Differential Cryptanalysis of Salsa20/8

Yukiyasu Tsunoo[1], Teruo Saito[2], Hiroyasu Kubo[2],
Tomoyasu Suzaki[2], and Hiroki Nakashima[2]

[1] NEC Corporation
1753 Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa 211-8666, Japan
tsunoo@BL.jp.nec.com
[2] NEC Software Hokuriku Ltd.
1 Anyoji, Hakusan, Ishikawa 920-2141, Japan
{t-saito@qh, h-kubo@ps, t-suzaki@pd, h-nakashima@vt}.jp.nec.com

Abstract. This paper presents a cryptanalysis of the Salsa20 stream cipher proposed in 2005. Salsa20 was submitted to eSTREAM, the ECRYPT Stream Cipher Project. The cipher uses bitwise XOR, addition modulo 2^{32}, and constant-distance rotation operations on an internal state of 16 32-bit words.
It is reported that there is a significant bias in the differential probability for Salsa20's 4^{th} round internal state. It is further shown that using this bias, it is possible to break the 256-bit secret key 8-round reduced Salsa20 model with a lower computational complexity than an exhaustive key search. The cryptanalysis method exploits characteristics of addition, and succeeds in reducing the computational complexity compared to previous methods.
Key words: Salsa20, ECRYPT, eSTREAM, stream cipher, pseudo-random number generator, key rocovery attack

1 Introduction

Recently, there have been efforts around the world to standardize encryption. For example, the selection of AES as a standard cipher has been announced in America [1], and the NESSIE project is hoping for adoption in Europe [11]. The NESSIE project is aiming for the selection of secure encryption primitives, and stream ciphers have been chosen as one member of that category. However, after having undergone a 3-year evaluation phase, the stream ciphers offered by the NESSIE project have been tackled by numerous cryptanalyses and ultimately there is not even a single secure candidate stream cipher remaining. In response, the design and analysis of stream ciphers has been receiving increasing attention.

In February 2004, ECRYPT was established with the objective of encouraging cooperation among European researchers working on information security. The ECRYPT Stream Cipher Project (henceforth "eSTREAM") was established in 2005 by the ECRYPT working group STVL, and new stream ciphers were made public [7]. As a result 34 candidates were submitted to eSTREAM. The first phase of evaluation for these ciphers was completed at eSTREAM in February 2006 and the second phase of evaluation is currently underway.

Salsa20 is one of the stream ciphers submitted to eSTREAM and was proposed by Bernstein [3]. There are two types, handling secret keys of either 128 or 256 bits. With a 64-bit initial vector and a 64-bit counter as input, a 512-bit keystream is generated. However, from a security standpoint Bernstein recommends a 256-bit secret key [4]. Salsa20 is a stream cipher proposed for Profile 1 (SW) and Profile 2 (HW). The cipher uses bitwise XOR, addition modulo 2^{32}, and constant-distance rotation operations on an internal state of 16 32-bit words. It currently remains a second phase candidate for both profiles.

Crowley reported a differential cryptanalysis of the 5-round Salsa20 model in 2006 [6]. According to Crowley's method, it is possible to recover the secret key for the 256-bit secret key, 5-round Salsa20 model using 2^{165} computational complexity and 2^6 keystreams. Fisher et al. also reported a differential cryptanalysis of the 6-round model in 2006 [8]. According to the method of Fisher et al., it is possible to recover the secret key for the 256-bit 6-round Salsa20 model using 2^{177} computational complexity and 2^{16} keystream pairs. Fisher et al. also reported the potential applicability of related-key attacks on 256-bit 7-round Salsa20.

This paper reports that there is a significant bias in the differential probability for Salsa20's 4^{th} round internal state. It is further reported that the Salsa20 reduced-round model was broken by means of this bias. Using the method in this paper, 128-bit secret key 7-round Salsa20 was broken using less computational complexity than required by an exhaustive key search. It is also possible to break 256-bit 8-round Salsa20 using less computational complexity than required by an exhaustive key search. The necessary data are theoretically $2^{11.37}$ keystream pairs. Bernstein proposed the 8-round model as a high-speed version of Salsa20 [5], but there is a problem with the security of the model when faced with the attack described in this paper.

The Salsa20 stream cipher is explained in Section 2. The differential probability bias in the internal state of the 4^{th} round of Salsa20 is reported in Section 3, and the method for recovering the secret key is explained in Section 4. The paper is concluded in Section 5.

2 Salsa20

The Salsa20 stream cipher is explained in this section. For a more detailed description refer to the proposal [3], or cryptanalyses [6, 8].

2.1 Notation and Definition

The unit of processing in Salsa20 is a single 32-bit word, and the mapping from words to bytes is little endian. In this paper bitwise XOR and bitwise AND are denoted by $\oplus$ and $\cdot$ respectively. Addition modulo 2^{32} is denoted by $+$, and a t-bit left-shift rotation on a word X is denoted $X \lll t$. The individual bits in a 32-bit word X are identified as follows.

$$X_{(32)} = X_{\#0} \parallel X_{\#1} \parallel \cdots \parallel X_{\#31}$$

Here, the $\|$ symbol indicates concatenation.

2.2 Round Function

Firstly, the round 1 processing in Salsa20 is now defined. Given a 4×4 word matrix m,

$$m = \begin{pmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{pmatrix},$$

the round function $R(m)$ is defined as follows,

$$R(m) = (Q'^4(m))^T.$$

In this equation

$$Q' = \begin{pmatrix} m_5 & m_6 & m_7 & q_1 \\ m_9 & m_{10} & m_{11} & q_2 \\ m_{13} & m_{14} & m_{15} & q_3 \\ m_1 & m_2 & m_3 & q_0 \end{pmatrix}, \quad q = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = Q \begin{pmatrix} m_0 \\ m_4 \\ m_8 \\ m_{12} \end{pmatrix}.$$

The function Q, of the 4-word input $y = y_0 \| y_1 \| y_2 \| y_3$, and 4-word output $z = z_0 \| z_1 \| z_2 \| z_3$ effects the calculation below.

$$\begin{aligned} z_1 &= y_1 \oplus ((y_0 + y_3) \lll 7) \\ z_2 &= y_2 \oplus ((z_1 + y_0) \lll 9) \\ z_3 &= y_3 \oplus ((z_2 + z_1) \lll 13) \\ z_0 &= y_0 \oplus ((z_3 + z_2) \lll 18) \end{aligned}$$

2.3 Encryption Function

The Salsa20 encryption function is defined as follows,

$$Salsa20_k(v, i) = H \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ i_0 & i_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}.$$

Here, H is defined as a "hash function", and carries out the following processing.

$$H(m) = m + R^r(m)$$

where $k = k_0 \| k_1 \| \cdots \| k_7$ indicates the 256-bit secret key, and $v = v_0 \| v_1$ indicates the 64-bit initial vector. $i = i_0 \| i_1$ represents a 64-bit counter, and $c = c_0 \| c_1 \| c_2 \| c_3$ represent 128-bit constants. However, when the secret key

length is 128-bits, the assignment $(k_4, k_5, k_6, k_7) = (k_0, k_1, k_2, k_3)$ is adopted. The 128-bit constants vary depending on whether the secret key is 128-bit or 256-bit. $c = \tau$ or $c = \sigma$ is chosen for each case, respectively. τ and σ have the values "expand 16-byte k", and "expand 32-byte k", respectively, as ASCII coded values.

For the sake of convenience, in this paper the modified Salsa20 model with r rounds is denoted by Salsa20/r. The models with 8 and 12 rounds that were proposed for high speed with a focus on implementation are thus denoted Salsa20/8 and Salsa20/12, respectively.

2.4 Cryptanalysis of Salsa20 Variants

In 2006, Crowley proposed a differential attack on Salsa20/5. Crowley successfully obtained the secret key for Salsa20/5 by using differential characteristics formed with a high probability in the internal state of round 3. Crowley utilizes a characteristic of the algorithm that relies on the fact that for the inverse of the function Q (denoted Q^{-1}), part of the output data does not depend on the input. Specifically, the function Q^{-1} is expressed using the equations below,

$$\begin{aligned} y_0 &= z_0 \oplus ((z_2 + z_3) \lll 18) \\ y_3 &= z_3 \oplus ((z_1 + z_2) \lll 13) \\ y_2 &= z_2 \oplus ((y_0 + z_1) \lll 9) \\ y_1 &= z_1 \oplus ((y_0 + y_3) \lll 7) \end{aligned}$$

and thus y_0 is not influenced by z_1, and y_3 is not influenced by z_0. By using this characteristic and conjecturing values for the secret key word required to compute Q^{-1} in round 1 and round 2, the candidates for the secret key are narrowed down. According to Crowley's method 256-bit secret key Salsa20/5 was broken with 2^{165} computational complexity and 2^6 keystreams.

Fisher et al. proposed a differential attack on Salsa20/6 in 2006. They described an alternative LinSalsa20 model with addition modulo 2^{32} replaced by XOR, and introduced some ways of using a low-weight differential path. They thus exposed the non-randomness of Salsa20 by showing that LinSalsa20 behaves the same as the actual Salsa20 model under the assumption that addition does not yield a carry. According to Fisher et al.'s method, 256-bit secret key Salsa20/6 was broken in 2^{177} computational complexity with 2^{16} keystreams. When they allowed related-key attacks, Fisher et al. also reported that 256-bit secret key Salsa20/7 could be broken, and mentioned that non-randomness exists in the round 7 output.

3 Differential Characteristics

3.1 Differential Probabilities of Addition

Firstly, with $x, y, z \in GF(2)^n$, consider the function f for addition modulo 2^n,

$$f(x, y) = x + y = z \pmod{2^n}.$$

Next, with $x^*, y^*, z^* \in GF(2)^n$, call the difference values for each of x, y, z, $\Delta x = x \oplus x^*, \Delta y = y \oplus y^*, \Delta z = z \oplus z^*$, respectively. According to the differential probabilities of addition, $DP_f(\Delta x, \Delta y, \Delta z)$ is defined as follows.

Definition 1.

$$DP_f(\Delta x, \Delta y, \Delta z) = \underset{x,y}{P}[f(x, y) \oplus f(x \oplus \Delta x, y \oplus \Delta y) = \Delta z]$$

Next, the difference values $\Delta x_i, \Delta y_i, \Delta z_i$ of the i^{th} bits x_i, y_i, z_i are defined in the same way. The carry of the i^{th} bit is denoted by c_i. When x, y yield differences, a difference is also yielded by each bit. In such cases the difference value of the i^{th} bit's carry, c_i, is written Δc_i. Since the carry of the least significant bit is always 0, $c_{n-1} = \Delta c_{n-1} = 0$.

Theorem 1, Theorem 2 and Corollary 1 are expressed below [9, 10].

Theorem 1. *For* $0 \leq i \leq n-1$,

$$\Delta x_i \oplus \Delta y_i \oplus \Delta c_i = \Delta z_i.$$

Theorem 2. *For* $1 \leq i \leq n-2$, *and given values for* $(\Delta x_i, \Delta y_i, \Delta c_i)$, Δc_{i-1} *is determined as follows.*

$(\Delta x_i, \Delta y_i, \Delta c_i)$	Δc_{i-1}	Probability
(0, 0, 0)	0	1
(0, 0, 1)	0 / 1	0.5 / 0.5
(0, 1, 0)	0 / 1	0.5 / 0.5
(0, 1, 1)	0 / 1	0.5 / 0.5
(1, 0, 0)	0 / 1	0.5 / 0.5
(1, 0, 1)	0 / 1	0.5 / 0.5
(1, 1, 0)	0 / 1	0.5 / 0.5
(1, 1, 1)	1	1

Corollary 1. *Given values for* $(\Delta x_{n-1}, \Delta y_{n-1})$, Δc_{n-2} *is determined as follows.*

$(\Delta x_{n-1}, \Delta y_{n-1})$	Δc_{n-2}	Probability
(0, 0)	0	1
(0, 1)	0 / 1	0.5 / 0.5
(1, 0)	0 / 1	0.5 / 0.5
(1, 1)	0 / 1	0.5 / 0.5

Theorem 3 and Corollary 2 below are formulated similarly.

Theorem 3. *For* $1 \leq i \leq n-2$, *and given values for* $(\Delta x_i, \Delta y_i, \Delta c_i)$, Δz_{i-1} *is determined as follows.*

$(\Delta x_i, \Delta y_i, \Delta c_i)$	Δz_{i-1}	Probability
(0, 0, 0)	0	1
(0, 0, 1)	1	1
(0, 1, 0)	1	1
(0, 1, 1)	0	1
(1, 0, 0)	1	1
(1, 0, 1)	0	1
(1, 1, 0)	0	1
(1, 1, 1)	1	1

Corollary 2. *Given* $(\Delta x_{n-1}, \Delta y_{n-1})$, Δz_{n-2} *is determined as follows.*

$(\Delta x_{n-1}, \Delta y_{n-1})$	Δz_{n-2}	Probability
(0, 0)	0	1
(0, 1)	1	1
(1, 0)	1	1
(1, 1)	0	1

3.2 Round 4 Differential Probabilities

In this section, the differential probabilities for each bit in the round r internal state data (i.e., the data before applying the addition immediately prior outputting the keystream) for Salsa20 ($0 \leq r \leq 20$) are obtained rigorously, based on the differential probabilities occurring during addition that were revealed in Section 3.1. $r = 0$ denotes the initial value for internal state.

Firstly, a difference is introduced for an arbitrary bit of the initial vector v or counter i. In this case the differential probability for the bit with the difference is 1. The differential probabilities for the other bits are 0. The differential probabilities for each bit of internal state in round r of Salsa20 were calculated for such input using the theorems in Section 3.1. If the output bit is random, then the corresponding differential probability is $1/2$.

In the result, a bias was found in the differential probabilities for the round 4 internal state. Writing the round r internal state as

$$m^r = \begin{pmatrix} m_0^r & m_1^r & m_2^r & m_3^r \\ m_4^r & m_5^r & m_6^r & m_7^r \\ m_8^r & m_9^r & m_{10}^r & m_{11}^r \\ m_{12}^r & m_{13}^r & m_{14}^r & m_{15}^r \end{pmatrix},$$

the relationships among the input-output bits with the most biased differential probabilities are $(m^0_{7,\#0}, m^4_{1,\#17})$ and $(m^0_{8,\#0}, m^4_{6,\#17})$. The probabilities that each 1 bit difference in the round 4 internal state are 1, $P[\Delta m^4_{1,\#17} = 1 \mid \Delta m^0_{7,\#0} = 1]$, $P[\Delta m^4_{6,\#17} = 1 \mid \Delta m^0_{8,\#0} = 1]$ are

$$\begin{aligned} P[\Delta m^4_{1,\#17} = 1 \mid \Delta m^0_{7,\#0} = 1] &= P[\Delta m^4_{6,\#17} = 1 \mid \Delta m^0_{8,\#0} = 1] \\ &\approx \frac{1}{2}(1 - 2^{-5.24}). \end{aligned}$$

In this expression the N difference values $\Delta m^4_{1,\#17}$ are each independent and form a binary sequence obeying a distribution D_{BIAS} with the bias mentioned above. The amount of data, N, needed to break Salsa20 is shown by Theorem 4 below [2, 12] where D_{UNI} denotes a uniform distribution.

Theorem 4. *When the input to the optimal distinguisher is a binary random variable z_i $(0 \leq i < N)$ obeying the D_{BIAS} distribution, in order to achieve an advantage greater than 0.5, it is necessary to have at least as many samples of the optimal distinguisher as given by the following equation.*

$$N = 0.4624 \times M^2 \qquad where$$
$$P_{D_{BIAS}}[z_i = 1] - P_{D_{UNI}}[z_i = 1] = \frac{1}{M}$$

It is thus possible to estimate using Theorem 4, that the amount of data N, needed to judge that the difference value $\Delta m^4_{1,\#17}$ is biased with respect to a uniform distribution is theoretically $2^{11.37}$ keystream pairs. In short, by assigning differences to an initial vector such that $\Delta m^0_{7,\#0} = 1$ and accumulating $2^{11.37}$ keystream pairs, a significant bias was detected in the difference value $\Delta m^4_{1,\#17}$. A cryptanalysis method using this bias is discussed in the next section.

4 Key Recovery Method

4.1 Key Conjectures in Word Units

This section describes cryptanalysis methods for Salsa20/5 and Salsa20/6 using the differential characteristics of round 4 revealed in Section 3.2. When using the bias in $\Delta m^4_{1,\#17}$ that was revealed in Section 3.2 to calculate data traced back 1 round from the keystream, it is necessary to conjecture values for 32 bits (k_3). Calculating data traced back 2 rounds from the keystream requires that 224 bits must be conjectured $(k_0, k_1, k_2, k_4, k_5, k_6, k_7)$.

Thus for 128-bit secret keys and 256-bit secret keys, it is possible to recover a part of the key for Salsa20/5 by using conjectures for 32 key bits and $2^{11.37}$ keystream pairs. It is also possible to recover a part of the 256-bit secret key for Salsa20/6 by using conjectures for 224 key bits and $2^{11.37}$ keystream pairs. For the bits which cannot be obtained through conjecture, an exhaustive key search is appropriate. The computational complexity of an exhaustive key search is not exceeded.

4.2 On the Optimization of the Key Recovery Method

In this section the efficiency-optimization of the key recovery method is investigated with reference to the characteristics of addition and the construction of Salsa20.

First consider addition modulo 2^{32}. When computing the function Q^{-1}, it is necessary to conjecture a full 32 bits of input in order to obtain a single bit

according to the method in Section 4.1. However, one of the characteristics of addition is that when calculating output bit i $(0 < i)$, the information in the higher bits (in positions j, such that $(0 \leq j < i)$ is not required. Thus it is unnecessary to make a conjecture for the whole 32 bits of input, except in the case that the bit calculated is the most significant bit.

Furthermore, when addition modulo 2^{32} is expressed using single bit variables, each bit of the output can be expressed as a polynomial of order at most 31. When an individual bit is expressed as a polynomial, the probability that the higher order terms are zero increases with the order. In particular, it is possible to ignore the d^{th} order term except in the case that d parameters have value 1. Indeed, the d^{th} order term may be ignored with probability $1 - 2^{-d}$. There is thus the possibility of a low order polynomial approximation.

Having considered these two factors, we anticipated the possibility of reducing the computational complexity by making conjectures for bit units rather than for word units. The results of analyzing Salsa20 using these ideas are reported in the next section.

4.3 Key Conjectures in Bit Units

This section describes a method for reducing the number of conjectures needed to compute the function Q^{-1} according to Section 4.2. Firstly, three types of approximation to addition modulo 2^{32} are made below.

2^{nd} order approximation
When calculating $z = x + y$, approximate using
$z_i = z_i^{(2)} = x_i \oplus y_i \oplus x_{i+1} \cdot y_{i+1}$.
When $i = 31$ approximate using $z_i = z_i^{(2)} = z_i^{(1)} = x_i \oplus y_i$.

3^{rd} order approximation
When calculating $z = x + y$, approximate using
$z_i = z_i^{(3)} = z_i^{(2)} \oplus x_{i+1} \cdot x_{i+2} \cdot y_{i+2} \oplus y_{i+1} \cdot x_{i+2} \cdot y_{i+2}$.
When $i = 30, 31$ approximate using $z_i = z_i^{(3)} = z_i^{(2)}$.

4^{th} order approximation
When calculating $z = x + y$, approximate using
$z_i = z_i^{(4)} = z_i^{(3)} \oplus x_{i+1} \cdot x_{i+2} \cdot x_{i+3} \cdot y_{i+3} \oplus x_{i+1} \cdot y_{i+2} \cdot x_{i+3} \cdot y_{i+3} \oplus y_{i+1} \cdot x_{i+2} \cdot x_{i+3} \cdot y_{i+3} \oplus y_{i+1} \cdot y_{i+2} \cdot x_{i+3} \cdot y_{i+3}$.
When $i = 29, 30, 31$ approximate using $z_i = z_i^{(4)} = z_i^{(3)}$.

The number of key bits needed to trace back to $m^4_{1,\#17}$ was obtained for Salsa20/r $(6 \leq r \leq 8)$ using these approximations. The addition immediately prior to the output of the keystream was also considered. The results for 128-bit and 256-bit secret keys are shown in Tables Table 1 and Table 2.

For a 128-bit secret key, it was found that the number of bits needed to trace back Salsa20/8 was 128, so the result was omitted from Table 1. As can be seen from Table 1 and Table 2, if the secret key candidates can be narrowed

Table 1. Number of bits needed to trace back with a 128-bit secret key

r	2^{nd} order approximation	3^{rd} order approximation	4^{th} order approximation
6	31 bits	50 bits	61 bits
7	93 bits	115 bits	119 bits

Table 2. Number of bits needed to trace back with a 256-bit secret key

r	2^{nd} order approximation	3^{rd} order approximation	4^{th} order approximation
6	35 bits	62 bits	83 bits
7	117 bits	171 bits	199 bits
8	228 bits	247 bits	250 bits

down correctly using these approximations, then a part of the secret key can be obtained more efficiently than with an exhaustive key search.

In the next section it is confirmed that the conjectured secret key candidates can be narrowed down to correct values, and the legitimacy of the discussion in Section 4.2 is examined.

4.4 Cryptanalysis Experiments

This section discusses an experimental confirmation of whether cryptanalysis based on the approximations in Section 4.3 is applicable. However, it is computationally problematic to conjecture all the key bits needed to trace back, so it was confirmed whether the solution could be correctly narrowed down from set of randomly selected candidate keys. Also, only 256-bit secret keys were used for the experiments in this section. The experimental procedure was as follows.

1. Set a 256-bit secret key KEY.
2. Set the initial vector V to 0.
3. Apply the difference $\Delta m^0_{7,\#0} = 1$ to the vector V in step 2, yielding the initial V^* vector.
4. Produce an $N \times 64$ byte keystream KS with (KEY, V, I) as input, for $0 \leq I < N$.
5. Produce an $N \times 64$ byte keystream KS^* with (KEY, V^*, I) as input, for $0 \leq I < N$.
6. Set $AKEY$ such that the number of conjectured bits shown in Table 2 are randomly assigned, and the bits whose values are not subject to conjecture are set to 0.
7. Trace back to $m^4_{1,\#17}$ from KS using $(AKEY, V, I)$. Perform the calculation for each of the N cases.

8. Trace back to $m^{4*}_{1,\#17}$ from KS^* using $(AKEY, V^*, I)$. Perform the calculation for each of the N cases.
9. Perform each of the N difference value calculations $\Delta m^4_{1,\#17} = m^4_{1,\#17} \oplus m^{4*}_{1,\#17}$ and count the number of times the result is 0.
10. Attempt the $AKEY$ conjectured in step 6. Repeat steps 7 to 9 with a different $AKEY$ 2^{14} times. One repetition is performed with the same value as KEY except that all of the bits not subject to conjecture are also set to 0.
11. Confirm whether KEY is contained among the $AKEY$ candidates for which the counter obtained in step 9 is maximal. If KEY is contained among the $AKEY$ candidates then cryptanalysis is considered successful, otherwise cryptanalysis is considered a failure.
12. Change the KEY set in step 1 a total of 100 times. Repeat steps 2 to 11 for each and calculate the success rate.

Experimental results are shown in Table 3 $\sim$ Table 5.

Table 3. Cryptanalysis success rate for Salsa20/6

N	2^{nd} order approximation	3^{rd} order approximation	4^{th} order approximation
2^{10}	0 %	15 %	47 %
2^{11}	0 %	23 %	56 %
2^{12}	0 %	34 %	62 %
2^{13}	0 %	45 %	65 %
2^{14}	0 %	53 %	67 %

Table 4. Cryptanalysis success rate for Salsa20/7

N	2^{nd} order approximation	3^{rd} order approximation	4^{th} order approximation
2^{10}	0 %	27 %	59 %
2^{11}	0 %	45 %	64 %
2^{12}	0 %	57 %	67 %
2^{13}	0 %	61 %	69 %
2^{14}	0 %	64 %	73 %

It can be seen from Table 3 $\sim$ Table 5 that increasing the order of the approximation improves the success rate of cryptanalysis. Thus, as discussed in Section 4.2, it is possible to perform analysis using approximations, but the

Table 5. Cryptanalysis success rate for Salsa20/8

N	2^{nd} order approximation	3^{rd} order approximation	4^{th} order approximation
2^{10}	0 %	57 %	60 %
2^{11}	0 %	61 %	63 %
2^{12}	0 %	63 %	66 %
2^{13}	0 %	65 %	67 %
2^{14}	0 %	72 %	73 %

cryptanalysis accuracy varies according to the order of the approximation. Using a relatively high order approximation yields a high cryptanalysis success rate, but the number of key bits needed to trace back increases accordingly, so care should be taken regarding the computational complexity.

When all the additions are approximated with a 3^{rd} order polynomial, the computational complexity required to cryptanalyze Salsa20/8 with a 256-bit secret key exceeds that of an exhaustive key search. However, it is also possible to reduce the computational complexity by modeling some additions with a 2^{nd} order polynomial and modeling the rest using 3^{rd} order polynomials. As an example of this kind of method, we selected the additions immediately prior to the output of the keystream for 2^{nd} order approximations, and the additions of the function Q^{-1} for 3^{rd} order approximations. As a result, for Salsa20/8 it was confirmed that the correct key was narrowed down from random candidates with a 44% success rate by conjecturing 245 key bits and using 2^{10} keystream pairs. [1]

5 Conclusion

This paper presented a cryptanalysis of the Salsa20 stream cipher proposed in 2005. We discovered a significant differential probability bias in the Salsa20 round 4 internal state, yielded by assigning single bit differences to the initial vector which may be freely chosen by an attacker.

Further more, we have used this bias to attack Salsa20/r ($5 \leq r \leq 8$). Using our method, Salsa20/7 with a 128-bit secret key and Salsa20/8 with a 256-bit secret key were broken using less computational complexity than required for an exhaustive key search. The amount of data needed for both cryptanalyses is theoretically $2^{11.37}$ keystream pairs. Our method of attack uses a new technique of approximating polynomials of addition, and succeeds in the reducing the computational complexity compared to previous methods.

[1] In this example, the computational complexity becomes approximately 2^{255} ($= 2^{245} \times 2^{10} \times 2 \times \frac{4}{8}$) times of Salsa20/8 encryption and it is lower than 2^{256} of exhaustive key search.

Finally, Bernstein proposed Salsa20/8 and Salsa20/12 as models paying consideration to implementation, but faced with our attack the security of Salsa20/8 is insufficient, and even considering implementation, a model of at least 12 rounds should be used. However, the attack in this paper does not directly threaten the security of the full-round Salsa20/20 specification, although caution is still required regarding the issue of whether there is a sufficient margin of safety.

Acknowledgement

The authors would like to thank the anonymous referees of SASC 2007 whose helpful remarks that improved the paper. The authors also would like to thank Etsuko Tsujihara for her useful comments.

References

1. AES, the Advanced Encryption Standard, NIST, FIPS-197.
   Available at http://csrc.nist.gov/CryptoToolkit/aes/
2. T. Baigneres, P. Junod, and S. Vaudenay: "How Far Can We Go Beyond Linear Cryptanalysis?," *Advances in Cryptology - Asiacrypt 2004*, LNCS 3329, pp.432-450, 2004.
3. D.J. Bernstein: "The Salsa20 Stream Cipher," *SKEW 2005 - Symmetric Key Encryption Workshop, 2005*, Workshop Record.
   Available at http://www.ecrypt.eu.org/stream/salsa20p2.html
4. D.J. Bernstein: "Notes on the Salsa20 Key Size," *eSTREAM, the ECRYPT Stream Cipher Project*, Report 2005/066, 2005.
   Available at http://www.ecrypt.eu.org/stream/salsa20.html
5. D.J. Bernstein: "Salsa20/8 and Salsa20/12," *eSTREAM, the ECRYPT Stream Cipher Project*, Report 2006/007, 2006.
   Available at http://www.ecrypt.eu.org/stream/salsa20.html
6. P. Crowley: "Truncated Differential Cryptanalysis of Five Rounds of Salsa20," *SASC 2006 - Stream Ciphers Revisited*, Workshop Record, pp.198-202, 2006.
   Available at http://www.ecrypt.eu.org/stvl/sasc2006/
7. eSTREAM, the ECRYPT Stream Cipher Project.
   Available at http://www.ecrypt.eu.org/stream/
8. S. Fischer, W. Meier, C. Berbain, J.-F. Biasse, and M. Robshaw: "Non-Randomness in eSTREAM Candidates Salsa20 and TSC-4," *Progress in Cryptology - Indocrypt 2006*, LNCS 4329, pp.2-16, Springer-Verlag, 2006.
9. H. Lipmaa and S. Moriai: "Efficient Algorithms for Computing Differential Properties of Addition," *Fast Software Encryption, FSE 2001*, LNCS 2355, pp.336-350, Springer-Verlag, 2001.
10. S. Moriai: "Cryptanalysis of Twofish (I)," *In Proceedings of the Symposium on Cryptgraphy and Information Security, SCIS 2000*, No. A01, 2000. (In Japanese)
11. NESSIE, the New European Schemes for Signatures, Integrity, and Encryption.
    Available at https://www.cosic.esat.kuleuven.ac.be/nessie/
12. S. Paul, B. Preneel, and G. Sekar: "Distinguishing Attacks on the Stream Cipher Py," *Fast Software Encryption, FSE 2006*, LNCS 4047, pp.405-421, Springer-Verlag, 2006.