Salsa and ChaCha20 Algorithm

Abeer D. Salman
University of Anbar
21 PUBLICATIONS **24** CITATIONS

Salsa and ChaCha20 Algorithm

By

Dr. Abeer Dawood Salman

Supervised by

Prof.Dr.Hala Bahjat Abdulwahab

A thesis submitted to the Department of Computer Science at the University of Technology in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science

This lecture is part of my PhD thesis, I published it for anyone who needs details on these algorithms because there are no adequate explanations in other research.

We developed chacha20 algorithm, which is explained in the research cited below.

Salman, Abeer Dawood, and Hala Bahjat Abdulwahab. "Dynamic, Secure, and Invariant Watermarking System for Multiview Plus Depth Video." 2019 First International Conference of Computer and Applied Sciences (CAS). IEEE, 2019.
https://ieeexplore.ieee.org/abstract/document/9075651

for more information don't hesitate to contact me:

alnuaimiabeer@gmail.com
abeer.dawood@uoanbar.edu.iq

1. Stream cipher Technique

The common goal of such techniques is generating the infinite random key-stream to be used in cryptographic. Stream cipher cryptography deals with one bit or byte of key-stream to encrypt one bit or byte of the input data at a time, and it avoids dividing the input data into blocks. There are some stream cipher algorithms like One Time Pad, RC4, Salsa, and ChaCha etc. The stream cipher has the speed for transforming the data over the internet compared with the block cipher. So, it widely used for transmitting multimedia data.

In this section, we will focus on the Salsa and ChaCha algorithms, as well as Chaos theory, that will be used for creating a random sequence that was utilized to design a security watermark system for MVD video as discussed in chapters four and five.

1.1 Salsa Algorithm

Bernstein designed Salsa20 in 2005. It treats 256-bit stream cipher. Initially, the Salsa algorithm is introduced as a cipher with 20 rounds (Salsa20), then it reduced to the 12 round and finally to 8 rounds (Salsa20/12 and Salsa20/8) [65]. For applications that prefer the speed than the confidence, Salsa20/8 can be applied where it considers the fastest stream ciphers available [66]. A brute-force attack is a famous attack against Salsa20/20 [67]. The hash function is the core of the Salsa20 algorithm, it used in the counter mode. Salsa20 hashing the input parameters (key, nonce, constants, and block number), and finally apply XOR operation between the result of hashing operation with a 64-byte block of plaintext. Figure 3.7 depicts the block diagram of the Salsa20 algorithm. Salsa 20/8 and Salsa 20/12 operate exactly like the original Salsa20, except they perform 4 or 6 "double rounds function", respectively rather than of 10 as in salsa 20 [68]. The input to the function is [67] [69]:

1. ***secret key*** k = (k0, k1,. . . , k7), a 256-bit (32 byte), or 16 byte
2. ***nonce*** **(**unique message number**)** v = (v0, v1), 64-bit (8 byte)
3. ***Block number***, 8 bytes which values change from 0 to 2^{64}-1.***counter*** t = (t0, t1). Block number increased by one each call to the expansion function.
4. ***Constants***, c = (c0, c1, c2, c3), 16 byte, c= (0x61707865 0x3320646e, 0x79622d32, 0x6b206574).

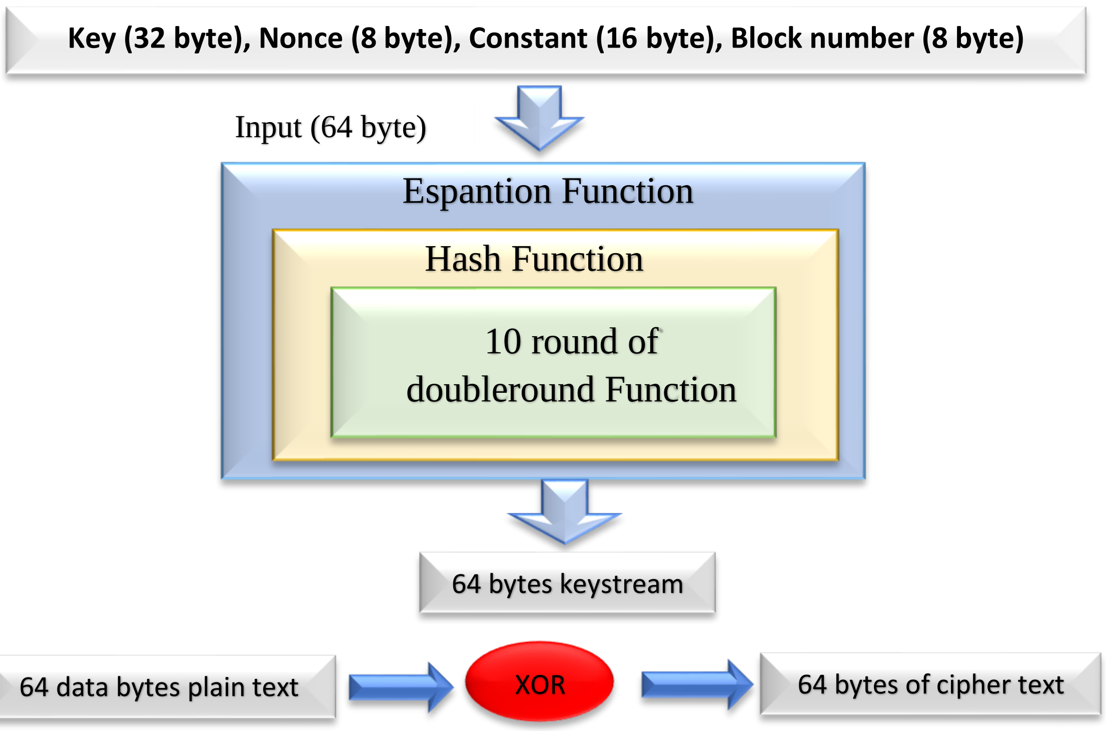


Figure 3.7: General structure of Salsa algorithm

The input acts as a 4 × 4 matric written as [66] [69]:

$$X=\begin{array}{|c|c|c|c|}\hline x_0 & x_1 & x_2 & x_3 \\ \hline x_4 & x_5 & x_6 & x_7 \\ \hline x_8 & x_9 & x_{10} & x_{11} \\ \hline x_{12} & x_{13} & x_{14} & x_{15} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline c_0 & k_0 & k_1 & k_2 \\ \hline k_3 & c_1 & v_0 & v_1 \\ \hline t_0 & t_1 & c_2 & k_4 \\ \hline k_5 & k_6 & k_7 & c_3 \\ \hline \end{array}$$

Each block has a specified key, nonce, and block number that changed continuously for other blocks [67]. Key-stream generation includes the following operations [69].

1. **Salsa20 Expansion Function:** The Salsa20 Expansion Function accepts 64 bytes blocks and output another 64 bytes. The Salsa20 Expansion function is specified using the Salsa20 Hash function as follows:

$$\textbf{\textit{Salsa20Expansion = Salsa20Hash}}\ (c_0, k_0{:}k_3, c_1, v_0, v_1, t_0, t_1\ c_2, k_4{:}k_7, c_3)$$

2. **Salsa20 Hash Function** deals with 64 bytes sequence as input and output. Firstly, the hash function produces 16 words, and each word contains 4 bytes with values ranging from 0 to 2^{32}-1. If **input** = (b_0, b_1, b_2, ..., b_{63}) 64 bytes sequence, then, as follows, 16 words will be created:

$$w_0 = \text{little-endian}\ (b_0, b_1, b_2, b_3)$$
$$w_1 = \text{little-endian}\ (b_4, b_5, b_6, b_7)$$
$$[...]$$
$$w_{15} = \text{little-endian}\ (b_{60}, b_{61}, b_{62}, b_{63})$$

3. **Little-endian Function** adjust 4-byte sequence order. Consider **b** is a series of 4 bytes [b = (b0, b1, b2, b3)], and then the little -endian function is computed:

$$\textit{little-endian}\ (b) = b_0 + 2^8 b_1 + 2^{16} b_2 + 2^{24} b_3$$

The Function Little-endian is invertible. It just shifts the byte order in one word. Then, 10 iterations of the Double Round Function update all 16 words:

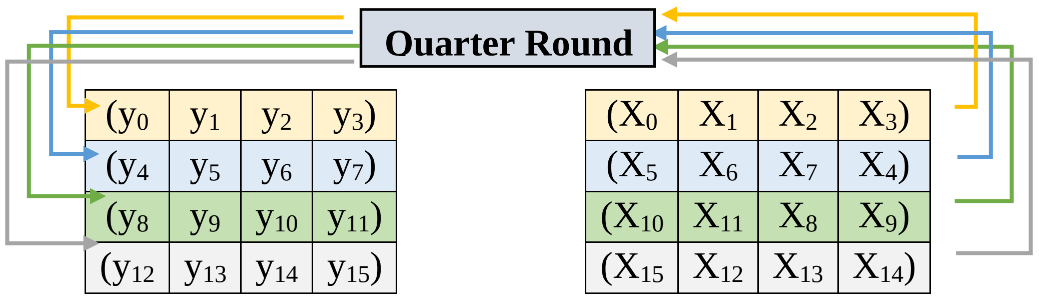
$$X(x_0, x_1, ..., x_{15}) = \textit{doubleround}^{10}(w_0, w_1, ..., w_{15})$$

4. **Double round Function** Gets 16 words as input, and outputs a series of 16 words. The Double Round Function is as follows:

$$\textit{doubleround}(x) = \textit{Row Round}(\textit{Column Round}(X))$$

5. **Row Round Function** accept 16 words data, transforms them and produces a series of 16 words (y_0,…,y_{16}). In even numbers of rounds, The non-linear process shall apply to rows (x0, x1, x2, x3), (x5, x6, x7, x4), (x10, x11, x8, x9), (x15, x12, x13, x14) as follow[69].

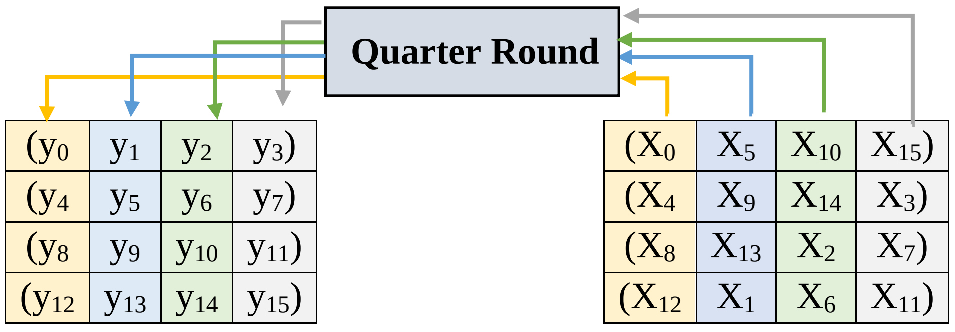
***Row Round* Y(y_0,…,y_{15})= Quarter Round X(x_0,…,x_{15})**



Quarter Round
(y_0 y_1 y_2 y_3)
(y_4 y_5 y_6 y_7)
(y_8 y_9 y_{10} y_{11})
(y_{12} y_{13} y_{14} y_{15})
(X_0 X_1 X_2 X_3)
(X_5 X_6 X_7 X_4)
(X_{10} X_{11} X_8 X_9)
(X_{15} X_{12} X_{13} X_{14})

6. **Column Rounds Function** occupy 16 words as input and introduces a 16-word sequence as output. This function operates on the words in a different order, the nonlinear operation is applied to columns (x0, x4, x8, x12), (x5, x9, x13, x1), (x10, x14, x2, x6), (x15, x3, x7, x11) as follow:

***Column Round* (y_0,…,y_{15})= Quarter Round X(x_0,…,x_{15})**



Quarter Round
(y_0 y_1 y_2 y_3)
(y_4 y_5 y_6 y_7)
(y_8 y_9 y_{10} y_{11})
(y_{12} y_{13} y_{14} y_{15})
(X_0 X_5 X_{10} X_{15})
(X_4 X_9 X_{14} X_3)
(X_8 X_{13} X_2 X_7)
(X_{12} X_1 X_6 X_{11})

7. The **Quarter Round Function:** this function receives 4 words as input and produced 4-word as output. If x is input: x = (x0, x1, x2, x3). the quarter round function implements the nonlinear transformation to convert the vector (x0, x1, x2, x3) to (y0, y1, y2, y3) as follows:

***(y0, y1, y2, y3)= Quarter Round*(**x_0, x_1, x_2 x_3**)**

$$y_1 = x_1 \textbf{ XOR } ((x_0 + x_3) <<< 7)$$

$$y_2 = x_2 \textbf{ XOR } ((y_1 + x_0) <<< 9)$$

$$y_3 = x_3 \textbf{ XOR } ((y_2 + y_1) <<< 13)$$

$$y_0 = x_0 \textbf{ XOR } ((y_3 + y_2) <<< 18)$$

Initially, x_1 changes to y_1, then, x_2 altered to y_2, then x_3 modified to y_3, and finally x_0 changes to y_0. Being that all the above modifications are invertible, Quarter Round Function is also invertible. The main operations of Salsa20 encryption function are:

- **Total of two words:** ($a + b \bmod 2^{32}$) of two 32-bit words. The consequence is a valid word that is 4 bytes long.
- **Exclusive-Or of two words**: (a XOR b), to perform XOR addition requires a bit-wise comparison of two words before performing XOR addition for each pair. The consequence is a proper word that is 4 bytes long
- **Rotation** the 32-bit word to a constant number, (a <<< b). Leftmost bits 'a' is rotated to rightmost of 'b' positions. The result of this operation is a word with 4-byte.

The modified 16 words above are added with the 16 words that entered as input to the salsa function and lastly Little-endian Function is implemented to change the result of the addition to new 64 bytes as follow:

$$\textit{Output} = \textit{little-endian}^{-1}(x_0+w_0) + \ldots + \textit{little-endian}^{-1}(x_{15}+w_{15})$$

To generated 64-byte cipher-text, the result of the Salsa20 expansion function XOR with the 64-bytes input plaintext. For the decryption process, the 64-byte cipher-text XOR with the 64-bytes key-stream that generated in the same way.

1.2 ChaCha20 algorithm

The ChaCha family was first proposed in 2008 by Bernstein [70] as a variant of the Salsa family. Because they use slightly better hash functions, they offer better security compared to the original Salsa20 cipher. The arrangement of the input data for the hash function enables more effective implementation of the algorithms. Following Google's implementation, ChaCha is acquisition popularity. ChaCha supports both 128- and 256-bit keys depending on the Salsa20/8 8-round cipher [66]. The goal of development ChaCha is to keep the same performance as the Salsa family while the diffusion at a single round is increased [70]. With some improvements, ChaCha follows the same basic design concepts as Salsa20. ChaCha algorithm has two versions ChaCha20 and ChaCha12 that are alteration of Salsa20/20 and Salsa20/12. ChaCha algorithm faster than the salsa algorithm with a minimum number of rounds for the same level of security [66]. The difference between the sala and ChaCha algorithm listed below:

1. The input of the ChaCha algorithm is that the 4x4 matrix includes 16 words arranged as follows: [70].

$$\mathbf{X}= \begin{array}{|c|c|c|c|}\hline \boldsymbol{x_0} & \boldsymbol{x_1} & \boldsymbol{x_2} & \boldsymbol{x_3} \\ \hline \boldsymbol{x_4} & \boldsymbol{x_5} & \boldsymbol{x_6} & \boldsymbol{x_7} \\ \hline \boldsymbol{x_8} & \boldsymbol{x_9} & \boldsymbol{x_{10}} & \boldsymbol{x_{11}} \\ \hline \boldsymbol{x_{12}} & \boldsymbol{x_{13}} & \boldsymbol{x_{14}} & \boldsymbol{x_{15}} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline \boldsymbol{c_0} & \boldsymbol{c_0} & \boldsymbol{c_1} & \boldsymbol{c_2} \\ \hline \boldsymbol{k_0} & \boldsymbol{k_1} & \boldsymbol{k_2} & \boldsymbol{k_3} \\ \hline \boldsymbol{k_4} & \boldsymbol{k_5} & \boldsymbol{k_6} & \boldsymbol{k_7} \\ \hline \boldsymbol{t_0} & \boldsymbol{t_1} & \boldsymbol{v_7} & \boldsymbol{v_1} \\ \hline \end{array}$$

2. X is put through a hash function that has the quarter-round function (QRF) as its main component. The source of non-linearity for the ChaCha stream cipher comes from using the QRF that contains addition modulo 2^{32}, rotation, and XOR operations [70]. These operations are executed in a different order. In the ChaCha algorithm, each word is updated twice instead once. Assume the inputs to the QRF are (a, b, c, and d) words, ChaCha will update a, b, c, d in the following way [69]:

   Quarter Round function:

   $$a=a+b,\ d=(a \text{ XOR } d) <\!<\!<16$$
   $$c=c+d,\ b=(c \text{ XOR } b) <\!<\!<12$$
   $$a=a+b,\ d=(a \text{ XOR } d) <\!<\!<8$$
   $$c=c+d,\ b=(c \text{ XOR } b) <\!<\!<7$$

3. Column Rounds Function is defined in a different manner; the nonlinear operation is only applicable to the columns [69].

- **Quarter Round** (x0, x4, x8,x12)
- **Quarter Round** (x1, x5, x9,x13)
- **Quarter Round** (x2, x6,x10,x14)
- **Quarter Round** (x3, x7,x11,x15)

X0	X1	X2	X3
X4	X5	X6	X7
X8	X9	X10	X11
X12	X13	X14	X15

4. Row Round Function: Here, nonlinear procedure is only applicable for the rows [69].

- **Quarter Round** (x0, x5,x10,x15)
- **Quarter Round** (x1, x6,x11,x12)
- **Quarter Round** (x2, x7, x8,x13)
- **Quarter Round** (x3, x4, x9,x14)

X0	X1	X2	X3
X4	X5	X6	X7
X8	X9	X10	X11
X12	X13	X14	X15

The ChaCha QFR offers every input word an opportunity to impact every output word, unlike in the Salsa20. It is also unclear that the ChaCha QRF diffuses quicker through bits than the QRF of Salsa20. The distance of rotation in ChaCha algorithm is changed from 7, 9, 13, 18 to 16, 12, 8, and 7. for security issues, 7, 9, 13, 18 looks marginally better in some diffusion measurements while 16, 12, 8, 7 looks marginally better in others [66]. Ciphers of the randomization stream usually use feedback shift registers (FSRs). The activity of the stream cipher group ChaCha and Salsa doesn't rely on FSRs; rather, their main reliance is on a hash function for pseudo-random key-stream generation [70]. Salsa20/20 is a more moderate design compared to AES, and this cipher appears to have acquired community interest regarding its security [66]. The Salsa20/20 20-round stream cipher is regularly speedier than AES and is advised for cryptographic applications by the designer [67].