SALSA20 : Implementation

Indian Statistical Institute, Kolkata

April 19^{th}, 2024

Contents

Background

Background

- Use of network based applications are growing at a rapid speed.
- *Pseudo-random* numbers are at the core of any network security application.
- *Osvik*, *Shamir* and *Tromer* used cache-timing attacks to steal *AES* keys from a Linux disk-encryption device.
- Serious key collision & leakage in the hardware implementation of AES ciphers was found.
- *A.Shamir*, *I.Mantin* and *S.fluhrer* revealed weaknesses in key scheduling algorithm of *RC*4.
- Cipher should be **"GENERIC"** compatible on both Hardware and Software platforms.

This way *Salsa*20 came to the picture

Rise of Salsa20

History

- **eStream** : The *Ecrypt* Stream Cipher Project, called for submissions of stream ciphers in *November 2004*.
- **Salsa20** : Family of $256 - bit$ stream ciphers designed in 2005 and submitted to *eStream* by *Daniel J. Bernstein*.
- Salsa20 progressed to the *third round* of *eSTREAM* without any further changes.
- The final *eStream* portfolio, containing four software stream ciphers and four hardware stream ciphers, were announced in April *2008*.
- It is not **patented**, and Bernstein has written several public domain implementations optimized for common architectures.

Overview

- Long chain of simple operations, rather than a shorter chain of complicated operations.
- This software-oriented stream cipher is built on a *Pseudorandom* function based on *ADD–ROTATE–XOR* (ARX) operations.
- It undergoes the following set of operations :
  - $\mathbf{32-bit}$ Addition producing the sum $a + b\ mod 2^{32}$ of two $32-bit$ words $a,\ b$.
  - $\mathbf{32-bit}$ Exclusive-Or, producing the $a \oplus b$ of two $32-bit$ words $a,\ b$.
  - Constant-distance **32**-bit rotation, producing the rotation $a \lll b$ of a $32-bit$ word a by b bits to the **left** (where b is constant).

Agility of Salsa20

				Cycles/byte					
			Salsa20	Salsa20/8		Salsa20/12		Salsa20/20	
Arch	MHz	Machine	software	long	576	long	576	long	576
amd64	3000	Xeon 5160 (6f6)	amd64-xmm6	1.88	2.07	2.80	3.25	3.93	4.25
amd64	2137	Core 2 Duo (6f6)	amd64-xmm6	1.88	2.07	2.57	2.80	3.91	4.33
ppc32	533	PowerPC G4 7410	ppc-altivec	1.99	2.14	2.74	2.88	4.24	4.39
x86	2137	Core 2 Duo (6f6)	x86-xmm5	2.06	2.28	2.80	3.15	4.32	4.70
amd64	2000	Athlon 64 X2 (15,75,2)	amd64-3	3.47	3.65	4.86	5.04	7.64	7.84
ppc64	2000	PowerPC G5 970	ppc-altivec	3.28	3.48	4.83	4.87	7.82	8.04
amd64	2391	Opteron (f5a)	amd64-3	3.78	3.96	5.33	5.51	8.42	8.62
amd64	2192	Opteron (f58)	amd64-3	3.82	4.18	5.35	5.73	8.42	8.78
x86	2000	Athlon 64 X2 (15,75,2)	x86-1	4.50	4.78	6.27	6.55	9.80	10.07
x86	900	Athlon (622)	x86-athlon	4.61	4.84	6.44	6.65	10.04	10.24
ppc64	1452	POWER4	merged	6.83	7.00	8.35	8.51	11.29	11.47
hppa	1000	PA-RISC 8900	merged	5.82	5.97	7.68	7.85	11.39	11.56
amd64	3000	Pentium D (f64)	amd64-xmm6	5.38	5.87	7.19	7.84	10.69	11.73
x86	1300	Pentium M (695)	x86-xmm5	5.30	5.53	7.44	7.70	11.70	11.98
x86	3000	Xeon (f26)	x86-xmm5	5.30	5.86	7.41	8.21	11.64	12.55
x86	3200	Xeon (f25)	x86-xmm5	5.30	5.84	7.40	8.15	11.63	12.59
x86	2800	Xeon (f29)	x86-xmm5	5.33	5.95	7.44	8.20	11.67	12.65
x86	3000	Pentium 4 (f41)	x86-xmm5	5.76	6.92	8.12	9.33	11.84	13.40
x86	1400	Pentium III (6b1)	x86-mmx	6.37	6.79	8.88	9.29	13.88	14.29
sparc	1050	UltraSPARC IV	sparc	6.65	6.76	9.21	9.33	14.34	14.45
x86	3200	Pentium D (f47)	x86-athlon	7.13	7.66	9.90	10.31	15.29	15.94
ia64	1500	Itanium II	merged	8.49	8.87	12.42	12.62	18.07	18.27
ia64	1400	Itanium II	merged	8.28	8.65	12.56	12.76	18.21	18.40

Figure: Speed on different platforms

Where we have :

- $\textit{Cycles / byte} = \dfrac{\textit{cycles per Sec}}{\textit{speed}}$
- $\textit{speed} = \dfrac{\textit{data size}}{\textit{time}}$

Speed

- *Salsa*20/20 runs at 3.93 cycles/byte for long streams. Whereas the fastest *AES* takes 9.2 cycles/byte for just 10 rounds of long stream.
- *Salsa*20 runs at only 5.14 cycles/byte on a *Qualcomm Snapdragon S4 processor*, compared to 18.62 cycles/byte for $AES - 128$ in counter mode.
- 3 cycles/byte for *Cryptography* on *Core 2 Salsa*20/12 rounds takes 2.8 cycles/byte, one can afford at most 3 rounds of *AES* for any security at all.

Initial State of Salsa20

A **word** is an element of $\{0, 1, \ldots, 2^{32} - 1\}$
The internal state is made of **sixteen** 32-bit words arranged in a 4×4 matrix. The initial state contains **eight** words of *key* , **two** words of *stream position* , **two** words of *nonce* (essentially additional stream position bits), and **four** *fixed words*:

"expa"	key	key	key
key	"nd 3"	nonce	nonce
block	block	"2-by"	key
key	key	key	"te k"

Table: *Salsa*20's initial state (**IS**) for 32 byte keys.

The Quarterround function

If x is a 4-word sequence then $quarterround(x)$ is also a 4-word sequence.

Definition

If $x = (x_0, x_1, x_2, x_3)$ then $quarterround(x) = (y_0, y_1, y_2, y_3)$ where :

$$y_1 = x_1 \oplus ((x_0 + x_3) \lll 7)$$

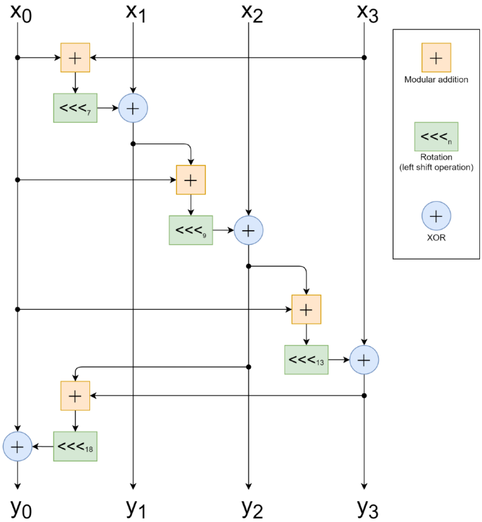
$$y_2 = x_2 \oplus ((x_1 + x_0) \lll 9)$$

$$y_3 = x_3 \oplus ((x_2 + x_1) \lll 13)$$

$$y_0 = x_0 \oplus ((x_3 + x_2) \lll 18)$$

N.B. : Each modification is **invertible**, so the entire function is **invertible**.

Diagram



quarterround(0x00000001, 0x00000000, 0x00000000, 0x00000000) = (0x08008145, 0x00000080, 0x00010200, 0x20500000)

The Rowround function

If y is a 16-word sequence then $rowround(y)$ is a 16-word sequence.

Definition

If $y = (y_0, y_1, y_2, y_3, \ldots, y_{15})$ then
$rowround(y) = (z_0, z_1, z_2, z_3, \ldots, z_{15})$, where

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

The Columnround function

If x is a 16-word sequence then $columnround(x)$ is a 16-word sequence.

Definition

If $x = (x_0, x_1, x_2, x_3, \dots, x_{15})$ then $columnround(x) = (y_0, y_1, y_2, y_3, \dots, y_{15})$ where,

$$(y_0, y_4, y_8, y_{12}) = quarterround(x_0, x_4, x_8, x_{12})$$

$$(y_5, y_9, y_13, y_1) = quarterround(x_5, x_9, x_{13}, x_1)$$

$$(y_{10}, y_{14}, y_2, y_6) = quarterround(x_{10}, x_{14}, x_2, x_6)$$

$$(y_{10}, y_{14}, y_2, y_6) = quarterround(x_{10}, x_{14}, x_2, x_6)$$

The Doubleround function

If x is a 16-word sequence then $doubleround(x)$ is a 16-word sequence.

Definition

A *doubleround* function is the composition of *columnround* followed by the *rowround* function, So we have

$$doubleround(x) \;=\; rowround(columnround(x))$$

The Littleendian function

If b is a 4-byte sequence then $littleendian(x)$ is a word.

Definition

If b $= (b_0, b_1, b_2, b_3)$ then we have,

$$littleendian(b) \ = \ b_0 \ + \ 2^8 \cdot b_1 \ + \ 2^{16} \cdot b_2 \ + \ 2^{24} \cdot b_3$$

Example

$$littleendian(255,\ 250,\ 126,\ 96) \ = \ 0x\ \underline{60}\ \underline{7e}\ \underline{fa}\ \underline{ff}$$

$$(255)_{2^8} \ = \ (\underline{1111}\ \underline{1111})_2 \ = \ (ff)_{16}$$

$$(250)_{2^8} \ = \ (\underline{1111}\ \underline{1100})_2 \ = \ (fa)_{16}$$

$$(126)_{2^8} \ = \ (\underline{0111}\ \underline{1110})_2 \ = \ (7e)_{16}$$

$$(96)_{2^8} \ = \ (\underline{0110}\ \underline{0000})_2 \ = \ (60)_{16}$$

The Salsa20 Expansion function

If k is a 32-byte and n is a 16-byte sequence then $Salsa20_k(n)$ is a 64-byte sequence.

Definition

Define $\sigma_0 = (101, 120, 112, 97)$, $\sigma_1 = (110, 100, 32, 51)$, $\sigma_2 = (50, 45, 98, 121)$, and $\sigma_3 = (116, 101, 32, 107)$. If k_0, k_1, n are 16-byte sequences then we have :

$$Salsa20_{k_0,k_1}(n) = Salsa20(\sigma_0, k_0, \sigma_1, n, \sigma_2, k_1, \sigma_3)$$

N.B. : Expansion refers to the expansion of (k, n) into $Salsa20_k(n)$. The constants $\sigma_0\ \sigma_1\ \sigma_2\ \sigma_3$ is "*expand* $32 -$ *byte k*" in **ASCII**.

The *Salsa*20 Hash function

If x is a 64-byte sequence then $Salsa20(x)$ is a 64-byte sequence.

Definition

$$Salsa20(x) \; = \; x \; + \; doubleround^{10}(x)$$

Where each 4-byte sequence is viewed as a word in $little - endian$ form. Starting with $x \; = \; (x[0], \; x[1], \; \ldots, \; x[63])$. Lets, define

$$x_0 \; = \; \text{littleendian}(x[0], \; x[1], \; x[2], \; x[3])$$

$$\vdots$$

$$x_{15} \; = \; \text{littleendian}(x[60], \; x[61], \; x[62], \; x[63])$$

Define $(z_0, \; z_1, \; \ldots, \; z_{15}) \; = \; doubleround^{10}(x_0, \; x_1, \; \ldots, \; x_{15})$
Then $Salsa20(x)$ is the concatenation of :

$$littleendian^{-1}(z_0 + x_0) \,||\, littleendian^{-1}(z_1 + x_1) \,||\ldots||\, littleendian^{-1}(z_{15} + x_{15})$$

The Salsa20 Encryption function

- Let k be a 32-byte sequence of key, v be a 8-byte sequence of nonce and m be an l-byte sequence of input, (for some $l \in \{0, 1, \ldots, 2^{70}\}$).
- The *Salsa*20 *Encryption/Decryption* of *Input* is denoted by $\mathbf{Salsa20_k(v)} \oplus \mathbf{m}$, is an l-byte sequence.

Definition

$Salsa20_k(v)$ is a 2^{70}-byte sequence

$$Salsa20_k(v, 0) \ || \ Salsa20_k(v, 1) \ || \ \ldots \ || \ Salsa20_k(v, 2^{64} - 1)$$

$Salsa20_k(v) \oplus m$ implicitly truncates $Salsa20_k(v)$ to the same length as m. In other words : (where $c[i] = m[i] \oplus Salsa20_k(v, \lfloor \frac{i}{64} \rfloor)[i \ mod 64])$

$$Salsa20_k(v) \oplus (m[0], m[1], \ldots, m[l-1]) = (c[0], c[1], \ldots, c[l-1])$$

Brute-force attacks

The Quarterround (QR) function takes a 128 bit binary number. The total possible combinations of inputs are 2^{128}.

A complete search would thus take about: (Assuming we can know the cryptographic nonce used)

$$\frac{2^{128} QR}{10000 QR/s} = 3.4 \cdot 10^{34} \text{ seconds} \approx 10^{27} \text{ years}$$

A complete search of the 256 bit key space would take:

$$\frac{2^{256} runs}{14 runs/s} = 2.4 \cdot 10^{76} \text{ seconds} \approx 10^{68} \text{ years}$$

Hamming distance differential analysis

For an *Encryption function* $f : f(X) \rightarrow Y$

If $HD(X, X') = n$, then $HD(Y, Y') \stackrel{?}{=} m$

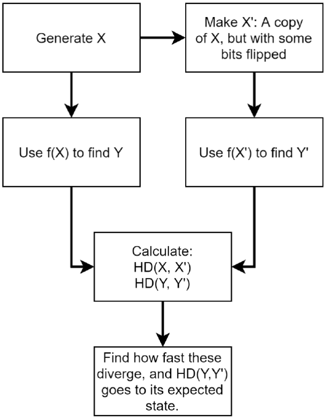


Figure: Hamming Distance

- If the algorithm has a good avalanche effect, we would expect the *HD* to be about the same as the *HD* between two random values: About half the bits.
- If P and Q are two random binary numbers of length n, we would expect: $HD(Q, P) \approx \frac{n}{2}$

Differential analysis on QR function

As $QR(X) \rightarrow y$ and we get X' by flipping some random bits of X. As n increases, m tend towards the expected equilibrium of half the length of Y. Analysis based on measuring $HD(QR(x'),\ y)$:

- The bit flipping is given by x-axis. i.e. n times.
- The HD between 2 values are given by y-axis.
- Legend of the Plot :
  0. $HD(X, Y)$
  1. Convergence point of the Hamming distance between two random values
  2. $HD(Y, Y')$
  3. $HD(X, X')$

     Assuming $HD(X, X') < \frac{key\_size}{8}$, $HD(X,\ X')$ should be easily distinguishable.

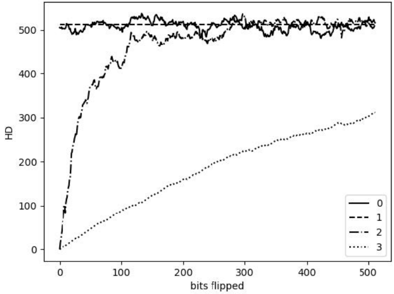


Figure: Flipping of random bits in X

Contd.

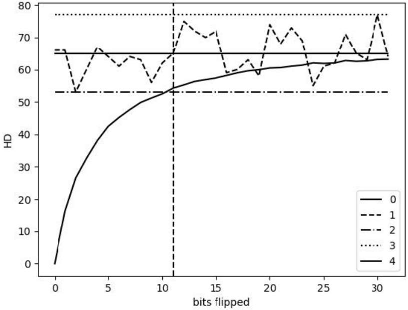


Figure: Averaging of effects on Y when bits are flipped in X.

Legend of the Plot :

0. $HD(X', Y')$ as n bits in X' are flipped.
1. $HD(X, Y)$ of random Xs
2. *Minimum expected difference* between two random Xs
3. *Maximum expected difference* between two random Xs
4. *Expected average difference* between two random Xs

The vertical line is where the bits flipped needed for the Hamming distance to be within the range of expected random distances

Analysis on Salsa20's PRG

- Line 1, shows how, $HD(input_{original},\ input_{next})$ is roughly equal to 1 per flipped bits.
- Line 2, is the expected value for random inputs and outputs.
- Line 0, seems to be no correlation or pattern between the amounts of bits flipped in the $input(n)$ and the HD between the two values.

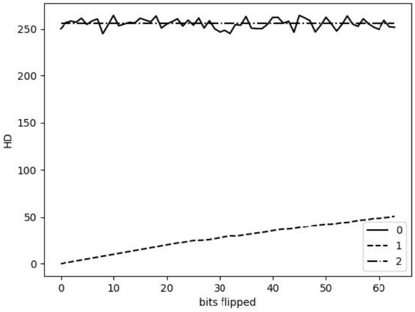


Figure: Averaging of effects on the *PRG* output when bits are flipped in key

The Invincible Salsa20

*Salsa*20 is highly resistant and secure against all the well known attacks :

- *Algebraic* attack
- *Weak* – *Key* attack
- *Equivalent* – *Key* attack
- *Related* – *Key* attack
- *Correlation power analysis*
- *Context aggregation* network analysis

Conclusion

After going through all this discussions, we conclude with the following points :

- *SALSA*20 is faster and efficient as compared to *AES*.
- Been secure to both *KPA* and *CPA*.
- Efficient in both *software* and *hardware*.
- *Brute force* attack are not easily implementable.

References

- D. J. Bernstein, *"The Salsa20 Family of Stream Ciphers"*, New Stream Cipher Des., pp. 84–97, 2008.
- D. J. Bernstein, *"Salsa20 specification"*, eSTREAM Proj. algorithm Descr., pp. 2–10, 2005.
- "Calculating cycles per byte." Stream cipher - Calculating cycles per byte - Cryptography Stack Exchange. N.p., 2 Oct. 2012. Web. 3 Mar. 2017. `http://crypto.stackexchange.com/questions/3943/calculating-cycles-per-byte`
- "How secure is Salsa20?" Algorithm design - How secure is Salsa20? - Cryptography Stack Exchange. N.p., 8 Oct. 2016. Web. 10 Mar. 2017. `http://crypto.stackexchange.com/questions/40542/how-secure-is-salsa20/40543`