Response to "On the Salsa20 core function"

Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago
Chicago, IL 60607–7045
snuffle6@box.cr.yp.to

1 Summary

The paper "On the Salsa20 core function" by Hernandez-Castro, Tapiador, and Quisquater consists entirely of plagiarism of an observation that was made by Matt Robshaw in June 2005, that was independently posted to `sci.crypt` by David Wagner in September 2005, and that has been mentioned near the top of `http://cr.yp.to/salsa20.html` since May 2007.

The paper describes the observation as a "weakness" and "vulnerability" in the Salsa20 core function. In fact, exactly the opposite is true. The observation has no impact whatsoever on security: it bounces off a wall that was built into the Salsa20 design and that was already explained in the original "Salsa20 security" document.

Section 2 of this response reviews the wall—Salsa20's built-in protection against this type of problem. Section 3 reviews the comments by Robshaw and Wagner. Section 4 pinpoints the errors in the paper "On the Salsa20 core function."

2 How Salsa20 already avoids all of these problems

The Salsa20 stream cipher produces a 64-byte block by feeding a 32-byte key, an 8-byte nonce, and an 8-byte block counter to the "Salsa20 core," also called the "Salsa20 hash function." The Salsa20 core maps 64 bytes to 64 bytes.

The astute reader will already have observed that 32 plus 8 plus 8 is smaller than 64. The other 16 bytes of input to the Salsa20 core are 16 constant bytes appearing in the Salsa20 specification (and in the Salsa20 software, of course). Similarly, when the Rumba20 compression function uses the Salsa20 core, it provides only 48 varying input bytes to the Salsa20 core; the other 16 input bytes are 16 constant bytes appearing in the Rumba20 specification.

What role do these constants play? Read on.

The first example in the "Salsa20 specification" document (posted in April 2005 and submitted to eSTREAM), Section 8 ("The Salsa20 hash function"),

* Permanent ID of this document: `0a55a912218fd413ddb6b356665825a0`. Date of this document: 2008.02.24. This work was supported by the National Science Foundation under grant ITR–0716498.

shows that the Salsa20 core maps an all-0 input to an all-0 output. The second and third examples show that the Salsa20 core preserves diagonal shifts.

The "Salsa20 security" document (also posted in April 2005 and submitted to eSTREAM), Section 4 ("Notes on the diagonal constants"), comments on the diagonal-shift structure and then says

> The Salsa20 expansion function eliminates this shift structure by limiting the attacker's control over the hash-function input. In particular, the input diagonal is always `0x61707865`, `0x3320646e`, `0x79622d32`, `0x6b206574`, which is different from all its nontrivial shifts.

The all-0 structure is also eliminated by the diagonal constants, although I didn't bother to mention this in the "Salsa20 security" document. I did mention a third structure that is eliminated in the same way by the diagonal constants.

I didn't try to make a comprehensive list of all the things that could have gone wrong with Salsa20 (or Rumba20) if I hadn't included the diagonal constants. As a cryptographer, once I've found a solid low-cost solution to an entire class of potential problems, I simply adopt the solution and stop worrying about the problems. Perhaps other designers haven't adopted the solution yet; I don't mind cryptanalysts continuing to explore the problems with those other designs; but I *do* mind cryptanalysts overstating the impact of their work and pretending that the solution doesn't exist.

3 Collisions

I updated `http://cr.yp.to/salsa20.html` in May 2007 to include a paragraph on collision resistance:

> I originally introduced the Salsa20 core as the "Salsa20 hash function," but this terminology turns out to confuse people who think that "hash function" means "collision-resistant compression function." The Salsa20 core does not compress and is not collision-resistant. If you want a collision-resistant compression function, look at Rumba20.

Here is the background for this paragraph.

In May 2005, two out of five anonymous eSTREAM-workshop referees said that Salsa20 belonged in a SHA-replacement competition rather than an AES-replacement competition, simply because part of Salsa20 was labelled as a "hash function." I wrote back to the program chair as follows: "I feel compelled to add that these documents focus entirely on Salsa20's merits as a stream cipher. They say nothing about collision resistance. I have no idea how any referee could think that this submission belongs in a collision-resistance workshop rather than a stream-cipher workshop."

In June 2005, Matt Robshaw sent me a 2-page note "The Salsa20 hash function is not collision-free" saying that "$\text{Salsa20}(x) = \text{Salsa20}(x + \Delta)$" where $\Delta = (\texttt{0x80000000}, \texttt{0x80000000}, \ldots)$. I had never claimed that the Salsa20 hash function was collision-free; I told him so.

The same differential "`X ^ X' = (every word is 1<<31)`" was posted by David Wagner to `sci.crypt` in September 2005, along with the comment that the Salsa20 core "should not be used as a general-purpose hash function." See `http://groups.google.com/group/sci.crypt/msg/0692e3aaf78687a3`. In the same public discussion, Paul Rubin suggested that "the doc should be updated to warn people not to use the Salsa20 primitive as a general purpose hash function."

I responded as follows: "The documentation already (1) points out that the Salsa20 hash function has visible structure and (2) explains exactly how the Salsa20 encryption function chooses diagonal constants to eliminate all visible structure." Wagner's posting had already stated that "The salsa20 encryption algorithm doesn't allow inputs to be of this form"; there's no dispute regarding the underlying facts!

Matthijs van Duin asked "Is there a good way to make Salsa20 into a collision-resistant hash?" Subsequent discussions led to my introduction of the Rumba20 compression function ("What output size resists collisions in a xor of independent expansions?", posted April 2007 and appearing at the May 2007 ECRYPT Workshop on Hash Functions). Rumba20, unlike Salsa20, comes with a claim of collision-resistance.

In the meantime, reading many documents from other designers, I became convinced that the phrase "hash function" should be avoided in any serious technical discussion. I renamed the "Salsa20 hash function" as the "Salsa20 core" and updated `http://cr.yp.to/salsa20.html` as quoted above. In June 2007, after seeing further examples of "hash function" confusion, I expanded the paragraph slightly:

> I originally introduced the Salsa20 core as the "Salsa20 hash function," but this terminology turns out to confuse people who think that "hash function" means "collision-resistant compression function." The Salsa20 core does not compress and is not collision-resistant. If you want a collision-resistant compression function, look at Rumba20. (I wonder what the same people think of the FNV hash function, perfect hash functions, universal hash functions, etc.)

The paragraph hasn't changed since then.

4 The latest paper

The paper "On the Salsa20 core function" claims to "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" and to point out "another weakness in the form of a differential characteristic with probability one that proves that the Salsa20 core does not have 2nd preimage resistance."

In fact, both of these "weaknesses" consist of exactly the known collisions $\text{Salsa20}(x) = \text{Salsa20}(x + \Delta)$ with the same difference $\Delta = (\texttt{0x80000000}, \ldots)$ found by Robshaw and posted by Wagner in 2005. See Theorem 7 of the paper.

The Salsa20 diagonal constants prevent these collisions. If x is an input to Salsa20, then x has the specified diagonal constants, so $x + \Delta$ does not have the specified diagonal constants, so $x + \Delta$ is never an input to Salsa20—even if the key, nonce, and block counter are allowed to vary. These collisions are not a "weakness" in Salsa20; quite the opposite! They are an illustration of how the Salsa20 design successfully eliminates an entire class of potential problems.

The paper claims that the differential Δ has "some common points with ... equivalent keys for TEA made by Kelsey et al. ... [and] the exact truncated differential found by Crowley." That's incorrect. Kelsey's differential is in *key* bits. Crowley's differential is in *attacker-controlled* input bits. The differential Δ is not just in key bits and attacker-controlled input bits but in *constants* that cannot vary.

The paper claims that "there is a speed up by a factor of 2 in any exhaustive key/input search attack." That's incorrect. The paper has no impact whatsoever on the difficulty of brute-force attacks, or any other attacks, against Salsa20.

The paper manages to stretch the formula $\text{Salsa20}(x) = \text{Salsa20}(x+\Delta)$ into a nine-page paper, mostly through a tedious analysis of the special case presented in "Theorem 6." The paper's special case consists of any pair $x, x+\Delta$ where x has the form $(Z, -Z, Z, -Z, -Z, Z, -Z, Z, Z, -Z, Z, -Z, -Z, Z, -Z, Z)$; for example, if $Z = 0$, then x is the all-0 input and $x + \Delta$ is the all-`0x80000000` input. The paper claims that an attacker can "take advantage of this." That's incorrect, for two independent reasons. The first reason is that two Salsa20 inputs never have difference Δ. The second reason is that a Salsa20 input never has the form $(Z, -Z, Z, -Z, -Z, Z, -Z, Z, Z, -Z, Z, -Z, -Z, Z, -Z, Z)$.

The paper claims that these 2^{32} cases, out of 2^{512} possible states, "could be useful in mounting an impossible fault analysis," like Finney states for RC4. That's incorrect. The attacker can reasonably hope to enter a Finney state of RC4 through random faults; Finney states are a non-negligible fraction 2^{-16} of all RC4 states. In contrast, 2^{32} states are a negligible fraction 2^{-480} of all states and will never be entered through random faults.

The paper claims more broadly that it "shows some vulnerabilities in [the] underlying cryptographic core" of "Bernstein's approach." That's incorrect. The paper doesn't show any vulnerabilities in the Salsa20 core, or in anything else that I've designed.

The paper claims that it shows "an undesirable linear behavior over a large subset of the input space." That's incorrect. The inputs considered by the paper are inputs that were already prohibited by the Salsa20 specification.

The paper claims that "the Salsa20 'hash' function was never really intended for hashing." That's incorrect. The Salsa20 hash function was always intended for hashing, exactly as the name says. The Salsa20 hash function is used for hashing inside the Salsa20 stream cipher, the Rumba20 compression function, etc. The problem is that some people, including the authors of the paper, don't understand how many different types of operations are called "hashing."

The paper claims that it proves that "Salsa20 is not to be used as-is as a hash function." What the paper means, apparently, is that the 64-byte-to-64-

byte Salsa20 core is not to be used as a collision-resistant compression function. But what kind of idiot would think that a 64-byte-to-64-byte function is meant as a collision-resistant compression function?

I understand that the phrase "hash function" confuses some ignorant people—which is why I switched terminology to "Salsa20 core" in May 2007 and added an explicit warning to `http://cr.yp.to/salsa20.html` saying "The Salsa20 core ... is not collision-resistant." How can someone write a paper several months later claiming novelty for the observation that the Salsa20 core is not collision-resistant? The paper says that "[Bernstein] acknowledges that the Salsa20 core is not collision-free"; the paper inexplicably fails to say that this "acknowledgment" predates the paper.