Salsa20/8 and Salsa20/12

Daniel J. Bernstein *

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

Introduction. This document formally proposes two variants of the Salsa20 stream cipher:

- Salsa20/8, which is Salsa20 reduced from 20 rounds to 8 rounds; and
- Salsa20/12, which is Salsa20 reduced from 20 rounds to 12 rounds.

There are no known attacks on Salsa20/8, Salsa20/12, or the original Salsa20/20. Paul Crowley has published an attack (taking 2^{165} operations) on Salsa20/5, so the security margin in Salsa20/8 is obviously not huge; but the other eSTREAM submissions generally have no security margin at all.

The sole purpose of reducing the number of rounds is to save time. Salsa20/8 is spectacularly fast in software; see below for detailed timings of Salsa20/8 in various situations. One can also expect Salsa20/8 to perform very well in hardware; no timings are available at this point, but this document discusses the resources required for a Salsa20/8 hardware implementation.

Do Salsa20/8 and Salsa20/12 replace Salsa20/20? No. This issue was already covered in the original Salsa20 design document: "Should there be fewer rounds? I'm comfortable with the 20 rounds of Salsa20 as being far beyond what I'm able to break. Perhaps it will turn out that, after more extensive attempts at cryptanalysis, the community is comfortable with a smaller number of rounds; I can imagine using a smaller number of rounds for the sake of speed. On the other hand, Salsa20 will still have its place as a conservative design that is fast enough for practically all applications."

I'd be utterly astonished to see a successful attack on Salsa20/20, the original 20-round Salsa20. I can't express the same confidence about the other ciphers submitted to eSTREAM, or about AES/10, or about Salsa20/8. The literature has many examples of ciphers that weren't designed with large security margins, that seemed to withstand cryptanalysis for a while, and that were finally broken by a slight advance in cryptanalysis.

On the other hand, 10-round AES has survived without a large security margin. Perhaps cryptography doesn't need large security margins. Perhaps

* The author carried out this work while visiting Denmark Technical University. The author was also supported by the Alfred P. Sloan Foundation. Date of this document: 2006.02.07. Permanent ID of this document: 6975039c50783e7d4d93229a58305aa6. This document is final and may be freely cited.

Salsa20/8 will survive too. Even if Salsa20/8 is broken, I wouldn't be surprised to see Salsa20/12 withstanding all attacks.

One can draw an analogy here between Salsa20 and Serpent. The original 20-round Salsa20, like the original 32-round Serpent, was designed to achieve the maximum possible confidence subject to specified performance goals. In both cases, it's interesting to consider reduced-round variants that don't inspire as much confidence but that provide better performance.

One flaw in the analogy is that Salsa20/20 is, in absolute terms, more than twice as fast as Serpent/32. (It's clear to me that Serpent suffers from being a 16-byte block cipher; Salsa20 diffuses changes through a much larger block.) For example, on the Pentium III, Salsa20/8 streams at about 6 cycles per byte; Salsa20/20 and Serpent/13 stream at about 14 cycles per byte; Serpent/32 streams at about 35 cycles per byte. Some applications are unable to afford 35 cycles per byte; fewer applications are unable to afford 14 cycles per byte.

What about Salsa20/9, Salsa20/10, Salsa20/11, etc.? I don't think there's any point in taking such tiny steps, except to mark advances in cryptanalysis. Assume, for example, that Salsa20/r is secure for all $r \geq 10$. What application would notice the slowdown from Salsa20/10 to Salsa20/12? Salsa20/10 would also annoy software and hardware implementors who want to unroll 4 rounds for the sake of speed.

How fast is Salsa20/8 in software? I added a Salsa20/8 implementation to version 156 of ECRYPT's stream-cipher timing suite. I timed Salsa20/8 in several different situations:

- "40k": set up key, set up nonce, and encrypt 40-byte packet.
- "40": set up nonce and encrypt 40-byte packet.
- "576": set up nonce and encrypt 576-byte packet.
- "1500": set up nonce and encrypt 1500-byte packet.
- "long": encrypt one long stream.
- "agility": encrypt many parallel streams in 256-byte blocks.

The following table shows the results, all expressed in cycles per encrypted byte, as in [1]:

	PPC G4	PPC G5	A64	Athl Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	12.4	12.0	14.2	16.2	19.7	24.3	27.9	21.6	22.4	25.2	25.1	26.1	29.0	36.9
40	10.7	11.3	12.6	14.2	18.8	22.4	26.0	20.2	20.8	23.1	23.0	23.8	25.4	34.2
576	2.1	3.3	3.9	5.0	6.2	5.9	6.0	6.8	6.8	7.0	7.0	8.0	9.2	8.4
1500	2.2	3.4	3.9	5.1	6.3	5.9	6.3	6.9	6.9	7.0	6.9	8.1	9.2	8.3
long	2.0	3.2	3.6	4.8	6.0	5.5	5.8	6.5	6.6	6.6	6.6	7.6	9.3	7.6
agility	2.7	5.8	4.9	6.7	6.9	6.5	6.4	7.9	7.8	7.6	7.9	9.7	10.7	10.1

For example, to set up a nonce and encrypt a 576-byte packet, Salsa20/8 takes 2.1 cycles per encrypted byte (about 1200 cycles overall) on a PowerPC G4, and 8.4 cycles per encrypted byte on a Pentium 4 f29.

How does this speed compare to other submissions? The following table shows the speedup factor in switching from ABC version 2 to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10
40	1.30	1.42	1.02	0.94	0.57	0.75	0.45	2.42	1.14	0.83	0.83	0.95	0.90	0.73
576	2.57	1.79	0.97	1.12	0.61	0.92	0.83	1.00	0.96	1.03	0.91	0.71	1.10	0.83
1500	2.32	1.62	0.90	1.04	0.57	0.85	0.76	0.93	1.10	0.94	0.84	0.62	1.03	0.75
long	2.45	1.59	0.92	0.92	0.57	0.76	0.78	0.78	0.89	0.77	0.74	0.51	0.95	0.57
agility	7.19	>10	5.61	3.91	3.65	5.23	5.14	8.84	>10	2.97	6.27	2.40	2.89	2.62

For example, for a 576-byte packet, Salsa20/8 is 2.57 times faster than ABC on a PowerPC G4, but 0.83 times faster (i.e., 1.2 times slower) on a Pentium 4 f29. The table shows that, on most machines, Salsa20/8 is somewhat slower than ABC for long streams (for example, losing 3 cycles per byte on the Pentium 4 f29), but provides better key agility (for example, saving 16 cycles per byte on the Pentium 4 f29) and is much faster at key setup (for example, saving 50000 cycles on the Pentium 4 f29). Note that ABC has only a 128-bit key.

The following table shows the speedup factor in switching from Dragon to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	5.66	6.37	4.34	6.51	4.02	3.44	2.25	5.07	4.41	4.34	3.76	3.62	5.42	2.91
40	6.25	6.34	4.65	7.04	3.89	3.58	2.23	5.10	4.39	4.33	3.93	3.63	5.85	2.89
576	>10	>10	6.15	7.46	5.65	4.71	4.33	5.21	6.79	5.63	4.67	3.88	6.55	4.12
1500	>10	9.88	5.72	6.71	5.30	4.31	3.90	4.68	6.38	5.21	4.35	3.57	5.57	3.87
long	4.20	2.62	2.25	2.79	1.43	2.35	1.07	2.20	1.33	2.17	2.12	1.70	2.80	1.70
agility	3.74	2.16	2.12	2.51	1.46	2.32	1.27	2.52	1.55	2.16	2.19	1.81	2.54	1.82

The table shows that Salsa20/8 is always faster than Dragon.

The following table shows the speedup factor in switching from HC-256 to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10
40	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10
576	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10	>10
1500	>10	>10	>10	>10	>10	8.14	7.08	8.49	>10	8.46	8.46	>10	>10	9.96
long	3.10	1.91	1.22	1.19	1.03	0.80	0.91	1.00	1.36	0.88	0.86	0.66	1.25	0.63
agility	>10	5.03	3.71	5.18	3.38	3.28	2.78	6.99	7.33	3.14	4.48	2.79	3.13	2.95

The table shows that Salsa20/8 provides much better performance than HC-256 for small packets, for large packets, and for parallel streams. Performance for long streams may be better or worse, depending on the machine.

The following table shows the speedup factor in switching from LEX to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	2.46	2.23	1.68	2.04	1.26	1.49	0.75	1.71	1.17	1.56	1.48	1.56	2.47	1.34
40	2.12	1.67	1.44	1.78	0.98	1.24	0.55	1.27	0.99	1.35	1.25	1.00	2.26	0.94
576	5.86	3.30	2.46	2.74	1.69	2.29	1.27	2.07	1.66	2.16	2.01	1.52	3.39	1.73
1500	5.18	2.97	2.28	2.49	1.52	2.10	1.10	1.87	1.51	1.96	1.86	1.36	3.09	1.60
long	5.40	2.94	2.33	2.50	1.53	2.13	1.10	1.86	1.47	1.98	1.83	1.36	2.98	1.59
agility	4.74	2.24	2.18	2.31	1.59	2.15	1.34	2.08	1.69	2.01	1.90	1.46	2.70	1.55

The table shows that Salsa20/8 is generally faster than LEX. Note that LEX has only a 128-bit key, although I'm told that 256-bit keys are possible with slower setup.

The following table shows the speedup factor in switching from NLS to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	3.69	3.14	4.69	4.28	3.06	1.96	1.54	2.42	3.00	2.96	2.92	3.48	3.02	2.40
40	2.86	2.06	3.34	3.15	2.09	1.40	1.12	1.63	2.08	2.13	2.12	2.20	2.40	1.64
576	3.86	2.27	1.82	1.52	1.31	1.27	1.30	1.10	1.28	1.21	1.20	1.41	1.65	1.40
1500	3.18	1.94	1.38	1.14	1.03	1.03	1.06	0.88	1.03	0.89	0.88	1.15	1.22	1.18
long	3.45	2.09	1.39	1.17	1.07	1.09	1.10	0.92	1.11	0.91	0.91	0.93	1.09	0.92
agility	3.59	1.90	1.92	1.73	1.51	1.45	1.45	1.48	1.59	1.49	1.53	1.52	1.31	1.50

The table shows that Salsa20/8 is generally faster than NLS. Note that NLS has only a 128-bit key and is now believed vulnerable to a 64-bit attack.

The following table shows the speedup factor in switching from Phelix to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	5.57	5.99	2.05	2.07	3.54	2.79	2.68	1.88	4.11	2.99	2.99	2.20	2.15	1.67
40	4.89	4.28	1.66	1.75	2.60	2.40	2.30	1.49	3.24	2.50	2.50	1.66	1.81	1.28
576	8.14	4.91	1.56	1.46	2.52	2.53	2.73	1.32	3.09	2.44	2.43	1.50	1.54	1.50
1500	7.05	4.35	1.38	1.27	2.19	2.22	2.29	1.16	2.74	2.17	2.20	1.35	1.38	1.33
long	4.80	3.12	1.36	1.25	2.07	2.20	3.97	1.15	2.56	2.24	2.24	1.33	1.30	1.33
agility	4.56	2.34	1.47	1.42	2.09	2.23	3.86	1.47	2.56	2.34	2.25	1.46	1.37	1.45

The table shows that Salsa20/8 is always faster than Phelix. Note that Phelix claims only 128-bit security.

Beware that the above comparison is unfair to Phelix in one important way: Phelix provides free message authentication. On the other hand, when the goal is to survive a flood of forged packets, Phelix isn't as good as a fast cipher plus a fast authenticator: a separate authenticator allows forged packets to be discarded without being decrypted.

The following two tables show the speedup factor in switching from Py and Py6 to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	>10	>10	>10	>10	>10	>10	6.04	>10	>10	>10	>10	>10	>10	9.62
40	>10	>10	>10	>10	7.08	7.54	4.27	8.15	>10	7.94	7.90	>10	9.95	7.18
576	7.90	4.79	4.23	3.90	2.23	2.39	1.88	2.13	3.32	2.27	2.26	3.00	3.08	2.38
1500	4.41	2.65	2.59	2.20	1.35	1.19	1.06	1.10	1.91	1.19	1.19	1.68	2.21	1.24
long	2.70	1.47	1.11	1.04	0.88	0.49	0.72	0.49	1.02	0.67	0.48	0.63	1.16	0.50
agility	>10	3.47	5.10	6.15	2.71	4.11	2.30	9.78	2.77	4.39	5.99	2.91	3.69	2.69

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	6.75	8.15	6.47	5.88	4.25	5.60	2.38	7.69	5.08	5.19	5.71	4.68	4.32	3.43
40	5.83	5.49	4.96	4.63	2.72	4.20	1.63	4.81	3.67	4.43	4.27	3.79	2.99	2.46
576	4.38	2.70	2.56	2.08	1.34	1.54	1.10	1.44	1.68	1.49	1.40	1.46	1.89	1.07
1500	3.09	1.88	1.97	1.53	1.00	0.88	0.79	0.84	1.23	0.89	0.84	1.06	1.71	0.66
long	2.65	1.50	1.08	0.92	0.88	0.53	0.74	0.51	0.98	0.58	0.50	0.59	1.01	0.46
agility	4.96	1.93	2.45	2.51	1.42	1.65	1.30	3.20	1.87	1.74	2.09	1.53	1.68	1.22

The Py and Py6 tables are comparable to the ABC table. Note that Py and Py6 are now believed vulnerable to a 64-bit attack.

The following table shows the speedup factor in switching from Rabbit to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	5.20	3.70	1.61	1.82	1.87	1.42	1.37	1.60	2.15	1.50	1.43	2.31	2.12	1.86
40	4.01	2.61	1.18	1.34	1.36	1.02	0.97	1.08	1.45	1.06	1.05	1.57	1.49	1.22
576	8.48	3.52	1.28	1.30	1.68	1.24	1.42	1.15	1.85	1.23	1.17	1.45	1.63	1.45
1500	7.64	3.21	1.21	1.20	1.57	1.15	1.27	1.07	1.72	1.13	1.10	1.32	1.50	1.30
long	8.05	3.25	1.25	1.21	1.52	1.15	1.24	1.08	1.74	1.08	1.06	1.28	1.41	1.30
agility	6.37	2.31	1.24	1.22	1.61	1.18	1.33	1.28	1.76	1.13	1.18	1.32	1.37	1.29

The table shows that Salsa20/8 is generally faster than Rabbit. Note that Rabbit has only a 128-bit key.

The following table shows the speedup factor in switching from Sosemanuk to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	6.21	8.19	3.82	3.09	3.63	2.78	1.68	3.32	3.71	2.98	3.00	4.57	3.11	3.90
40	2.98	3.24	1.93	1.96	1.95	1.61	1.14	1.90	2.09	1.77	1.78	1.94	2.13	1.50
576	4.90	3.21	2.13	1.76	1.56	1.63	1.47	1.76	2.06	1.77	1.71	1.60	1.88	1.61
1500	4.05	2.62	1.87	1.47	1.27	1.37	1.17	1.48	1.77	1.49	1.48	1.31	1.65	1.30
long	3.10	2.19	1.22	1.17	0.93	0.95	1.05	0.95	1.26	0.92	0.92	0.82	1.18	0.86
agility	2.67	1.57	1.16	1.10	0.94	0.97	1.14	0.99	1.31	0.93	0.94	0.84	1.13	0.87

The table shows that Salsa20/8 is faster than Sosemanuk for packet encryption, slightly slower for streaming performance on some machines, and slightly faster for streaming performance on other machines. Note that Sosemanuk has a 256-bit key but claims only 128-bit security and has been shown vulnerable to a 224-bit attack.

The following table shows the speedup factor in switching from TRIVIUM to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	5.32	2.52	1.86	3.75	1.10	2.32	0.71	3.70	1.48	3.77	3.14	4.34	5.23	2.47
40	6.03	2.51	1.97	4.11	1.06	2.43	0.67	3.84	1.45	4.01	3.32	4.66	5.82	2.57
576	8.00	2.12	1.46	2.58	0.66	3.27	0.68	2.90	1.12	3.57	3.31	3.62	3.93	2.49
1500	6.68	1.79	1.23	2.12	0.49	3.00	0.56	2.48	0.96	3.13	3.00	3.16	3.74	2.13
long	6.65	1.69	1.17	3.10	0.43	1.07	0.53	2.37	0.91	1.00	1.00	3.07	3.31	2.11
agility	5.48	1.43	1.18	2.61	0.54	1.12	0.66	2.27	0.99	1.07	1.08	2.67	2.84	1.92

The table shows that Salsa20/8 is faster than TRIVIUM except on a few old CPUs. TRIVIUM performs somewhat fewer bit operations than Salsa20/8 (11 xors and 3 ands for each output bit, compared to 8 xors and 9 adds-with-carry), but it doesn't exploit the many fast addition circuits built into modern CPUs. Note that TRIVIUM has only an 80-bit key.

How does this speed compare to the official "benchmark" ciphers? The following table shows the speedup factor in switching from 10-round AES to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	3.28	3.09	2.17	3.02	1.72	1.75	1.11	2.05	1.93	1.89	1.88	1.67	3.59	1.44
40	3.26	2.65	1.99	3.02	1.54	1.62	0.98	1.89	1.59	1.78	1.77	1.50	3.46	1.30
576	>10	6.85	5.00	6.64	3.73	4.46	2.97	4.22	3.71	4.31	4.20	3.56	7.22	3.58
1500	>10	6.65	4.97	6.51	3.63	4.44	2.81	4.13	3.65	4.26	4.25	3.53	7.55	3.59
long	>10	7.03	5.36	6.85	3.80	4.71	3.03	4.35	3.80	4.44	4.41	3.70	7.22	3.88
agility	>10	4.78	4.43	5.54	3.70	4.43	3.17	4.23	3.76	4.22	4.09	3.35	6.18	3.43

The following table shows the speedup factor in switching from RC4 to Salsa20/8:

	PPC G4	PPC G5	A64	Athl	Alpha	PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	>10	>10	>10	>10	>10	>10	2.58	>10	5.29	>10	>10	>10	>10	>10
40	>10	>10	>10	>10	>10	>10	2.67	>10	5.53	>10	>10	>10	>10	>10
576	>10	8.97	9.82	7.68	6.05	5.25	1.87	4.56	3.24	4.63	4.66	6.20	5.17	5.12
1500	7.91	5.06	6.05	4.67	3.49	2.85	1.37	2.48	2.58	2.54	2.55	3.36	3.03	2.98
long	5.60	2.97	4.03	3.04	2.17	1.42	1.19	1.26	2.29	1.26	1.23	1.88	1.72	1.72
agility	5.19	2.47	4.53	4.22	2.74	2.69	1.83	5.19	2.58	2.39	1.77	2.01	1.70	1.77

The following table shows the speedup factor in switching from SNOW 2.0 to Salsa20/8:

	PPC G4	PPC G5	A64	Athl Alpha		PM 695	HP	P3 68a	SP III	P3 673	P3 6b1	P4 f12	P1 52c	P4 f29
40k	2.79	2.60	1.90	2.15	2.07	1.49	1.02	1.81	1.97	1.75	1.72	1.84	2.36	1.34
40	2.98	2.58	1.92	2.23	1.86	1.50	0.92	1.80	1.95	1.76	1.72	1.79	2.42	1.32
576	4.00	2.39	1.54	1.32	1.23	1.07	0.90	1.12	1.44	1.16	1.11	0.93	1.55	0.94
1500	3.55	2.15	1.38	1.16	1.13	0.92	0.79	0.97	1.32	1.00	0.99	0.78	1.42	0.78
long	3.55	2.09	1.36	1.06	1.00	0.84	0.78	0.91	1.18	0.91	0.89	0.70	1.31	0.71
agility	3.11	1.69	1.35	1.13	1.04	0.94	0.87	1.15	1.33	0.97	1.05	0.88	1.26	0.85

Is Salsa20/8 suitable for hardware? Yes. Salsa20/8 and Salsa20/12, like the original Salsa20/20, offer a wide range of attractive options for the hardware implementor. They can fit into a very small circuit area; alternatively, they can be parallelized for extremely high throughput; either way, Salsa20/8 and Salsa20/12 offer even better price-performance ratios than Salsa20/20.

A hardware implementation of Salsa20/8, like a hardware implementation of Salsa20/20, needs the following resources:

- Storage for the key, or several keys for a multiple-key chip. This storage has minimal size (for example, 256 bits for the recommended 256-bit keys); Salsa20 does not need space to store, or gates to compute, expanded keys. The key is read only twice for each 512-bit block, once at the beginning and once at the end, so one can save area without much loss of performance by storing the key in RAM rather than registers.
- Storage for the nonce and block counter, or several nonces and block counters for a multiple-session chip. As above, this storage has minimal size; Salsa20 does not use expanded nonces.
- Temporary storage used while generating a 512-bit output block. Typical implementations will use 512 flip-flops here. A tiny circuit could instead use 512 bits of RAM. Of course, no storage is required for a large high-throughput low-delay circuit that generates the entire output block combinatorially.
- 32-bit adders used in generating an output block. Salsa20/20 involves 0.65625 additions per bit of output; Salsa20/8 involves 0.28125 additions per bit of output. There are many options here: a minimum-area circuit using a 32-bit adder 16 times per round (with a simple pattern of RAM access); a circuit performing an entire round combinatorially, with 16 separate 32-bit adders; a 2-combinatorial-round circuit with 32 separate 32-bit adders; a 4-combinatorial-round circuit with 64 separate 32-bit adders; etc. At a lower level, there are many different ASIC adder structures offering various combinations of area and speed.
- Other gates used in generating an output block and encrypting data. There is some cost here—for example, each 32-bit addition is accompanied by a 32-bit xor—but Salsa20 does not use expensive operations such as multiplications or accesses to large tables.

- Wires and other overheads. Salsa20 was designed to allow shorter wires than a typical circuit, potentially saving both space and time, if hardware blocks are placed in Salsa20's 4×4 pattern.

Obviously Salsa20 can achieve reasonable performance in a small circuit, and higher performance in larger circuits. I don't know whether Salsa20/8 is as fast as today's best hardware-oriented ciphers, especially ciphers designed for a lower security level; but it shouldn't be omitted from hardware benchmarks.

For applications that need maximum streaming throughput, Salsa20 offers a huge advantage: it can be parallelized across any number of blocks. One can, for example, generate 256 blocks from a single stream in parallel with 256 copies of the Salsa20 hardware, either on a single chip or spread across chips. (This parallelization should also improve price-performance ratio somewhat: one does not need to store 256 copies of the key, for example.) Traditional LFSR-based stream ciphers offer the same feature, but most other stream ciphers don't.

References

1. Daniel J. Bernstein, *Comparison of 256-bit stream ciphers at the beginning of 2006* (2006). URL: `http://cr.yp.to/papers.html#stream256`. ID `eff0eb8eebacda584 62948ab97ca48a0`. Citations in this paper: §1.