Cryptanalysis of Reduced Word Variants of Salsa

Sylvain Pelissier

EPFL LACAL, Lausanne, Switzerland

Abstract. The Salsa stream cipher was proposed by Bernstein as a candidate of the eSTREAM project. As far as we know, the best cryptanalysis of this cipher is the one proposed by Aumasson et al. at FSE 08 which uses a new concept called probabilistic neutral bits. This method allows an adversary to recover a key when Salsa uses up to 8 rounds instead of 20. Bernstein has asked cryptanalysts to apply their attack against Salsa on versions with smaller word size (instead of the original 32-bit version) in order to check the validity of their attack. That is why we decided to implement the version of Aumasson et al. attack working with smaller words. In addition to check validity of this attack, applying the attack on smaller word sizes allowed us to give evidences that the probabilistic neutral bits can be applied successfully in practice. We applied this method to Salsa when it uses words reduced to 16 bits and 8 bits. For the 16-bit version of this attack we obtained complexity of 2^{117} when Salsa uses 6 rounds and complexity 2^{71} for 5 rounds. For the 8-bit version we obtained a complexity of 2^{38} for 5 rounds of Salsa. We implemented a practical key recovery for Salsa on 8-bit words in order to show the efficiency of this new method in practice.

Keywords: Stream cipher, Salsa20, Probabilistic neutral bits.

1 Introduction

The aim of the eSTREAM project [6] was to identify new stream ciphers which could be used in a wild range of applications. This project was organised by the European Network of Excellence in Cryptology (ECRYPT) from 2004 to 2008. The Salsa20 stream cipher [3] was proposed by Bernstein as a candidate of this project and was accepted for the final portfolio. The original version of Salsa20 is composed of 20 rounds, works with 32-bit words and uses modular additions, bitwise XOR and rotation operations. Bernstein later proposed two variants Salsa20/8 and Salsa20/12 [4], which work exactly as Salsa20 but are composed respectively of 8 and 12 rounds. These ciphers were not official candidates of the eSTREAM project.

Several cryptanalytic results of reduced versions of Salsa cipher were obtained, none of which applies to the full version of Salsa. In 2005, Crowley [5] proposed an attack on Salsa20/5 which uses a 3-round differential. This attack has a time complexity of 2^{165} trials. In 2006, Fischer et al. [7] used a differential

after 4 rounds to construct an attack against Salsa20/6 with a time complexity of 2^{177} trials.

In this paper, we study the best attack on Salsa20 with 256-bit keys due to Aumasson et al. [1] which can break only up to 8 rounds. This attack uses a new concept called probabilistic neutral bits to recover the key faster than exhaustive search in a known keystream differential attack scenario. In other words, the adversary needs to be able to collect the keystream bits for some pairs with a desired input difference in the controllable public variables. For Salsa the public variables are the IV (initial value) and the counter. The authors constructed a key recovery attack against Salsa20/8 and Salsa20/7 with respective complexity 2^{251} and 2^{151}. They also proposed an attack on Chacha, a slightly modified version of Salsa, with complexity 2^{139} over 6 rounds.

In the very early publication of Salsa, Bernstein recommended the cryptanalysts [2] to evaluate their attack also on Salsa reduced to words of w bits with $w < 32$ and to state the success probability of the attack for every different w. However, to the best of our knowledge, the previous attacks against Salsa were never tested with smaller words length. In addition, since reducing the length of the words means reducing the attack complexity, there is also an opportunity to bypass the outrageous complexity of 32-bit version. Hence we can see if the concept of probabilistic neutral bits works in practice.

That is our motivation to explore how this new method can be applied for word-reduced variants of Salsa. In particular, we applied this method when Salsa is defined on 16-bit and 8-bit words. We can not break the 16-bit version (with a 128-bit key) when it uses 7 rounds. Although we can attack 6-round and 5-round variants with complexities 2^{117} and 2^{71} respectively, these numbers are still practically hard to reach. For the 8-bit version (with a 64-bit key) we can not break the 6-round variant, however, the 5-round version can be broken in time 2^{38}. We focus on the 5-round 8-bit version of Salsa20, implement the attack and extract information which shows that the probabilistic neutral bits method indeed works in practice. However, some care needs to be taken if we want to exactly predict the attack complexity.

For the practical attack, an unforeseen phenomenon appears; some false positive key candidates are able to pass the optimal distinguisher. We investigate how this affects the theoretical numbers of the attacks by Aumasson et al. [1] on reduced round Salsa20 (defined over 32-bit words). Moreover, we provide a way for reducing the increase of the complexity given by this phenomenon.

2 Specification of Salsa

We use the notation $\text{Salsa}_{(w,R)}$ for the stream cipher which operates on w-bit words and is composed of R rounds. The $8w$-bit key of this cipher is denoted by the 8-word vector $k = (k_0, k_1, ..., k_7)$. The $\text{Salsa}_{(w,R)}$ cipher takes as parameters the key and a two-word nonce $v = (v_0, v_1)$. It outputs a sequence of 16-word keystream blocks where the t-th block of this sequence is the output of the *Salsa function* which takes as parameter a two-word counter (t_0, t_1) corresponding to

the integer t in addition to the key and the nonce. This function acts on the 4×4 matrix of words given by

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_5 & k_7 & c_3 \end{pmatrix} ,$$

where the c_i's are known constants. For a matrix X, the *Salsa function* returns the keystream block Z defined by $Z = X + \text{Round}_R(X)$, where "+" denotes the matrix modular addition and the Round function is based on a function called the *quarterround* function. The *quarterround* function takes as parameter a vector (x_0, x_1, x_2, x_3) and return a vector (y_0, y_1, y_2, y_3) which is defined by

$$\begin{aligned} y_1 &= x_1 \oplus [(x_3 + x_0) \lll s_0] \ , \\ y_2 &= x_2 \oplus [(x_0 + y_1) \lll s_1] \ , \\ y_3 &= x_3 \oplus [(y_1 + y_2) \lll s_2] \ , \\ y_0 &= x_0 \oplus [(y_2 + y_3) \lll s_3] \ , \end{aligned}$$

where $\lll$ denotes the left rotation on w-bit words and the values s_i are the shift values. Then Round_R function applies sequentially this *quarterround* function for rounds $i = 1, ..., R$ on columns (x_0, x_4, x_8, x_{12}), (x_5, x_9, x_{13}, x_1), $(x_{10}, x_{14}, x_2, x_6)$ and $(x_{15}, x_3, x_7, x_{11})$ if i is odd. For even rounds, this function is applied to the rows (x_0, x_1, x_2, x_3), (x_5, x_6, x_7, x_4), $(x_{10}, x_{11}, x_8, x_9)$ and $(x_{15}, x_{12}, x_{13}, x_{14})$.

For the following attack against Salsa, we use the fact that the function *quarterround* is invertible.

Note that the Salsa function is completely specified once we know the constant and rotation values. The version of Salsa submitted at the eSTREAM project called Salsa20, corresponds to $\text{Salsa}_{(32,20)}$ with constants $(c_0, c_1, c_2, c_3) = (\texttt{0x61707865}, \texttt{0x3320646E}, \texttt{0x79622D32}, \texttt{0x6B206574})$, and rotation values $(s_0, s_1, s_2, s_3) = (7, 9, 13, 18)$.

For $\text{Salsa}_{(16,R)}$ we keep the same *Salsa function* but for the *quarterround* we set $(c_0, c_1, c_2, c_3) = (\texttt{0x6170}, \texttt{0x7865}, \texttt{0x3320}, \texttt{0x646E})$ and $(s_0, s_1, s_2, s_3) = (4, 5, 7, 9)$. We also define $\text{Salsa}_{(8,R)}$ by setting $(c_0, c_1, c_2, c_3) = (\texttt{0x61}, \texttt{0x70}, \texttt{0x78}, \texttt{0x65})$ and $(s_0, s_1, s_2, s_3) = (2, 3, 4, 5)$.

The constants have been simply chosen by keeping the begining of those of the original Salsa20. The rotation values have been selected to keep almost the same ratio between the shift values and the word size. More precisely we fixed the new value of $s_i(w)$ for w-bit version of Salsa from the value of $s_i(32)$ of the 32-bit version of Salsa by the formula $s_i(w) = \left\lceil \frac{w}{32} \cdot s_i(32) \right\rceil$.

3 Differential analysis

3.1 Attack model

We consider an adversary who wants to recover a key k of the stream cipher $\text{Salsa}_{(w,R)}$. As shown in figure 1 we allow an adversary to interact with an oracle

as follows. The adversary can submit as many nonces v and counters t of his

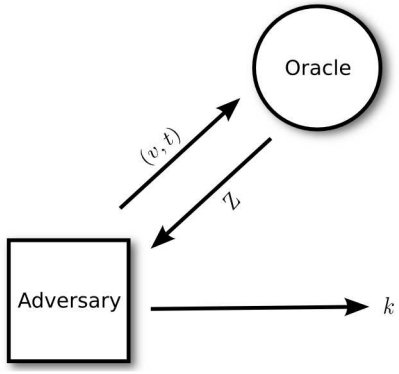

Fig. 1: Attack model for the Salsa analysis

choice as he wants. Then the oracle answers to each request the corresponding keystream block Z created from a fixed random key k, unknown to the adversary.

When the adversary has collected enough keystream blocks then he tries to guess the secret key k. The data complexity is given in this paper in number of pairs collected by the adversary. The time complexity will be explained later.

3.2 Truncated differentials

The concept of truncated differentials was introduced by Knudsen in 1995 [8]. A truncated differential on a single bit consists in introducing some difference in the input of a cipher and looking at the output difference restricted to one output bit.

In the case of Salsa analysis, if x_i and x'_i are the i-th words of the initial matrices X and X' then we define the difference between x_i and x'_i to be $\Delta_i^0 = x_i \oplus x'_i$. We denote the j-th bit of x_i by $[x_i]_j$. The attack presented in [1] uses r-round truncated differential for the matrix input X. If X^r and X'^r, respectively, are the transformation after r rounds of X and X' then we define x_p^r and $x_p'^r$ to be the p-th words of the matrices X^r and X'^r and we define the output difference Δ_p^r to be $\Delta_p^r = x_p^r \oplus x_p'^r$. The attack sets a single bit input difference *i.e.,* $[\Delta_i^0]_j = 1$, where $i \in \{6, 7, 8, 9\}$ and $j \in \{0, ..., w-1\}$, and then considers the single bit output difference $[\Delta_p^r]_q$, where $p \in \{0, ..., 15\}$ and $q \in \{0, ..., w-1\}$ after r rounds. We denote such a truncated differential by $([\Delta_p^r]_q | [\Delta_i^0]_j)$. In this context, the bias ϵ_d of a key is defined by

$$\Pr\left\{[\Delta_p^r]_q = 1 | [\Delta_i^0]_j\right\} = \frac{1}{2}(1 + \epsilon_d) \ ,$$

where we consider v and t as random variables. In addition, the value ϵ_d^* is defined to be the median value of ϵ_d when the key is considered random.

Since the only values controlled by a user are v and t, we allow an attacker to use only truncated differentials with $i \in \{6, 7, 8, 9\}$ whereas we allow $j \in \{0, ..., w-1\}$.

For finding significant differentials we choose some random keys and for each of these keys we try all the possible truncated differentials $([\Delta_p^r]_q|[\Delta_i^0]_j)$. Since $i \in \{6, 7, 8, 9\}$, $p \in \{0, ..., 15\}$ and $j, q \in \{0, ..., w-1\}$ it gives $4 \cdot 16 \cdot w^2$ trials for each key. Then we have to adjust the number of samples according to our desired confidence level. The complexity for estimating the bias of the truncated differentials is negligible compared with the complexity of the whole attack.

3.3 Backward computations

As explained before, if k, v and t are known, it is possible to invert the operation $Z = X + \text{Round}_R(X)$ since $\text{Round}_R(X)$ is invertible and X is a matrix depending on the three previous parameters. This means we can recover X from a block Z. We denote the invert operation of Round_R by Round_R^{-1}. We can also access to the value of the r-th round with $r < R$ since we have $X^r = \text{Round}_{R-r}^{-1}(Z - X)$. We can define the function f which outputs the q-th LSB of the word p of the matrix $\text{Round}_{R-r}^{-1}(Z - X) \oplus \text{Round}_{R-r}^{-1}(Z' - X')$. That is,

$$f(k, v, t, Z, Z') = [\Delta_p^r]_q \ .$$

If Z and Z' are outputs when we use a differential $([\Delta_p^r]_q|[\Delta_i^0]_j)$ of bias ϵ_d, we have $\Pr\{f(\hat{k}, v, t, Z, Z') = 1\} = \frac{1}{2}(1 + \epsilon_d)$ if $\hat{k} = k$ (the correct encryption key) and since f must be unbiased then $\Pr\{f(\hat{k}, v, t, Z, Z') = 1\} = \frac{1}{2}$ for almost all $\hat{k} \neq k$. To sum up, we can verify the correctness of a key candidate. So far there is nothing which allows us to recover the key faster as f may depend on all the $8 \cdot w$ key bits in general. However, the following attack consists of searching over a space of subkey of $m = 8 \cdot w - n$ bits. If $\overline{k}$ is the subkey of m bits of the key k we try to construct g, an approximation of f and we define the bias ϵ_a to be

$$\Pr\{f(k, v, t, Z, Z') = g(\overline{k}, v, t, Z, Z')\} = \frac{1}{2}(1 + \epsilon_a) \ .$$

We also define the bias of g to be

$$\Pr\{g(\overline{k}, v, t, Z, Z') = 1\} = \frac{1}{2}(1 + \epsilon) \ .$$

In [1], the authors claim that under reasonable independence assumptions we have $\epsilon = \epsilon_a \cdot \epsilon_d$. Again ϵ^* denotes the median bias over all subkeys.

3.4 Probabilistic neutral bits

Previously we stated that we want to compute $g(\overline{k}, v, t, Z, Z')$ but first we need to introduce the notion of neutral bits which is the cornerstone concept of the attack. Informally a key bit k_l is a neutral bit if complementing its value in

the key k does not change the output of $f(k, v, t, Z, Z')$. More generally some key bits may not have big influence on the output of f. These bits are called non-significant key bits. The neutrality measure defined in the following can be used to identify non-significant key bits from significant ones. The function g can then be constructed by setting all the non-significant key bits to some arbitrary value, e.g. zero.

Definition 1. *The neutrality measure γ_l of a key bit k_l is defined by*

$$Pr\{f(k, v, t, Z, Z') = f(\tilde{k}, v, t, Z, Z')\} = \frac{1}{2}(1 + \gamma_l) \ ,$$

where $\tilde{k}$ is k except that the l-th bit is complemented.

A key bit which has a neutrality measure equal to 1 is called a neutral bit; that is, it has no effect on the output of the function. More generally, in [1] for a given threshold value γ, $0 < \gamma < 1$, the key bits with $\gamma_l > \gamma$ are referred to as probabilistic neutral bits (PNBs); whereas the other key bits are called significant key bits. The PNBs are also called non-significant key bits. Intuitively, the bigger γ is the less effect the PNBs have on the output of the function. For the attack against Salsa, a threshold γ is chosen and then the key bits are divided into the set of significant and non-significant key bits according to this threshold value and their neutrality measure. Then g is defined to be the same as f except that the non-significant key bits are set to zero in k. The algorithm for estimating the neutrality measure of a key bit is given in Appendix A.

3.5 Attack and Complexity

Now we can explain the complete attack against $\text{Salsa}_{(w,R)}$ and estimate its complexity. The attack is divided into two phases: the precomputation and the effective attack. The precomputation is not key related hence it is sufficient to run this phase once for each differential we want to analyse. The second phase is the effective attack which allows to recover the key.

Precomputation

1. Find a high probability r-round truncated differential as explained in section 3.2.
2. Choose a threshold γ.
3. Construct the function f as defined in section 3.3.
4. Estimate the neutrality measure γ_l for each key bit with Algorithm 1.1.
5. Put all the key bits which have $\gamma_l < \gamma$ in the set of significant bits.
6. Construct the function g from f by assigning a fixed value to all the non-significant bits.
7. Estimate the median ϵ^* by measuring the bias of g using random keys.
8. Estimate the number N of keystream block pairs needed for the effective attack.

At step one we want to find a r-round truncated differential with the highest bias ϵ_d to have then the lowest complexity in the effective attack. For estimating the number N of keystream block pairs, we need to consider the following problem of hypothesis testing. We are considering 2^m sequences of random variable where $2^m - 1$ of them are verifying the hypothesis H_0, that is the candidate is not the correct subkey. One of them is verifying the hypothesis H_1 that it is the correct subkey. In our case we look at a realisation a of random variable A and then we have to find a decision rule $\mathcal{D}(a) = s$ which gives the hypothesis H_s to accept. It is well-known that there are two kinds of error in hypothesis testing. The first one is when $\mathcal{D}(a) = 0$ but $A \in H_1$ which is called a Non-detection event. We will denote the probability of this event by p_{nd}. The second kind of possible error is when $\mathcal{D}(a) = 1$ but $A \in H_0$ which is called a False alarm event. This event happens with probability p_{fa}. We define the value α to be such that $p_{fa} = 2^{-\alpha}$. The Neyman-Pearson lemma gives us a result to estimate the number N of keystream blocks we need in order to have a high probability to succeed during the effective attack. Indeed, it can be shown that for

$$N \approx \left(\frac{\sqrt{\alpha \log 4} + 3\sqrt{1 - \epsilon^2}}{\epsilon} \right)^2$$

we have $p_{nd} = 1.3 \times 10^{-3}$ and $p_{fa} = 2^{-\alpha}$.

Effective attack

1. For an unknown key k, collect N pairs of keystream blocks with the corresponding input difference of the precomputation.
2. **First phase:** For each choice of the m significant bits:
   - Estimate the bias of g using the N keystream block pairs.
   - If the optimal distinguisher indicates this candidate to be the correct one then add it to the key candidate list.
3. **Second phase:** For each key candidate $\tilde{k}$ in the list:
   - Perform an exhaustive search over all the n remaining non-significant bits of $\tilde{k}$.
   - If the correct key is found display it and stop.

Let us estimate the complexity of the effective attack. We have that the **first phase** is executed 2^m times. During this phase we estimate the bias of g. To achieve that we need to compute the backward computation of each of the N keystream blocks pairs among $R - r$ round. This is almost the same thing as an encryption of Salsa. So for the first phase we need $N \cdot 2^m$ Salsa encryptions. If during the first phase the right subkey is inside the set of key candidates, then the second phase is executed once for the right subkey and for each of the false positives in the worst case. The second phase is a brute force on the n non-significant bits so it takes 2^n Salsa encryptions. Since there are on average $p_{fa} \cdot 2^m = 2^{m-\alpha}$ false positives, we have a total complexity of $2^m \cdot N + 2^n + 2^{8 \cdot w - \alpha}$

Since N is a function of α, in practice we chose α such that the above formula

is minimised. The details of the construction of an optimal distinguisher are given in [9].

In [1], the authors remarked that a false positive subkey which is correlated to the correct subkey can introduce a high bias. It results in an increase of the estimated value of p_{fa}. In fact this bias happens if there are some significant bits which have a high neutrality measure. In this case the bias would almost remain unchanged for all combination of those key bits.

4 Experimental results

4.1 Salsa over 32-bit words

We have made an implementation of the attack presented in [1]. This version of Salsa is Salsa$_{(32,7)}$, *i.e.,* Salsa with 32-bit word and with 7 rounds. We used the 4-round truncated differential $\left([\Delta_1^4]_{14}|[\Delta]_{31}\right)$ with $|\epsilon_d^*| = 0.131$. We found the same results in term of complexity as the ones given in [1]. We can see that we obtain the same result as in the original paper. The best attack is given when we choose $\gamma = 0.5$, then we have an attack which uses 2^{26} keystream blocks and 2^{151} Salsa encryptions.

4.2 Salsa over 16-bit words

Truncated differentials: We have made a version of the differential analysis for Salsa$_{(16,7)}$. In this case, we did not find a truncated differential after four rounds forward bigger than 10^{-4}. This shows that the version of Salsa defined over 16-bit words is more resistant to differential analysis than the version over 32-bit words.

We have found several high probability differentials after three rounds of Salsa. We give the two highest of them in Table 1.

ϵ_d	$\mathcal{ID}$	$\mathcal{OD}$
0.977	$[\Delta_7^0]_{15}$	$[\Delta_9^3]_{12}$
0.969	$[\Delta_7^0]_{14}$	$[\Delta_9^3]_{11}$

Table 1: Truncated differentials over three rounds (16-bit version)

Backward computation: For an attack against Salsa$_{(16,6)}$, we used the first differential of Table 1, it does not give a lot of PNB's after three rounds backward computations but it was sufficient to have an attack better than brute force. The results of this attack are given in Table 2.

The last line of Table 2 gives an attack of Salsa$_{(16,6)}$ with time complexity 2^{117} and data complexity 2^{16}.

We also decided to analyse Salsa$_{(16,5)}$, *i.e.,* we looked only after two round backward computations. We give in Table 3 the result of the attack using the first differential of Table 1. This gives an attack of Salsa$_{(16,5)}$ with time complexity 2^{71} and data complexity 2^7.

γ	n	$\lvert\epsilon_a^*\rvert$	$\lvert\epsilon^*\rvert$	α	Time	Data
0.50	17	0.383	$0,3738$	10	2^{120}	2^8
0.40	23	0.091	0.0888	13	2^{118}	2^{13}
0.30	28	0.023	0.0225	14	2^{117}	2^{16}

Table 2: Attack on Salsa$_{(16,6)}$

γ	n	$\lvert\epsilon_a^*\rvert$	$\lvert\epsilon^*\rvert$	α	Time	Data
1.00	46	1.000	0.9761	41	2^{89}	2^6
0.99	64	0.988	$0,9631$	60	2^{71}	2^7
0.90	81	0.753	0.7356	64	2^{81}	2^8

Table 3: Attack on Salsa$_{(16,5)}$

4.3 Salsa over 8-bit words

Truncated differentials: We have also analysed Salsa$_{(8,5)}$. As for the 16-bit case, we failed to find a useful truncated differential after four round forward computations and the number of PNB's were to small after three round backward computations. Table 4 gives the differentials found after three round forward.

ϵ_d	$\mathcal{ID}$	$\mathcal{OD}$
0.7607	$[\Delta_7^0]_7$	$[\Delta_4^3]_3$
0.8205	$[\Delta_7^0]_7$	$[\Delta_4^3]_4$
0.7533	$[\Delta_8^0]_7$	$[\Delta_9^3]_3$
0.8146	$[\Delta_8^0]_7$	$[\Delta_9^3]_4$

Table 4: Truncated differentials

γ	n	$\lvert\epsilon_a^*\rvert$	$\lvert\epsilon^*\rvert$	α	Time	Data
1.00	23	1.000	0.7607	18	2^{48}	2^7
0.90	31	0.880	0.6696	25	2^{41}	2^7
0.80	37	0.706	0.5372	30	2^{38}	2^8
0.70	40	0.706	0.5371	32	2^{40}	2^8
0.50	42	0.397	0.3019	32	2^{42}	2^{10}

Table 5: Attack on Salsa$_{(8,5)}$

Backward computation We use the first differential of Table 4 in the attack. As shown in Table 5, it gives an attack on Salsa$_{(8,5)}$ with time complexity 2^{38} and data complexity 2^8. These complexities are low enough to construct a practical key recovery.

4.4 Practical key recovery on Salsa$_{(8,5)}$

We have implemented a practical attack against Salsa$_{(8,5)}$. The machine we used for this experiment was a PC with an Intel Pentium 4 CPU with a 3.00GHz clock. We used the first differential of Table 4 and the neutrality measure of 0.8 with the parameters given in Table 5. We made 232 experiments of this attack with random key each time.

The first phase outputs on average 153 key candidates. This is not what was predicted by the theory because we expected to have average $2^{m-\alpha} = 2^{-7}$ false positive, that is, we expected to have almost no false positive. This phenomenon is due to what we explained in Section 3.5, there are some significant bits which have a high neutrality measure. Figure 2 shows the distribution of the number of subkey candidates after the first phase. If we consider this distribution as a Gaussian distribution then the average of this distribution is 152.66, the variance is 121.55 and with probability 0.997 the average is in the range $[149, 154]$. We were able to well approximate the value of the number of subkey candidate after the first phase.

This first phase lasts on average 7 hours and 15 minutes on our workstation.

Then for each candidate, the brute-force phase takes about 16 hours which gives an overall attack in about 102 days (16×153 hours) whereas a brute-force attack would take on average 124538 years on the same computer. We did not run the whole second phase on our work station since it takes too long. However, we ran it for the right subkey in order to verify that the program outputs the correct key at the end. We also ran the brute force phase on some false subkeys in order to measure the time it takes to finish this phase.

We have tried to increase the value of α to 65 instead of 30 in order to reduce the set of key candidates. This gives that on average the set of key candidates is 58 and the first phase takes about 14 hours and 20 minutes. With this modification, the whole attack lasts about 39 days. We have found that it is useless to increase the value of α, the set size of the key candidates will stay almost the same.

This practical key recovery attack shows that the concept of probabilistic neutral bits can be applied successfully in practice but the number of false positive is higher than the one predicted by the theory. Therefore, the theoretical complexity does not describe *exactly* the running time of the attack and some simulation needs to be performed to refine the complexity estimation.

Reducing the average practical attack running time The distinguisher of Section 3.5 is constructed by comparing the correlation measures of the subkeys with a fixed threshold. In other words the ones having a correlation measure greater than the predefined threshold pass the filter. This distinguisher does not need extra memory to store the filtered subkeys. They can be tested for the second phase right after they pass the filter. One can also consider a distinguisher which works based on the ranking of the subkeys according to their correlation measure which require extra memory to store the candidates which pass the filter but performs better than the earlier.

We measured the number of wrong subkeys which had bias larger than the bias of the correct subkey. We give the results of this experiment in Figure 3. With these results we were able to deduce that the number of false positive candidates which have a correlation measure bigger than the one of the right key are on average 30 for the same setup as before. After the end of the first phase, if we sort the subkey candidates in function of their correlation measure then it gives on average less computation for the second phase and so we can construct a modified effective attack which is completely given in Appendix B.

In this new attack, the second phase is executed on average only 30 times so if we use $\alpha = 65$ it reduces the whole running time to 21 days on average. Despite decreasing the total complexity, it is still higher than what is theoretically expected. In Table 6 we give the comparison between the theoretical complexity and practical results. We can see that the complexity in practice is higher than the one predicted by the theory but it still gives an effective attack moreover our modified attack reduces on average this complexity.

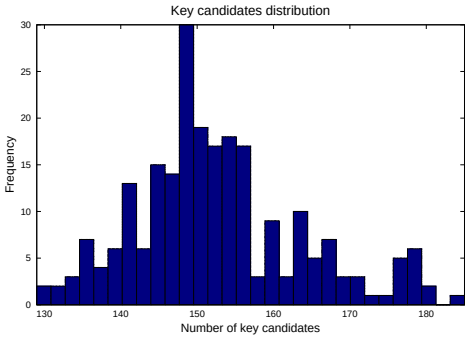

Fig. 2: Subkey candidate distribution

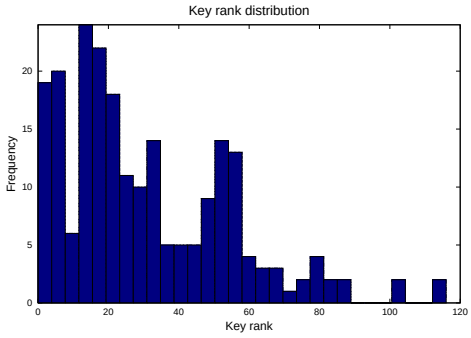

Fig. 3: Right subkey rank

	Theoretical complexity	Practical attack	Modified attack
Salsa encryptions	2^{38}	2^{45}	2^{42}
Key stream blocks	2^{8}	2^{8}	2^{8}

Table 6: Complexity comparisons

Effect on the complexity of the attack on Salsa20/7 and Salsa20/8

We have seen that in the previous attack some unforeseen false positives can appear at the end of the first phase increasing the complexity of the attack. Therefore, it is necessary to investigate how the complexity 2^{251} of the attack on Salsa20/8 proposed by Aumasson et al. is affected in light of this phenomenon. Especially since it is quite close to the average complexity of 2^{255} of a full brute force attack.

We made the experiment of computing the first phase for the 32-bit version of Salsa20/7 but we fixed most of the key bits to the correct key bits value and we permutated the remaining key bits which have the highest neutrality measure. We did that because we have seen that the false positives are always given by combination of significant key bits which have a high neutrality measure. In fact we sorted the m significant key bits in the descending order of their neutrality measure then we set the first $m - t$ of them to the value of the right key and we tried for each possibility of the t remaining key bits if the key candidate pass the optimal distinguisher or not.

For Salsa20/7, we saw that after $t = 4$ the number of false positive stops to increase and there is on average about 10 key candidates which pass the optimal distinguisher. The time complexity of the attack presented by Aumasson et al. was stated to be 2^{151} but in practice it should be 2^{152}. With the improved attack we presented in Section 4.4, the previous complexity should be unchanged compared the complexity in [1]. In addition, we have made this previous simulation on $\text{Salsa}_{(8,5)}$ and after $t = 15$ the number of false positive is 157 and stays constant. This number of false positives, is close to the one estimated in

Section 4.4. It shows that this simulation should be a solution for estimating the number of false positive.

However, the simulations were too long to be applied for the attack on Salsa20/8 due to huge data complexity of 2^{31} . Nevertheless, we have seen that false positive were given by significant key bits with a high neutrality measure. In the attack on Salsa20/8, the thereshold γ is 0.12 so it is likely that there are no significant key bits with a neutrality measure enough to pass the optimal distinguisher. However, even with 2^3 false positives at the end of the first phase, the complexity of the attack would remain unchanged since the complexity is dominated by the first phase. Hence we still consider Salsa20/8 as a theoretically broken cipher in time 2^{251}.

5 Conclusion

We gave concrete evidence that the concept of probabilistic neutral bits can be applied successfully in practice and since this concept is quite general we can imagine that it could be applied to the analysis of other ciphers. As explained by Bernstein, it is necessary to validate new attacks on reduced versions to see the non predicted effects of the attack and to see if the predicted complexity agrees with the practical running time. We showed that in this new kind of attack, an unpredicted phenomenon could arise. After the first phase of the attack, some false positives which are related to the key happen and they are not predicted by the theory. This phenomenon needs to be taken into account to compute the complexity of such attacks and compare the results with other attacks specially when the attack is very competitive with the other known ones. We proposed a new method to limit its impact. In particular, we show that this phenomenon makes the best attack on Salsa20/7 two times slower but we expect that the attack on Salsa20/8 remains unchanged.

We also see that Salsa on bigger word size needs more number of rounds to achieve its theoretically expected security level. The minimum number of rounds for 8, 16 and 32-bit versions is respectively 6, 7 and 9 to provide respective security of 64, 128 and 256 bits.

Acknowledgments. I would like to thank Prof. Arjen Lentra who allowed me to work on this project. I also would like to thank Shahram Khazaei and Martijn Stam for all their help from the begining to the end of this project.

References

1. J.P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In *Fast Software Encryption: 15th International Workshop*, volume 5086/2008 of *Lecture Notes in Computer Science*, pages 470 – 488, 2008.
2. D. J. Bernstein. Snuffle 2005: the Salsa20 encryption function, 2005. `http://cr.yp.to/snuffle.html`.

3. D.J. Bernstein. Salsa20 specication. 2005. http://cr.yp.to/snuffle/spec.pdf.
4. D.J. Bernstein. Salsa20/8 and Salsa20/12. *eSTREAM, the ECRYPT Stream Cipher Project*, 2006. http://www.ecrypt.eu.org/stream/papersdir/2006/007.pdf.
5. P. Crowley. Truncated differential cryptanalysis of five rounds of Salsa20. *eSTREAM, the ECRYPT Stream Cipher Project*, 2005. http://www.ecrypt.eu.org/stream/papersdir/073.pdf.
6. ECRYPT. The eSTREAM project. http://www.ecrypt.eu.org/stream/.
7. S. Fischer, W. Meier, C. Berbain, J.F. Biasse, and M.J.B. Robshaw. Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In *Progress in Cryptology - INDOCRYPT 2006*, volume 4329/2006 of *Lecture Notes in Computer Science*, pages 2–16, 2006.
8. L.R. Knudsen. Truncated and higher order differentials. In *Fast Software Encryption: Second International Workshop*, volume 1008/1995 of *Lecture Notes in Computer Science*, pages 196–196, 1995.
9. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, 34:81–85, 1985.

A Computation of the Neutrality Measure

Algorithm 1.1 Neutrality measure estimation

Require: The number of total rounds R of Salsa20, the number of foreword rounds r, the key bit index l, the truncated differential chosen and the number of samples T.
Ensure: Determine an estimation of neutrality measure γ_l

```
ctr ← 0
for i = 1 to T do
    Pick a random X (with fixed constants).
    Compute X' from X by complementing bit [x_i]_j in matrix X.
    Z ← X + Round_R(X)
    Z' ← X' + Round_R(X')
    u ← Round^{-1}_{r-R}(Z − X) ⊕ Round^{-1}_{r-R}(Z' − X)
    Flip the l-th key bit in X and X'
    u* ← Round^{-1}_{r-R}(Z − X) ⊕ Round^{-1}_{r-R}(Z' − X)
    if [u_p]_q = [u*_p]_q then
        ctr ← ctr + 1
    end if
end for
γ_l ← 2 · ctr/T
```

B Modified effective attack

Effective attack:

1. For an unknown key k, collect N pairs of keystream blocks with the corresponding input difference of the precomputation.

2. **First phase:** For each choice of the m significant bits:
   - Estimate the bias of g using the N keystream block pairs.
   - If the optimal distinguisher indicates this candidate to be the correct one then add it to the key candidate list with its bias value.
3. **Second phase:** Sort the list of subkey candidate in the descending order of their bias.
4. For each key candidates $\tilde{k}$ in the list:
   - Perform an exhaustive search over all the n remaining non-significant bits of $\tilde{k}$.
   - If the correct key is found display it and stop.