

# CS 6530 Applied Cryptography

July-Nov 2025

## Introduction to Cryptography and Data Security

11<sup>th</sup> August 2025 – Session 1

Dr. Manikantan Srinivasan

(Material covered is based on Chapter 2 of  
[Understanding Cryptography – Second Edition](#),  
[Chapter 5 of Serious Cryptography – Second Edition](#) )

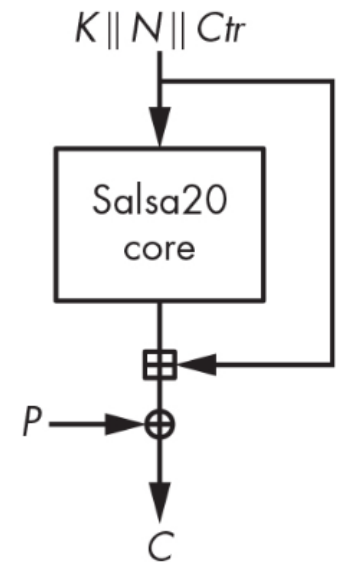
# Contents of Chapter 2

- ◆ • **Introduction to Stream Ciphers**
- ◆ • Random Number Generators (RNGs)
- ◆ • One-Time-Pad (OTP)
- ◆ • **Salsa20, ChaCha20**
- ◆ • Trivium

Salsa20

# Salsa20

- ◆ Salsa20 is a family of software-efficient stream ciphers
- ◆ Salsa20 developed by Daniel J Bernstein in 2005.
- ◆ The cipher uses a pseudo random function based on 32-bit additions, rotations and XOR operations. (*add-rotate-XOR* (ARX) ciphers). It is a counter based stream cipher as well.
- ◆ Salsa20/20 – original cipher has 20 rounds of quarter-round functions
- ◆ Salsa supports key lengths of 256 and 128 bits
- ◆ Core of Salsa20 is a function with 512-bit input and 512-bit output.
- ◆ Salsa20 core algorithm transforms a 512-bit block using a key (K), a nonce (N), and a counter value (Ctr). Salsa20 then adds the result to the original value of the block to produce a keystream block



Salsa20's encryption scheme for a 512-bit plaintext block

# Salsa20 – Quarter-round function

- ◆ Salsa20's core permutation uses a function called quarter-round (QR), which transforms four 32-bit words ( $a$ ,  $b$ ,  $c$ , and  $d$ ) as follows:

$$\begin{aligned}b &= b \oplus [(a + d) \lll 7] \\c &= c \oplus [(b + a) \lll 9] \\d &= d \oplus [(c + b) \lll 13] \\a &= a \oplus [(d + c) \lll 18]\end{aligned}$$

- ◆ Another way / definition can be as follows:

If  $y = (y_0, y_1, y_2, y_3)$  then  $\text{quarterround}(y) = (z_0, z_1, z_2, z_3)$  where

$$\begin{aligned}z_1 &= y_1 \oplus ((y_0 + y_3) \lll 7), \\z_2 &= y_2 \oplus ((z_1 + y_0) \lll 9), \\z_3 &= y_3 \oplus ((z_2 + z_1) \lll 13), \\z_0 &= y_0 \oplus ((z_3 + z_2) \lll 18).\end{aligned}$$

# Transforming Salsa20's 512-Bit State

- ◆ Salsa20's core permutation transforms a 512-bit internal state viewed as a  $4 \times 4$  array of 32-bit words.
- ◆ The initial state, using a key (K) of eight words (256 bits), a nonce (V) of two words (64 bits), a counter (T) of two words (64 bits), and C - four fixed constant words (128 bits) that are identical for each encryption/decryption and all blocks.
- ◆ The Constant C is given by the ASCII encoded string "expand 32-byte k"
- ◆ If 128-bit key consisting of 4 words is used, then same key is concatenated to form 8 words.

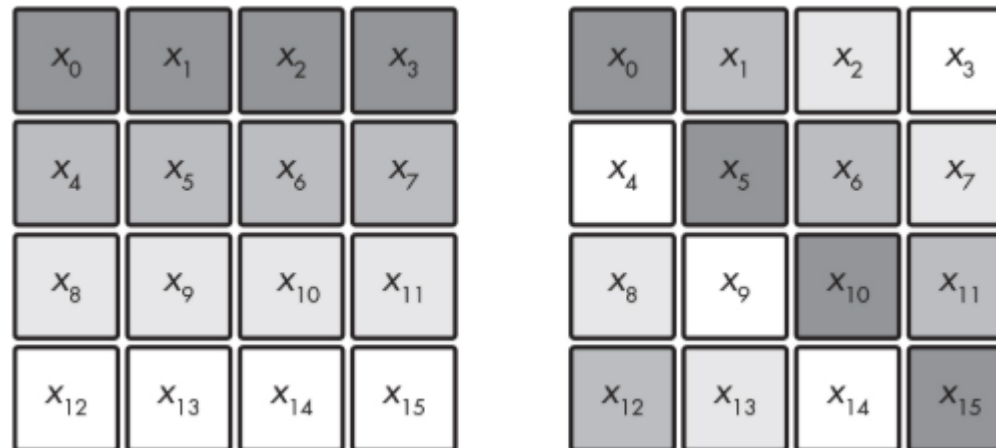
$c_0$	$k_0$	$k_1$	$k_2$
$k_3$	$c_1$	$v_0$	$v_1$
$t_0$	$t_1$	$c_2$	$k_4$
$k_5$	$k_6$	$k_7$	$c_3$

The initialization of Salsa20's state

# Transforming Salsa20's 512-Bit State

- ◆ The initial 512-bit state is transformed.
- ◆ Salsa20 first applies the **QR** transform to all four columns independently (known as the *column-round*) and then to all four rows independently (the *row-round*).
- ◆ The sequence column-round/row-round is a *double-round*. Salsa20 repeats 10 double-rounds, for 20 rounds in total, which is the reason for the *20* in *Salsa20*.

◆ .



Columns and rows transformed by Salsa20's quarter-round (QR) function

# Transforming Salsa20's 512-Bit State

- ◆ The column-round transforms the four columns as:

$$\mathbf{QR}(x_0, x_4, x_8, x_{12})$$

$$\mathbf{QR}(x_1, x_5, x_9, x_{13})$$

$$\mathbf{QR}(x_2, x_6, x_{10}, x_{14})$$

$$\mathbf{QR}(x_3, x_7, x_{11}, x_{15})$$

- ◆ The row-round transforms the four rows as:

$$\mathbf{QR}(x_0, x_1, x_2, x_3)$$

$$\mathbf{QR}(x_5, x_6, x_7, x_4)$$

$$\mathbf{QR}(x_{10}, x_{11}, x_8, x_9)$$

$$\mathbf{QR}(x_{15}, x_{12}, x_{13}, x_{14})$$

# Evaluating Salsa

- ◆ Salsa20's initial states for the first and second blocks when initialized with an all-zero key (00 bytes) and an all-one nonce (ff bytes).
- ◆ These two states differ in only 1 bit, in the counter, which is in bold: specifically, 0 for the first block and 1 for the second.

```
61707865 00000000 00000000 00000000
00000000 3320646e ffffffff ffffffff
00000000 00000000 79622d32 00000000
00000000 00000000 00000000 6b206574
```

```
61707865 00000000 00000000 00000000
00000000 3320646e ffffffff ffffffff
00000001 00000000 79622d32 00000000
00000000 00000000 00000000 6b206574
```

- ◆ internal states after 10 double-rounds

```
e98680bc f730ba7a 38663ce0 5f376d93
85683b75 a56ca873 26501592 64144b6d
6dcb46fd 58178f93 8cf54cfe cfdc27d7
68bbe09e 17b403a1 38aa1f27 54323fe0
```

```
1ba4d492 c14270c3 9fb05306 ff808c64
b49a4100 f5d8fbbd 614234a0 e20663d1
12e1e116 6a61bc8f 86f01bcb 2efead4a
77775a13 d17b99d5 eb773f5b 2c3a5e7d
```

# Salsa Secure Cipher compared to RC4

- ◆ *differential cryptanalysis*: the study of the differences between states rather than their actual values
- ◆ These two states differ in only 1 bit, in the counter, which is in bold: specifically, 0 for the first block and 1 for the second.

```
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

- ◆ After one round, the difference propagates across the first column to two of the three other words in that column:

```
80040003 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000001 00000000 00000000 00000000
00002000 00000000 00000000 00000000
```

# Salsa Secure Cipher compared to RC4

- ◆ After two rounds, differences further propagate across the rows that already include a difference, which is all but the second row

```
9ed7eb7f 060002c0 18028b0c 57ca83c0
00000000 00000000 00000000 00000000
00000001 0000e000 801c0006 00000000
00002000 00400000 04000008 0060f300
```

- ◆ After three rounds, the differences between the states become more dense,

```
3ab3c25d 9f40a5c9 10070e30 07bd03c0
db1ee2ce 43ee9401 21a7022c3 48fd800c
403c1e72 00034003 4dc843be 700b8857
5625b75b 09c00e00 06000348 23f712d4
```

# Salsa Secure Cipher compared to RC4

- ◆ After four rounds, differences look random to a human observer, and they are also almost random statistically as well:

```
d93bed6d a267bf47 760c2f9f 4a41d54b
0e03d792 7340e010 119e6a00 e90186af
7fa9617e b6aca0d7 4f6e9a4a 564b34fd
98be796d 64908d32 4897f7ca a684a2df
```

- ◆ After only four rounds, a single difference propagates to most of the bits in the 512-bit state. In cryptography, this is called *full diffusion*
- ◆ The differences propagate across all states, also according to complex equations that make future differences hard to predict. The mix of XOR, addition, and rotation cause the highly nonlinear relations in the state's evolution.
- ◆ In XORs, we can have many differences propagate, but the process would be linear and therefore insecure.

ChaCha20

# ChaCha20

---

- ◆ ChaCha20 is a stream cipher intended to be extremely efficient in s/w, introduced in 2008.
- ◆ It is available as a replacement for RC4 in many systems.
- ◆ It is combined with the Poly1305 message authentication code to construct an authenticated encryption (AE) scheme widely used in the TLS protocol

# ChaCha20 Quarter Round

- ◆ Let  $y = (a, b, c, d)$  be a sequence of four 32-bit words.
- ◆ Then a ChaCha quarter-round updates  $(a, b, c, d)$  as follows:
  1.  $a = a + b;$        $d = d \oplus a;$      $d \ll 16;$
  2.  $c = c + d;$        $b = b \oplus c;$      $b \ll 12;$
  3.  $a = a + b;$        $d = d \oplus a;$      $d \ll 8;$
  4.  $c = c + d;$        $b = b \oplus c;$      $b \ll 7;$

# ChaCha20 Doble Round

- ◆ ChaCha20 also runs 10 double-rounds.
- ◆ However, a ChaCha double-round consists of a **column-round** and a **diagonal-round**.
- ◆ A ChaCha double-round is defined by the 8 ChaCha quarter-rounds

column-round	quarter-round( $y_0, y_4, y_8, y_{12}$ ) quarter-round( $y_1, y_5, y_9, y_{13}$ ) quarter-round( $y_2, y_6, y_{10}, y_{14}$ ) quarter-round( $y_3, y_7, y_{11}, y_{15}$ )
diagonal-round	quarter-round( $y_0, y_5, y_{10}, y_{15}$ ) quarter-round( $y_1, y_6, y_{11}, y_{12}$ ) quarter-round( $y_2, y_7, y_8, y_{13}$ ) quarter-round( $y_3, y_4, y_9, y_{14}$ )

# ChaCha20 State Array

- ◆ The State Array  $S$

$$S = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ k_1 & k_2 & k_3 & k_4 \\ k_5 & k_6 & k_7 & k_8 \\ b & n_1 & n_2 & n_3 \end{pmatrix}$$

- ◆ The ChaCha20 stream cipher takes a 256-bit key  $k = (k_1, \dots, k_8)$  and a unique 96-bit message number  $n = (n_1, n_2, n_3)$  (nonce) as input.
- ◆ A 32-bit block counter  $b$  is initially set to zero
- ◆ The four 32-bit constants  $y_0 = 61707865$ ,  $y_1 = 3320646E$ ,  $y_2 = 79622D32$ ,  $y_3 = 6B206574$

$$\text{ChaCha}_k(n, b) = S + \text{double-round}_{10}(S).$$

Thank you – Q&A