w>} \oplus [t_j]^{<w>} = u | H_w(a_i) \oplus H_w(a_j) = v),$$

where $l = u \oplus v$. Thus, we have $\sum_{l \in \{0,1\}^w} p_l = 1$.

To show how to compute the distribution of $[t_i]^{<w>} \oplus [t_j]^{<w>}$ we consider two cases separately, namely when the least significant bit of l is 0 and 1 respectively. We let $T_{(w,i,j)}$ and $H_{(w,i,j)}$ denote $[t_i]^{<w>} \oplus [t_j]^{<w>}$ and $H_w(a_i) \oplus H_w(a_j)$, respectively.

Case 1: The least significant bit (LSB) of l is 1.

In this case we can write

$$\begin{aligned}
&\Pr(T_{(w,i,j)} = l) \\
&= \sum_{q \in \{0,1\}^w} \Pr(T_{(w,i,j)} = l | H_{(w,i,j)} = q) \Pr(H_{(w,i,j)} = q) \\
&= \frac{1}{2^w}(1 - \frac{1}{2^{80}}) \sum_{q \in \{0,1\}^w \setminus \{0\}^w} \Pr(T_{(w,i,j)} = l | H_{(w,i,j)} = q) \\
&= \frac{1}{2^w}(1 - \frac{1}{2^{80}}).
\end{aligned}$$

The last equation follows from $\Pr(T_{(w,i,j)} = l | H_{(w,i,j)} = 0) = 0$. We see that the probability is always the same when the LSB of l is 1.

Case 2: The LSB of l is 0.

$$\begin{aligned}
&\Pr(T_{(w,i,j)} = l) \\
&= \sum_{q \in \{0,1\}^w} \Pr(T_{(w,i,j)} = l | H_{(w,i,j)} = q) \Pr(H_{(w,i,j)} = q) \\
&= \Pr(T_{(w,i,j)} = l | H_{(w,i,j)} = 0) \Pr(H_{(w,i,j)} = 0) \\
&+ \sum_{q \in \{0,1\}^w \setminus \{0\}^w} \Pr(T_{(w,i,j)} = l | H_{(w,i,j)} = q) \Pr(H_{(w,i,j)} = q) \\
&= \frac{1}{2^w}(1 + \frac{2^w - 1}{2^{80}}) p_l + \frac{1}{2^w}(1 - p_l)(1 - \frac{1}{2^{80}}) \\
&= \frac{1}{2^w}(1 + \frac{1}{2^{80}}(p_l 2^w - 1)).
\end{aligned}$$

Finding $\Pr(T_{(w,i,j)} = l)$ in this case reduces to finding p_l.

5.1 The Effect of Carry Bits in the Linear Approximation

We now analyze the influence of the carry introduced in (10). When only the least significant bit is considered there is no influence of any carry. In our case, this analysis is needed in order to compute p_l when $w > 1$. First we note that there are two '+' in (6) that are approximated by $\oplus$. Linear approximations of modulo 2^w additions of k integers was studied by Staffelbach and Meier [11]. Maitra et al. [9] extended this idea to approximate the feedback functions g_1 and g_2 by replacing the two '+' with $\oplus$ operators. However they considered the effect of the approximation on only one bit. We extend this idea to words of several bits.

Let X, Y, Z be three w-bit integers and let $A = (X + Y + Z) \mod 2^w$ and $B = X \oplus Y \oplus Z$. We consider the carries that are produced in each bit position of $A \oplus B$. Let $C = A \oplus B$, so that C_i represents the carry generated at the i^{th} least significant bit due to the linear approximation. The least significant bit of A will always be equal to the least significant bit of B, because there are no carries that affect this bit position. We consider the carries from the w least significant bits and give a recursive formula for the distribution of the w least significant carry bits.

We denote the probability $\Pr(A \oplus B = s)$ by p_k^i, where i is the integer representation of s and k is the number of bits under consideration, $s = s_{k-1}s_{k-2}\cdots s_1 s_0$. As a starting point, it is easy to see that $\Pr(A \oplus B = 00) = p_2^0 = \frac{1}{2}$, $\Pr(A \oplus B = 01) = p_2^1 = 0$, $\Pr(A \oplus B = 10) = p_2^2 = \frac{1}{2}$ and $\Pr(A \oplus B = 11) = p_2^3 = 0$. Now, suppose we know the distribution of $A \oplus B$ for the k least significant bits, then we can find the distribution of $A \oplus B$ for the $k+1$ least significant bits. Note that $\Pr(A \oplus B = s_k 0 s_{k-2}s_{k-3}\cdots s_0) = \frac{1}{2}\Pr(A \oplus B = 0 s_{k-2}s_{k-3}\cdots s_0)$, since the carry value at position $(k-1)$ does not affect the carry value at position k. Hence, $p_{k+1}^i = \frac{1}{2}p_k^i$ for $0 \le i < 2^{k-1}$, when bit $(k-1)$ is zero and $p_{k+1}^i = \frac{1}{2}p_k^{i-2^k}$ for $2^k \le i < 3 \cdot 2^{k-1}$. If the carry bit at position $k-1$ is 1, then the Table 1 lists the possible carry bit values at position k. We see that $p_{k+1}^i = \frac{1}{4}p_k^i$

Table 1. The values of carry bits at position k.

X_k	Y_k	Z_k	C_{k-1}	C_k
0	0	0	1	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

for $2^{k-1} \le i < 2^k$ and $p_{k+1}^i = \frac{3}{4}p_k^{i-2^k}$ for $3 \cdot 2^{k-1} \le i < 2^{k+1}$.

We can now give a recursive formula for generating carries and thus for approximating the '+' by $\oplus$ operators. Starting out with the basic case $\Pr(A \oplus B = 00) = p_2^0 = \frac{1}{2}$, $\Pr(A \oplus B = 01) = p_2^1 = 0$, $\Pr(A \oplus B = 10) = p_2^2 = \frac{1}{2}$, $\Pr(A \oplus B = 11) = p_2^3 = 0$, the carry generated for the least

$k+1$ significant bits is given by

$$p_{k+1}^i = \begin{cases} \frac{1}{2}p_k^i & 0 \le i < 2^{k-1}, \\ \frac{1}{4}p_k^i & 2^{k-1} \le i < 2^k, \\ \frac{1}{2}p_k^{i-2^k} & 2^k \le i < 3 \cdot 2^{k-1}, \\ \frac{3}{4}p_k^{i-2^k} & 3 \cdot 2^{k-1} \le i < 2^{k+1}. \end{cases}$$

As an example, for word size $w = 3$ we have $\Pr(A \oplus B = 000) = \frac{1}{4}$, $\Pr(A \oplus B = 001) = 0$, $\Pr(A \oplus B = 010) = \frac{1}{8}$, $\Pr(A \oplus B = 011) = 0$, $\Pr(A \oplus B = 100) = \frac{1}{4}$, $\Pr(A \oplus B = 101) = 0$, $\Pr(A \oplus B = 110) = \frac{3}{8}$, $\Pr(A \oplus B = 111) = 0$.

Utilizing this recursive expression we have computed the distribution of $[t_i]^{<w>} \oplus [t_j]^{<w>}$ for $1 \le w \le 18$. The inverse of the relative entropy between the cipher distribution and the uniform distribution is given in Table 2. This corresponds to the number of samples needed in the distinguisher. Note that the original attack by Wu corresponds to word size 1. It is clear that the attack is improved, though the gain is small. Still, this settles the open problem of determining the advantage of considering a word-based distinguisher for HC-128. We can note that when the least significant two bits are considered then the required number of samples is same as that for $w = 1$. This is consistent with the result of Maitra et al. [9] who stated that the second least significant bit does not affect the bias in any way.

When implementing the word-based distinguisher in practice, one may use 2^w counters. From all keystream words, compute the w-bit result according to (10) and increase the corresponding counter. When finished counting, perform the hypothesis test on the resulting distribution (the counter values) to see if the HC-128 or the ideal probability distribution yields the closest fit.

6 A New Vector-Based Distinguisher

In [13] Wu found a bias in t_i,

$$t_i = [s_i]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23} \oplus [s_{i-1024}]^0, \tag{11}$$

by considering pairs $(t_i, t_j), i \neq j$, and xoring them to form $t_i \oplus t_j$. We now extend this idea to binary vectors of length n with $2 \le n \le 501$. Take n consecutive t_i's and form the vector

$$(t_i, t_{i+1}, \ldots, t_{i+n-1}) .$$

Table 2. Word size and number of samples (32-bit keystream words) required by the distinguisher.

Word size	Samples	Word size	Samples
1	$2^{160.471}$	10	$2^{160.002}$
2	$2^{160.471}$	11	$2^{159.940}$
3	$2^{160.449}$	12	$2^{159.878}$
4	$2^{160.396}$	13	$2^{159.818}$
5	$2^{160.331}$	14	$2^{159.758}$
6	$2^{160.264}$	15	$2^{159.700}$
7	$2^{160.198}$	16	$2^{159.642}$
8	$2^{160.131}$	17	$2^{159.585}$
9	$2^{160.066}$	18	$2^{159.529}$

Consider the weight $w = \sum_{k=0}^{n-1} t_{i+k}$ of such vectors. We will study weight probability distributions W to show that HC-128 behavior is less than ideal. That is, we want to find $P(W = w)$ for all values of $0 \leq w \leq n$, both for the cipher and the ideal case. Once this is done, we can assert the efficiency of our distinguisher by comparing the two distributions.

We will first go through the conceptual structure of our problem and state the ideal weight probability distribution in Section 6.1. We will then show how to calculate the cipher's weight probability distribution for a simplified case in Section 6.2. We then construct the full weight probability distribution for the cipher by learning how to combine different weight probability distributions. This is covered in Section 6.3, where we also derive the resulting distinguisher performance.

6.1 The Ideal Vector Weight Probability Distribution

Each bit t_i in the vector is derived from several lookups into the same table Q (or P). Recall the expression

$$t_i = [h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h_1'(z_{i-1023})]^{23} \oplus [h_1'(z_{i-1024})]^0.$$

Table Q (or P) contains 512 32-bit words, and lookup into Q (P) is divided into the upper and lower half according to function h_1 (h_2). Note that h_1 and h_1' denote lookup into different tables, and note also that we only use one bit from each 32-bit lookup value for t_i so that the table bit

positions differ for each application of h_1. Thus, t_i is essentially derived by xoring lookups from ten *different* lookup tables of size 256 (with binary entries). One may view t_i as the result of xoring the output of ten different (non-invertible) 8-to-1-bit S-boxes. The conceptual composition of t_i is illustrated in Fig. 2.

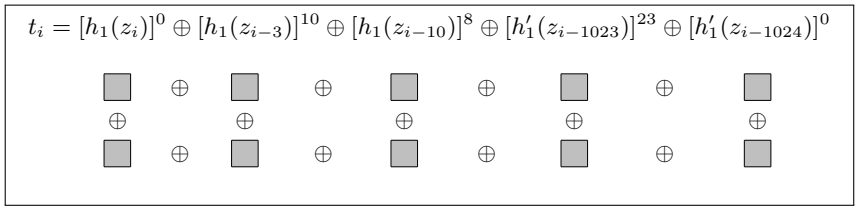

Fig. 2. The conceptual anatomy of t_i.

HC-128 reuses the same tables with fixed values during several consecutive time instances, so t_i and t_j with $i \neq j$ are pairwise dependent for this period of time. But we first consider the ideal case, in which *fresh* 8-to-1-bit tables are used for every time instance, so that t_i and t_j with $i \neq j$ are independent.

Let W_{ideal} denote a stochastic variable that returns the weight of a vector produced according to the above stated conditions. Every vector is equally likely in the ideal case, so the resulting vector weight probability distribution is combinatorially determined by

$$\Pr(W_{\text{ideal}} = w) = \binom{n}{w} 2^{-n}.$$

6.2 The Cipher's Vector Weight Probability Distribution

Calculating the weight probability distribution of the cipher is much more complicated. We now reuse the same tables, so t_i and t_j with $i \neq j$ are pairwise dependent. We begin by simplifying this case by assuming that only *one* 8-to-1-bit table is used to produce each vector bit t_i.

At first glance, deriving this probability distribution for vector lengths n approaching 500 is no walk in the park. We will first show how this distribution can be recursively determined for the simplified case with one table. We hope that this algorithm will clearly illustrate how the probabilities are derived. The recursiveness is an impediment as n approaches

500, of course, so we also show how to compute the same distribution more efficiently using dynamic programming.

With the recursive approach, one would hope that full knowledge of the weight distribution for vectors of length k would be easily deduced from the distribution for vectors of length $k-1$. This is, however, not the case. A vector of length k and weight w can be derived from a vector of length $k-1$ with

a) weight w if
   i) a zero is read from a previously opened table entry, or
   ii) a zero is read from a new table entry, or
b) weight $w-1$ if
   i) a one is read from a previously opened table entry, or
   ii) a one is read from a new table entry.

We use the variables a_0 and a_1 to keep track of the number of opened table entries of each sort. The recursive algorithm is given in Fig. 3 in pseudocode resembling C.

Algorithm 2 – *CipherDistribution* **(Recursive)**

Input: maximum depth n (vector length), current depth d, current probability pr, probability distribution container p of length $n+1$, weight w, number of opened table slots with zeros a_0, number of opened table slots with ones a_1.
Output: probability distribution p.
Initial recursion parameters: p zeroized, $(d, pr, w, a_0, a_1) = (0, 1, 0, 0, 0)$.

```
if (d == n) {
    p[w] += pr;
    return;
}
CipherDistribution(p, n, d + 1, pr · a0/256, w, a0, a1); /* old 0 with prob a0/256 */
CipherDistribution(p, n, d + 1, pr · a1/256, w + 1, a0, a1); /* old 1 with prob a1/256 */
if (a0 + a1 < 256) { /* table not exhausted */
    /* new 0 with prob (256−(a0+a1))/512 */
    CipherDistribution(p, n, d + 1, pr · (256−(a0+a1))/512, w, a0 + 1, a1);
    /* new 1 with prob (256−(a0+a1))/512 */
    CipherDistribution(p, n, d + 1, pr · (256−(a0+a1))/512, w + 1, a0, a1 + 1);
}
```

Fig. 3. Cipher distribution calculation (recursive) for one table.

When translating the recursive algorithm into a dynamic programming variant, we use an intermediate storage for all probabilities for vectors of length $k-1$ to deduce the corresponding probabilities for a vector

of length k. Consider the tuple (w, a_0, a_1), for which w indicates weight and the a's indicate how many table entries of each sort that have been opened. The intermediate storage contains one probability for each such tuple, so it is necessary to be able to translate a tuple into an index in the temporary storage and vice versa. This translation is performed by the functions *getTuple* and *getIndex* in the algorithm detailed in Fig. 4.

Algorithm 3 – *CipherDistribution* **(Dynamic Programming)**

Input: vector length n.
Output: probability distribution *distr*.

$m = \frac{257^2}{2}(n+1)$; /* max num probability entries at depth n */
$p = (1, 0, \ldots, 0)$; /* length m */
$q = (0, 0, \ldots, 0)$; /* length m */
for $(d = 0;\ d < n;\ d{+}{+})$ { /* depth d */
  $m_d = \frac{257^2}{2}(d+1)$; /* max num probability entries at depth d */
  for $(e = 0;\ e < m_d;\ e{+}{+})$ { /* entry index e */
    $(w, a_0, a_1) = getTuple(d, e)$;
    $q[getIndex(d+1, (w, a_0, a_1))]$ += $p[e] \cdot \frac{a_0}{256}$; /* old 0 with prob $\frac{a_0}{256}$ */
    $q[getIndex(d+1, (w+1, a_0, a_1))]$ += $p[e] \cdot \frac{a_1}{256}$; /* old 1 with prob $\frac{a_1}{256}$ */
    if $(a_0 + a_1 < 256)$ { /* table not exhausted */
      /* new 0 with prob $\frac{256-(a_0+a_1)}{512}$ */
      $q[getIndex(d+1, (w, a_0+1, a_1))]$ += $p[e] \cdot \frac{256-(a_0+a_1)}{512}$;
      /* new 1 with prob $\frac{256-(a_0+a_1)}{512}$ */
      $q[getIndex(d+1, (w+1, a_0, a_1+1))]$ += $p[e] \cdot \frac{256-(a_0+a_1)}{512}$;
    }
  }
  $p \leftrightarrow q$; /* swap p and q */
  $q = (0, 0, \ldots, 0)$; /* clear q */
}
$distr = (0, 0, \ldots, 0)$; /* length $n+1$ */
for $(e = 0;\ e < m;\ e{+}{+})$ {
  $(w, a_0, a_1) = getTuple(m, e)$;
  $distr[w]$ += $p[e]$;
}

Fig. 4. Cipher distribution calculation (dynamic programming) for one table.

For HC-128 one may note that an upper bound on the memory requirement of a dynamic programming implementation is given by

$$257^2(n+1) \cdot L$$

bytes of memory, where L denotes the size of the number type that is being used for storing the probabilities[4]. The computational complexity[5] is bounded by

$$\frac{257^2}{2}\sum_{k=0}^{n}(k+1) = \frac{257^2(n+1)(n+2)}{4}$$

instructions involving multiple precision multiplications. This algorithm dominates the experiment, both in terms of memory requirements and computational complexity. Reaching the HC-128 limit for $n = 501$ is feasible with a reasonable computer and a healthy measure of patience.

Now that we can obtain the weight probability distribution of the cipher for a single S-box (table), we need to learn how to derive the corresponding distribution for the xor of all ten S-boxes.

6.3 Xoring Weight Probability Distributions

Our simplification so far is that we assume only one S-box. How do we find the corresponding weight probability distribution when we are xoring the output of two S-boxes? Algorithm $XorDistributions$ in Fig. 5 shows how to combine two distributions. Given two different S-boxes, potentially with different weight probability distributions, we still know precisely what to expect from their xored output in terms of vector weight.

The algorithm works as follows. Consider vectors v_1 and v_2 with weights i and j, respectively. If k ones are overlapping, the xored vector $v_1 \oplus v_2$ will contain $i + j - 2k$ ones. The algorithm sums the probabilities over all possible tuple values (i, j, k). The binomial expression $\frac{\binom{i}{k}\binom{n-i}{j-k}}{\binom{n}{j}}$ in the algorithm states the probability of a k-bit overlap given two vectors of weight i and j.

Using this algorithm we can easily and successively deduce the probability distributions for the resulting vector weight over, in turn, 2, 4, 8 and 10 tables.

Last but not least, we need to compare the ideal and the biased probability distributions. We do this by using the divergence measure according to (2). For various vector lengths n and number of S-boxes, the required

[4] 64-bit data types do not provide sufficient precision to measure differences in probability distributions when considering the full 10-table case.

[5] More general upper bound formulas for memory and computational complexity for vectors of size N derived from a table of size T are given by $\frac{T^2NL}{2}$ and $\frac{T^2N^2}{4}$, respectively. These formulas do leave some room for improvement.

Algorithm 4 – *XorDistributions*

Input: vector length n, probability distributions p and q ($n+1$ values each).
Output: probability distribution r.

$r = (0, 0, 0, \ldots, 0)$; /* length $n+1$ */
for ($i = 0$; $i < n+1$; i++) {
  for ($j = 0$; $j < n+1$; j++) {
    for ($k = \max(0, i+j-n)$; $k < \min(i, j)+1$; k++) {
      /* vector v_1 with weight i,
       * vector v_2 with weight j,
       * k ones overlapping $\implies$ $v_1 \oplus v_2$ has weight $i+j-2k$ */
      $r[i+j-2k] \mathrel{+}= p[i] \cdot q[j] \cdot \frac{\binom{i}{k}\binom{n-i}{j-k}}{\binom{n}{j}}$;
    }
  }
}
return r;

Fig. 5. Xoring two weight probability distributions p and q.

number of samples for our distinguisher are presented in Table 3. Note the reappearance of the value $2^{160.471}$ for $n = 2$ over 10 S-boxes that was derived in Section 5 and also corresponds to Wu's attack. As a verification, the case $n = 10$ over one S-box was also simulated to ensure correctness. The simulation results matched the theoretical findings.

Table 3. Number of samples required for various vector lengths n and number of S-boxes.

		vector length n					
		2	10	50	100	250	501
# S-boxes	1	$2^{16.471}$	$2^{11.009}$	$2^{6.380}$	$2^{4.517}$	$2^{2.232}$	$2^{0.665}$
	2	$2^{32.471}$	$2^{26.979}$	$2^{22.213}$	$2^{20.199}$	$2^{17.549}$	$2^{15.544}$
	4	$2^{64.471}$	$2^{58.979}$	$2^{54.213}$	$2^{52.198}$	$2^{49.545}$	$2^{47.537}$
	8	$2^{128.471}$	$2^{122.979}$	$2^{118.213}$	$2^{116.198}$	$2^{113.545}$	$2^{111.537}$
	10	$2^{160.471}$	$2^{154.979}$	$2^{150.213}$	$2^{148.198}$	$2^{145.545}$	$2^{143.537}$

Using this analysis, the actual distinguishing attack on HC-128 would proceed as follows. Instantiate $n+1$ counters, one for each possible vector weight. From all keystream words, compute the n-bit vector samples by repeatedly applying (11). Based on the resulting vector weight, increase the corresponding counter. When finished counting, perform the hypoth-

esis test on the resulting distribution (the counter values) to see if the HC-128 or the ideal probability distribution yields the closest fit.

Our best result for the full HC-128 with 10 S-boxes is a requirement of $2^{143.537}$ samples for $n = 501$. Each sample is derived from the least significant bits of n keystream words, so the total complexity is no more than $2^{152.537}$ keystream words.

The reader may further note that the time units used here correspond to very simple operations involving only xor and shifts of keystream words. These operations are *much* cheaper than the initializations considered in a brute-force search. One key initialization takes about 27300 clock cycles [13], involving both a key expansion and 1024 table updates. For comparison, one single key initialization corresponds to processing about 2^{10} keystream words using our simple operations.

7 Conclusion

We have presented two new distinguishing attacks on the eSTREAM portfolio stream cipher HC-128. Both techniques use the underlying idea of using more information from the keystream than obtained from a simple xor of the least significant bit. The first idea is based on looking at several bits in the keystream words simultaneously. This gives a slight improvement over the original attack. The second technique puts bits in a long vector and considers the distribution of the weight of this vector. Using non-trivial techniques we are able to compute the distribution of the vector weight and we show that the corresponding distinguishing attack requires $2^{152.537}$ keystream words. This is the most efficient distinguishing result known for HC-128. It exploits more information than any other distinguisher, and it is an open problem if it is possible to improve this result even further.

Acknowledgements
This work was sponsored in part by the Swedish Research Council (Vetenskapsrådet) under grant 621-2006-5249.

References

1. Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Advances in Cryptology—ASIACRYPT 2004, *Lecture Notes in Computer Science*, vol. 3329, pp. 432–450. Springer-Verlag (2004)
2. Cover, T., Thomas, J.A.: Elements of Information Theory. Wiley series in Telecommunication. Wiley (1991)

3. Dunkelman, O.: Phorum5: ECRYPT forum, post 'A small observation on HC-128'. Available at *http://www.ecrypt.eu.org/stream/phorum/read.php?1,1143*. Last accessed on July 3, 2011
4. ECRYPT: D.SYM.3 – The eSTREAM Portfolio 2009 Annual Update, ICT-2007-216676. Available at *http://www.ecrypt.eu.org/stream/D.SYM.3-v1.1.pdf*. Last accessed on January 14, 2011
5. ECRYPT: eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932. Available at *http://www.ecrypt.eu.org/stream/*. Last accessed on January 14, 2011
6. Hell, M., Johansson, T., Brynielsson, L.: An overview of distinguishing attacks on stream ciphers. Cryptography and Communications **1**(1), 71–94 (2008)
7. Kircanski, A., Youssef, A.M.: Differential fault analysis of HC-128. In: Africacrypt 2010, *Lecture Notes in Computer Science*, vol. 6055, pp. 360–377. Springer (2010)
8. Liu, Y., Qin, T.: The key and IV setup of the stream ciphers HC-256 and HC-128. In: International Conference on Networks Security, Wireless Communications and Trusted Computing, pp. 430–433 (2009)
9. Maitra, S., Paul, G., Raizada, S., Sen, S., Sengupta, R.: Some observations on HC-128. Designs, Codes and Cryptography pp. 1–15 (2010)
10. Paul, G., Maitra, S., Raizada, S.: A Combinatorial Analysis of HC-128. Cryptology ePrint Archive: Report 2010/387
11. Staffelbach, O., Meier, W.: Cryptographic significance of the carry for ciphers based on integer addition. In: A. Menezes, S.A. Vanstone (eds.) CRYPTO, *Lecture Notes in Computer Science*, vol. 537, pp. 601–614. Springer (1990)
12. Wu, H.: Phorum5: ECRYPT forum, post 'Re: A small observation on HC-128'. Available at *http://www.ecrypt.eu.org/stream/phorum/read.php?1,1143*. Last accessed on July 3, 2011
13. Wu, H.: The Stream Cipher HC-128. In: New Stream Cipher Designs, *Lecture Notes in Computer Science*, vol. 4986, pp. 39–47. Springer-Verlag (2008)