IJRAR.ORG E-ISSN: 2348-1269, P-ISSN: 2349-5138

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG

An International Open Access, Peer-reviewed, Refereed Journal

IMPLEMENTATION OF AUTHENTICATION OF WEB OF THINGS BY USING SECURE HASH ALGORITHM-3 AND SALSA20 ALGORITHM

[1]Mrs. Chaithanya S, [2]Jagadhish, [3]Manjunatha S, [4]Manoj P G, [5]Naveen Kumar N
[1]Assistant Professor, [2]Student, [3]Student, [4]Student, [5]Student
[1]Electronics and Communication Engineering,
[1]Rajarajeswari College of Engineering, Bangalore, India

Abstract**:** The increasingly developing field of Web of things has led to lot of sensitive user data being exchanged over the web. Web of things enables the user to interact with their connected environment such as sensors, digital devices, automation systems, security camera monitoring without any hazels. But the growing number of users in the mere future poses a threat to the data being transmitted so there needs to be strong authentication and encryption measures put forth to ensure the integrity of the data it is also essential to create a way to identify the intruder has not manipulated the data. This paper deals with implementation a multi factor authentication scheme by using stream encryption algorithms for the encryption and authentication of the data there by ensuring faster validation of data and a better data monitoring. There has been various stream encryption algorithms developed to both authenticate and encrypt the data consists of SHA-1,2,3 for authentication and RC4, A5/1, Panama, Salsa20 etc. In this Paper we utilize Secure Hash Alogorithm-3 (SHA-3) algorithm in combination with Salsa20 with facilitates high speed data validation.

IndexTerms – WoT (Web of Things),encryption, SHA-1, SHA-2, SHA-3, RC4, A5/1, Panama, Salsa20

I. INTRODUCTION

Web of things is extension of internet of things where the data from different objects, sensors, applications gets transmitted over the web. The web of things ensures interoperability of different internet of things and application by describing specific set of standards. The WoT provides several building blocks which makes implementation of the system to follow conventions of the WoT architecture. These blocks are Thing description, Binding Templates, Scripting API. Security and privacy is a key aspect in WoT architecture where each building block in the WoT architecture contains specifications for security and privacy. This is essential because IoT only deals with transportation of information through the internet and does not bother with how the information travels or what happens to the data while in transmission in contrast the WoT establishes several rules that needs to be followed for the transmission of information. As more and more devices get integrated with the internet the switch from WoT to IoT becomes pretty evident. The WoT provides flexibility to implement different encryption algorithms to be embedded into its architecture. In this paper we implement a authentication scheme as well as an encryption scheme for WoT applications by utilizing SHA-3 which belongs to a family of Secure Hash Algorithms and is a latest member. It is a one way function used for generating digital prints of select length (128, 224, 256, 384 or 512 in bits) in our case it is 256 bits it is a algorithm which generates a one way function that we will be using for authenticating the integrity of the data being transmitted in combination with another algorithm Salsa20 which is a pseudorandom function generator based oof of add-rotate-XOR (ARX) operation which we will be using for data encryption and decryption. Use of both the algorithms makes it easier the any manipulation in the transmitted data and also send a encrypted data over the network.

II. LITERATURE SURVEY

1.They basically created a novel architecture for the WSNs environment, upon which a proposed technique for user authentication and key agreement has been given. Despite the fact that paper has a higher efficiency than other systems, it compromises key security elements.
2.This study examines the use of homomorphic encryption to encrypt client data on a cloud server and to perform necessary computations on this encrypted data. This paper focuses solely on analysis and does not include any recommendations for action.
3.The SHA-3 algorithm is written in Verilog HDL and simulated with Xilinx ISE 14.2. Only combinational circuits were used to implement the SHA-3 architecture.
4.The maximum frequency of the design, f-max is 224.339MHz. The design uses one clock cycle for each round and throughput obtained is:10170Mbps.
5.This paper has algorithm which has been upgraded version of the traditional ChaCha algorithm that increase cryptanalysis resistance. The super ChaCha cipher is suitable for security of IoT devices which required high security but suffering from low energy and limited storage space.

6.In this paper, the image is encrypted using ChaCha symmetric stream cipher with Hyperchaotic Map. The suggested lightweight picture encryption takes roughly 3.5 seconds to complete, indicating that it can be used in real-time apps.
7.They have principally designed a novel architecture for the WSNs environment, on which they have provided a proposed technique for user authentication and key agreement. Though, this paper presents relatively better efficiency comparing with the related systems, they compromised several security aspect.
8.This is a deep neural network-based framework that enables real-time authentication of wireless nodes by detecting the impacts of inherent process variation on RF parameters of wireless transmitters (Tx) at the receiver (Rx) end using in-situ machine learning. The stability of the RF-PUF is not examined in the presence of temperature and supply voltage variations.
9.To ensure secure communication in healthcare applications, the Anonymous Biometric Based User Authentication Scheme (SAB-UAS) is proposed. It is shown that the proposed SAB-UAS scheme experience more congestion when the number of message transmission increases proportionally i.e., adding (20) sensors in a row.
10. Stream ciphers are the quickest encryption algorithms, because they use an XOR between the algorithm output and the message to encrypt the data. Only side channel attacks, which are not algorithm-dependent, can break it.

III. SYSTEM DESIGN

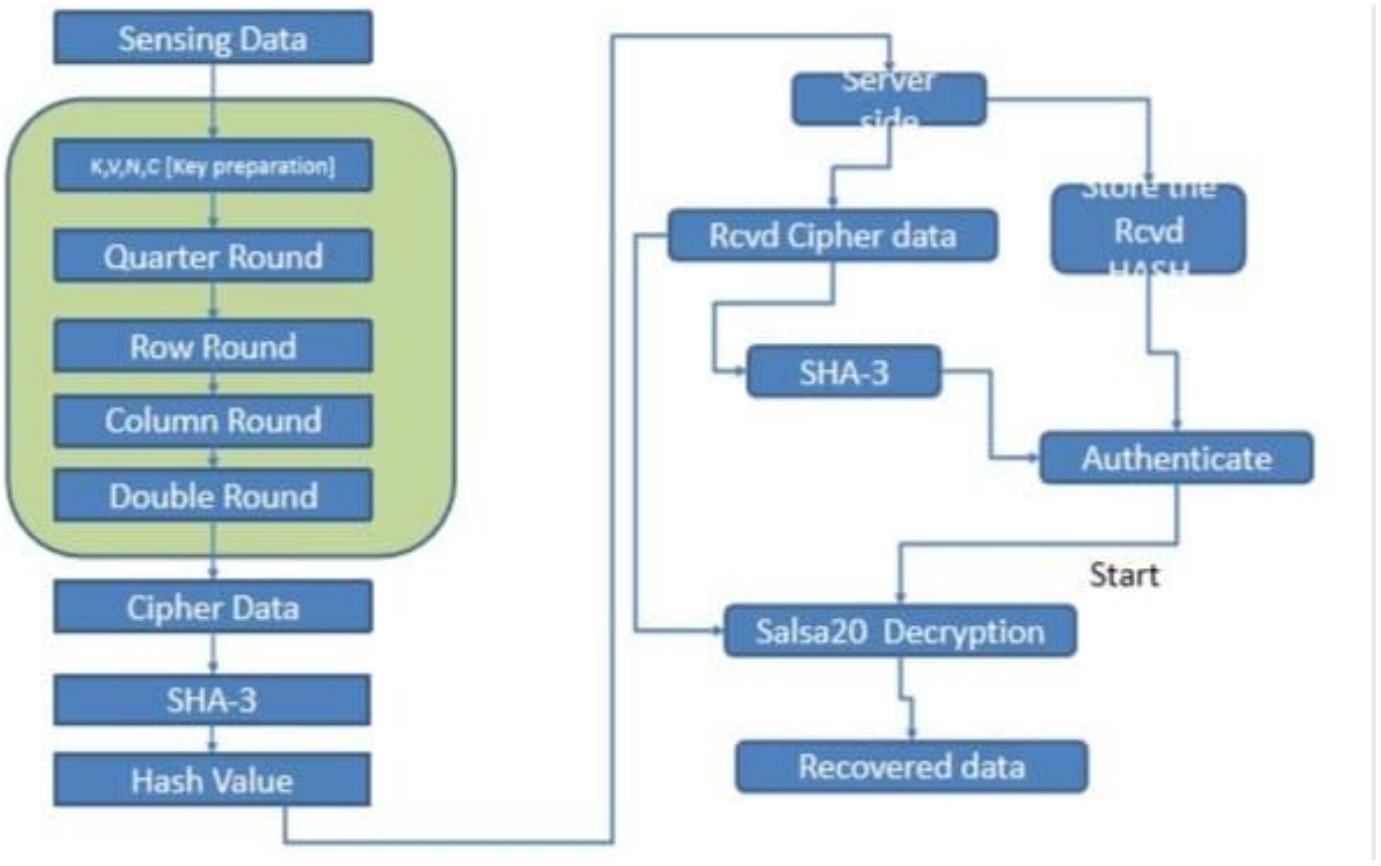


Figure 1: Block diagram

On the left side of the block diagram, the salsa20 Algorithm sub modules and data originating side processes are shown. To encrypt generated/sensor data, the Salsa20 method is employed. Encrypted data is hashed using the SHA3 algorithm. The encrypted and hashed data is sent to the server or the end customer. On the left side of the block diagram, the salsa20 Algorithm sub modules and data originating side processes are shown.
To encrypt generated/sensor data, the Salsa20 method is employed. Encrypted data is hashed using the SHA3 algorithm. They will receive hash and encrypted data are saved on the server. The SHA3 algorithm is used to hash the encrypted data once more. To ensure data authenticity, the generated and received hashes are compared.
If the authentication is successful, the decoding of the encrypted files can begin. Otherwise, the data received would be rejected because it did not come from a reliable source. The Salsa20 decryption block retrieves the original data, which can then be utilized for WoT.

IV. METHODOLOGY

In this work, we use two algorithms:
1 SHA 3 - it can be used in place of SHA-2 in present applications if necessary, and it improves the general resilience of NIST's hash algorithm toolkit greatly.
2 Salsa20 Algorithm-It uses less computing power because its ranking is the same as a weighted in/out degree ranking. The algorithm's processing cost is critical since HITS and SALSA are computed at query time and so have a major impact on a search engine's response time.

Secure Hash Algorithm 3:

SHA-3, commonly called Keccak, is a unidirectional algorithm for creating digital printouts of a certain length. It accepts 224, 256, 384, or 512 bits from any quantity of input data.
The Secure Hash Algorithm-3 was created by Keccak. It contains various SHA-3 models, including SHA-3 (224-bit, 512-bit, 384-bit, and 256-bit). It consists of several rounds, each with its own set of logical processes. The sponge function allows input to be absorbed first, then compressed to achieve a specified goal.
There are a variety of implementations that address certain constraints and trade-offs. There's something for everyone, from the simplest round-based implementation through pipelines and unrolled version targeted at throughput to serial variants geared at lightweight programs using a folding layout.
For low-area designs, various solutions have been offered. Folded structures are defined by the folding factor, which determines the orientation and degree of folding (FF). In their initial implementation, they offered a lane-oriented design in which the state is handled lane by lane. The issue with this method is that it takes precise control logic to schedule the calculations of and permutations. Jungk and Apfelbeck proposed the first slice-based folding implementation, in which the state is processed slice by slice.
In This paper shows an implementation using FF=8, which processes eight slices in parallel. A similar strategy was used. employed in, where several folding factors were investigated, with

Distributed RAM was used to implement this state.

- At the zero-input step, and input padding is performed to generate a 1600-bit arbitrary input block.
- The input matrix is XORed in the absorption phase, and all 24 iterations are completed.
- During the squeeze process, the input matrix is truncated to produce the appropriate output length.

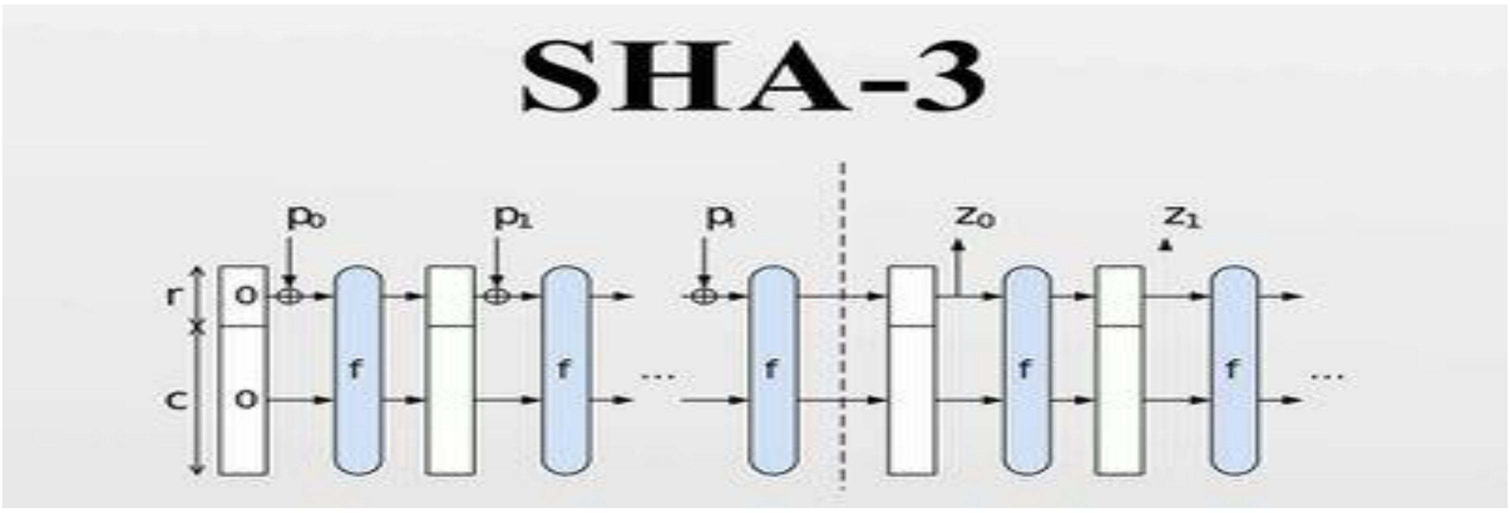


Figure 2: SHA-3 Algorithm

Salsa20 Algorithm:

Salsa20 algorithm are listed in ascending order from the simplest to the most difficult. In the algorithm explanation, the addition of two 4-byte words is written as a + b. The result is modulo 232 divided (so, by the maximum value that can be stored in one word).
a+b mod 232 equals the sum of two words a and b. The result was a 4-byte long valid word.
The algorithm denotes the a-bit left rotation of a 4-byte word was w a. The parts on the left are moved to the right.

A left w a rotation of bits in a 4-byte word w can be represented as the multiplication:
2a·w mod (232-1)
The result is a 4-byte long valid word.
The Quarter round function follows four words and returns a four-word sequence.

The function can be defined as follows if x is a four-word input: x = (x0, x1, x2, x3).
where quarter round(x) = (y0, y1, y2, y3)

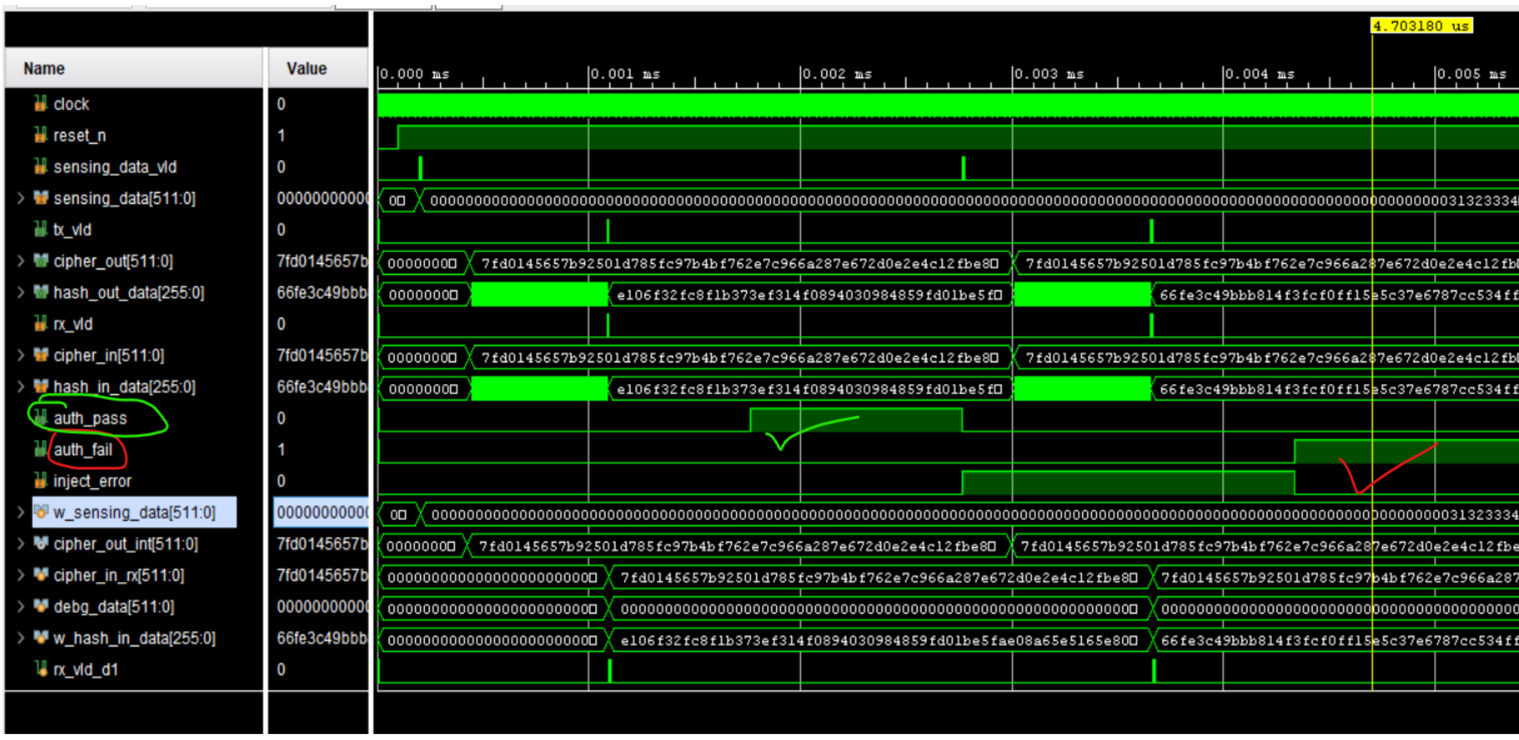
$$\begin{aligned} z1 &= y1 \oplus ((y0 + y3) \lll 7),\\ z2 &= y2 \oplus ((z1 + y0) \lll 9),\\ z3 &= y3 \oplus ((z2 + z1) \lll 13),\\ z0 &= y0 \oplus ((z3 + z2) \lll 18). \end{aligned}$$

Without allocating any more memory, the Quarter round function can be performed in place. First, x1 becomes y1, then x2 becomes y2, then x3 becomes y3, and finally x0 becomes y0. Because all of the adjustments above are invertible, it is invertible.

These algorithm stream cypher devised by Daniel J. Bernstein and submitted to the e-Stream (ECRYPT Stream Cipher Project) in 2005. Everywhere, the Salsa20 algorithm permits a different output block from any original input block. It's an asynchronous stream cypher that generates 64-block output based on the key, nonce, and block number. The Salsa20 algorithm provides an output block that applies the method directly to key and nonce input values because it does not pre-process any S-boxes or inputs.
It has 20 rounds and 32-byte (256-bit) memory. It can, however, be used in a minor key with its 8 and 12 round versions. The Salsa20 algorithm's designer does not recommend any small adjustments

V. EXPERIMENTAL RESULT

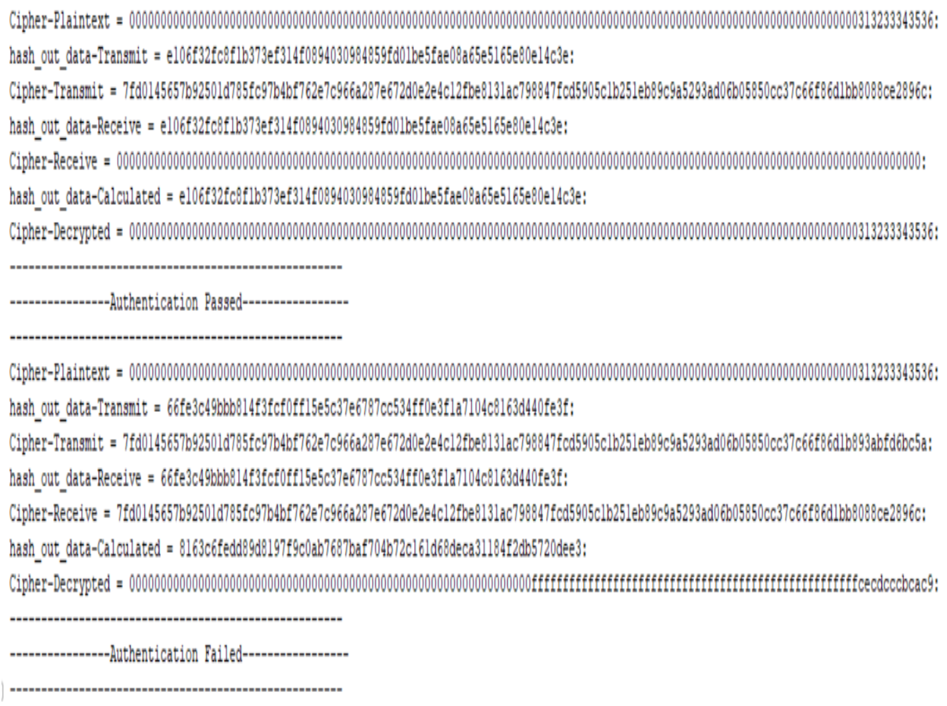
We have implemented our project using ModelSim and Vivado software. The code used to simulate the project written in Verilog HDL.
Simulation waveform looks like this.



Name	Value
clock	0
reset_n	1
sensing_data_vld	0
sensing_data[511:0]	00000000000
tx_vld	0
cipher_out[511:0]	7fd0145657b
hash_out_data[255:0]	66fe3c49bbb
rx_vld	0
cipher_in[511:0]	7fd0145657b
hash_in_data[255:0]	66fe3c49bbb
auth_pass	0
auth_fail	1
inject_error	0
w_sensing_data[511:0]	00000000000
cipher_out_int[511:0]	7fd0145657b
cipher_in_rx[511:0]	7fd0145657b
debg_data[511:0]	00000000000
w_hash_in_data[255:0]	66fe3c49bbb
rx_vld_d1	0

Figure 3: Overall simulation result with Authentication pass and fail conditions covered

Initially Authentication will pass & 2nd time the authentication fail will be asserted as inject error is enabled for the 2nd time. The simulation console also gives the data in & out prints during the simulation



```
Cipher-Plaintext = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000313233343536:
hash_out_data-Transmit = e106f32fc8f1b373ef314f0894030984859fd01be5fae08a65e5165e80e14c3e:
Cipher-Transmit = 7fd0145657b92501d785fc97b4bf762e7c966a287e672d0e2e4c12fbe8131ac798847fcd5905c1b251eb89c9a5293ad06b05850cc37c66f86d1bb8088ce2896c:
hash_out_data-Receive = e106f32fc8f1b373ef314f0894030984859fd01be5fae08a65e5165e80e14c3e:
Cipher-Receive = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000:
hash_out_data-Calculated = e106f32fc8f1b373ef314f0894030984859fd01be5fae08a65e5165e80e14c3e:
Cipher-Decrypted = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000313233343536:
----------------------------------------------------
---------------Authentication Passed----------------
----------------------------------------------------
Cipher-Plaintext = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000313233343536:
hash_out_data-Transmit = 66fe3c49bbb814f3fcf0ff15e5c37e6787cc534ff0e3f1a7104c8163d440fe3f:
Cipher-Transmit = 7fd0145657b92501d785fc97b4bf762e7c966a287e672d0e2e4c12fbe8131ac798847fcd5905c1b251eb89c9a5293ad06b05850cc37c66f86d1b893abfd6bc5a:
hash_out_data-Receive = 66fe3c49bbb814f3fcf0ff15e5c37e6787cc534ff0e3f1a7104c8163d440fe3f:
Cipher-Receive = 7fd0145657b92501d785fc97b4bf762e7c966a287e672d0e2e4c12fbe8131ac798847fcd5905c1b251eb89c9a5293ad06b05850cc37c66f86d1bb8088ce2896c:
hash_out_data-Calculated = 8163c6fedd89d8197f9c0ab7687baf704b72c161d68deca31184f2db5720dee3:
Cipher-Decrypted = 0000000000000000000000000000000000000000000000000000000000000000ffffffffffffffffffffffffffffffffffffffffffffffffffffcecdcccbcac9:
----------------------------------------------------
---------------Authentication Failed----------------
----------------------------------------------------
```

Figure 4: Console prints to visualize the input and output data during the simulation

- **Implementation results**:

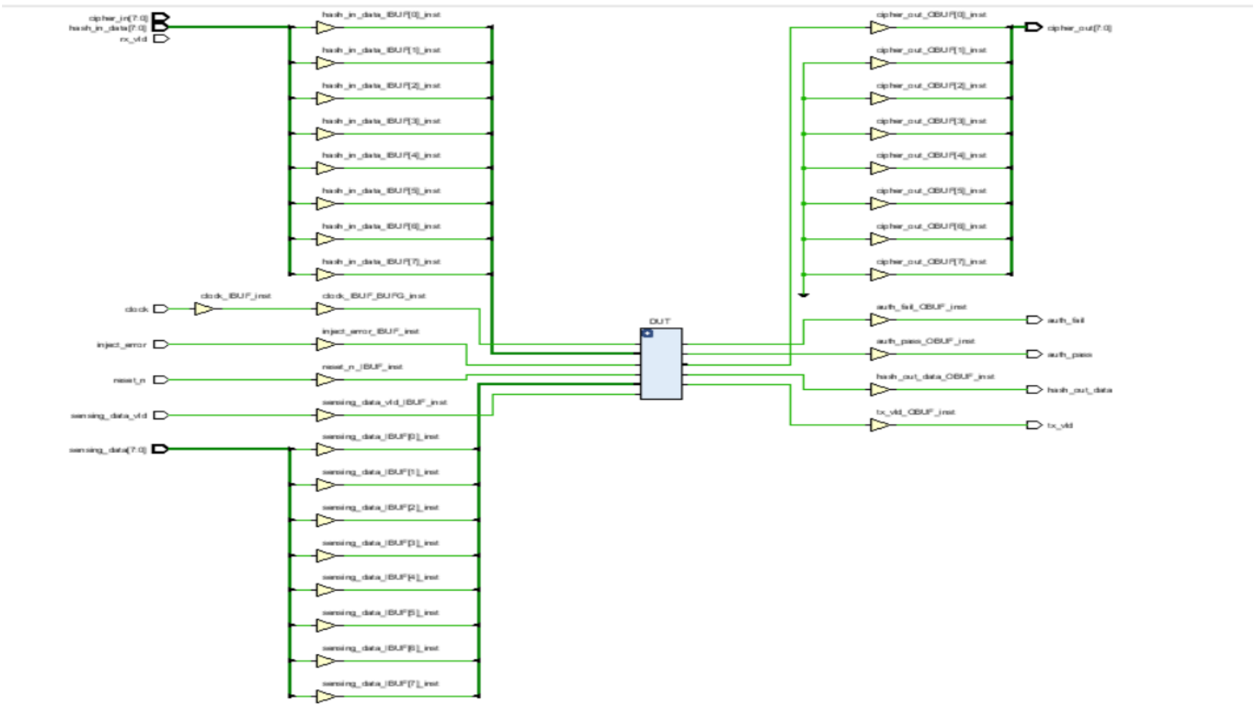
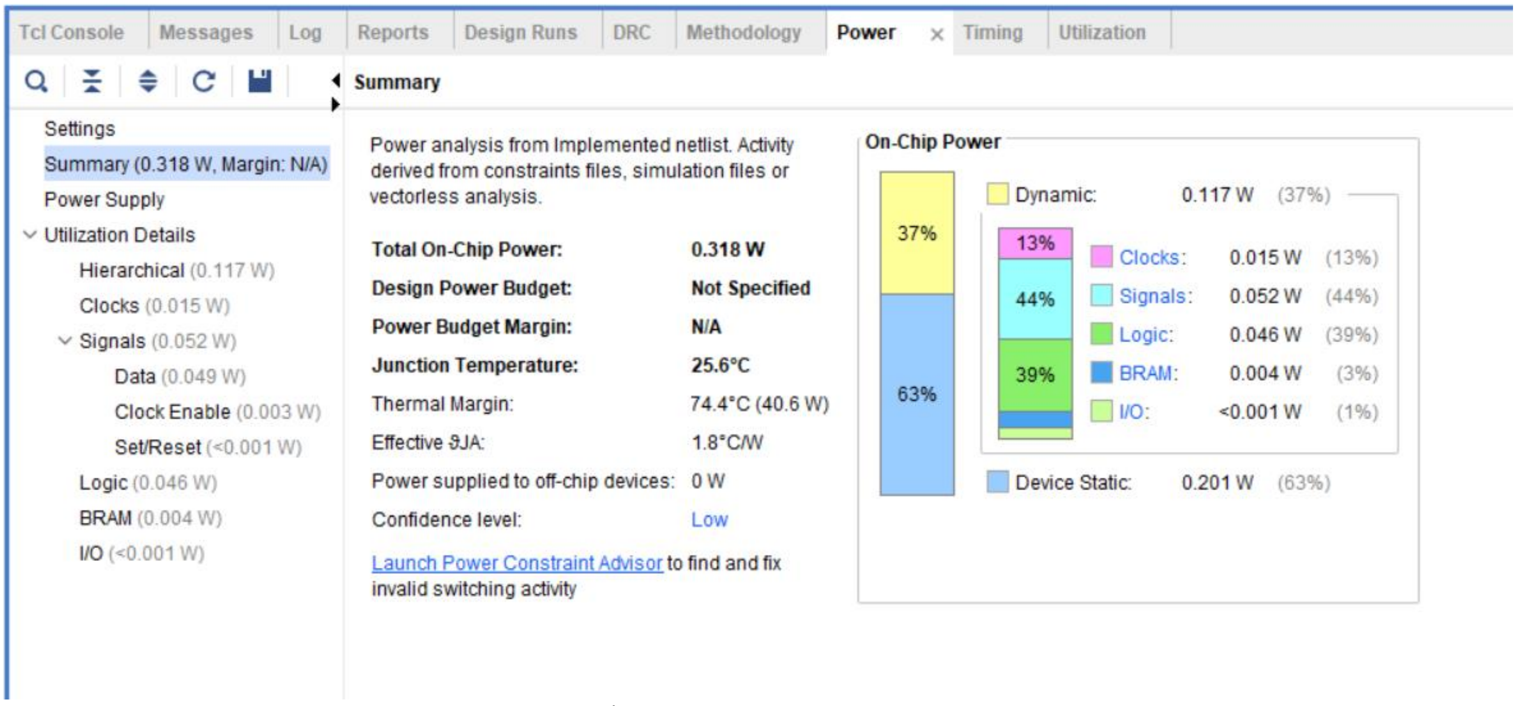


Figure 5: Implemented Vivado schematics

- **Power Summary:**

Detailed power breakdowns are mentioned in the above report.



Settings
Summary (0.318 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.117 W)
Clocks (0.015 W)
Signals (0.052 W)
Data (0.049 W)
Clock Enable (0.003 W)
Set/Reset (<0.001 W)
Logic (0.046 W)
BRAM (0.004 W)
I/O (<0.001 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.318 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.6°C**
Thermal Margin:	74.4°C (40.6 W)
Effective ϑJA:	1.8°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	0.117 W	(37%)
🟪	Clocks:	0.015 W	(13%)
🟦	Signals:	0.052 W	(44%)
🟩	Logic:	0.046 W	(39%)
🟦	BRAM:	0.004 W	(3%)
🟩	I/O:	<0.001 W	(1%)
🟦	Device Static:	0.201 W	(63%)

Figure 6: Power Report

Overall power consumption: 0.318W
Dynamic power: 0.117W
Static power: 0.201W

VI. Acknowledgment

We give our deepest thanks to our mentor, Raja Rajeshwari college of engineering for her unfailing support, guidance, and encouragement.

VII. REFERENCE

1. A secure light weight scheme for user authentication and key agreement in multi gateway based wireless sensor networks. Ad Hoc Networks, 2015, Rahul Amin, G P Biswas.

2. Using fully Homomorphic Encryption to secure cloud computing, Research Gate, 2016, Ihsan Jabbar.

3. Low power and pipelined secure hashing algorithm-3, IEEE-2016, Jayanthi Sharma.

4. Design and characterization of SHA-3 IP core Science Direct, 2019, Jeethu James.

5. An improved ChaCha algorithm for securing data on IoT devices, SN applied sciences, 2021, Mohammed Salih Mahdi.