On the Salsa20 Core Function

Julio Cesar Hernandez-Castro[†], Juan M. E. Tapiador[‡], and
Jean-Jacques Quisquater[†]

[†]Crypto Group, DICE, Universite Louvain-la-Neuve
Place du Levant, 1 B-1348 Louvain-la-Neuve, Belgium
[‡]Computer Science Department, Carlos III University
Avda. de la Universidad, 30, 28911 Leganes, Madrid, Spain
julio.hernandez@uclouvain.be, jestevez@inf.uc3m.es,
jjq@uclouvain.be

Abstract. In this paper, we point out some weaknesses in the Salsa20 core function that could be exploited to obtain up to 2^{31} collisions for its full (20 rounds) version. We first find an invariant for its main building block, the **quarterround** function, that is then extended to the **rowround** and **columnround** functions. This allows us to find an input subset of size 2^{32} for which the Salsa20 core behaves exactly as the transformation $f(x) = 2x$. An attacker can take advantage of this for constructing 2^{31} collisions for any number of rounds. We finally show another weakness in the form of a differential characteristic with probability one that proves that the Salsa20 core does not have 2^{nd} preimage resistance.

Key words: Salsa20, hash function, cryptanalysis, collision.

1 Introduction

Salsa20 is a very interesting design by Daniel Bernstein [1]. It is mostly known because of its submission to the eSTREAM Project, where it passed to Phase 3 without major known attacks, although some interesting weaknesses over reduced-round versions have been pointed out [6, 8, 11]. As mentioned in [2], *"The core of Salsa20 is a hash function with 64-byte input and 64-byte output. The hash function is used in counter mode as a stream cipher: Salsa20 encrypts a 64-byte block of plaintext by hashing the key, nonce, and block number and xor'ing the result with the plaintext."* Note, however, that in spite of its name, the Salsa20 "hash" function was never really intended for hashing.

Reduced-round versions Salsa20/12 and Salsa20/8 (respectively using 12 and 8 rounds) have been proposed [3], although the author acknowledges that the security margin for Salsa20/8 is not huge, in view of the attack against Salsa20/5 presented in [6]. However, the speed gain over the full Salsa20 is very significant. Unfortunately, serious doubts over the security of Salsa20/8 were raised later over the publication of [11], which essentially breaks Salsa20/6 and successfully attacks Salsa20/7.

Salsa20 represents quite an original and flexible design, where the author justifies the use of very simple operations (addition, xor, constant distance rotation) and the lack of multiplication or S-boxes to develop a very fast primitive. Moreover, its construction protects it from timing attacks.

We find that, although this paper shows some vulnerabilities in its underlying cryptographic core, Bernstein's approach is indeed valuable and should be further investigated. For more information about Salsa20 design, please refer to the rationale presented by its author in [4].

The rest of the paper is organized as follows. Section 2 presents the main results in the form of various theorems, and Section 3 shows how these results can be practically used to find collisions for the full Salsa20 "hash" function. Section 4 ends the paper with some conclusions. In the Appendix, we show two collisions (out of the 2^{31} presented in this paper) for testing purposes.

2 Main Results

The main building block of the Salsa20 "hash" is the **quarterround** function, defined as follows:

Definition 1 *If* $y = \begin{pmatrix} y_0 & y_1 \\ y_2 & y_3 \end{pmatrix}$ *then* **quarterround**$(y) = \begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$, *where:*

$$z_1 = y_1 \oplus ((y_0 + y_3) \lll 7) \quad (1)$$
$$z_2 = y_2 \oplus ((z_1 + y_0) \lll 9) \quad (2)$$
$$z_3 = y_3 \oplus ((z_2 + z_1) \lll 13) \quad (3)$$
$$z_0 = y_0 \oplus ((z_3 + z_2) \lll 18) \quad (4)$$

and $X \lll n$ *is the rotation of the 32-bit word* X *to the left by* n *positions.*

Theorem 1 *For any 32-bit value* A*, an input of the form* $\begin{pmatrix} A & -A \\ A & -A \end{pmatrix}$ *is left invariant by the* ***quarterround*** *function, where* $-A$ *represents the only 32-bit integer satisfying* $A + (-A) = 0 \pmod{2^{32}}$.

Proof. Simply by substituting in the equations above, we obtain that every rotation operates over the null vector, so $z_i = y_i$ for every $i \in (0..3)$ □

Similarly, the **rowround** function, defined below, suffers from the same problem:

Definition 2 *If* $y = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}$ *then* **rowround**$(y) = \begin{pmatrix} z_0 & z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 & z_7 \\ z_8 & z_9 & z_{10} & z_{11} \\ z_{12} & z_{13} & z_{14} & z_{15} \end{pmatrix}$

where:

$$(z_0, z_1, z_2, z_3) = \textbf{quarterround}(y_0, y_1, y_2, y_3) \tag{5}$$
$$(z_5, z_6, z_7, z_4) = \textbf{quarterround}(y_5, y_6, y_7, y_4) \tag{6}$$
$$(z_{10}, z_{11}, z_8, z_9) = \textbf{quarterround}(y_{10}, y_{11}, y_8, y_9) \tag{7}$$
$$(z_{15}, z_{12}, z_{13}, z_{14}) = \textbf{quarterround}(y_{15}, y_{12}, y_{13}, y_{14}) \tag{8}$$

Theorem 2 *Any input of the form* $\begin{pmatrix} A & -A & A & -A \\ B & -B & B & -B \\ C & -C & C & -C \\ D & -D & D & -D \end{pmatrix}$*, for any 32-bit values* A, B, C *and* D*, is left invariant by the* ***rowround*** *transformation.*

Proof. This trivially follows from the repeated application of Theorem 1 to the four equations above. □

Remark. It is important to note that any other rearrangement of the equations from its canonical form:

$$(z_{4*i}, z_{4*i+1}, z_{4*i+2}, z_{4*i+3}) = \textbf{quarterround}(y_{4*i}, y_{4*i+1}, y_{4*i+2}, y_{4*i+3}) \tag{9}$$

will suffer from the same problem whenever the rearranging permutation keeps on alternating subindex oddness.

It is worth observing that this result implies that, from the 2^{512} possible inputs, at least one easily characterizable subset of size 2^{128} remains invariant by the **rowround** transformation.

The same happens with the **Columnround** function, which is defined below:

Definition 3 *If* $y = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}$ *then* $\textbf{columnround}(y) = \begin{pmatrix} z_0 & z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 & z_7 \\ z_8 & z_9 & z_{10} & z_{11} \\ z_{12} & z_{13} & z_{14} & z_{15} \end{pmatrix}$ *where:*

$$(z_0, z_4, z_8, z_{12}) = \textbf{quarterround}(y_0, y_4, y_8, y_{12}) \tag{10}$$
$$(z_5, z_9, z_{13}, z_1) = \textbf{quarterround}(y_5, y_9, y_{13}, y_1) \tag{11}$$
$$(z_{10}, z_{14}, z_2, z_6) = \textbf{quarterround}(y_{10}, y_{14}, y_2, y_6) \tag{12}$$
$$(z_{15}, z_3, z_7, z_{11}) = \textbf{quarterround}(y_{15}, y_3, y_7, y_{11}) \tag{13}$$

Theorem 3 *Any input of the form* $\begin{pmatrix} A & -B & C & -D \\ -A & B & -C & D \\ A & -B & C & -D \\ -A & B & -C & D \end{pmatrix}$*, for any 32-bit values* A, B, C *and* D*, is left invariant by the* ***columnround*** *transformation.*

Proof. This follows directly from the repeated application of Theorem 1, and can be seen as a dual of Theorem 2.

Theorem 4 *Any input of the form* $\begin{pmatrix} A & -A & A & -A \\ -A & A & -A & A \\ A & -A & A & -A \\ -A & A & -A & A \end{pmatrix}$ *for any 32-bit value* A, *is left invariant by the* ***doubleround*** *transformation.*

Proof. This is quite obvious. The point is that, due to the arrangement of the indexes in the **columnround** and the **rowround** function, we cannot have as free a hand. Here we are forced to make $B = -A$, $C = A$, and $D = -A$.

Taking into account that **doubleround** is defined as the composition of a **columnround** and a **rowround** operation:

$$\mathbf{doubleround}(x) = \mathbf{rowround}(\mathbf{columnround}(x)) \tag{14}$$

a common fixed point should be also a fixed point of its composition. □

3 Collision finding for the Salsa20 "hash" function

Theorem 5 *For any input of the form* $\begin{pmatrix} A & -A & A & -A \\ -A & A & -A & A \\ A & -A & A & -A \\ -A & A & -A & A \end{pmatrix}$ *and for any 32-bit value* A, *the Salsa20 core function behaves as a linear transformation of the form* $f(x) = 2x$, *and this happens independently of the number of rounds.*

Proof. As the Salsa20 "hash" is defined as:

$$Salsa20(x) = x + \mathbf{doubleround}^{10}(x) \tag{15}$$

and every input of the above form is an invariant (fixed point) for the **doubleround** function, then:

$$Salsa20(x) = x + \mathbf{doubleround}^{10}(x) = x + x = 2x \tag{16}$$

(And this happens independently of the number of rounds) □

The previous result is of great use in collision finding. All what is left now is to find two different nontrivial inputs, x and x', of the said form such that:

$$x \neq x' \quad \text{but} \quad 2x = 2x' \tag{17}$$

Fortunately, this is possible thanks to modular magic, i.e. the fact that all operations in Salsa20 are performed mod 2^{32}.

3.1 Modular magic

Let us assume that X is a 32-bit integer such that $X < 2^{31}$. Then, we define $X' = X + 2^{31}$. The interesting point here is that, even though $X \neq X'$, $2X = 2X'$ (mod 2^{32}).

Theorem 6 *Any pair of inputs* $\begin{pmatrix} Z & -Z & Z & -Z \\ -Z & Z & -Z & Z \\ Z & -Z & Z & -Z \\ -Z & Z & -Z & Z \end{pmatrix}$ *and* $\begin{pmatrix} Z' & -Z' & Z' & -Z' \\ -Z' & Z' & -Z' & Z' \\ Z' & -Z' & Z' & -Z' \\ -Z' & Z' & -Z' & Z' \end{pmatrix}$, *such that* $Z < 2^{31}$ *and* $Z' = Z + 2^{31}$, *generate a collision for any number of rounds of the Salsa20 "hash" function, producing* $\begin{pmatrix} 2Z & -2Z & 2Z & -2Z \\ -2Z & 2Z & -2Z & 2Z \\ 2Z & -2Z & 2Z & -2Z \\ -2Z & 2Z & -2Z & 2Z \end{pmatrix}$ *as a common hash value.*

Proof. This follows directly from the observations and definitions above. Substitution of the proposed input values into the formulæ for the Salsa20 "hash" will confirm this hypothesis. □

Corollary 1 *Theorem 6 implies that there are at least (these conditions are sufficient but probably not necessary)* 2^{31} *input pairs that generate a collision in the output, proving that indeed Salsa20 is not to be used as-is as a hash function. As an example, two of these pairs are provided in the Appendix.*

Corollary 2 *Let us call inputs of the form discussed by Theorem 5* A-states. *Then, as a direct consequence of Theorem 6 the output by the Salsa20 "hash" function of any* A-state *is also an* A-state *(where, in this case, A is even). It could be interesting to check whether these states could be reached at any intermediate step during a computation beginning with a* non-A state. *This would have important security implications. However, it could be easily shown that this is not the case, so any state leading to an* A-state *should be an* A-state *itself.*

This property has an interesting similitude with Finney-states *for RC4 [7] and could be useful in mounting an impossible fault analysis for the Salsa20 stream cipher, as* Finney-states *were of key importance on the impossible fault cryptanalysis of RC4 [5].* A-states, *on the other hand, have the interesting advantage over* Finney-states *that their influence over the output is immediately recognized, so they can be detected in an even simpler way. On the other hand, it is much less likely to reach an* A-state *by simply injecting random faults, as the set of conditions that should hold is larger than for the RC4 case.*

Once we have shown that the Salsa20 "hash" function is not collision resistant, we focus on its security against 2^{nd} preimage attacks. The next result[1] reveals that 2^{nd} preimage attacks are not only possible but even easy.

Theorem 7 *Any pair of inputs A, B with a difference of*

$$A - B = A \bigoplus B = \begin{pmatrix} \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} \\ \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} \\ \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} \\ \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} & \mathit{0x80000000} \end{pmatrix}$$

will produce the same output over any number of rounds.

[1] This property was presented informally before by Robshaw [10] and later by Wagner [12]

Proof. This depends on two interesting observations. The first one is that *addition* behaves as *xor* over the most significant bit (that changed by adding `0x80000000`). So the result in each of the four additions on Definition 1 is the same when both its inputs are altered by adding 2^{31} (differences cancel out mod 2^{32}).
The second one is that in the **quarterround** function, all partial results $z_0, ..., z_3$ are computed after an odd number (three in this case) of *addition/xor* operations. As a result, **quarterround** conserves the input difference, and so it does **rowround**, **columnround** and **doubleround**. As in the last stage of the Salsa20 core function the input is added to the output; This forces input differences to cancel out. □

Corollary 3 *Theorem 7 could now be seen as a particular instance of 6 (because* $2 * \mathtt{0x80000000} = \mathtt{0x00000000}$*). It is interesting to point out that this result has some common points with the one on the existence of equivalent keys for TEA made by Kelsey et al. [9], and also with the exact truncated differential found by Crowley in [6] for a reduced-round version of the Salsa20 stream cipher.*

A direct consequence of this result is that the effective key/input space of the Salsa20 "hash" is reduced by half, so there is a speed up by a factor of 2 in any exhaustive key/input search attack. This also means that $Salsa20(x) = y$ has solution for no more than (at most) half of the possible y's.

4 Conclusions

The Salsa20 "hash" function was never intended for cryptographic hashing, and some previous results showed that finding a good differential for the core function was not as hard as might have been expected [10]. Even though its author acknowledges that the Salsa20 core is not collision-free, to the best of our knowledge no work has so far focused on finding and characterizing these collisions. In this paper we explicitly show that there is a relevant amount (2^{31}) of easily characterizable collisions, together with an undesirable linear behavior over a large subset of the input space. In a sense, Theorem 6 is a generalization of Robshaw's previous observation.

Since the stream cipher uses four diagonal constants to limit the attacker's control over the input (thus making unreachable the differences needed for a collision), these results have no straightforward implications on its security. However, these undesirable structural properties might be useful to mount an impossible fault attack for the stream cipher. Particularly, what we have called A-states could play a role analogous to Finney states for RC4, in way similar to that presented by Biham et al. at FSE'05 [5]. We consider this as an interesting direction for future research.

That being said, we still consider that Salsa20 design is very innovative and well-motivated. Further work along the same guidelines should be encouraged. Particularly, we believe that a new, perhaps more complex and time consuming definition of the **quarterround** function should lead to a hash that would not be vulnerable to any of the presented attacks and could, in fact, provide a high-level security algorithm. This will, obviously, be more computationally expensive, but there may exist an interesting trade-off between incrementing the complexity of the **quarterround** function and decreasing the total number of rounds. The use of the add-rotate-xor chain at every stage of the **quarterround** function considerably eases the extension of these bad properties to any number of rounds. Although the author justified this approach because of performance reasons, we believe that alternating this structure with xor-rotate-add and making all output words depending on all input words will present the cryptanalyst with a much more difficult task. This should be the subject of further study.

On the other hand, in the light of our results we can also conclude that the inclusion of the diagonal constants is absolutely mandatory. An additional conclusion from our results is that less diagonal constants might suffice for stopping these kinds of undesirable structural properties, with a significant efficiency improvement that can vary from a 16% (from processing 384 bits to 448 bits in the same amount of time, that is, using only two diagonal constants) up to a 33% (in the extreme case of fixing the most significant bit of two diagonal 32-bit values).

Acknowledgments

The authors want to thank the anonymous reviewers, who contributed to improve this paper with their comments and suggestions. We specially want to thank Orr Dunkelman for his insights and many useful remarks.

References

1. D.J. Bernstein. “The Salsa20 Stream Cipher.” SKEW2005, Symmetric Key Encryption Workshop, 2005, Workshop Record. Available at `http://www.ecrypt.eu.org/stream/salsa20p2.html`
2. D.J. Bernstein. “Salsa20 Specification.” Available at `http://cr.yp.to/snuffle/spec.pdf`
3. D.J. Bernstein. “Salsa20/8 and Salsa20/12.” Available at `http://cr.yp.to/snuffle/812.pdf`
4. D.J. Bernstein. “Salsa20 design.” Available at `http://cr.yp.to/snuffle/design.pdf`
5. E. Biham, L. Granboulan, P. Q. Nguyen. “Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4.” Proceedings of the Fast Software Encryption Workshop 2005. LNCS 3557, pages 359–367.
6. P. Crowley. “Truncated Differential Cryptanalysis of Five Rounds of Salsa20.” In eSTREAM, ECRYPT Stream Cipher Project, Report 2005/073.
7. H. Finney. “An RC4 Cycle that Cant Happen.” sci.crypt newsgroup, September 1994.
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. J. Kelsey, B. Schneier and D. Wagner. “Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES.” Advances in Cryptology-CRYPTO ’96 California, USA, August 1996. Lecture Notes in Computer Science, 1109: 237-251, 1996.
10. M. Robshaw. “The Salsa20 Hash Function is Not Collision-Free”. June 22nd, 2005.
11. Y. Tsunoo, T. Saito, H. Kubo, T. Suzaki, and H. Nakashima. “Differential Cryptanalysis of Salsa20/8”. Submitted 2007-01-02. Available at `http://www.ecrypt.eu.org/stream/papersdir/2007/010.pdf`
12. D. Wagner. Message from discussion “Re-rolled Salsa-20 function” in the sci.crypt newsgroup on September 26th, 2005. Available at `http://groups.google.com/group/sci.crypt/msg/0692e3aaf78687a3`

Appendix: Collisions for the full Salsa20 hash function

Here we show a couple of collisions for testing purposes:

If

$$Z = \begin{pmatrix} \texttt{0xAAAAAAAA} & \texttt{0x55555556} & \texttt{0xAAAAAAAA} & \texttt{0x55555556} \\ \texttt{0x55555556} & \texttt{0xAAAAAAAA} & \texttt{0x55555556} & \texttt{0xAAAAAAAA} \\ \texttt{0xAAAAAAAA} & \texttt{0x55555556} & \texttt{0xAAAAAAAA} & \texttt{0x55555556} \\ \texttt{0x55555556} & \texttt{0xAAAAAAAA} & \texttt{0x55555556} & \texttt{0xAAAAAAAA} \end{pmatrix}$$

and

$$Z' = \begin{pmatrix} \texttt{0x2AAAAAAA} & \texttt{0xD5555556} & \texttt{0x2AAAAAAA} & \texttt{0xD5555556} \\ \texttt{0xD5555556} & \texttt{0x2AAAAAAA} & \texttt{0xD5555556} & \texttt{0x2AAAAAAA} \\ \texttt{0x2AAAAAAA} & \texttt{0xD5555556} & \texttt{0x2AAAAAAA} & \texttt{0xD5555556} \\ \texttt{0xD5555556} & \texttt{0x2AAAAAAA} & \texttt{0xD5555556} & \texttt{0x2AAAAAAA} \end{pmatrix}$$

then, the common Salsa20 hash value is

$$Salsa20(Z) = Salsa20(Z') = \begin{pmatrix} \texttt{0x55555554} & \texttt{0xAAAAAAAC} & \texttt{0x55555554} & \texttt{0xAAAAAAAC} \\ \texttt{0xAAAAAAAC} & \texttt{0x55555554} & \texttt{0xAAAAAAAC} & \texttt{0x55555554} \\ \texttt{0x55555554} & \texttt{0xAAAAAAAC} & \texttt{0x55555554} & \texttt{0xAAAAAAAC} \\ \texttt{0xAAAAAAAC} & \texttt{0x55555554} & \texttt{0xAAAAAAAC} & \texttt{0x55555554} \end{pmatrix}$$

Alternatively, if

$$W = \begin{pmatrix} \texttt{0xFFFFFFFF} & \texttt{0x00000001} & \texttt{0xFFFFFFFF} & \texttt{0x00000001} \\ \texttt{0x00000001} & \texttt{0xFFFFFFFF} & \texttt{0x00000001} & \texttt{0xFFFFFFFF} \\ \texttt{0xFFFFFFFF} & \texttt{0x00000001} & \texttt{0xFFFFFFFF} & \texttt{0x00000001} \\ \texttt{0x00000001} & \texttt{0xFFFFFFFF} & \texttt{0x00000001} & \texttt{0xFFFFFFFF} \end{pmatrix}$$

and

$$W' = \begin{pmatrix} \texttt{0x7FFFFFFF} & \texttt{0x80000001} & \texttt{0x7FFFFFFF} & \texttt{0x80000001} \\ \texttt{0x80000001} & \texttt{0x7FFFFFFF} & \texttt{0x80000001} & \texttt{0x7FFFFFFF} \\ \texttt{0x7FFFFFFF} & \texttt{0x80000001} & \texttt{0x7FFFFFFF} & \texttt{0x80000001} \\ \texttt{0x80000001} & \texttt{0x7FFFFFFF} & \texttt{0x80000001} & \texttt{0x7FFFFFFF} \end{pmatrix}$$

then, the common Salsa20 hash value is

$$Salsa20(W) = Salsa20(W') = \begin{pmatrix} \texttt{0xFFFFFFFE} & \texttt{0x00000002} & \texttt{0xFFFFFFFE} & \texttt{0x00000002} \\ \texttt{0x00000002} & \texttt{0xFFFFFFFE} & \texttt{0x00000002} & \texttt{0xFFFFFFFE} \\ \texttt{0xFFFFFFFE} & \texttt{0x00000002} & \texttt{0xFFFFFFFE} & \texttt{0x00000002} \\ \texttt{0x00000002} & \texttt{0xFFFFFFFE} & \texttt{0x00000002} & \texttt{0xFFFFFFFE} \end{pmatrix}$$