Number Theory And Cryptography Assignment2

VIVEK K P

B210473CS

Salsa20

Salsa20 is a stream cipher developed by Daniel J. Bernstein that expands a 256-bit key into 2^64 randomly accessible streams, each containing 2^64 randomly accessible 64-byte (512 bits) blocks. Salsa20 has been selected as a Phase 3 design for Profile 1 (software) by the eSTREAM project.

Salsa20 is a symmetric key stream cipher designed to provide high security and performance. It operates on variable-length keys (128 or 256 bits) and a 64-bit nonce. The core of Salsa20 is based on a hash function that employs a series of rounds to mix and diffuse input data.

Brief explanation about each of the functions:

1. **`rotl` Function:**

   ```
   static uint32_t rotl(uint32_t value, int shift)
   ```

   This function performs a bitwise left rotation on a 32-bit value. The `shift` parameter determines the number of positions to rotate.
2. **`s20_quarterround` Function:**

   ```
   static void s20_quarterround(uint32_t *y0, uint32_t *y1, uint32_t *y2,
   uint32_t *y3)
   ```

   Implements the quarter-round operation of the Salsa20 algorithm. It updates four input words based on a series of bitwise operations and additions.
3. **`s20_rowround` Function:**

   ```
   static void s20_rowround(uint32_t y[static 16])
   ```

   Applies the quarter-round operation to each of the four rows in the Salsa20 state matrix.
4. **`s20_columnround` Function:**

```
static void s20_columnround(uint32_t x[static 16])
```

Applies the quarter-round operation to each of the four columns in the Salsa20 state matrix.

5. `s20_doubleround` **Function:**

```
static void s20_doubleround(uint32_t x[static 16])
```

Performs a double round, which consists of a column round followed by a row round.

6. `s20_littleendian` **Function:**

```
static uint32_t s20_littleendian(uint8_t *b)
```

Creates a 32-bit little-endian word from a 4-byte array.

7. `s20_rev_littleendian` **Function:**

```
static void s20_rev_littleendian(uint8_t *b, uint32_t w)
```

Moves a little-endian word into a 4-byte array.

8. `s20_hash` **Function:**

```
static void s20_hash(uint8_t seq[static 64])
```

The core function of Salsa20. It performs 20 rounds of the Salsa20 algorithm on a 64-byte block (`seq`), updating the state in place.

9. `s20_expand16` **Function:**

```
static void s20_expand16(uint8_t *k, uint8_t n[static 16], uint8_t
keystream[static 64])
```

Key expansion function for a 16-byte (128-bit) key. It incorporates constants specified by the Salsa20 specification and generates a keystream block.

10. `s20_expand32` **Function:**

```
static void s20_expand32(uint8_t *k, uint8_t n[static 16], uint8_t
keystream[static 64])
```

Key expansion function for a 32-byte (256-bit) key. Similar to `s20_expand16` but uses different constants.

11. **`s20_crypt` Function:**

```
enum s20_status_t s20_crypt(uint8_t *key, enum s20_keylen_t keylen,
uint8_t nonce[static 8], uint32_t si, uint8_t *buf, uint32_t buflen)
```

   The main encryption/decryption function. It takes a key, key length, nonce, stream index, input buffer (`buf`), and buffer length (`buflen`). It encrypts or decrypts the data based on the Salsa20 algorithm and returns a status.

These functions collectively implement the Salsa20 stream cipher, providing core operations such as rotation, quarter-round, key expansion, and the main hashing function for encryption and decryption.

Encryption Function:

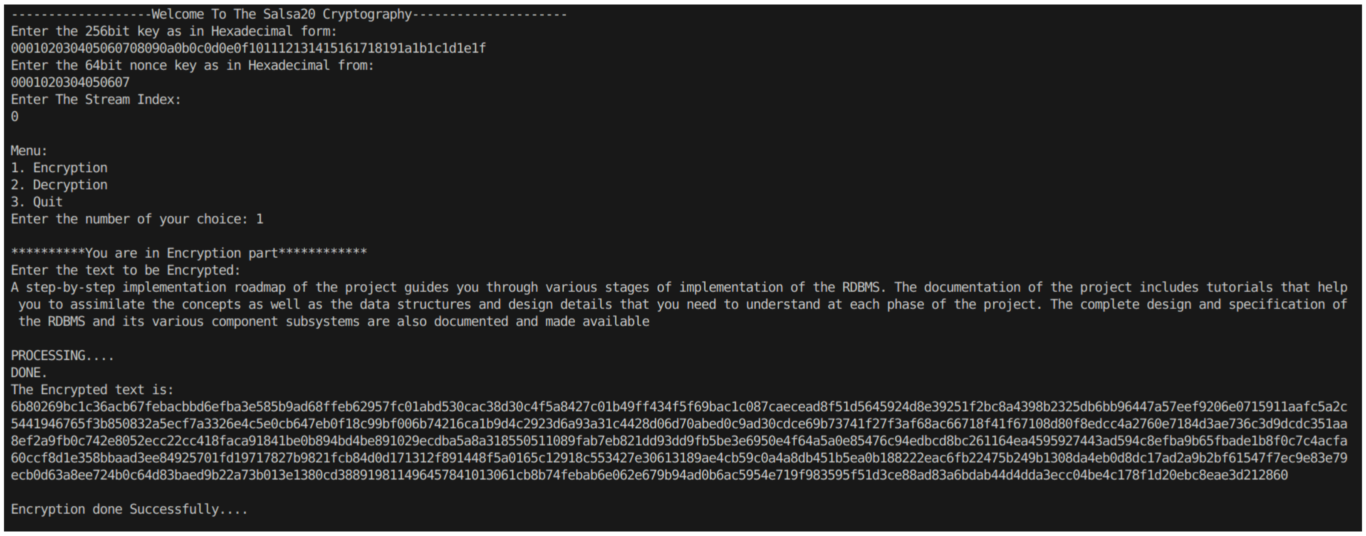
The `s20_crypt` function is responsible for encrypting data using the Salsa20 algorithm. It takes the following parameters:

- `uint8_t *key`: Pointer to the encryption key (256 bits or 128 bits).
- `enum s20_keylen_t keylen`: Key length (S20_KEYLEN_256 or S20_KEYLEN_128).
- `uint8_t nonce[static 8]`: Nonce (64 bits).
- `uint32_t si`: Stream index.
- `uint8_t *buf`: Pointer to the data to be encrypted.
- `uint32_t buflen`: Length of the data to be encrypted.

The function performs the following steps:

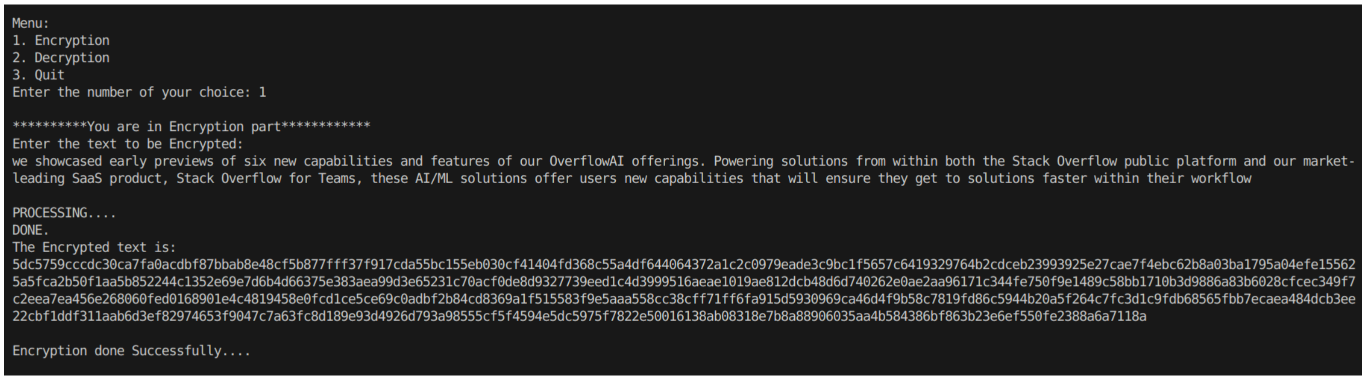
1. It determines the key length and selects the appropriate key expansion function (`s20_expand16` or `s20_expand32`).
2. It initializes the nonce and computes the initial keystream block using the key expansion function.
3. It XORs the plaintext data with the keystream to produce the ciphertext.
4. If the data length extends beyond one keystream block, additional keystream blocks are generated as needed.

Example-1:



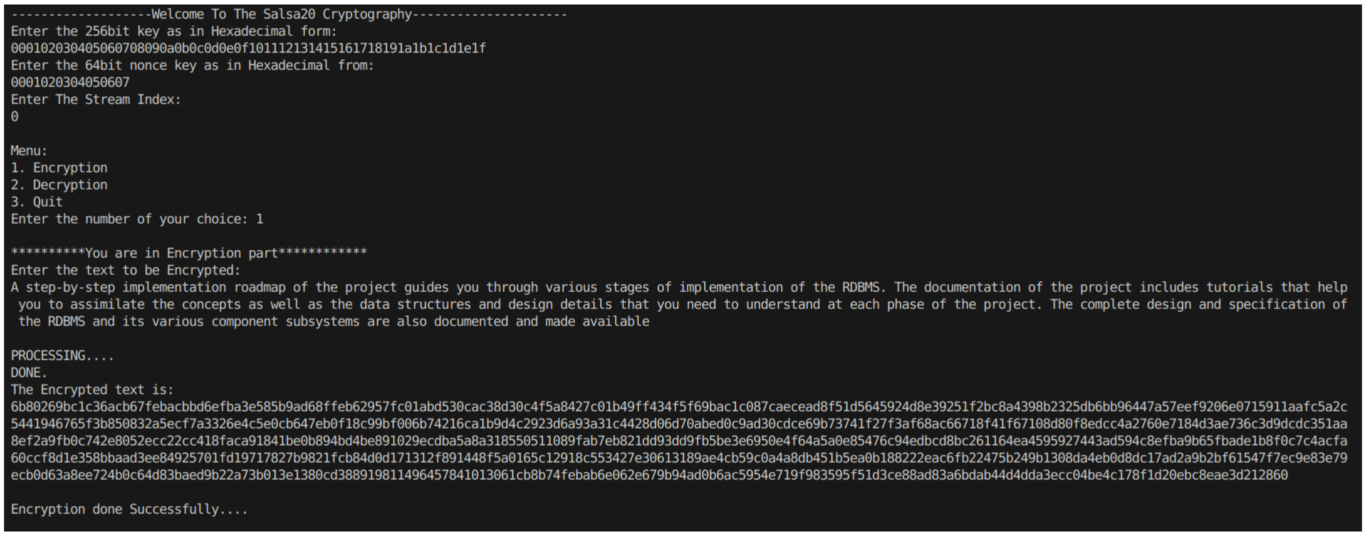
```
-------------------Welcome To The Salsa20 Cryptography--------------------
Enter the 256bit key as in Hexadecimal form:
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
Enter the 64bit nonce key as in Hexadecimal from:
0001020304050607
Enter The Stream Index:
0

Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 1

**********You are in Encryption part************
Enter the text to be Encrypted:
A step-by-step implementation roadmap of the project guides you through various stages of implementation of the RDBMS. The documentation of the project includes tutorials that help
 you to assimilate the concepts as well as the data structures and design details that you need to understand at each phase of the project. The complete design and specification of
 the RDBMS and its various component subsystems are also documented and made available

PROCESSING....
DONE.
The Encrypted text is:
6b80269bc1c36acb67febacbbd6efba3e585b9ad68ffeb62957fc01abd530cac38d30c4f5a8427c01b49ff434f5f69bac1c087caecead8f51d5645924d8e39251f2bc8a4398b2325db6bb96447a57eef9206e0715911aafc5a2c
5441946765f3b850832a5ecf7a3326e4c5e0cb647eb0f18c99bf006b74216ca1b9d4c2923d6a93a31c4428d06d70abed0c9ad30cdce69b73741f27f3af68ac66718f41f67108d80f8edcc4a2760e7184d3ae736c3d9dcdc351aa
8ef2a9fb0c742e8052ecc22cc418faca91841be0b894bd4be891029ecdba5a8a318550511089fab7eb821dd93dd9fb5be3e6950e4f64a5a0e85476c94edbcd8bc261164ea4595927443ad594c8efba9b65fbade1b8f0c7c4acfa
60ccf8d1e358bbaad3ee84925701fd19717827b9821fcb84d0d171312f891448f5a0165c12918c553427e30613189ae4cb59c0a4a8db451b5ea0b188222eac6fb22475b249b1308da4eb0d8dc17ad2a9b2bf61547f7ec9e83e79
ecb0d63a8ee724b0c64d83baed9b22a73b013e1380cd388919811496457841013061cb8b74febab6e062e679b94ad0b6ac5954e719f983595f51d3ce88ad83a6bdab44d4dda3ecc04be4c178f1d20ebc8eae3d212860

Encryption done Successfully....
```

Example-2:



```
Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 1

**********You are in Encryption part************
Enter the text to be Encrypted:
we showcased early previews of six new capabilities and features of our OverflowAI offerings. Powering solutions from within both the Stack Overflow public platform and our market-
leading SaaS product, Stack Overflow for Teams, these AI/ML solutions offer users new capabilities that will ensure they get to solutions faster within their workflow

PROCESSING....
DONE.
The Encrypted text is:
5dc5759cccdc30ca7fa0acdbf87bbab8e48cf5b877fff37f917cda55bc155eb030cf41404fd368c55a4df644064372a1c2c0979eade3c9bc1f5657c6419329764b2cdceb23993925e27cae7f4ebc62b8a03ba1795a04efe15562
5a5fca2b50f1aa5b852244c1352e69e7d6b4d66375e383aea99d3e65231c70acf0de8d9327739eed1c4d3999516aeae1019ae812dcb48d6d740262e0ae2aa96171c344fe750f9e1489c58bb1710b3d9886a83b6028cfcec349f7
c2eea7ea456e268060fed0168901e4c4819458e0fcd1ce5ce69c0adbf2b84cd8369a1f515583f9e5aaa558cc38cff71ff6fa915d5930969ca46d4f9b58c7819fd86c5944b20a5f264c7fc3d1c9fdb68565fbb7ecaea484dcb3ee
22cbf1ddf311aab6d3ef82974653f9047c7a63fc8d189e93d4926d793a98555cf5f4594e5dc5975f7822e50016138ab08318e7b8a88906035aa4b584386bf863b23e6ef550fe2388a6a7118a

Encryption done Successfully....
```

Example-3:



```
-------------------Welcome To The Salsa20 Cryptography--------------------
Enter the 256bit key as in Hexadecimal form:
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
Enter the 64bit nonce key as in Hexadecimal from:
0001020304050607
Enter The Stream Index:
0

Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 1

**********You are in Encryption part************
Enter the text to be Encrypted:
A step-by-step implementation roadmap of the project guides you through various stages of implementation of the RDBMS. The documentation of the project includes tutorials that help
 you to assimilate the concepts as well as the data structures and design details that you need to understand at each phase of the project. The complete design and specification of
 the RDBMS and its various component subsystems are also documented and made available

PROCESSING....
DONE.
The Encrypted text is:
6b80269bc1c36acb67febacbbd6efba3e585b9ad68ffeb62957fc01abd530cac38d30c4f5a8427c01b49ff434f5f69bac1c087caecead8f51d5645924d8e39251f2bc8a4398b2325db6bb96447a57eef9206e0715911aafc5a2c
5441946765f3b850832a5ecf7a3326e4c5e0cb647eb0f18c99bf006b74216ca1b9d4c2923d6a93a31c4428d06d70abed0c9ad30cdce69b73741f27f3af68ac66718f41f67108d80f8edcc4a2760e7184d3ae736c3d9dcdc351aa
8ef2a9fb0c742e8052ecc22cc418faca91841be0b894bd4be891029ecdba5a8a318550511089fab7eb821dd93dd9fb5be3e6950e4f64a5a0e85476c94edbcd8bc261164ea4595927443ad594c8efba9b65fbade1b8f0c7c4acfa
60ccf8d1e358bbaad3ee84925701fd19717827b9821fcb84d0d171312f891448f5a0165c12918c553427e30613189ae4cb59c0a4a8db451b5ea0b188222eac6fb22475b249b1308da4eb0d8dc17ad2a9b2bf61547f7ec9e83e79
ecb0d63a8ee724b0c64d83baed9b22a73b013e1380cd388919811496457841013061cb8b74febab6e062e679b94ad0b6ac5954e719f983595f51d3ce88ad83a6bdab44d4dda3ecc04be4c178f1d20ebc8eae3d212860

Encryption done Successfully....
```

Decryption Function:

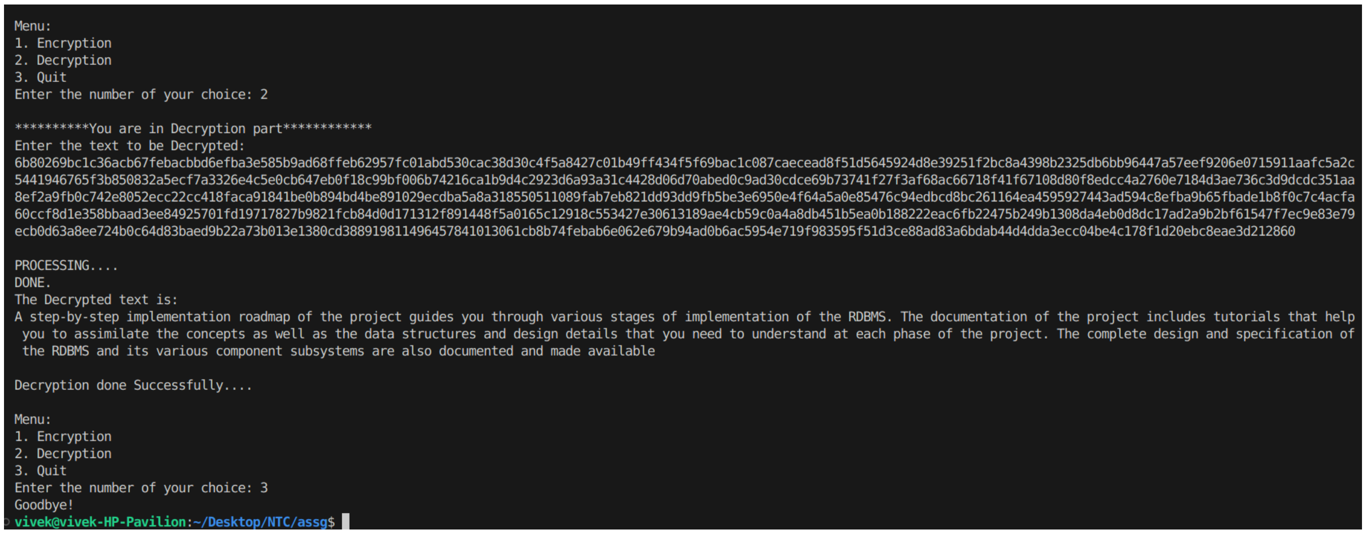
1. **Keystream Generation:**
   - Salsa20 employs a core hash function that operates on a 64-byte block of data (which is often referred to as the "keystream block").

- During encryption, this keystream block is generated based on the key, nonce, and a stream index.
- During decryption, the same keystream block is regenerated using the key, nonce, and the same stream index used during encryption.

2. **XOR Operation with Ciphertext:**
   - Just like in encryption, the keystream is XORed with the ciphertext to produce the plaintext.
   - The keystream is generated in blocks, and each block is XORed with the corresponding block of ciphertext.
3. **Stream Index:**
   - The stream index (si) is used to determine which block of the keystream to generate.
   - The stream index must be the same during both encryption and decryption to ensure the correct keystream is generated.
4. **Nonce:**
   - The nonce is a value used to ensure that the keystream is unique for each encryption/decryption session.
   - It is crucial that the nonce is the same during encryption and decryption for a given session.
5. **Key Expansion:**
   - Salsa20 uses key expansion functions (`s20_expand16` or `s20_expand32`) based on the key length.
   - These functions expand the original key and nonce into a larger keystream block.
6. **Block Processing:**
   - The keystream block is processed through multiple rounds of quarter-rounds, row-rounds, and column-rounds using the Salsa20 core hash function.
7. **XOR with Ciphertext:**
   - The generated keystream block is XORed with the corresponding block of the ciphertext to produce the corresponding block of plaintext.
8. **Repeating Blocks:**
   - The process is repeated for each block of the ciphertext.
9. **Output:**
   - The result is the decrypted plaintext.

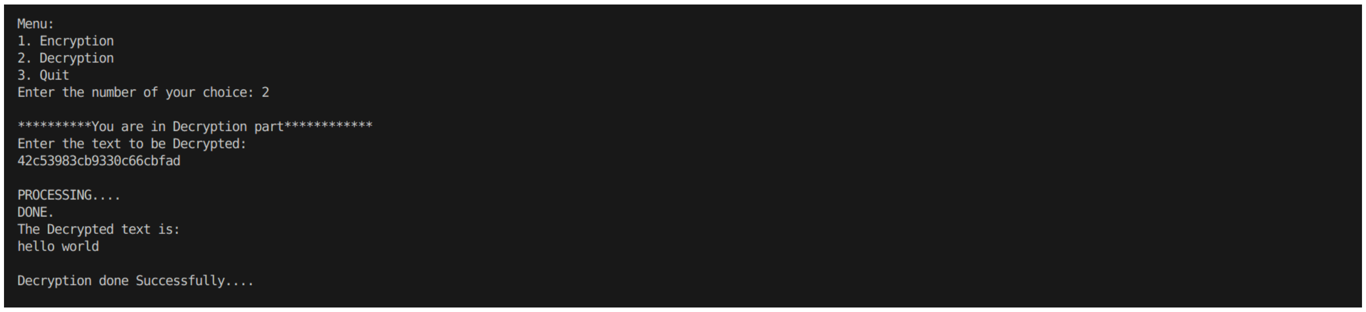
It's important to note that the decryption process in Salsa20 is quite similar to the encryption process, and the main difference lies in using the same keystream generation parameters (key, nonce, and stream index) during both encryption and decryption. The XOR operation with the keystream effectively reverses the effect of XORing during encryption, recovering the original plaintext.

Example-1:



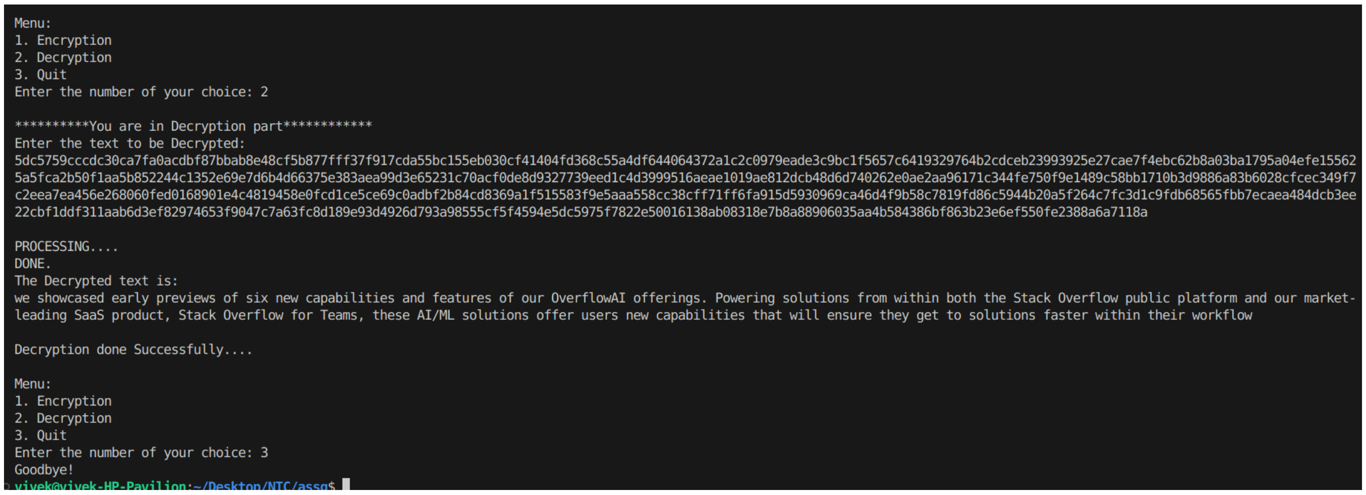
```
Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 2

**********You are in Decryption part************
Enter the text to be Decrypted:
6b80269bc1c36acb67febacbbd6efba3e585b9ad68ffeb62957fc01abd530cac38d30c4f5a8427c01b49ff434f5f69bac1c087caecead8f51d5645924d8e39251f2bc8a4398b2325db6bb96447a57eef9206e0715911aafc5a2c
5441946765f3b850832a5ecf7a3326e4c5e0cb647eb0f18c99bf006b74216ca1b9d4c2923d6a93a31c4428d06d70abed0c9ad30cdce69b73741f27f3af68ac66718f41f67108d80f8edcc4a2760e7184d3ae736c3d9dcdc351aa
8ef2a9fb0c742e8052ecc22cc418faca91841be0b894bd4be891029ecdba5a8a318550511089fab7eb821dd93dd9fb5be3e6950e4f64a5a0e85476c94edbcd8bc261164ea4595927443ad594c8efba9b65fbade1b8f0c7c4acfa
60ccf8d1e358bbaad3ee84925701fd19717827b9821fcb84d0d171312f891448f5a0165c12918c553427e30613189ae4cb59c0a4a8db451b5ea0b188222eac6fb22475b249b1308da4eb0d8dc17ad2a9b2bf61547f7ec9e83e79
ecb0d63a8ee724b0c64d83baed9b22a73b013e1380cd388919811496457841013061cb8b74febab6e062e679b94ad0b6ac5954e719f983595f51d3ce88ad83a6bdab44d4dda3ecc04be4c178f1d20ebc8eae3d212860

PROCESSING....
DONE.
The Decrypted text is:
A step-by-step implementation roadmap of the project guides you through various stages of implementation of the RDBMS. The documentation of the project includes tutorials that help
 you to assimilate the concepts as well as the data structures and design details that you need to understand at each phase of the project. The complete design and specification of
 the RDBMS and its various component subsystems are also documented and made available

Decryption done Successfully....

Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 3
Goodbye!
vivek@vivek-HP-Pavilion:~/Desktop/NTC/assg$
```

Example-2:



```
Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 2

**********You are in Decryption part************
Enter the text to be Decrypted:
42c53983cb9330c66cbfad

PROCESSING....
DONE.
The Decrypted text is:
hello world

Decryption done Successfully....
```

Example-3:



```
Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 2

**********You are in Decryption part************
Enter the text to be Decrypted:
5dc5759cccdc30ca7fa0acdbf87bbab8e48cf5b877fff37f917cda55bc155eb030cf41404fd368c55a4df644064372a1c2c0979eade3c9bc1f5657c6419329764b2cdceb23993925e27cae7f4ebc62b8a03ba1795a04efe15562
5a5fca2b50f1aa5b852244c1352e69e7d6b4d66375e383aea99d3e65231c70acf0de8d9327739eed1c4d3999516aeae1019ae812dcb48d6d740262e0ae2aa96171c344fe750f9e1489c58bb1710b3d9886a83b6028cfcec349f7
c2eea7ea456e268060fed0168901e4c4819458e0fcd1ce5ce69c0adbf2b84cd8369a1f515583f9e5aaa558cc38cff71ff6fa915d5930969ca46d4f9b58c7819fd86c5944b20a5f264c7fc3d1c9fdb68565fbb7ecaea484dcb3ee
22cbf1ddf311aab6d3ef82974653f9047c7a63fc8d189e93d4926d793a98555cf5f4594e5dc5975f7822e50016138ab08318e7b8a88906035aa4b584386bf863b23e6ef550fe2388a6a7118a

PROCESSING....
DONE.
The Decrypted text is:
we showcased early previews of six new capabilities and features of our OverflowAI offerings. Powering solutions from within both the Stack Overflow public platform and our market-
leading SaaS product, Stack Overflow for Teams, these AI/ML solutions offer users new capabilities that will ensure they get to solutions faster within their workflow

Decryption done Successfully....

Menu:
1. Encryption
2. Decryption
3. Quit
Enter the number of your choice: 3
Goodbye!
vivek@vivek-HP-Pavilion:~/Desktop/NTC/assg$
```

CryptAnalysis:

Salsa20 [2] was designed by Bernstein in 2005 as a candidate for eStream [9] and Salsa20/12 has been accepted in the eStream software portfolio. It has generated serious attention in the domain of cryptanalysis and quite a few results have been published in this direction [3, 4, 8, 1, 5, 7, 6] that show weaknesses of this cipher in reduced rounds. The main ideas in all these papers are circled around the following two issues.

- Put some input difference at the initial state and then try to obtain certain bias in some output differential
- Once you can move forward a few rounds with the above strategy, try to come back a few rounds from a final state after a certain rounds obtaining further non-randomness.

Combining the above ideas, one may find some non-randomness in Salsa20 for forward plus backward many rounds. The first result on Salsa20 cryptanalysis by [3] used this idea to move forward 3 rounds from the initial state and to come back 2 rounds from the final state for an attack on 5-round Salsa20. Later in [1], biased differentials till 4 forward rounds have been exploited and then the attack considered moving backwards from the final state for 4 rounds to obtain an 8-round attack on Salsa20. Before proceeding further, let us briefly explain the structure of Salsa20 stream cipher. This cipher considers 16 words, each 32-bit. We may refer to this as the Salsa state and a state can be written in 4 × 4 matrix format as follows:

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

The rightmost matrix shows the initial state, that takes four predefined constants c0, . . . , c3,1 256-bit key k0, . . . , k7, 64-bit nonce v0, v1 and 64-bit counter t0, t1. Since this is with 256-bit keys, we can refer it as 256-bit Salsa. Similarly, one can use the same cipher with 128-bit key, where ki = ki+4, for 0 ≤ i ≤ 3 and the constants are c0, . . . , c3.2 One may note the little differences in c1, c2 for the 128-bit and 256-bit version. In this paper, we are considering 256-bit key throughout. However, studying similar techniques for 128-bit key will also be interesting. The basic nonlinear operation of Salsa20 is the quarterround function. Each quarterround(a, b, c, d) consists of four ARX rounds, each of which comprises of one addition (A), one cyclic left rotation (R) and one XOR (X) operation as given below.

$$\left.\begin{aligned} b &= b \oplus ((a + d) \lll 7), \\ c &= c \oplus ((b + a) \lll 9), \\ d &= d \oplus ((c + b) \lll 13), \\ a &= a \oplus ((d + c) \lll 18). \end{aligned}\right\}$$

Each columnround works as four quarterrounds on each of the four columns of the state matrix and each rowround works as four quarterrounds on each of the four rows of the state matrix. In Salsa20 (one can call it Salsa20/20), ten times the columnround and ten times the rowround

are applied alternatively on the initial state. One may note that this can be considered as application of the columnround and transpose of the state matrix [6] twenty times. This helps in understanding the cipher better as in this case every round of Salsa20 becomes identical.
To be precise, in each round, we first apply quarterround on all the four columns in the following order: quarterround(x0, x4, x8, x12), quarterround(x5, x9, x13, x1),quarterround(x10, x14, x2, x6), and quarterround(x15, x3, x7, x11), and then a transpose(X) as follows.

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \rightarrow X^T = \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix}.$$

By X(r), we mean that r such rounds have been applied on the initial state X. Hence X(0) is the same as the initial state X. Finally, after R rounds we have X(R). Then a keystream block of 16 words or 512 bits is obtained as
`Z = X + X(R).`
For Salsa20, R = 20. However, the one accepted in eStream [9] software portfolio is Salsa20/12, where R = 12. Naturally, more rounds will provide better security and less rounds will provide higher speed.One may note that each Salsa20 round is reversible as the state-transition operations are reversible. In other words, if X(r+1) = round(X(r)), then X(r) = reverseround(X(r+1)), where reverseround is the inverse of round and consists of first transposing the state and then applying the inverse of quarterround for each column as follows.

$$\left.\begin{aligned} a &= a \oplus ((d + c) \lll 18), \\ d &= d \oplus ((c + b) \lll 13), \\ c &= c \oplus ((b + a) \lll 9), \\ b &= b \oplus ((a + d) \lll 7). \end{aligned}\right\}$$

Consider that one obtains a state X(1) after one round of Salsa20. Now to know whether it is a valid state after one round, one needs to come back by one reverse round and then check whether the constants in the diagonal elements are indeed the specified ones. This is the constraint for 256-bit Salsa20. However, for 128-bit Salsa20, one needs to have another constraint related to the key words apart from matching the constants. That is, we need to have ki = ki+4, for 0 ≤ i ≤ 3. Let us now present a few more notations. We have already denoted xi as the i-th word of the matrix X. By xi,j , we mean the j-th bit (0-th bit is the least significant bit) of xi. Given two states X(r), X′(r), we denote Δ(r)
i = x(r) i ⊕ x′(r) i . By Δ(r) i,j = x(r) i,j ⊕ x′(r) i,j , we mean the difference between two states at the j-th bit of the i-th word after r many rounds. That is, ‘Δ(0) 7,31 = 1’ means that we have two

initial states X(0), X′(0) that differ at the 31-st (most significant) bit of the 7-th word (a nonce word) and they are same at all the other bits of the complete state. In such a case, one can obtain significant biases after 4 rounds. That is, in general, we are interested to input a difference at the initial state (call it Input Differential or ID) and then try to obtain some bias corresponding to combinations of some output bits (call it Output Differential or OD). Most naturally, one can compute

where the probability is estimated for a fixed key and all the possible choices of nonce and counter words 0, v1 and t0, t1 respectively, other than the constraints imposed due to the input differences on keys or nonces/counters3. The term ♫d is a measure of the amount of the bias away from the probability of a random event and is naturally referred as the bias in the literature. Now we present the advantage of coming back one round. Consider that one likes to put some input differences in a state to obtain some output differential after a few rounds. Let X(1), X′(1) be the two initial states that differ in a few places and we obtain some bias Pr(Δ(r)
p,q = 1|Δ(1)
i,j = 1) = 1
That is actually we consider (r − 1) rounds here. If it is possible to get back one round each rom X(1), X′(1) such that X(0), X′(0) are valid initial states with differentials Δ(0)
i1,j1 , Δ(0) i2,j2 , . . . etc, then we can consider the situation Pr(Δ(r) p,q = 1|Δ(0) i1,j1 = 1, Δ(0) i2,j2 = 1, . . .) = 1 That is, we can exploit the bias as if it s after r rounds. This helps in analysing Salsa20 by one additional round.