D. J. Bernstein
Hash functions and ciphers

Snuffle 2005: the Salsa20 encryption function

The Salsa20 encryption function, also known as Snuffle 2005, uses the Salsa20 core to encrypt data.

Salsa20 specification; Salsa20 design

The following paper explains how Salsa20 works:

- [salsafamily] 15pp. (PDF) D. J. Bernstein. The Salsa20 family of stream ciphers. Document ID: 31364286077dcdff8e4509f9ff3139ad. URL: http://cr.yp.to/papers.html#salsafamily. Date: 2007.12.25.

The original Salsa20 documentation split this information into two documents (including many extra examples in the specification):

- Salsa20 specification. This document contains the complete definition of the Salsa20 core, the Salsa20 expansion function, and the Salsa20 encryption function.
- Salsa20 design. This document discusses the Salsa20 design, explaining why this particular encryption function is better than other encryption functions.

The following paper defines XSalsa20, a variant of Salsa20 with a longer nonce, and proves that XSalsa20 is secure if Salsa20 is secure:

- [xsalsa] 14pp. (PDF) D. J. Bernstein. Extending the Salsa20 nonce. Document ID: c4b172305ff16e1429a48d9434d50e8a. URL: http://cr.yp.to/papers.html#xsalsa. Date: 2011.02.04. Supersedes: (PDF) 2008.11.28.

Salsa20 speed; Salsa20 software

There are two documents analyzing Salsa20 performance:

- Salsa20 speed. This document discusses a range of benchmarks relevant to cryptographic speed; estimates Salsa20's performance on those benchmarks; and explains, at a lower level, techniques to achieve this performance. This document is

somewhat out of date: it does not reflect some speedups achieved by the current Salsa20 software.

- Salsa20/8 and Salsa20/12. This document proposes reduced-round (8-round and 12-round) variants of Salsa20; presents detailed software benchmarks comparing Salsa20/8 to ABC version 2, 10-round AES, Dragon, HC-256, LEX, NLS, Phelix, Py, Py6, Rabbit, RC4, SNOW, SOSEMANUK, and TRIVIUM; and discusses Salsa20 from a hardware perspective.

There are several Salsa20 implementations:

- `test`, an initial test implementation. This consists of ecrypt-config.h, ecrypt-machine.h, ecrypt-portable.h, and ecrypt-sync.h from the ECRYPT Stream Cipher Project; ecrypt.c, which encrypts and decrypts; and bigtest.c, which tries various Salsa20 encryptions and prints occasional checksums of the results, starting with e2d22467015c0ffb0adc5fac0ee88ccf8d467a7f07ab53d4efeac8da47fd833e.
- `ref`, a reference implementation that fits into the eSTREAM performance-testing framework: salsa20.c, Makefile, ecrypt-sync.h.
- `regs`: salsa20.c, Makefile, ecrypt-sync.h. Similar to `salsa20/ref` but uses separate temporary variables instead of a temporary array.
- `merged`: salsa20.c, Makefile, ecrypt-sync.h. Similar to `salsa20/regs` but inlines the Salsa20 core.
- `x86-1`, specific to the Pentium, Athlon, and other x86 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.
- `x86-2`, specific to the Pentium, Athlon, and other x86 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.
- `x86-3`, specific to the Pentium, Athlon, and other x86 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Similar to x86-2 but organizes the stack differently.
- `x86-pm`, specific to the Pentium, Athlon, and other x86 chips: salsa20.s, Makefile, ecrypt-sync.h.
- `x86-athlon`, specific to the Pentium, Athlon, and other x86 chips: salsa20.s, Makefile, ecrypt-sync.h. Similar to x86-3 but pads instructions to 16-byte boundaries.
- `x86-mmx`, specific to the Pentium Pro and other CPUs with MMX registers: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Uses some MMX registers as substitutes for stack positions.
- `x86-xmm`, specific to the Pentium 4 and other CPUs with SSE2 instructions: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.
- `x86-xmm2`, specific to the Pentium 4 and other CPUs with SSE2 instructions: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Selects different SSE2 instructions.
- `x86-xmm4`, specific to the Pentium 4 and other CPUs with SSE2 instructions: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Handles four blocks in parallel. Based on code by Wei Dai.
- `x86-xmm5`, specific to the Pentium 4 and other CPUs with SSE2 instructions: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Some additional instruction scheduling.
- `amd64-1`, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.
- `amd64-2`, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.
- `amd64-3`, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.

- `amd64-xmm`, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.
- `amd64-xmm2`, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Selects different SSE2 instructions.
- `amd64-xmm5`, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Handles four blocks in parallel. Translation of x86-xmm5.
- `amd64-xmm6`, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h. Some additional instruction scheduling.
- `ppc-altivec`, specific to the PowerPC G4 and other CPUs with AltiVec instructions: salsa20.c, Makefile, ecrypt-sync.h. Based on code by Matthijs van Duin.
- `sparc`, specific to the UltraSPARC and other 64-bit SPARC chips: salsa20.q, salsa20.s, Makefile, ecrypt-sync.h.

Some older implementations, limited to the Salsa20 core and not including stream generation, appear on the Salsa20-core speed page.

ECRYPT's test framework reports the following speeds for encrypting a 576-byte packet (or a long stream) with a 256-bit key:

Salsa20 cycles/byte	Salsa20/12 cycles/byte	Salsa20/8 cycles/byte	4-round cycles/byte	Implementation	Machine
4.25 (3.93)	3.25 (2.80)	2.07 (1.88)	0.73 (0.68)	amd64-xmm6 in 20070618	amd64 3000MHz Intel Xeon 5160 (6f6) named td162
4.33 (3.91)	2.80 (2.57)	2.07 (1.88)	0.75 (0.68)	amd64-xmm6 in 20070618	amd64 2137MHz Intel Core 2 Duo (6f6) named katana
4.39 (4.24)	2.88 (2.74)	2.14 (1.99)	0.75 (0.75)	ppc-altivec in 20070618	ppc32 533MHz Motorola PowerPC G4 7410 named gggg
4.70 (4.32)	3.15 (2.80)	2.28 (2.06)	0.81 (0.75)	x86-xmm5 in 20070618	x86 2137MHz Intel Core 2 Duo (6f6) named katana32
7.84 (7.64)	5.04 (4.86)	3.65 (3.47)	1.39 (1.39)	amd64-3 in 20070618	amd64 2000MHz AMD Athlon 64 X2 (15,75,2) named mace
8.04 (7.82)	4.87 (4.83)	3.48 (3.28)	1.52 (1.51)	ppc-altivec in 20070618	ppc64 2000MHz IBM PowerPC G5 970 named geespaz
8.62 (8.42)	5.51 (5.33)	3.96 (3.78)	1.55 (1.55)	amd64-3 in 20070618	amd64 2391MHz AMD Opteron (f5a) named td159
8.78 (8.42)	5.73 (5.35)	4.18 (3.82)	1.53 (1.53)	amd64-3 in 20070618	amd64 2192MHz AMD Opteron (f58) named td189

10.07 (9.80)	6.55 (6.27)	4.78 (4.50)	1.76 (1.77)	x86-1 in 20070618	x86 2000MHz AMD Athlon 64 X2 (15,75,2) named mace32
10.24 (10.04)	6.65 (6.44)	4.84 (4.61)	1.80 (1.81)	x86-athlon in 20070618	x86 900MHz AMD Athlon (622) named thoth
11.47 (11.29)	8.51 (8.35)	7.00 (6.83)	1.49 (1.49)	merged in 20070618	ppc64 1452MHz IBM POWER4 named tigger
11.56 (11.39)	7.85 (7.68)	5.97 (5.82)	1.86 (1.86)	merged in 20070618	hppa 1000MHz HP PA-RISC 8900 named td191
11.73 (10.69)	7.84 (7.19)	5.87 (5.38)	1.95 (1.77)	amd64-xmm6 in 20070618	amd64 3000MHz Intel Pentium D (f64) named svlin001
11.98 (11.70)	7.70 (7.44)	5.53 (5.30)	2.15 (2.13)	x86-xmm5 in 20070618	x86 1300MHz Intel Pentium M (695) named whisper
12.55 (11.64)	8.21 (7.41)	5.86 (5.30)	2.23 (2.11)	x86-xmm5 in 20070618	x86 3000MHz Intel Xeon (f26) named td185
12.59 (11.63)	8.15 (7.40)	5.84 (5.30)	2.25 (2.11)	x86-xmm5 in 20070618	x86 3200MHz Intel Xeon (f25) named td186
12.65 (11.67)	8.20 (7.44)	5.95 (5.33)	2.23 (2.11)	x86-xmm5 in 20070618	x86 2800MHz Intel Xeon (f29) named svlin003
13.40 (11.84)	9.33 (8.12)	6.92 (5.76)	2.16 (2.03)	x86-xmm5 in 20070618	x86 3000MHz Intel Pentium 4 (f41) named svlin002
14.29 (13.88)	9.29 (8.88)	6.79 (6.37)	2.50 (2.50)	x86-mmx in 20070618	x86 1400MHz Intel Pentium III (6b1) named td152
14.45 (14.34)	9.33 (9.21)	6.76 (6.65)	2.56 (2.56)	sparc in 20070618	sparc 1050MHz Sun UltraSPARC IV named hald
15.94 (15.29)	10.31 (9.90)	7.66 (7.13)	2.76 (2.72)	x86-athlon in 20070618	x86 3200MHz Intel Pentium D (f47) named shell
18.27 (18.07)	12.62 (12.42)	8.87 (8.49)	3.13 (3.19)	merged in 20070618	ia64 1500MHz HP Itanium II named td178
18.40 (18.21)	12.76 (12.56)	8.65 (8.28)	3.25 (3.31)	merged in 20070618	ia64 1400MHz HP Itanium II named

					td156
19.93	12.73	9.14	3.59	x86-1	x86 133MHz Intel Pentium 1 (52c) named cruncher

AVR speeds: At the SASC 2007 workshop, Gordon Meiser reported a Salsa20 implementation for an 8MHz ATmega8 taking 292 cycles/byte and using 1514 bytes of flash memory. For comparison, Meiser reported an AES implementation for an 8MHz ATmega16 taking 786 cycles/byte and using 6664 bytes of flash memory.

ARM speeds: At the SASC 2007 workshop, Cedric Lauradoux reported a Salsa20 implementation for a 200MHz ARM920T taking 69 cycles/byte and using just 868 bytes of code. For comparison, Lauradoux reported an AES implementation taking 101 cycles/byte with 15920 bytes of code.

FPGA speeds: At the SASC 2007 workshop, Marcin Rogawski reported an unrolled-double-round Salsa20 implementation using 3510 logic elements on a Altera Cyclone EP1C20F324C6 (130nm process). The implementation is estimated to drain 450.14 mW at 30MHz and produce 1280 Mbps. For comparison, Rogawski reported an AES implementation using 5053 logic elements; the implementation is estimated to drain 1191.01 mW at 105MHz and produce 611 Mbps.

At the IEEE CCECE 2007 workshop, Yan and Heys reported a "compact" Salsa20 implementation using 194 CLB slices and 4 Block RAMs on a Xilinx 2V250fg256. The implementation is estimated to produce 38 Mbps.

ASIC speeds: At the SASC 2007 workshop, Tim Good reported an unrolled-double-round Salsa20 implementation for a 130nm ASIC, with area estimated as 18626 gate equivalents and speed estimated as 668 Mbps at 35.2 MHz. It's quite clear to me that these speeds are highly suboptimal; I look forward to seeing better Salsa20 ASIC implementations. For comparison, Good reported an AES implementation with area estimated as 5398 gate equivalents and speed estimated as 311 Mbps at 131.2 MHz.

At the IEEE CCECE 2007 workshop, Yan and Heys reported three Salsa20 implementations for a 180nm ASIC: a "compact" implementation with area estimated as 14100 gate equivalents and speed estimated as 71.2 Mbps, a "basic" implementation with area estimated as 23408 gate equivalents and speed estimated as 255 Mbps, and a "fast" implementation with area estimated as 470000 gate equivalents and speed estimated as 4800 Mbps.

Other languages: Larry Bugbee has implemented a Python wrapper for Salsa20: http://www.seanet.com/~bugbee/crypto/salsa20/.

Salsa20 security

I have several documents discussing the security of Salsa20:

- Salsa20 security. This document states the Salsa20 security conjectures and explains why the security conjectures are reasonable.

- Notes on the Salsa20 key size. This document discusses my recommendation to use 256-bit keys.
- Disproof of Li An-Ping's claims regarding Salsa20. This document illustrates the importance of computer verification of cryptanalytic claims.
- Response to "On the Salsa20 core function". This document illustrates the importance of Salsa20's diagonal constants.
- Response to "Slid pairs in Salsa20 and Trivium". This document illustrates the fact that "attacks" more expensive than brute force aren't useful.
- [bruteforce] 10pp. (PDF) (PS) (DVI) D. J. Bernstein. Understanding brute force. Document ID: 73e92f5b71793b498288efe81fe55dee. URL: http://cr.yp.to/papers.html#bruteforce. Date: 2005.04.25.

  This paper is not specific to Salsa20. It discusses the tremendous power of a parallel brute-force key-search machine, and corrects several related errors in the literature.

See also the Salsa20 diffusion page.

Cryptanalysts are strongly encouraged

- to generalize their attacks from 32-bit words to w-bit words for every w in {2,4,6,8,...,32};
- to generalize their attacks from 20 rounds to r rounds for every r in {2,4,6,8,...,20};
- to state the success probability of the attack for all pairs (w,r);
- to state the time taken by the attack for all pairs (w,r);
- to state the price of the attack machine (memory, etc.) for all pairs (w,r); and
- to publish software verifying these statements for as many pairs (w,r) as possible.

Every attack should be fast when w and r are small; and there is no excuse for failing to have a computer verify that a fast attack works.

Independent cryptanalysis:

- 2005.10: Crowley reported
  - a 2^165-operation attack on Salsa20/5.

  The attack works forwards from a small known input difference to a biased bit 3 rounds later, and works 2 rounds backwards from an output after guessing 160 relevant key bits.
- 2006.12: Fischer, Meier, Berbain, Biasse, and Robshaw reported
  - a 2^177-operation attack on Salsa20/6 and
  - a much faster attack on Salsa20/5, clearly breaking Salsa20/5.

  The Salsa20/6 attack works forwards from a small known input difference to a biased bit 4 rounds later, and works 2 rounds backwards from an output after guessing 160 relevant key bits.
- 2007.01: Tsunoo, Saito, Kubo, Suzaki, and Nakashima reported
  - a 2^184-operation attack on Salsa20/7 and
  - a much faster attack on Salsa20/6, clearly breaking Salsa20/6.

  The Salsa20/7 attack works forwards from a small known input difference to a biased bit 4 rounds later, and works 3 rounds backwards from an output after guessing 171

highly relevant key bits.

- 2007.12: Aumasson, Fischer, Khazaei, Meier, and Rechberger reported
  - a 2^249-operation attack on Salsa20/8,
  - a 2^248-operation attack on ChaCha7,
  - a 2^153-operation attack on Salsa20/7, and
  - a 2^139-operation attack on ChaCha6.

  The Salsa20/8 attack works forwards from a small known input difference to a biased bit 4 rounds later, and works 4 rounds backwards from an output after guessing 228 extremely relevant key bits. (See also Fischer's thesis, 2008.04.)
- 2012.11: Shi, Zhang, Feng, and Wu reported
  - a 2^250-operation attack on Salsa20/8 (while saying that the previous attack takes 2^251 operations),
  - a 2^246.5-operation attack on ChaCha7,
  - a 2^148-operation attack on Salsa20/7 (while saying that the previous attack takes 2^151 operations), and
  - a 2^136-operation attack on ChaCha6.
- 2015.07: Maitra reported
  - a 2^245.5-operation attack on Salsa20/8 and
  - a 2^239-operation attack on ChaCha7.
- 2016.10: Choudhuri and Maitra reported
  - a 2^244-operation attack on Salsa20/8,
  - a 2^233-operation attack on ChaCha7,
  - a 2^137-operation attack on Salsa20/7,
  - a 2^127.5-operation attack on ChaCha6, and
  - a 2^32-operation attack on Salsa20/6.

Salsa20 prizes

In May 2005 I announced that, at the end of 2005, I would award a $1000 prize for the public Salsa20 cryptanalysis that I considered most interesting. I didn't make any promises regarding what I'd find interesting, but I posted the following guidelines:

- Breaking larger r's wins bonus points. ``Breaking'' is defined by being faster than brute force *for the same cost of cryptanalytic hardware*.
- Extra speed wins bonus points.
- Early publication wins bonus points. Don't hold back!
- New ideas win bonus points.

I awarded the prize to Paul Crowley for his paper ``Truncated differential cryptanalysis of five rounds of Salsa20.''

Salsa20 formalities

My submission of Salsa20 to the ECRYPT Stream Cipher project (eSTREAM) included the following formalities:

- Salsa20 cover sheet.
- Salsa20 robustness statement.
- Salsa20 intellectual-property statement.

- CD packing list.