ChaCha, a variant of Salsa20

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

Abstract. ChaCha8 is a 256-bit stream cipher based on the 8-round cipher Salsa20/8. The changes from Salsa20/8 to ChaCha8 are designed to improve diffusion per round, conjecturally increasing resistance to cryptanalysis, while preserving—and often improving—time per round. ChaCha12 and ChaCha20 are analogous modifications of the 12-round and 20-round ciphers Salsa20/12 and Salsa20/20. This paper presents the ChaCha family and explains the differences between Salsa20 and ChaCha.

1 Introduction

1.1 Background

The Salsa20/20 stream cipher expands a 256-bit key into 2^{64} randomly accessible streams, each containing 2^{64} randomly accessible 64-byte blocks. Salsa20/20 is a more conservative design than AES, and the community seems to have rapidly gained confidence in the security of the cipher. See [2, Section 5] for a summary of third-party cryptanalysis.

Salsa20/20 is consistently faster than AES. I recommend Salsa20/20 for encryption in typical cryptographic applications. The Salsa20 family also includes reduced-round ciphers—the 12-round cipher Salsa20/12 and the 8-round cipher Salsa20/8—aimed at users who value speed more highly than confidence. See [2, Table 1.1] for a summary of the speeds of Salsa20/8, Salsa20/12, and Salsa20/20.

1.2 Contributions

This paper introduces the ChaCha family of stream ciphers, a variant of the Salsa20 family. ChaCha follows the same basic design principles as Salsa20, but I changed some of the details, most importantly to increase the amount of diffusion per round. I speculate that the minimum number of secure rounds for ChaCha is smaller (and not larger!) than the minimum number of secure rounds for Salsa20.

* Permanent ID of this document: 4027b5256e17b9796842e6d0f68b0b5e. Date of this document: 2008.01.20. This work was supported by the National Science Foundation under grant ITR–0716498.

Computer				Cycles/byte					
				8 rounds		12 rounds		20 rounds	
Arch	MHz	CPU	model	Salsa20	ChaCha	Salsa20	ChaCha	Salsa20	ChaCha
ppc32	533	PowerPC G4	7410	1.99	1.86	2.74	2.61	4.24	4.13
amd64	2137	Core 2 Duo	6f6	1.87	1.87	2.53	2.54	3.90	3.95
ppc64	2000	PowerPC G5	970	3.24	3.06	4.74	4.57	7.81	7.58
amd64	2000	Athlon 64 X2	15,75,2	3.47	3.29	4.86	4.61	7.64	7.23
amd64	3000	Pentium D	f64	5.39	3.87	7.16	5.27	10.65	8.23
x86	1300	Pentium M	695	5.30	4.88	7.44	7.06	11.70	11.08
x86	900	Athlon	622	4.62	5.36	6.44	7.58	10.04	12.21
sparc	1050	UltraSPARC IV		6.65	6.60	9.17	9.17	14.34	14.29
x86	3200	Pentium D	f47	7.13	6.75	9.77	9.33	14.33	14.27
x86	1900	Pentium 4	f12	5.41	7.08	8.21	9.72	11.74	15.03

Table 1.3. Salsa20 and ChaCha software speeds for long streams. Sorted by ChaCha8 column.

This extra diffusion does not come at the expense of extra operations. A ChaCha round has 16 additions and 16 xors and 16 constant-distance rotations of 32-bit words, just like a Salsa20 round. Furthermore, ChaCha has the same levels of parallelism and vectorizability as Salsa20, and saves one of the 17 registers used by a "natural" Salsa20 implementation. So it is reasonable to guess that a ChaCha round can achieve the same software speed as a Salsa20 round—and even better speed than a Salsa20 round on some platforms. Consequently, if ChaCha has the same minimum number of secure rounds as Salsa20, then ChaCha will provide better overall speed than Salsa20 for the same level of security.

Of course, performance should be measured, not guessed! I wrote and posted new public-domain software for ChaCha, and timed that software, along with the fastest available Salsa20 software, on several computers, using the latest version (20080120) of the eSTREAM benchmarking framework. Table 1.3 shows the results. Compared to the Salsa20/8 software, the ChaCha8 software is

- the same speed on a 64-bit amd64-architecture Core 2 (6f6),
- the same speed on a 64-bit sparc-architecture UltraSPARC IV,
- 5% faster on a 64-bit amd64-architecture Athlon 64 (15,75,2),
- 5% faster on a 32-bit x86-architecture Pentium D (f47),
- 5% faster on a 64-bit ppc64-architecture PowerPC G5 (750),
- 6% faster on a 32-bit ppc32-architecture PowerPC G4 (7410),
- 8% faster on a 32-bit x86-architecture Pentium M (695), and
- 28% faster on a 64-bit amd64-architecture Pentium D (f64).

It is 16% slower on a 32-bit x86-architecture Athlon (622) and 31% slower on a 32-bit x86-architecture Pentium 4 (f12), but I expect these numbers to change with further tuning.

This paper assumes that the reader is familiar with Salsa20, and focuses on the changes from Salsa20 to ChaCha. Specifically, Section 2 compares Salsa20

quarter-rounds to ChaCha quarter-rounds, and Section 3 compares the Salsa20 matrix to the ChaCha matrix.

2 The quarter-round

2.1 Review of the Salsa20 quarter-round

Salsa20 invertibly updates 4 32-bit state words `a`, `b`, `c`, `d` as follows:

```
b ^= (a+d) <<< 7;
c ^= (b+a) <<< 9;
d ^= (c+b) <<< 13;
a ^= (d+c) <<< 18;
```

This code is expressed in a common extension of the C programming language: `^` means xor, `+` means addition modulo 2^{32}, and `<<<`b means b-bit rotation of a 32-bit integer towards high bits.

Salsa20 actually has 16 state words, but it operates on only 4 words at a time. The communication cost of accessing 16 state words is troublesome on many platforms; Salsa20 reduces the access frequency by performing several operations on 4 words before continuing on to the next 4 words.

2.2 The ChaCha quarter-round

ChaCha, like Salsa20, uses 4 additions and 4 xors and 4 rotations to invertibly update 4 32-bit state words. However, ChaCha applies the operations in a different order, and in particular updates each word *twice* rather than once. Specifically, ChaCha updates `a`, `b`, `c`, `d` as follows:

```
a += b; d ^= a; d <<<= 16;
c += d; b ^= c; b <<<= 12;
a += b; d ^= a; d <<<= 8;
c += d; b ^= c; b <<<= 7;
```

Obviously the ChaCha quarter-round, unlike the Salsa20 quarter-round, gives each input word a chance to affect each output word. Less obvious is that the ChaCha quarter-round diffuses changes through *bits* more quickly than the Salsa20 quarter-round. One can see this by tracing every possible 1-bit input difference: in the absence of carries, the ChaCha quarter-round changes (on average) 12.5 output bits, while the Salsa20 quarter-round changes 8 output bits.

This double-speed update is the most important difference between ChaCha and Salsa20. After I announced this idea, Aumasson, Fischer, Khazaei, Meier, and Rechberger in [1] considered a "ChaCha" cipher obtained by applying this idea to Salsa20, and concluded that their state-of-the-art attack broke one fewer round of "ChaCha" than of Salsa20. I speculate that ChaCha has similar resistance to "ChaCha" against the attack, but of course this has to be checked carefully.

The above code also shows a much less important difference between ChaCha and Salsa20: I changed the rotation distances $7, 9, 13, 18$ to $16, 12, 8, 7$. The difference in security appears to be negligible: $7, 9, 13, 18$ appears marginally better in some diffusion measures, and $16, 12, 8, 7$ appears marginally better in others, but the margins are tiny, far smaller than the presumed inaccuracy of the diffusion measures as predictors of security. The change boosts speed slightly on some platforms while making no difference on other platforms.

3 The matrix

3.1 Review of the Salsa20 matrix

Salsa20 puts its 4 attacker-controlled input words (nonce and block counter), 8 key words, and 4 constant words into a 4×4 matrix:

constant	key	key	key
key	constant	input	input
input	input	constant	key
key	key	key	constant

Salsa20/r invertibly transforms the matrix through r rounds. It then adds the result to the original matrix to obtain a 16-word (64-byte) output block.

Several Salsa20 software implementations (my XMM implementation for the Pentium 4, for example, and Matthijs van Duin's AltiVec implementation for the PowerPC G4) internally permute the matrix, shifting the second column up one row, shifting the third column up two rows, and shifting the fourth column up three rows. The permuted initial matrix is as follows:

constant	constant	constant	constant
key	input	key	key
input	key	key	input
key	key	input	key

From this permuted perspective, the first round of Salsa20 applies a quarter-round to each column in parallel, modifying the second row, then the third row, then the fourth row, then the first row. If rows of the matrix are stored in vectors then these 4 quarter-rounds consist entirely of SIMD operations on the vectors.

The second round of Salsa20 applies a quarter-round to each northeast diagonal in parallel, modifying the fourth row, then the third row, then the second row, then the first row. Appropriate rotations of the entries in each row turn the diagonals into columns, again allowing SIMD operations at very low cost.

The third round is like the first; the fourth round is like the second; and so on. The matrix permutation needs to be undone before a block is output.

3.2 The ChaCha matrix

ChaCha$r$, like Salsa20/r, builds a 4×4 matrix, invertibly transforms the matrix through r rounds, and adds the result to the original matrix to obtain a 16-word (64-byte) output block.

There are three differences in the details. First, ChaCha permutes the order of words in the output block to match the permutation described above. This has no effect on security; it saves time on SIMD platforms; it makes no difference in speed on other platforms.

Second, ChaCha builds the initial matrix with all attacker-controlled input words at the bottom:

constant	constant	constant	constant
key	key	key	key
key	key	key	key
input	input	input	input

The key words are simply copied in order; the input words are the block counter followed by the nonce; the constants are the same as in Salsa20. The first round of ChaCha adds keys into constants, then xors the results into inputs, then rotates, then adds the results into keys, etc. (Tangential note: Modifying constants first is helpful for compression functions built on the same core.)

I chose the positions of inputs in Salsa20 so that I could easily see which words were affected by small changes to the inputs. I think that a word-level trail-width analysis would take much more work for ChaCha than for Salsa20, in part because of cancellations in the modified quarter-round and in part because of the modified input positions. On the other hand, I think that the spread of positions in Salsa20 gave unnecessary flexibility to attackers, and I don't see any evidence that it actually improves security. A word-level trail-width analysis is much less interesting than the type of bit-level analysis carried out in, e.g., [1].

Third, ChaCha sweeps through rows in the same order in every round. The first round modifies first, fourth, third, second, first, fourth, third, second along columns, and the second round modifies first, fourth, third, second, first, fourth, third, second along southeast diagonals:

```
QUARTERROUND( x0, x4, x8,x12)
QUARTERROUND( x1, x5, x9,x13)
QUARTERROUND( x2, x6,x10,x14)
QUARTERROUND( x3, x7,x11,x15)
QUARTERROUND( x0, x5,x10,x15)
QUARTERROUND( x1, x6,x11,x12)
QUARTERROUND( x2, x7, x8,x13)
QUARTERROUND( x3, x4, x9,x14)
```

The four quarter-round words are always in top-to-bottom order in the matrix. I speculate that this improves diffusion slightly.

References

1. Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, Christian Rechberger, *New features of Latin dances: analysis of Salsa, ChaCha, and Rumba* (2007). URL: `http://eprint.iacr.org/2007/472`. Citations in this document: §2.2, §3.2.
2. Daniel J. Bernstein, *The Salsa20 family of stream ciphers* (2007). URL: `http://cr.yp.to/papers.html#salsafamily`. Citations in this document: §1.1, §1.1.