Design of Cryptographic Primitives for Wireless Communication and Blockchain Mining

A

Thesis submitted

for the award of the degree of

DOCTOR OF PHILOSOPHY

By

Sushree Sila P. Goswami

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781 039, ASSAM, INDIA

OCTOBER 2023

Declaration

I hereby declare that the thesis entitled **"Design of Cryptographic Primitives for Wireless Communication and Blockchain Mining"**, submitted to the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, India, for the award of the degree of **Doctor of Philosophy**, has been carried out by me under the supervision and guidance of Dr. Gaurav Trivedi. The results embodied in this thesis are original and have not been submitted to any other University or Institute for the award of any degree or diploma.

Dated:

Guwahati.

Sushree Sila P. Goswami
Research Scholar
Dept. of Electronics and Electrical Engg.
Indian Institute of Technology Guwahati
Guwahati - 781 039, Assam, India.

Certificate

This is to certify that the thesis entitled "**Design of Cryptographic Primitives for Wireless Communication and Blockchain Mining**", submitted by **Sushree Sila P. Goswami** (156102025), a research scholar in the *Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati*, for the award of the degree of **Doctor of Philosophy**, is a record of an original research work carried out by him under my supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the institute and, in my opinion, has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Dated:
Guwahati.

Dr. Gaurav Trivedi
Associate Professor
Dept. of Electronics and Electrical Engg.
Indian Institute of Technology Guwahati
Guwahati - 781 039, Assam, India.

Dedicated to Almighty,

&

to my family members for their blessings, love and support.

Acknowledgements

First and foremost, I feel it as a great privilege in expressing my deepest and most gratitude to my supervisor Dr. Gaurav Trivedi, for his excellent guidance throughout my Ph. D. tenure. His kindness, dedication and attention to detail has been asource of great inspiration to me. My heartfelt thanks to my supervisor for the unlimited support and patience that he has shown towards me. His emphasis on clear communication helped me a lot. He has enriched my life in many significant ways. I thank him from the bottom of my heart for always being there with me during all kinds of distress.

I would like to thank my doctoral committee members, Prof. H. B. Nemade, Dr. Srinivasan Krishnaswami and Dr. A. Sahu, for sparing time out of their busy schedule to evaluate my progress and enrich this work with their valuable suggestions and feedback. I am also grateful to faculty members and the office staffs of the Department of Electronics and Electrical Engineering, IIT Guwahati, for their help in carrying out this research work.

My sincere thanks go to Dr. Bikram Paul for his continuous assistance in completion of the thesis. I am also thankful to Dr. Satyabrata Dash, Dr. Sunil Dutt, Dr. Swati Shukla, Mr. Praveen Tiwari, Dr. Meenali Janveja, Dr. Subrata Nandi, Mr. Rohit Jharia, Mr. Tejas K. Atreya, Mr. Yashwanth Kumar Tirupati, Mr. Shubham Garg, Mr. Rushik Parmar, Mr. Bipul Bodo, Ms. Ankita Tiwari, Mr. Saras Mani Mishra, Ms. Parmita Roy, Mr. Shubhadip Poria, Mr. Abhyuday Bharadwaj, Dr. Ananda YR, Mr. Ashwini Dongre for their ungrudging help and sympathetic encouragement.

Lastly, I attribute this achievement to my parents, spouse, sister, son and daughter for their constant support and silent prayers for my success and making me stand in this position.

Sushree Sila P. Goswami

Abstract

Nowadays, internet usage in every sector is increasing, which increases the need for security because cyber attackers may corrupt or change our data and misuse it. Cryptography is employed in everyday life, like authentication or digital signatures. Electronic money, also known as digital cash, can be transferred electronically from one person to another. This implies a need to enhance the security provided by various cryptographic algorithms. In this thesis, different cryptographic algorithms like DES, RSA, AES, ECC, and ECCDH are implemented on FPGA. In applications like secure wireless communication, stream ciphers are used for their implementation simplicity in hardware. In military cryptography, stream ciphers can be generated in a separate box under stringent security protocols and then fed into other devices, such as radio sets that can be designed and utilized in less strict areas. A general method to design a stream cipher is to use a pseudorandom number generator or the keystream generator and mask the plaintext using its output, which is typically a XOR operation. The cipher text is produced using the key stream output as a one-time-pad. A stream cipher is a cryptographic encryption method that converts plain text into an unreadable code to anyone without the correct key by processing byte by byte. The same key is used to encrypt and decrypt messages with stream ciphers as they are linear. The design is realized using Verilog Hardware Description language, and for the hardware implementation, an FPGA is employed. From the experimental performance and hardware resources results, it can be inferred that the modified SNOW 2.0 utilizes 13% less resources and is 19% more efficient than the traditional SNOW 2.0 architecture. Also, it is 104% more efficient than the most efficient existing architectures.

Similarly, security plays a vital role in electronic communication, particularly in wireless networks like LTE (Long Term Evolution), where safeguarding data and resources from malicious activities is crucial. Cryptographic algorithms are at the core of security mechanisms, ensuring the protection of sensitive information. While software implementations of these algorithms are relatively straight-

forward, they often need more speed in real-time applications for communication devices like mobile phones. Consequently, implementation of these cryptographic algorithms as hardware crypto processors becomes necessary. This thesis presents a novel implementation of the SNOW3G crypto processor architecture for 4G LTE security applications, focusing on area, power and efficiency. The modified SNOW3G architecture proposed in this thesis utilizes only 0.31% of the available area when implemented on Zynq ZC702 FPGA and achieves 28.34 efficiency, which quantifies the ratio of throughput to the area. Furthermore, it consumes a total power of $0.142mW$. The low power and area requirements make the proposed design highly suitable for integration into mobile devices, meeting their specific constraints and enabling efficient cryptographic operations.

However, to provide both confidentiality and reliability, cryptographic methods are not suitable. A method known as secret sharing is used for distributing cryptographic encryption keys among multiple people to enhance security. The secret sharing is a method of encoding a secret by distributing it amongst a group of participants. The secret can only be decoded when a sufficient number of participants coordinate. Secure Multi-Party Computation (SMPC) is a computationally intensive application of secret sharing, where various functions over the input data are computed by keeping it private. An effective way of addressing this problem is to implement secret sharing algorithms on hardware. Designing application-specific hardware can achieve significant performance gains, reduced power consumption, and better resource utilization. This thesis presents a resource and delay-efficient hardware realization of Shamir's linear secret sharing and Renval-Ding's nonlinear secret sharing schemes on an FPGA. These schemes are scalable with respect to secret size and the number of participants. It is found that the proposed hardware design of Shamir's linear secret sharing scheme is 98.49% more resource efficient than its contemporary schemes. The power consumption of the share generation unit (SGU) and secret reconstruction unit (SRU), the main constituents of Shamir's linear secret sharing scheme, are approximately 96.96% and 99.67% less than the existing method available in the literature. The implementations of Shamir's linear secret sharing and Renval-Ding's nonlinear secret sharing schemes are approximately 300× faster than their software realizations. The power consumption and area utilization of Renval-Ding's nonlinear secret sharing scheme are 9.7 mW and 1.72 mm^2 when implemented as an ASIC using SCL $180nm$ BULK CMOS PDKs.

The security provided by a blockchain relies on a computationally intensive algorithm employed in the Bitcoin mining. This algorithm effectively safeguards against double spending of bitcoins and any

attempts to manipulate confirmed transactions. This security is primarily achieved through the use of asymmetric encryption, hashing, and various other cryptographic algorithms within the blockchain system. This "proof-of-work" algorithm is energy-demanding. Thus, the high energy consumption increases cost, since high energy is dissipated in the form of heat and a specific hardware is required for the temperature control. Therefore, an ASIC with blockchain technology is necessary to reduce energy consumption, cost and to optimize computational resources. The aim of the proposed work is to implement a blockchain mining process using Verilog, which can be realized as an ASIC. This objective is to realized implementing the design on an FPGA and validating it for the parameters like power consumption, frequency, and throughput.

Contents

List of Figures

List of Tables

1

Introduction

Contents

1. Introduction

In today's world, smart devices and sensors, such as mobile phones, ATMs, and vending machines, are ubiquitous. These devices generate and process vast amounts of data, which require secure storage and protection. However, a simple and cost-effective platform is needed to safeguard these networks and devices. Therefore, developing low-power, effective, and secure cryptographic algorithms are becoming increasingly crucial. Cryptographic algorithms form the foundation of secure communication and data protection. The primary objective of cryptography is to transform plaintext (original data) into ciphertext (encoded data) using mathematical algorithms and keys. The process of encryption ensures that even if an unauthorized entity intercepts the ciphertext, it cannot understand or extract meaningful information from it without the corresponding decryption key. This provides a means of maintaining confidentiality, data integrity, authentication, and no repudiation and access control in communication channels and data storage. There are two main types of cryptographic algorithms: symmetric key encryption (also known as secret key encryption) and asymmetric key encryption (also known as public key encryption). Symmetric key encryption uses a single secret key to encrypt and decrypt data. The same key is shared between the sender and the recipient, making it important to keep the key secret to prevent unauthorized access. Symmetric encryption algorithms are typically faster and more efficient for bulk data encryption. On the other hand, asymmetric key encryption uses a pair of mathematically related keys: a public key and a private key. The public key is widely distributed, while its owner keeps the private key secret. Data encrypted with the public key can only be decrypted using the corresponding private key. Asymmetric encryption provides a way to securely exchange information and verify the authenticity of digital signatures. Cryptography also encompasses other important components, such as hash functions and digital signatures. Hash functions are mathematical algorithms that convert an input of any size into a fixed-size output, known as a hash value or hash code. They are used to ensure data integrity and verify the authenticity of information. Digital signatures, on the other hand, are used to authenticate the source of digital messages or documents and provide non-repudiation.

The concept of interconnecting networks and computers has been around for years, but terms like "Internet of Things" and "Blockchain mining" are relatively new. IoT is a vast network of devices and objects connected to the Internet through a router and network devices. IoT has created various opportunities as well as challenges across the world. Applications of IoT includes smart homes,

smart city, connected health, wearables, industrial Internet, smart farming, smart retails, smart supply chain, etc. With the increasing reliance on digital systems and the constant evolution of technology, cryptography plays a crucial role in safeguarding sensitive information and maintaining the security and privacy of individuals, organizations, and digital transactions. Understanding the principles and applications of cryptography is essential for designing and implementing secure communication protocols, secure storage mechanisms, and secure digital transactions in today's interconnected world. Cryptographic constructs like cryptographic algorithms, snow stream cipher, secret sharing schemes, and blockchain mining are fundamental components of modern information security systems. These technologies play crucial roles in protecting sensitive data, ensuring secure communication, and enabling the decentralized verification of transactions and for, building robust security solutions and harnessing the power of decentralized blockchain networks.

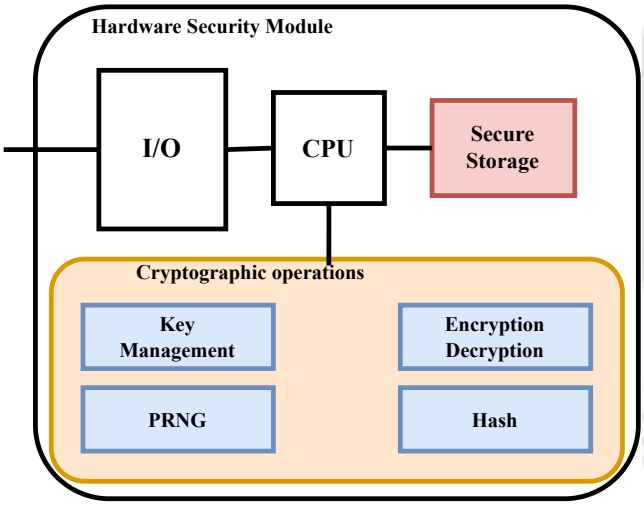


Figure 1.1: Hardware security module architecture

Cryptographic primitives are fundamental building blocks used in the construction of cryptographic systems. Research in security focuses on designing hardware security module architectures as shown in Figure. 1.1, which includes designing and analyzing encryption algorithms and protocols for various applications like secure communication, exploring techniques for key management and key exchange, hash functions, message authentication codes (MAC), digital signatures, and pseudo-random number generators (PRNG). Cryptography plays a vital role in the security and privacy of blockchain technologies and cryptocurrencies. Research is needed in designing secure and efficient transactions, privacy-preserving mechanisms, consensus algorithms, and smart contract security. Our research focuses on designing cryptographic algorithms for applications in cloud computing, wireless communications and

blockchain mining processes. However, complexity is a major concern for the implementation of cryptographic primitives. Focusing primarily on reducing the design complexity might lower the system's throughput without attaining the desired performance. Possible solutions might include reducing the complexity, which can reduce the power and energy consumption of the cryptographic primitives. In this thesis, four cryptographic primitive architectures based on different encryption algorithms are proposed. Our tasks included choosing the appropriate cryptographic algorithm, efficient implementation on FPGA, and realizing its ASIC layout for low-complexity, energy-efficient and high throughput design architectures.

1.1 Motivation

The implementation of cryptographic algorithms holds significant importance in today's digital landscape. As technology advances and our reliance on digital systems grows, the need for robust and secure information protection becomes increasingly vital. Cryptographic algorithms are the backbone of secure communication, data privacy, and integrity verification. By leveraging robust cryptographic algorithms, organizations can establish a secure and trustworthy environment, protect sensitive information, meet regulatory requirements, and instil confidence in users and customers. The implementation of cryptographic algorithms serves as a cornerstone for building a secure digital landscape in an increasingly interconnected world. Stream and block ciphers are cryptographic algorithms employing pseudo-random numbers to generate large keystreams, making the cryptosystems resilient against attacks. The block ciphers are bit-oriented, whereas the stream ciphers are word-oriented. These well-known stream cipher schemes, like SNOW, are used to secure wireless, IoT or Bluetooth devices. Maximizing the trade-off between security and performance on the hardware and software platforms is essential for an efficient cryptographic cipher scheme.

Further, secret sharing schemes can be used in distributing cryptographic encryption keys among multiple people to enhance security. Secret sharing is also used for distributed backup storage of cloud data to improve data security and prevent data loss in cases of failure of multiple nodes. Therefore, efficient linear and nonlinear secret-sharing scheme designs are essential in the distributed network cases.

The security and integrity of any blockchain are hinged on an enumerative algorithm for mining, which blocks the multiple expenditures of bitcoins and meddling with established transactions due to

asymmetric encryption, hashing and other algorithms used in blockchain. Currently, the blockchain process (mining) is performed using high-powered GPUs and CPUs, which consume a lot of energy and produce an enormous amount of greenhouse gases that can harm the environment. Thus, our focus is to design an ASIC, which can perform the blockchain mining process efficiently instead of bulky compute devices like GPUs.

1.2 Research Objectives

The main objective of the research is to develop various cryptographic primitives and implement them over FPGA and ASIC. All the objectives are given below in the chronological order.

(i) Design and implement various cryptographic algorithms to find suitable algorithms for cloud computing and blockchain technology.

(ii) Proposing fast and efficient modified SNOW stream ciphers:

- SNOW 2.0 stream Cipher.
- SNOW 3G stream Cipher.

(iii) Implementation of low power and resource-efficient Secret Sharing Schemes:

- Designing fast and efficient Linear secret sharing scheme.
- Implementing efficient Nonlinear secret sharing scheme.

(iv) Design of Blockchain Mining Algorithms for FPGA and ASIC Implementation.

1.3 Organization of the Thesis

The thesis is divided into seven chapters, and the construction of each chapter is given below.

The **first chapter** is the introduction part, where the security challenges and motivation to address them are discussed briefly.

The **second chapter** discusses the evolution of cryptographic algorithms and their applications in IoT. The impact on performance due to the evolution of security algorithms is discussed in this chapter. Further, the design of a suitable algorithm is elaborated that can be used in IoT applications.

Logic gate-based modified SNOW stream ciphers are discussed in the **third chapter**. These optimally designed stream ciphers are first implemented over a hardware (FPGA) platform and compared

with contemporary stream cipher implementations for various parameters. Verilog HDL is utilized to implement these stream cipher schemes on the FPGA platform. Further, an ASIC implementation of the proposed designs is presented.

The **fourth chapter** proposes a hardware design technique for nonlinear secret sharing schemes that can be used in dynamic environments where the number of players is constantly changing. The proposed design technique uses a modular arithmetic block as the primary computational block, and the Barret reduction and Montgomery reduction techniques are utilized to reduce the computation time. This design also proposes a scheme for modular inverse employing Gauss-Jordan elimination algorithm, which can be parallelized to reduce time complexity. The proposed design technique is scalable and can be implemented efficiently on FPGA and ASIC platforms.

In the **fifth chapter**, a novel blockchain mining encryption scheme based on the Hash function is discussed in detail. The mathematical construction, hardware (both FPGA and ASIC) implementation and performance analysis are depicted in the chapter.

The **sixth chapter**, reviews the various approximate multiplier designs. These multipliers are simulated, and performances are analyzed to determine a specific application multiplier. As multipliers are the integral and resource-consuming part of an architecture, replacing the traditional multiplier with the approximate multipliers makes any design resource-efficient provided it is error-resilient.

The **seventh chapter** concludes the work proposed in the thesis and narrates the future scope and perspectives of the proposed schemes.

2

Cryptographic Algorithms

Contents

This chapter compares various algorithms for application in the Internet of things (IoT). This technology contains a vast network of systems, sensors, and products, taking advantage of its low computing power and miniaturization of electronic networks and interconnected networks to provide extended capabilities that were not possible with the previously-used technology. Many IoT devices per the customer's use, such as Internet-enabled appliances, automation components, and management devices, are helping to lead the vision of a "smart life". This chapter aims to find out and implement cryptographic algorithms that can be used in various IoT applications. For the hardware implementation, a XILINX FPGA ZedBoard was used.

2.1 Introduction

It is the inherent need of the human beings to communicate data and share the information but selectively, and it gives rise to the art of coding the messages so that only the intended person gets to access the data or information or even if the messages in the scrambled form reach the unauthorized person, he could not be able to extract any information. This gave birth to the concept called Cryptography.

Cryptography is the art that can be used to obscure information to make the data secure. The word Cryptography originates from the combination of two greek words," Krypto"means *hidden* and "graphene" means *writing*, coined together to form the hidden writing. The Cryptography process provides confidentiality, data integrity, authentication, and no repudiation. Cryptography has been used to secure communication for centuries. Due to the worldwide growth of digital networks, many applications are using Cryptography to ensure their data, for example, in WLAN, WSN, and digital cards. Therefore, it is becoming crucial to generate low-power, efficient, and secure Cryptographic algorithms. Moreover, there are many cryptographic algorithms that are widely available and in use to secure the information.

The idea of connecting computer networks has a long history, but the term "Internet of Things" is a relatively recent addition. For example, connecting the electrical grid through the telephonic lines to monitor it remotely was already in use by the late 70s. By the 1990's progress in technology made machine-to-machine solutions and enterprise for operation and watching common. IoT is a vast network of devices and objects connected to the Internet through a router and network devices. IoT has created various opportunities as well as challenges across the world.

2.2 Related Work

The introduction of the new design either aims towards low memory consumption or high frequency or the optimization between both parameters. Security algorithms like AES can be implemented on Field Programmable Gate Array (FPGA) [3] [4] [5] [6] [7] [8] [9] [10] [11]. In [12], DES ECC algorithm is implemented and the FPGA Implementation of Pipelined Blowfish algorithm was done by S. Roy et al. [13]. The implementation of these encryption algorithms is proposed in terms of resource, and power optimization is implemented, while some focus on reducing area occupancy. Fast, resource efficient, and lightweight ECC algorithms are implemented for various applications like web sensors, IoT, wireless sensor networks [14] [15] [16] [17] [18] [19] [20] [21].

The implementation of advanced encryption standards (AES) on FPGA was efficiently done by Sridevi et al. [8]. The number of modes the AES performs is four, and these are cipher block chaining, output feedback mode, electronic code block, and Cipher feedback block. The variable key size of 128, 192, and 256 bits can be used for the plain text size of 128 bit with the number of rounds 10, 12, and 14, respectively. Sub bytes, shift rows, mix columns, and addroundkey are the steps involved in the algorithm. A Silicon platform was used to implement the AES, and it was pipelined to be implemented on the FPGA kit. T box was used to replace the S box, which in turn was also pipelined, showing higher throughput on the FPGA kit.

S. Roy et al. [13] proposed a pipelined structure for blowfish. Key size of 32 bits is used for the plain text of 64 bit of size. It also uses the Feistel network structure with the number of rounds to be 16 for improved Security, it provided that the throughput of the pipelined blowfish is less than that of non-pipelined blowfish. Blowfish shows better performance than the already existing AES and DES algorithm.

Reza et al. [11] Proposed a 2-slow retiming technique for the AES algorithm and implemented it on an FPGA kit. They proved that throughput could be increased by Pipelining concept, Rolling concept, and Sub stage pipelining concept. They also proved that the 2-slow retiming technique is better than the c-slow retiming technique. The 2-slow retiming technique breaks the critical path into the number of pieces. Due to that, registers are increased. To overcome the more number of registers, the author approached the multiplication process in the third stage of the AES algorithm.

Miguel et al. [18] presented a FPGA prototyped low power hardware accelerator scalar multiplier in elliptic curve cryptography (ECC), which provides security services in the IoT, such as confidentiality

and authentication. Their design is area efficient than other FPGA architectures and targeted elliptic curves defined over binary fields generated by trinomials. They used the IoT MicroZed FPGA board to deploy their ECC hardware accelerator, validated under a hardware/software codesign of the Diffie-Hellman key exchange protocol (ECDH).

Hamad et al. [20] implemented ECC, a symmetric key algorithm performed on the elliptical planar curve over the finite field and depended on point multiplication, which is followed by a series of addition and point doubling. The problem addressed is determined by the size of the elliptical curve. Kartusube-Hoffman method and modular arithmetic operation were performed to reduce the memory size. Higher throughput and low memory space are achieved by this.

Lightweight ECC for Internet of Things (IoT) to provide better security was proposed by He. and Sherli [21] analyzing the three schemes of existing cryptography algorithm and insisted that for IoT, lightweight schemes are suitable.

Due to the small size of the IoT devices and limited resources, there is a need to develop an algorithm that not only provides the maximum security but with as low as possible use of the hardware, the primary goal is ensuring the authentication and integrity of the message so that the message received by the user can be trusted, and to make sure the data is not being modified.

Standard cryptographic techniques are handy and essential tools to achieve the integrity goal. To ensure the Security of the system, its really important to have a good selection cryptographic algorithm. The choice of the cryptographic algorithm also depends on the application area, as one solution cannot be adapted everywhere. For example, it is not possible to implement schemes for complex Security in IoT applications where the main components are the sensors; therefore, feasible and flexible protocols are required to be adapted for different situations.

Millions of devices are connected to the IoT network, which in turn increases the number of communications that are taking place, leading to the issue of scalability and engineering. With this many devices connected to the network, another problem arises the issue of the cloud, making the Security of the devices more difficult. Distributed ledger maintaining a growing number of transactions and data records is called Blockchain, recording the blocks of transactions relating to the network participants, ensuring the correct sequences.

2.3 Architecture of Cryptographic Algorithms

Different Block ciphers are derived from the fiestel structure. In the encryption of the fiestel structure, the plain text is divided into two halves, the left and right. These two halves then go into the next step of the fiestel structure, called the rounds. In the first round, some special function is performed on the right half of the plain text in which the security key is added to it; then the output is XORed with the left half of the plain text, which is then sent to the next round as the right half. Also, the right half of the plain text is transferred to the next round as it is and as the left half to that round, and then the same operation is performed as above. The round Function is performed a number of times, and different is added for each round. After the last round, the cipher text is produced as shown in Figure. 2.1.

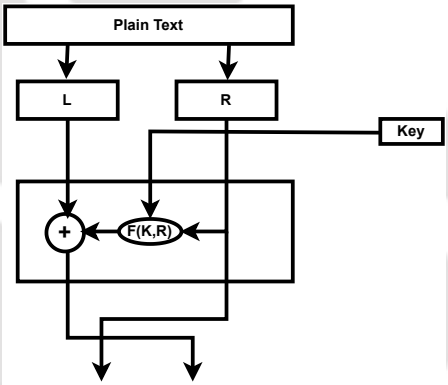


Figure 2.1: Fiestel Structure Block Diagram.

2.3.1 Advanced Encryption Standard

This algorithm is based on a permutation-substitution network. It consists of layers of operation which may involve exchanging input by some certain outputs(substitution)and shuffling the bits(permutations). AES algorithm does all its operation on bytes by treating 128 bits as 16 bytes.These bytes are processed as a matrix arranged in four rows and four columns. The block diagram of the AES algorithm is depicted in Figure. 2.2. The length of the key decides the number of rounds in the AES algorithm, for 128 bit key - 10 rounds, 192 bit key - 12 rounds, 256 bit key - 14 rounds.

Each round in the Encryption process comprises four sub-rounds, as discussed below,

- **Byte Substitution** The input is of 16 bytes, and in this process, each bit is substituted by looking into the table specified in the design, which in turn results in $4X4$ matrices.

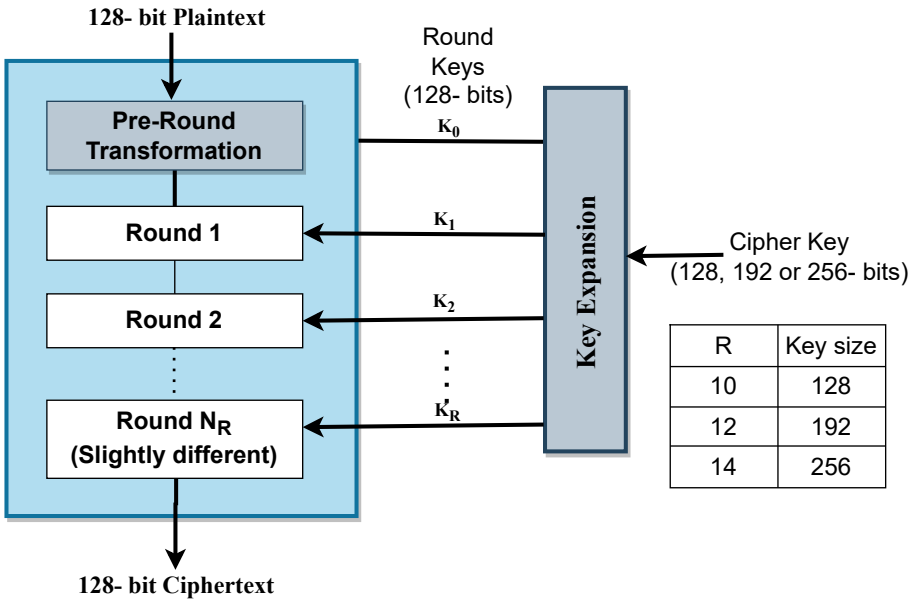


Figure 2.2: AES Algorithm Block Diagram.

- **Shift Rows** Each row is shifted towards the left as per the row number; any entries that fall off are reinserted on the right side.

  (i) first row - No Shift

  (ii) Second Row - Shifted towards left by one position

  (iii) Third Row - Shifted towards left by two positions

  (iv) Fourth Row - Shifted towards left by three positions

- **MixColumns** The input to this block is now each column of four bytes which is transformed to a set of new bytes using a unique mathematical function to replace the previous entries of the column, which in turn results in a matrix comprising of 16 new bytes. This round is not performed in the 10^{th} round.

- **AddRoundKey** This is the block where the key that we were talking about comes into play; as we have mentioned above, the 128 bits round key is used and is XORed with a matrix of 16 bytes which are considered as 128 bits here in this block. These 128 bits are now again interpreted as 16 bytes to be given as input to the next round, or this can also be the output cipher text if it is the last round.

2.3.2 Data Encryption Standard

Published by the National Institute of Standards and Technology (NIST) in 1975, it was a symmetric key block cipher. It is derived from the Feistel structure.

If the block size of 64 bit, i.e., at a time, a 64 bit plain text is converted to the cipher text. The number of Feistel structure rounds used in this process is 16, and the length of key size is 64 bit, but effectively only 56 of its bits are used.

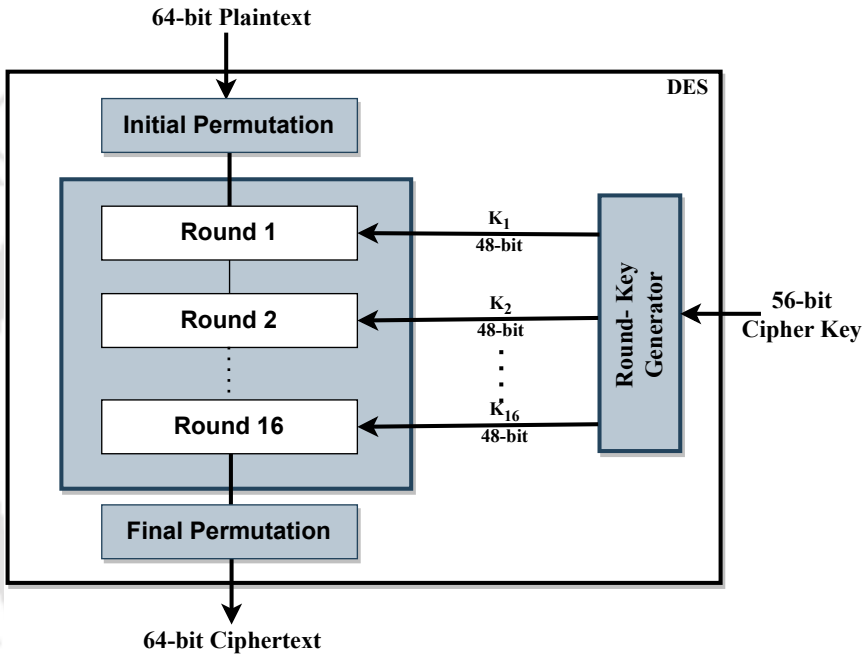


Figure 2.3: DES Algorithm Block Diagram.

The block diagram of the DES algorithm is depicted in Figure. 2.3. The essential steps for DES are Round functions, Key structure, and initial and final permutations. The permutation boxes are added to make the algorithm stronger; in this, the bits are permuted randomly, where the bits of the plain text is rearranged. The initial and final permutations are inverse of each other. The round Function is the heart of the DES algorithm; in the DES Algorithm, there are 16 round functions in which some special tasks are performed. In this, a 32 bit output is produced through the rightmost 32 bits by applying a 48 bit key. In the Permutation Box, the Right input needs to be expanded from 32-bit to 48 bits since the key is 48 bits.The expanded correct plain text is XORed with the round key after the expansion is performed. The S-boxes carry out the actual mixing, 8 S boxes are used in DES, and each s box is input with 6 bits and produces 4-bit output. Therefore the output is a total

of 32 bits.

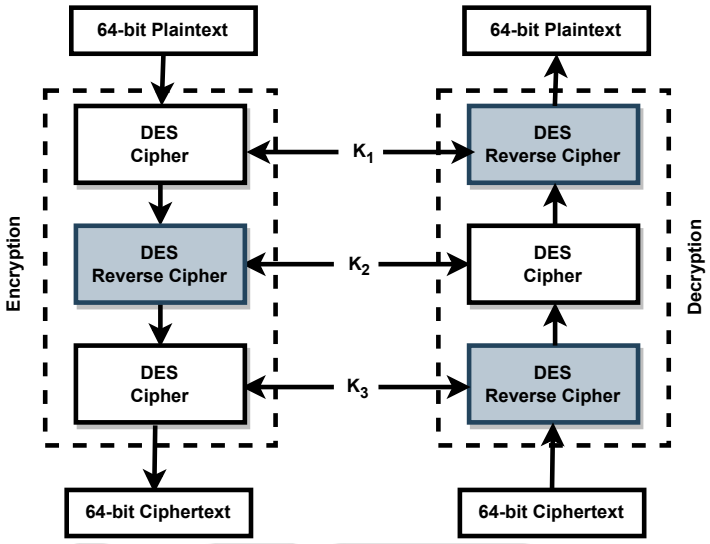


Figure 2.4: 3DES Algorithm Block Diagram.

2.3.3 Triple Data Encryption Standard

In the triple DES algorithm as shown in 2.4, first, the encryption is performed as per the above DES algorithm using key1, then decryption is performed using the key2, and again the decryption is performed using the key3 to produce the 64 bit cipher text.

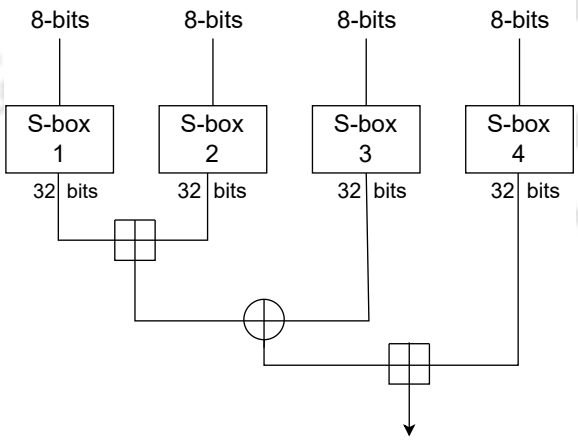


Figure 2.5: blowfish Algorithm Block Diagram.

2.3.4 Blowfish Algorithm

Blowfish contains a block size of 64 bit, and the key length can be any length from 32 bits to 48 bits.It has a huge dependency on the S boxes and follow the 16 round Feistel structure.

Each line in the Blowfish algorithm represents the 32 bits; two subkey arrays are used in this algorithm: four 256 entry S boxes and 18 entry P-array. Input is accepted in the form of 8 bits in the S boxes, and the output is produced in 32 bits. Every round uses one entry of the P array, and after the final round, every half of the data block is XORed with one of the remaining unused P arrays.

Four 8-bit quarters are formed by splitting the 32 bits input, and each quarter is input to the S-Boxes. Modulo 2^{32} is added to the outputs and XORed, which then produces the final output of 32 bits as shown in Figure. 2.5. Due to the Feistel structure of the blowfish algorithm, it is inverted by using P_{17} and P_{18} to the cipher text block.,then using the reverse-ordered P-entries.

2.3.5 RSA (Rivest Shamir-Adleman) Cryptosystem

It was invented in 1977 and is named after Ron Rivest, Adi Shamir, and Len Adleman. RSA is the world's most widely used public key cryptographic algorithm. It can be used not only for public key encryption but also for digital signatures. It is based on the principle that it is easy to calculate the product of two big prime numbers. Still, the inverse is very difficult, i.e., it is nearly impossible to factorize a huge number to its factors which are prime.

Generation of RSA key pair:

- Generate the RSA modulus(n):

  Select two large primes, a and b. Calculate the product of these two numbers, $n = a*b$, n should be a minimum of 512 bits for unbreakable solid encryption.

- Derive number (e):

  A number e is chosen such that it is greater than 1 and less than $(a-1)(b-1)$. e and $(a-1)(b-1)$ must not have any factor in common except for 1,that is these two numbers should be co-prime.

- Public key formation:

  RSA Public key is formed by the pair of numbers (n, e) and is made public. The public key has a part n in it, which is difficult to factorize. The RSA algorithm's strength is that it is difficult to find two primes (a and b) used to obtain n.

- Generate private key:

A unique number d is calculated using a, b, e which is the private key. It inverse of e modulo $(a-1)(b-1)$. This relation is written mathematically as:

$$ed = 1 \mod (a-1)(b-1) \tag{2.1}$$

RSA Encryption:

When the sender sends the text message to someone whose public key is (n, e), it is represented as the series of numbers less than n by the sender. To encrypt the plaintext P, which is a number modulo n, a simple the mathematical step is used to represent the encryption process:

$$C = P^e \mod (n) \tag{2.2}$$

RSA Decryption: It is a straightforward process when the cipher text C is received by the receiver of the public key (n, e). Cipher text C is raised by the power of the private key d, and then the plain text is obtained by modulo n the result as:

$$P = C^e \mod (n) \tag{2.3}$$

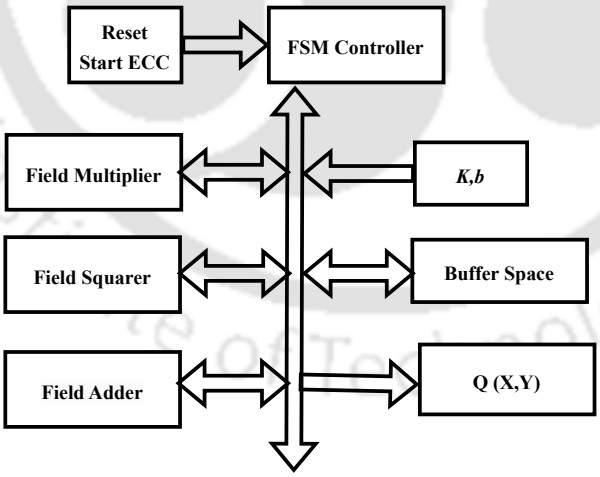


Figure 2.6: ECC Algorithm Block Diagram.

2.3.6 Elliptical Curve Cryptography

The working principle of Elliptical Curve Cryptography (ECC) is the Trapdoor function. We can get the same level of Security by using the 256-bits key size for ECC as we will get from the 3072-bits of the key length of RSA. US Government's top secrets are secured using the 384 bits of the key length

of ECC. The block diagram of ECC algorithm is given in FIgure. 2.6.

The curve is represented by the following;

$$y^2 = x^3 + ax + b \tag{2.4}$$

The number of points N is bounded by:

$$p + 1 - 2\sqrt{p} <= N <= p + 1 + 2\sqrt{p} \tag{2.5}$$

The elliptical curve should follow the equation;

$$4a^3 + 27b^2! = 0 \tag{2.6}$$

to have the curve 3 different roots. The curve requires its own algorithm for addition and multiplication. If we draw a straight line through the curve, it will intersect no more than 3 points. The elliptical curve is symmetrical about the x-axis as shown in Figure. 2.7.

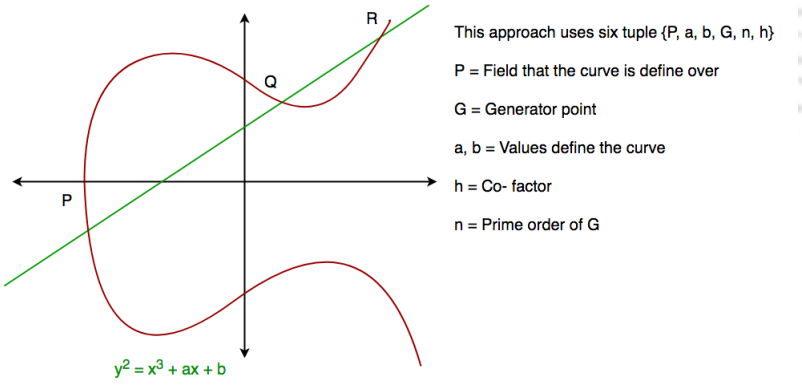


Figure 2.7: ECC Algorithm.

2.3.6.1 Addition in ECC

If we want to add the two points to the curve, a straight line is drawn through these two points, which intersects the curve at the third point, and a vertical line is drawn, which intersects the curve at another point. This point gives the addition of the given two points as shown in Figure. 2.8. As we know that multiplication is just repeated addition, we can perform the multiplication in the same way.

ECC gets its strength from this repetitive multiplication; we can get to the final point from the

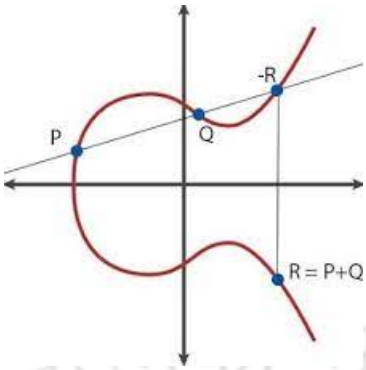


Figure 2.8: Addition in ECC.

initial point in an easy manner, but it is challenging to find the initial point from the final point. For it, we have to calculate all the points until we find the desired point, but it is nearly impossible when we are using sufficiently large values; it will take a a reasonable amount of time due to increased computational complexity that means encryption is easier but decryption is difficult.

2.3.7 Diffie-Hellman key exchange algorithm

Diffie-Hellman is a key exchange algorithm that is exponential, which allows the users to exchange a secret key without the requirement of any prior secrets in real time over an unsecured network, discrete computing logarithm for a large number is the principle behind the Diffie Hellman key exchange algorithm. There is not any successful attack found against this algorithm. It requires one prime number and one of its primitive roots; both should be very large, and the prime number should be at least 512 bits. This algorithm can be explained in the Figure. 2.9.

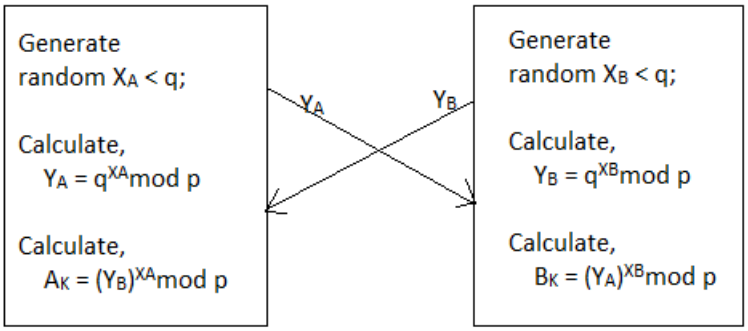


Figure 2.9: Diffie Hellman key exchange algorithm.

The public values can be computed by :

$$x = g^a \mod (p) y = g^b \mod (p) \tag{2.7}$$

where a and b are user picked values, x and y are values to be exchanged. The private key can be computed as:

$$ka = y^a \mod (p) kb = x^b \mod (p) \tag{2.8}$$

Algebraically it is seen that the user has a symmetric secret key to encrypt i.e.

$$ka = kb \tag{2.9}$$

2.3.8 Elliptical Curve Cryptography with Diffie-Hellman key exchange

ECCDH is the modified version of Diffie Hellman of passing the ECC keys; a new public key is generated by passing an old public key and then developing a shared public key. Calculations agreed upon elliptical curve is the basic idea on which the ECCDH public key algorithm. The block diagram of the ECCDH algorithm is given in Figure. 2.10. The point on the curve is taken as the starting point, which in turn derives a random number for the private key, and the product of all those values is sent to the other users. The curve and the starting point are known to the receiver, common product for both the user can be derived by using the sender's product which can be used as a key.

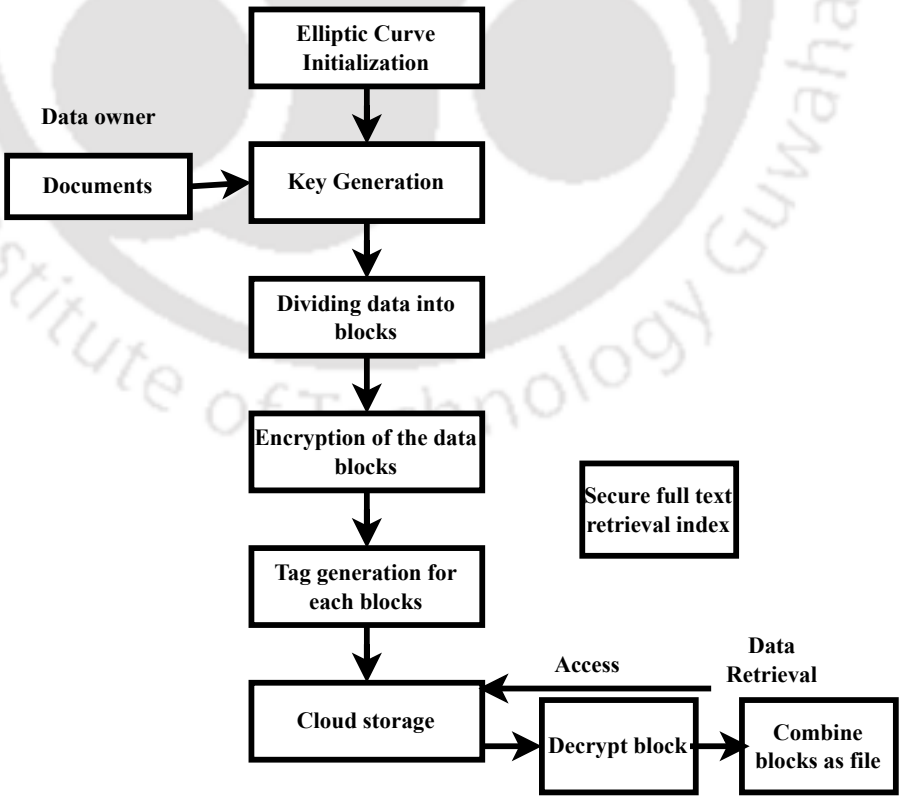


Figure 2.10: ECCDH Algorithm Block Diagram [1]

The key agreement in ECCDH can be understood as follows:
The prime number a, p, q are selected by both the users, which satisfies the elliptical curve group $E_a(p, q)$. The elliptical curve equation is;

$$y^2 = x^3 + px + q(mod(a)) \tag{2.10}$$

For example, elliptical curve group E_{211} created by $a = 211, p = 0, q = -4$ and the elliptical curve $y^2 = x^3 - 4$. A common coordinate is selected by users on the curve less than p. for example, $P(2, 2)$ is generated by using $x = 2$ and $y = 2$. This point p is the generator point in the Diffie- Hellman algorithm. A random number d (less than p) is selected by the sender, multiplied by the generator point P to generate the multiple points Q on the curve $Q = dP$. The sender sends the value Q, the public key, to the receiver. $S = dQ$ is computed by the receiver with the Q received from the sender. Now the sender and the receiver have the shared and the public key.

The algorithm can be used in the applications in which the bits for the Encryption and Decryption are shared between the devices and the cloud. For any encryption or decryption, the bits from the cloud from that particular device as well as the cloud, are required. If we are using the n-bit key, we can store x-bits in the device and other $n - x$ bits in the cloud. When the cloud and the device want to communicate, bits from both are required for the encryption and decryption of the text.

Table 2.1: Comparison of various Cryptographic Algorithms.

Algorithm	BlockSize	Key Size	Resource (LUTs)	Power (in mW)	Frequency (in MHz)
AES	128	128	500	189	51.33
DES	64	64	731	135	140
3DES	64	64	1035	141	107.38
Blowfish	64	64	2540	121	71.14
RSA	256	256	25164	760	100
ECC	233	32	23510	391	212.9
ECCDH	233	32	1030	90	234

Table 2.2: Comparison with Existing Architecture.

Design	Resource (LUT or Slices)	Frequency (in MHz)
Morales-Sandoval et. al. (2021) [18]	1809	62.5
Our Design	1030	234

2.4 Summary

In this chapter, designing and implementing IoT security algorithms, various cryptographic algorithms were implemented on the XILINX FPGA ZedBoard to find the suitable algorithm for the IoT security applications. IoT-enabled products need the algorithm to take the least power and area possible; frequency should be high for faster operations. As inferred from Table 2.1 the power consumption in ECCDH (Elliptical curve cryptography using Diffie-Hellman algorithm) is reduced by approximately 52% and 77% compared to the power consumption in AES and ECC, respectively. The operating frequency in ECCDH is increased by 455% and 9% as compared to the operating frequency in AES and ECC, respectively. Moreover, it is the most secure algorithm to date, as no known attacks have been found against this algorithm. From Table 2.2, we can see that our design uses 43% less resource and 374.4% more frequency than the existing ECCDH architecture [18]. Therefore our proposed fast and resource efficient ECCDH algorithm is suitable for IoT applications. The above results lead us to conclude that ECCDH is superior to other algorithms in terms of area, power, and performance.

3

SNOW Stream Cipher Architectures

Contents

In today's world, smart devices and sensors, such as mobile phones, ATMs, and vending machines, are ubiquitous [22]. These devices generate and process vast amounts of data, which require secure storage and protection. However, a simple and cost-effective platform is needed to safeguard these networks and devices. Therefore, developing low-power, effective, and secure cryptographic algorithms are becoming increasingly crucial. This chapter presents a modified SNOW 2.0 stream cipher scheme to address the abovementioned issue.

3.1 Introduction

The SNOW 2.0 stream cipher is composed of an LFSR and an FSM. It involves the secret key and the initialization vector (InitVec) to generate keystream bits [23]. The plaintext is encrypted by applying a group operation between plaintext and keystream, resulting in the ciphertext as shown in Figure. 3.1. Modern cipher schemes typically perform this group operation using a bitwise XOR operation. Most streaming ciphers in the literature only focus on analyzing LFSR [24]. This chapter presents an efficient realization of FSM, especially S-Box, resulting in better resource utilization and lesser delay of the SNOW 2.0 stream cipher scheme.

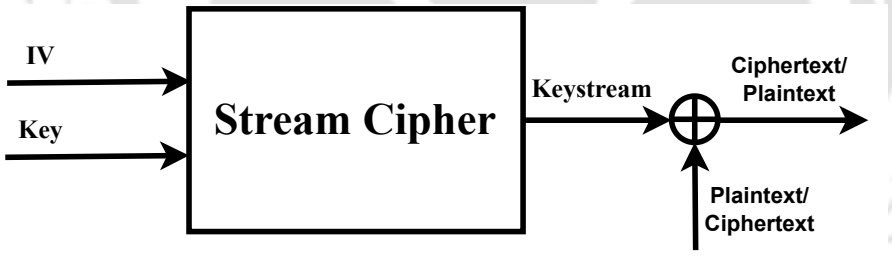


Figure 3.1: Stream cipher process

Improving the performance of a cryptographic scheme is essential for the security of IoT and wireless communication systems. Cipher schemes play a vital role in this context, and they leverage pseudorandom number generation to create extensive keystreams, providing the resilience of cryptosystems against various attacks. Block ciphers are bit-oriented, while stream ciphers are word-oriented cryptographic algorithms. These widely recognized schemes are employed to enhance the security of wireless, IoT, and Bluetooth devices. Achieving the optimal balance between security and performance across hardware and software platforms is crucial for an efficient cipher scheme. Block ciphers may not be suitable for software implementations due to their tendency to introduce significant delays. As a result, word-oriented stream ciphers find their utility in software platforms, facilitating the realization

of wireless standards such as IEEE 802.11b and Bluetooth.

The performance and area consumption of various stream ciphers, i.e. A5/1, E0, RC4 and Helix, are compared by Michalis D. et al. [25]. Further, the hardware efficiencies of standard block ciphers, such as AES, 3DES, Rijndael, Misty1 and a few stream ciphers like LILI-II, Helix, SNOW 2.0 are compared by L. Philippe et al. [26]. The FPGA implementations of four stream ciphers, MUGI [27], SNOW 2.0 [2], MICKEY 128 and TRIVIUM are proposed by [23]. S. Ashaq et al. depict a resource-efficient hardware architecture of S-Box for the PRESENT block cipher scheme [28]. Chen et al. propose an FPGA implementation of SNOW 2.0 using an area and power-optimal pipelined architecture. This design is realized on a Xilinx Virtex-6 FPGA exhibiting the throughput of 2.2 Gbps at 200 MHz. Lee et al. presents a hybrid FPGA/CPU implementation of SNOW 2.0 for wireless sensor networks.

This chapter presents a novel approach to implement the SNOW 2.0 stream cipher on the FPGA platform using hardware, where instead of using LUTs or ROMs for the S-Box implementation, logic circuits made up of boolean functions are utilized. The SNOW 2.0 stream cipher was implemented using 180nm ASIC technology through the synthesis tool provided by Synopsys. This marks the first-ever ASIC implementation of SNOW 2.0. Based on our findings, this SNOW 2.0 stream cipher implementation operates at a higher frequency and is more efficient than similar works previously reported.

3.2 SNOW Stream cipher Architecture

A stream-cipher is a text-based approach in which a cryptographic key and algorithm are added to each binary digit in the data stream, which is small at a time. This method is not widely used in modern cryptography. The primary approach is a cipher block where the key and algorithm are inserted into data blocks rather than individual bits in the stream. Stream cipher Converts plain text to cipher text by taking 1 Byte of plain text at a time.

SNOW, a stream cipher, was proposed in 2000, which has high-speed performance and implementation in hardware is also very fast. SNOW 1.0 is a word-oriented stream cipher with a word size of 32 bits [29]. It consists of a length 16 linear feedback shift register over $F(2^{32})$, feeding a finite state machine [30] as shown in Figure. 3.2. The FSM consists of two 32-bit registers. It suits IoT-enabled devices requiring less chip area and low-power cryptography design. Still, its vulnerability towards a

few algebraic attacks makes researchers design the SNOW version of SNOW, SNOW 2.0 by P. Ekdahl et al. [2], which is way faster in implementation and more secure than the original SNOW.

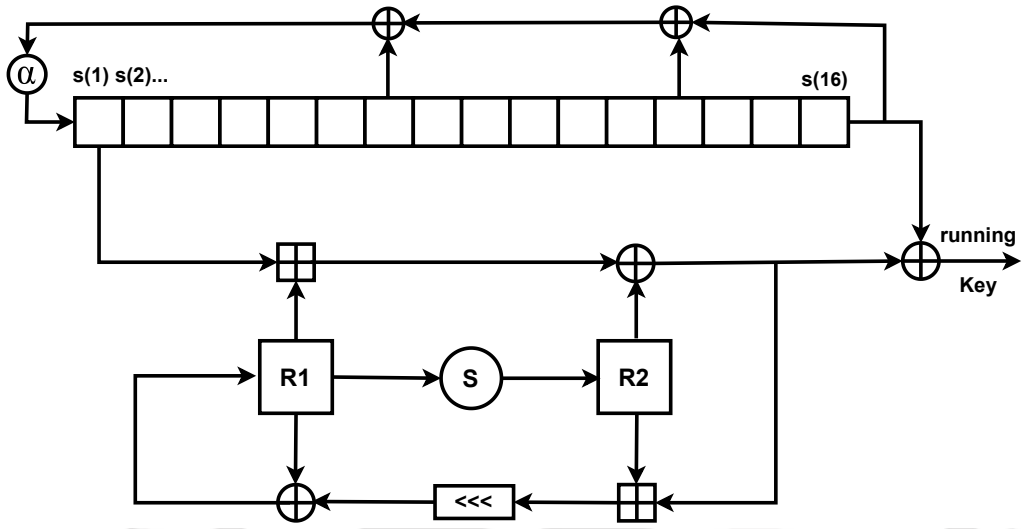


Figure 3.2: SNOW 1.0 Architecture.

SNOW 3G is a reliable and efficient stream cipher algorithm widely used in 3G and 4G cellular networks to provide secure communication over wireless channels [31]. SNOW 3G is designed to be implemented efficiently in hardware and software, making it suitable for low-power devices like mobile phones. SNOW V is a lightweight stream cipher for low-power devices such as wireless sensor networks, RFID tags, and smart cards [32]. It was developed by the same team that designed SNOW 3G but with a focus on efficiency and low power consumption. However, it may not be suitable for more demanding applications requiring higher security or larger key sizes.

SNOW 2.0 offers improved security, speed, flexibility, and standardization compared to SNOW 3G and SNOW V, making it a more robust and versatile stream cipher algorithm for general-purpose use cases. SNOW 2.0 is more secure against brute-force attacks. Additionally, SNOW 2.0 uses a different key schedule algorithm that provides better resistance against side-channel attacks. SNOW 2.0 can encrypt and decrypt data faster due to the use of a more efficient feedback shift register (FSR) design. SNOW 2.0 can be easily modified to support different key sizes, output lengths, and initialization vectors, making it more flexible and adaptable to other use cases. SNOW 2.0 has been standardized by the European Telecommunications Standards Institute (ETSI), which means that it has undergone extensive testing and review by experts in the field. While SNOW 3G is also standardized, it may be less suitable for general-purpose use cases due to its specific design for cellular networks.

The Stream cipher algorithm was designed in such a fashion that SNOW 2.0 get all the benefits of SNOW 1.0, and the weakness is somehow eradicated. In SNOW 1.0, multiplication can work in a single left shift of the word followed by XOR, which is possible with a known weight pattern 6. This means that the resulting word was often a change of first name. In SNOW 2.0, $F(2^{32})$ is defined as a field to add more $F(2^8)$, and Each multiplication can function as a byte switch once an unconditional XOR with 256 possible patterns. This improves the distribution of bits in the response loop and the resistance towards a particular attack of the encounter. The feedback loop also enhances the resistance of bitwise linear comparison attacks, as discussed above. There is no known mechanism to trick the polynomial of answers so that the multiplication of line results holds each position and has the correct minimum weight. Unconditional XOR also seems to improve speed by removing possible branch forecast errors in the pipeline processor.

FSM in SNOW 2.0 now takes two inputs and makes the guess-and-determine type of attack more difficult [33]. With the release of the FSM release, it's a plus with R1 and R2 it is no longer possible that we can directly output the next FSM state. The R1 update does not depend on the output of the FSM but on the name taken from the LFSR. This also suggests that the same phenomena as those in it will be very weak. The S box in SNOW 1.0 was also redirected, but the latter was slightly more compatible. In SNOW 1.0, for each installation, the S-box only touched eight pieces of the subtitle. New S-box selection, based on Rijndael's spherical performance [34] [35] [6] [28], provides a more robust distribution. Each issue now depends on each entry.

Several studies have explored different aspects of the SNOW 3G cipher, including its design, security analysis, and implementation in various communication systems. Johansson et al. provide an overview of the cipher [36], while Laurent et al. focuses on its security analysis, particularly its linearization method [37]. Conti et al. present an efficient FPGA implementation for 4G/LTE systems [38], and Xu et al. propose improvements for the cipher in 4G/LTE networks [39]. Ding et al. analyze the security of SNOW 3G in 5G NR systems [40]. Madani et al. propose an enhancement to SNOW-3G using a hyperchaotic generator to improve the randomness of its output keystream [41]. Shen et al. present a high-throughput SNOW3G-based design, achieving 10Gbps speed [42].

The SNOW3G single core produced by Helion company is integrated on the Quartus II 13.0 software platform and compared with Altera's Stratix III C2 and Stratix IV C2 FPGA devices [43]. An implementation of SNOW 3G on a graphics processing unit (GPU) [44] to accelerate the cryptographic

process is described by H.Liang et al. Ashaq et al. proposed a hardware architecture for the S-Box of the PRESENT block cipher [28]. Their goal was to reduce the resources required for the implementation, and they achieved this by optimizing the boolean expressions at the gate level for the S-Box.

3.2.1 SNOW 1.0

SNOW 1.0 is a word-oriented stream cipher with a word size of 32 bits. "This cipher is described with two possible key sizes, 128 and 256 bits. As we know in cryptography, the encryption of any data starts with a key initialization and gives the cipher its initial key values. In this description, we will only concentrate on the cipher in operation. The details of the key initialization will be explained later.

It consists of a length 16 linear feedback shift register over $F(2^{32})$, feeding a finite state machine. The FSM consists of two 32-bit registers, called R1 and R2, and some operations to calculate the output and the next state (the next value of R1 and R2). The operation of the cipher is as follows. First, key initialization is done.The LFSR has a primitive feedback polynomial over $F(2^{32})$ which is given by,

$$f(x) = x^{16} + x^{13} + x^7 + \alpha^{-1} \tag{3.1}$$

where $F(2^{32})$ is obtained from irreducible polynomial

$$p(x) = x^{32} + x^{29} + x^{20} + x^{15} + x^{10} + x + 1 \tag{3.2}$$

over F_2 and p(α)=0 let s(1), s(2), S(3)...S(16) belongs to$F(2^{32})$ be state of LFSR.

$$FSM_{out} = (s(1) \boxplus R1) \oplus R2 \tag{3.3}$$

The output of the FSM is XORed with s(16) to form the keystream, i.e.,

$$running\ key = FSM_{out} \oplus s(16) \tag{3.4}$$

The keystream is finally XORed with the plaintext, producing the ciphertext.
Inside the FSM, the values of R1 and R2 are given as follows,

$$newR1 = ((FSM_{out} \boxplus R2) \lll) \oplus R1, \tag{3.5}$$

$$R2 = S(R1) \tag{3.6}$$

$$R1 = newR1. \tag{3.7}$$

By the notation $x \boxplus y$, we mean the integer addition of x and y mod2^{32}. The notation x<<< is a cyclic shift of x 7 steps to the left, and the addition sign in $x \oplus y$ represents bitwise addition (XOR) of the words x and y.

3.2.2 Attacks on SNOW 1.0

SNOW 1.0 is designed to be secure as a 256-bit key cipher, but two different attacks have been presented. One is a Guess-and-Determine attack by [33]. This is a key recovery attack, and the complexity of the attack is O(2^{225}). Another attack is a distinguishing attack by Coppersmith et al. [45]. The complexity of this attack is about 2^{100}. The basic idea of this attack, developed by Goli'c, applies the technique of linear cryptanalysis on block ciphers for distinguishing the outputs of a KSG from a truly random bit sequence."

Guess and Determine attack- Malice user can guess the state value of LFSR; after that, determine the other state of LFSR based on earlier values. The attacker continues to determine the value until he gets the entire LFSR state; in the end, check whether the state is correct or not and produces the correct result or repeat this process. The data complexity of the attack is $F2^{95}$ words, and process complexity of $F2^{224}$ operations. Other than Hawkes and Rose made guessing the initial choice cleverly. Actually, two properties of SNOW 1.0 are used to reduce the complexity by which attack below exhaustive key search [33]. First, the FSM has only one input, s(1).

The second property leading to weakness is choosing the feedback polynomial in SNOW 1.0. The linear recurrence equation is given by;

$$s_{t+16} = \alpha(s_{t+9} + s_{t+3} + s_t) \tag{3.8}$$

There is a distance of 3 words between s_t and s_{t+3} and a distance of $6 = 23$ between s_{t+3} and s_{t+9}. Thus, by squaring the previous equation, we will get the following;

$$s_{t+32} = \alpha^2(s_{t+18} + s_{t+6} + s_t) \tag{3.9}$$

where $s_{t+i}+s_{t+i+6}$ can be considered as one; the sum of both is the only unknown; the attacker doesn't need to find both explicitly.

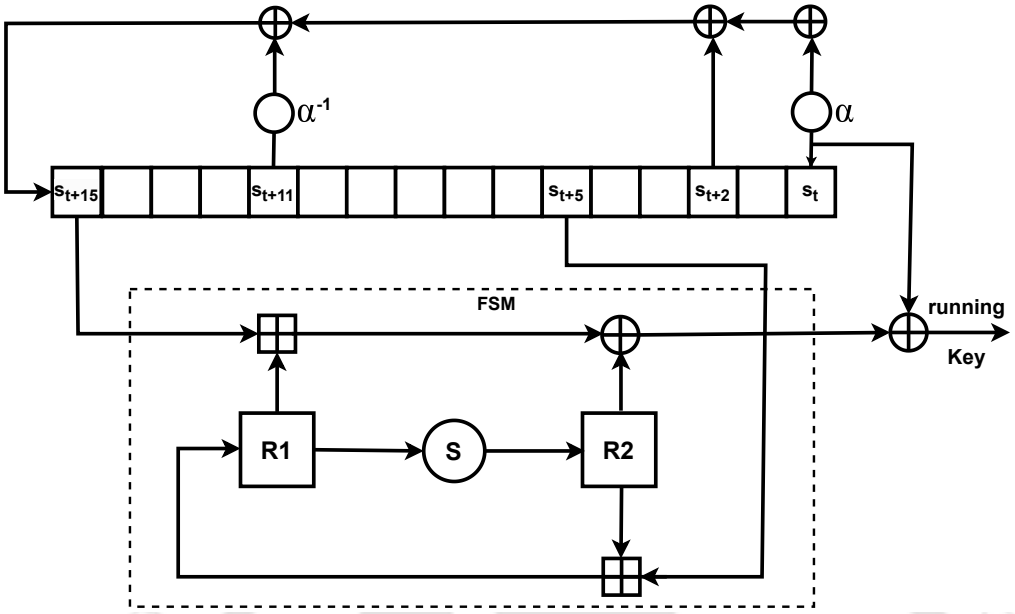


Figure 3.3: SNOW 2.0 Architecture.

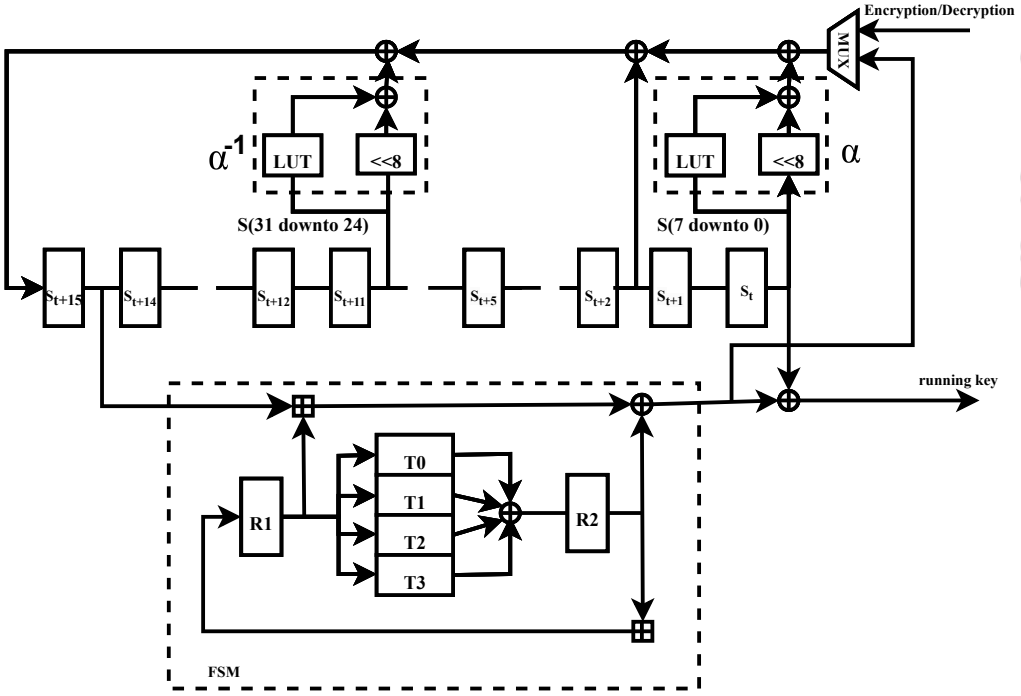


Figure 3.4: Hardware Architecture of SNOW 2.0.[[2]]

3.3 SNOW 2.0 stream cipher Architecture

SNOW 2.0 is very similar to SNOW 1.0, as shown in Figure. 3.3. Still, they changed the polynomials and the two input words in FSM they took from LFSR. The size of the words remains the

same as 32 bits, the LFSR size is also the same as 16, and the key obtained by XOR Functional output of FSM output and LFSR final object, as in the previous version. The hardware architecture design of SNOW 2.0 is given in Figure. 3.4. The cipher operation is as follows- first, we make the key initialization. This also uses the internal FSM register R1 and R2, its initialization value, and LFSR with the state. After that, the cipher will be restored to the first keystream we called out_1. And we get the second key S_0. The main reason why the polynomial of a particular response is selected in SNOW 1.0 is to get the software to understand quickly. We can see the multiplication by shifting one left and the known pattern possible with XOR by multiplying the original objects as the base is made. However, this choice unlocks potential weaknesses. In SNOW 2.0, two variables are involved in the response line, and 1 is now the root of the first phase of the polynomial of degree 4 over $F2^8$. For greater clarity, the SNOW 2.0 polynomial is given by

$$\pi(x) = \alpha x^{16} + x^{14} + \alpha^{-1}x^5 + 1 \in F_{2^{32}}[x] \tag{3.10}$$

Where α is a root of $x^4+\beta^{23}x^3+\beta^{245}x^2+\beta^{48}x+\beta^{239} \in F_{2^8}[x]$,
β is a root of $x^8+x^7+x^5+x^3+1 \in F_2[x]$. The value of the registers $R1$ and $R2$ at time $t > 0$ are denoted R_1t and R_2t, respectively. FSM input (s_{t+15}, s_{t+5}) and the output of the FSM, denoted F_t, is calculated as

$$F_t = (s_{t+15} \boxplus R1_t) \oplus R2_t, \qquad t \geqslant 0 \tag{3.11}$$

and the keystream is given by,

$$z_t = F_t \oplus s_t, \qquad t \geqslant 1 \tag{3.12}$$

Here we use the $\boxplus$ notation for integer addition modulo 2^{32} and XOR for bit-wise addition. The registers R1 and R2 will be modified with new data accordingly.

$$R1_{t+1} = s_{t+5} \boxplus R2_t \tag{3.13}$$

and

$$R2_{t+1} = S(R1_t) \qquad t \geqslant 0 \tag{3.14}$$

3.3.1 Sbox

The Sbox S(w) is a permutation on F(2^{32}) based on the round function of Rijndael. "Let w = (w_3, w_2, w_1, w_0) be the input into the Sbox, where w_i, i = 0...3 is the four bytes of w. Assume w_3 is the most significant byte, and

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} \tag{3.15}$$

is the vector representation of the input into the S-Box. First, we apply the Rijndael Sbox, denoted S_R, to each byte, giving us the vector.

$$\begin{bmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_3] \end{bmatrix}$$

In the MixColumn transformation of Rijndael's round function, each 4-byte word is considered a polynomial in y over $F(2^8)$, defined by the irreducible polynomial

$$x^4 + x^3 + x^1 + 1 \in \ F_2[x] \tag{3.16}$$

Each word can be represented by a polynomial of at most degree 3. Next we consider the vector as representing a polynomial over $F2^8$ and multiply with a fixed polynomial $c(y) = (x+1)y^3 + y^2 + y + x \in F_2^8$[y] modulo $y^4 + 1F_2^8[y]$. This polynomial multiplication also (as done in Rijndael) be calculated as matrix multiplication,

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \begin{bmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_2] \\ S_R[w_3] \end{bmatrix} \tag{3.17}$$

where (r_0, r_1, r_2, r_3) are the output bytes from the S-box. These bytes are concatenated to form the word output from the S-box, r = S(w).

3.3.2 Proposed modified architecture of SNOW 2.0 stream cipher

We first start the implementation of LFSR's build with 16, 32-bit registers with the feedback polynomial over $F(2^{32})$, the feedback taken from s_t with multiplication of α, s_{t+2} and s_{t+11} but multiplied with α inverse XoRed and feeded back to the s_{t+16}.The multiplication with α *and* α^{-1} are implemented with LUTs and 8-bit left or right shifting, respectively. The values of α *and* α^{-1} are computed by the following equations.

$$\alpha = (w \ll 8) \oplus \text{XORMUL}_{\alpha}[w \gg 24] \tag{3.18}$$

$$\alpha^{-1} = w \gg 8 \oplus \text{XORMUL}_{\alpha^{-1}}[w] \tag{3.19}$$

The secret key is either 128 or 256 bits and a publicly known 128 bit initialization variable IV. The IV value is four-word input IV = (IV3, IV2, IV1, IV0), where IV0 is the least significant word. The range for IV will be 0 .... $2^{128} - 1$. That means that the secret key provided for k, SNOW2.0, uses a function that extends the pseudorandom length from a set of IV values to a possible output. Using an IV value is not mandatory, and applications requiring an IV are specified and reuse the cipher more often with a fixed key, but the value of the IV has changed. This can be the case if two people have agreed on a common secret key but want to send multiple messages; for example, it can be given while using it in a frame-based setting .

The key initialization is done as follows.The registers in the LFSR named by , (s_{15},s_{14},...,s_0) from left to right, Thus, s_{15} corresponds to the element which has s_{t+15} during the normal operation of the cipher. lets assume the secret key be denoted as k = (k_3,k_2,k_1,k_0) in the 128 bit case and by k = (k_7,k_6,k_5,k_4, k_3,k_2,k_1,k_0) in the 256-bit case, where each k_i is a word and k0 is the least significant word. First, the shift register is initialized with k and IV according to the operation given below;

$s_{15} = k_3 \oplus IV_0$	$s_{14} = k_2$	$s_{13} = k_1$	$s_{12} = k_0 \oplus IV_1$
$s_{11} = k_3 \oplus 1$	$s_{10} = k_2 \oplus 1 \oplus IV_2$	$s_9 = k_1 \oplus 1 \oplus IV_3$	$s_8 = k_0 \oplus 1$
$s_7 = k_3$	$s_6 = k_2$	$s_5 = k_1$	$s_4 = k_0$
$s_3 = k_3 \oplus 1$	$s_2 = k_2 \oplus 1$	$s_1 = k_1 \oplus 1$	$s_0 = k_0 \oplus 1$

where **1** denotes all one vector (32 bits).

(i) Procedure to compute the cipher text of the input.

- Step 1- First, the key initialization is done, then the LFSR has been initialized, and two registers, R1 and R2 of FSM, have been put to zero.
- Step 2- Now stream cipher runs 32 times, and it doesn't produce any output, but output from FSM is fed again into LFSR.
- Step 3- The actual work of the cipher starts.

The next element, which is inserted later into the LFSR, is given by

$$s_{t+16} = \alpha^{-1} s_{t+11} \oplus s_{t+2} \oplus \alpha s_t \oplus F_t \tag{3.20}$$

Table 3.1: Common Subexpressions for Gate-level S-Box.

Factor	Expression
F_1	$(C' + B)$
F_2	AB
F_3	$A'C$
F_4	$A \oplus D$

Table 3.2: Final Expressions for Gate-level S-Box.

Output	Expression
S_3	$[D'F_1 + F_2 + F_3D)] \oplus A$
S_2	$[A'(C' + D) + B'D] \oplus B \oplus C$
S_1	$[DF_1 + F_2 + F_3D'] \oplus F_4$
S_0	$BC' \oplus F_4$

After 32 clocks, the cipher was moved back to regular operation and was introduced once before the first sign of the complex. The high number of allowed keywords is set to 2^{50}; then, the cipher

must be rewritten. This limit provides cryptographic analysis and means nothing works restrictions on cipher operations. Generating more than 2^{50} words using the same key is almost impossible.

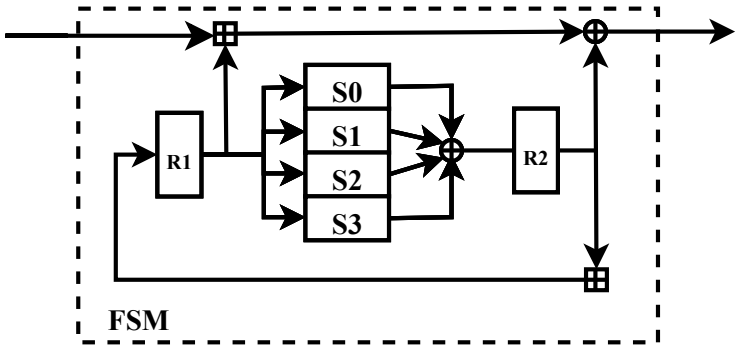


Figure 3.5: Hardware Architecture of FSM part of proposed SNOW 2.0 Architecture.

The main challenge is implementing the FSM part, which includes the implementation of two identical 32-bit registers and, the Substitution box and the integer addition over mod 2^{32}. The FSM consists of two 32-bit registers, two integer adders modulo 2^{32}, two 32-bit bitwise XOR and boolean functions for S-box instead of four LUTs. The implementation of the FSM part of the proposed SNOW 2.0 stream cipher architecture is represented in Figure. 3.5. The Substitution box of the SNOW stream cipher demands the highest hardware resources. This box contains multilevel logic. This box would use most space on devices with low storage capacity. The literature contains descriptions of numerous hardware implementations of Substitution boxes. They include LUT/ROM-based implementation, modified look-up tables, Boolean logic, computational techniques, multiplexers, etc. There is little room for optimization when using typical LUT or BRAM for S-Box implementation.

With the implementation using logic gates, there are further opportunities for area optimization, as presented by S. Ashaq et al. Using the Karnaugh map, they produced the minimized SOP expressions for S-Box, which, again using Boolean theorems, reduced to the gate-level logic. Then, the standard functions found from the expressions are used in a shared manner to minimize the number of gates for the output of S-Box implementation as shown in table 3.1 and table 3.2. It should be noted that here $A, B, C, andD$ are taken as inputs to the S-Box with A as MSB and D as LSB and $S3, S2, S1$, and $S0$ as the outputs of the S-Box with $S3$ being the MSB and $S0$ being the LSB. For hardware implementation of the S-box, 9 AND gates, 7 OR gates, 6 XOR gates, and 4 NOT gates are required instead of ample memory space or resources utilized in the case of LUT or BRAM-based S-box implementation.

3.3.3 Simulation Results and Performance Analysis

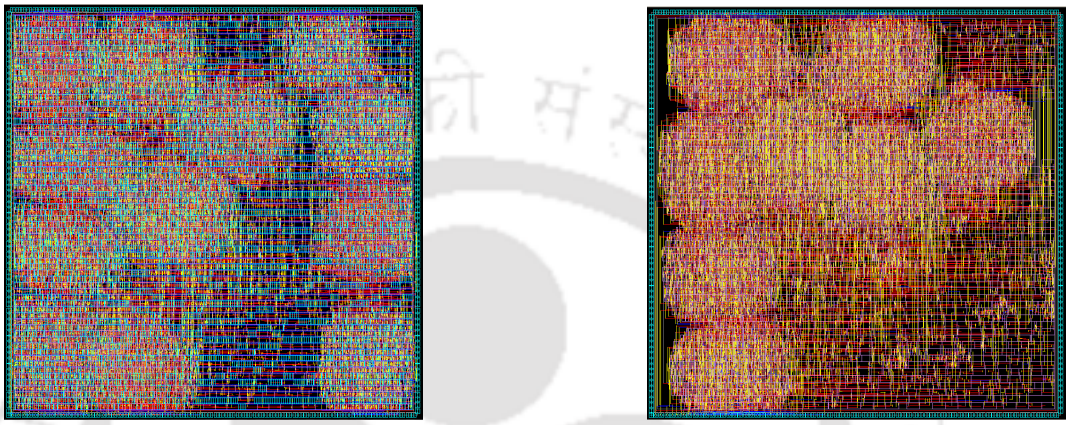
In this chapter, along with the traditional SNOW 2.0 architecture, the modified SNOW 2.0 stream cipher based on a fast resource-efficient S-box is designed and implemented on FPGA Zynq ZC702. The proposed implementation is simulated using the test vectors provided by the cipher specification [2] for checking the correct operations. The resource utilization of the proposed architecture on the FPGA Zynq ZC702 is shown in the table. 3.3, which shows that 0.50% of the total available resources are being utilized by the SNOW 2.0 (LUT) architecture, whereas the proposed modified SNOW 2.0 architecture uses 0.41% of total available resources making it resource-efficient.

Table 3.3: FPGA Implementation Resource utilization of Proposed SNOW 2.0 Architectures (FPGA Zynq ZC702)

Resources	Total Available	SNOW 2.0 (LUT)		Proposed Modified SNOW 2.0	
		Resources Utilized	Percentage Utilization (%)	Resources Utilized	Percentage Utilization (%)
Slice LUT	53200	293	0.55	197	0.37
Slice Reg	106400	467	0.44	467	0.21
F7MUX	26600	56	0.21	0	0
F8MUX	13300	24	0.18	0	0
BRAM	140	2	1.43	2	1.43
RAMB18E1	280	4	1.43	4	1.43
IOB	200	161	80.5	161	80.5
BUFGCTRL	32	1	3.13	1	3.13
Total Resources	**200152**	**1008**	**0.50**	**832**	**0.41**

Further, implementing SNOW 2.0 using SCL 180nm PDK ASIC technology with the help of the Synopsys synthesis tool VCS involves several steps. The first step is designing the SNOW 2.0 cipher utilizing a hardware description language like Verilog. The design is then optimized using the synthesis tool, which helps to reduce the size and power consumption of the resulting ASIC. After optimization, the physical layout of the ASIC is created using specialized software tools, which define the placement of the various components on the chip and the routing of the interconnects between them. The layout is then verified using simulation and tested to ensure it meets the desired specifications. Synopsys Design Compiler generates a gate-level netlist, which is then subjected to placement and routing using the ICC compiler. Figure. 3.6 illustrates the post-placement and routing layout of the proposed architectures for both SNOW 2.0 and modified SNOW 2.0 architectures. Finally, the resulting ASIC can provide significant performance and power consumption advantages over general-purpose solutions, making it an attractive option for many applications. Table 3.4 presents the ASIC implementation

specifications of the proposed SNOW 2.0 architectures. The overall design of the SNOW 2.0 (LUT) architecture utilizes $0.094mm^2$ area and consumes $14.95mW$ power at 1.98V and 270MHz frequency, while the proposed modified SNOW 2.0 architecture uses $0.076mm^2$ area and consumes$13.91mW$ power at 1.98V and $273MHz$ frequency.



(a) SNOW 2.0 (LUT) **(b)** Modified SNOW 2.0

Figure 3.6: Layout Design of SNOW 2.0 stream cipher

Table 3.4: ASIC Implementation specifications of Proposed SNOW 2.0 Architectures

Architecture	SNOW 2.0 (LUT)	Modified SNOW 2.0
CMOS process	SCL $180nm$	SCL $180nm$
Area	$0.094mm^2$ ($0.307mmx0.307mm$)	$0.076mm^2$ ($0.276mmx0.276mm$)
Voltage	$1.98V$	$1.98V$
Frequency	$270MHz$	$273MHz$
Power	$14.95mW$	$13.91mW$

Performance characteristics like resource utilization in terms of the number of slices, frequency, throughput and efficiency of the proposed modified SNOW 2.0 architecture and traditional SNOW 2.0 stream ciphers are compared with the existing architectures as shown in the table. 3.5. As shown in the table. 3.5, the efficiency of SNOW 2.0 architecture by [26]. SNOW 2.0 (LUT) and SNOW 2.0 (ROM) by [23] are 5.57, 2.5, and 11.5, respectively, while the efficiency of the proposed modified SNOW 2.0 architecture is 11.8. Though the efficiency of SNOW 2.0(ROM) and modified SNOW 2.0 are comparable, the maximum possible frequency of SNOW 2.0(ROM) by [23] is $141MHz$. The proposed modified SNOW 2.0 can reach up to $245MHz$ maximum possible frequency.

Work	stream cipher Architecture	Platform	Number of Resources	Comparative Resource Utilization	Frequency (MHz)	Throughput (Mbps)	Efficiency (Mbps/slice)	Comparative Efficiency
Proposed Modified SNOW 2.0		**Zynq ZC702**	**664**	**X**	**245**	**7840**	**11.8**	**Y**
SNOW 2.0 (LUT) (In house Realization)			**760**	**1.14X**	**236**	**7552**	**9.93**	**0.84Y**
[46]	SNOW 1.0	Virtex-II	752	$1.13X$	66.5	2128	2.83	$0.24Y$
[25]	RC4	Virtex-II	140	$0.21X$	60.8	120.8	0.86	$0.07Y$
	E0		895	$1.34X$	189	189.0	0.21	$0.02Y$
	A5/1		32	$0.05X$	188.3	188.3	5.88	$0.49Y$
	Helix		418	$0.62X$	32.0	1024.0	2.45	$0.21Y$
[23]	MUGI	**Virtex-E**	2092	$3.15X$	95	6080	2.9	$0.24Y$
	MICKEY128		167	$0.25X$	166	166	0.99	$0.08Y$
	TRIVIUM		144	$0.21X$	211	211	1.5	$0.13Y$
	SNOW 2.0 (LUT)		**1545**	**2.32X**	**122**	**3904**	**2.5**	**0.21Y**
	SNOW 2.0 (ROM)		**391**	**0.59X**	**141**	**4512**	**11.5**	**0.97Y**
[26]	**SNOW 2.0**	**Virtex-II**	**1015**	**1.53X**	-	**5659**	**5.57**	**0.47Y**Y

Table 3.5: Performance Analysis of SNOW 2.0 stream ciphers

3.4 SNOW 3G stream cipher Architecture

Designing efficient and effective security solutions for advanced mobile embedded systems, particularly those with high-security demands, is challenging. These systems impose strict constraints on power consumption and chip area coverage while requiring high-performance capabilities. To address this challenge, one approach is to implement cryptographic algorithms as hardware modules or dedicated crypto processors. By implementing in hardware, achieving faster and more optimized cryptographic operations is possible. This hardware-centric approach ensures that the stringent resource utilization and power consumption constraints are met while still delivering the required performance levels. Such implementations allow for the efficient execution of cryptographic tasks, enabling mobile embedded systems to meet their security requirements effectively.

Regarding cryptographic methods, their functionality depends on how fast they can be used on various platforms and the chip's resource and power consumption. Extensive research has been conducted in early and modern cryptographic eras to balance security and performance across hardware and software platforms. Bit-oriented stream ciphers are not ideal for software, hence the need for a different approach, such as using word-oriented stream ciphers. Stream ciphers are favoured in wireless standards such as IEEE 802.11b and Bluetooth due to their speed and efficiency compared to block ciphers. This chapter focuses on implementing and discussing the SNOW 3G stream ciphers, adopted as a standard ISO/IEC 18033-4:2005 standard [47].

SNOW 3G is a stream cipher algorithm for 3G and 4G cellular networks. It was developed collaboratively by researchers from Norway, Finland, and Germany [31]. The SNOW 3G algorithm employs a linear feedback shift register (LFSR) and a nonlinear feedback function to generate a pseudorandom bit sequence. Ciphertext is produced using the XOR operation of this generated sequence and the plaintext. One of the key advantages of SNOW 3G is that the generated keystream can only be predicted with knowledge of the secret key. This property makes it highly suitable for secure wireless communications, providing robust encryption for protecting sensitive data.

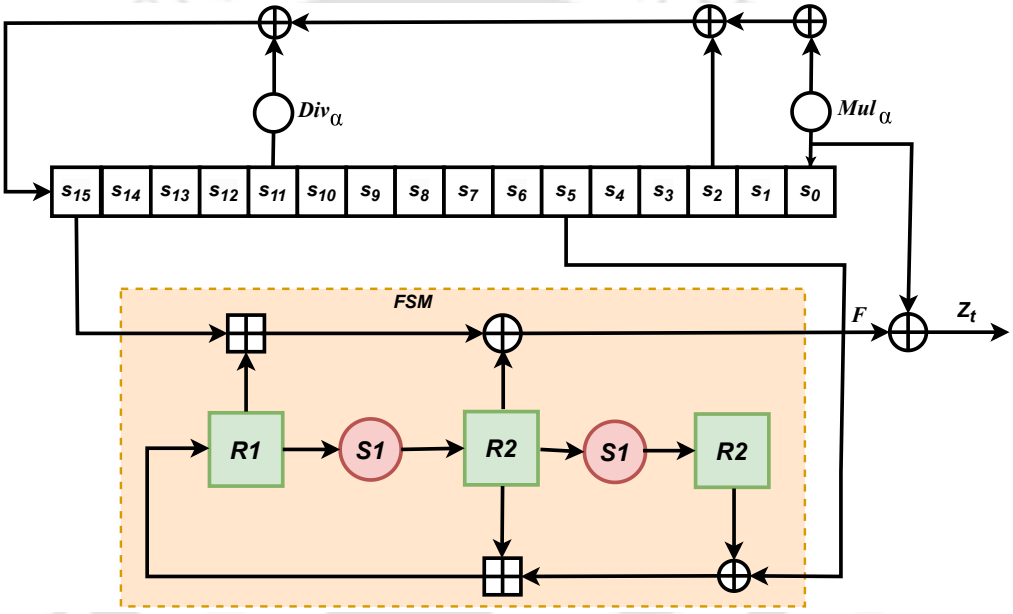


Figure 3.7: Schematic of SNOW 3G stream cipher.png

3.4.1 Proposed SNOW 3G Stream Cipher Architecture

The SNOW 3G stream cipher utilizes a 128-bit key and a 128-bit initialization variable to generate a sequence of 32-bit words, as depicted in Figure. 3.7. The cipher operates in two main steps. First, a key initialization process occurs, clocking the cypher without generating any output. Then, it switches to key-generation mode, producing a 32-bit ciphertext or plaintext word output for each clock cycle. This ensures the secure and controlled generation of the output. Algorithm 1 describes the step-by-step process of working with SNOW 3G architecture, which is explained in detail in the following sections.

3.4.2 Linear Feedback Shift Register (LFSR)

The LFSR comprises sixteen registers, s0, s1, s2, . . . , and s15, each holding 32 bits.

3.4.2.1 Key Initialization:-

Key Initialization in the SNOW 3G stream cipher involves initializing the LFSR (Linear Feedback Shift Register) and internal FSM (Finite State Machine) registers with their initial values. This process starts with a 128-bit key, represented by four 32-bit words: k_0, k_1, k_2, and k_3. A 128-bit initialization variable consists of four 32-bit words: IV_0, IV_1, IV_2, and IV_3. These values are used to set up the initial state of the cipher before the key-generation mode begins. The initialization is performed as follows:

Table 3.6: The initial values of the LFSR.

s15 = $k3 \oplus IV0$	s14 = $k2$	s13 = $k1$	s12 = $k0 \oplus IV1$
s11 = $k3 \oplus 1$	s10 = $k2 \oplus 1 \oplus IV2$	s9 = $k1 \oplus 1 \oplus IV3$	s8 = $k0 \oplus 1$
s7 = $k3$	s6 = $k2$	s5 = $k1$	s4 = $k0$
s3 = $k3 \oplus 1$	s2 = $k2 \oplus 1$	s1 = $k1 \oplus 1$	s0 = $k0 \oplus 1$

The all-ones word (0xffffffff) is used to compute the values of LFSR, R2, and R3, and the importance of R1 is set equal to the initialization variable IV. Furthermore, during initialization, the FSM registers R1, R2, and R3 are set to zero. The clocking process is carried out for 32 clock cycles to complete the initialization. This ensures that the initialization process is executed successfully.

3.4.2.2 Key Generation:-

After completing the key-generation process in the SNOW 3G stream cipher, the system is ready to encrypt or decrypt data. The cipher is clocked once, and the output generated in this clock cycle is discarded. Following the initialization, the generated output sequence, the running key, is bitwise combined with the plaintext sequence to produce the ciphertext sequence. This same process is applied during decryption as well. In each clock cycle, a 32-bit ciphertext word denoted as $z_t = F \oplus s_0$ is produced, where F represents the running key and s_0 suggests the current plaintext word being processed. The bitwise XOR operation between the running key and the plaintext word generates the corresponding ciphertext word. This process is repeated for each clock cycle, ensuring the encryption or decryption of the entire data sequence.

3.4.3 Finite State Machine (FSM)

The Finite State Machine (FSM) in the SNOW 3G stream cipher consists of three 32-bit registers: R1, R2, and R3. Additionally, it incorporates two S-boxes, S1 and S2, each with 32x32 bits, as depicted in the hardware architecture of SNOW 3G shown in Figure. 3.8. These S-boxes play a crucial role in

Algorithm 1 SNOW3G Algorithm

InputInput OutputOutput text() 0.6
plaintext, key, count ciphertext
initializeLFSRinitializeLFSR initializeFinitializeF getNextStategetNextState getNextFValueget-NextFValue shiftLeftshiftLeft
$LFSR \leftarrow (key)$
$F \leftarrow (key)$
$keystream \leftarrow$ empty string
$i \leftarrow 1$ to $count$ $state \leftarrow (LFSR)$
$z \leftarrow$ output$(state)$
$f \leftarrow (F, z)$
$keystream \leftarrow keystream||f$
$ciphertext \leftarrow$ plaintext $\oplus$ keystream
$ciphertext$
FnFunction: key $LFSR \leftarrow$ empty string
$i \leftarrow 1$ to 16 $LFSR[i] \leftarrow$ bit(key, i)
$LFSR$
FnFunction: key $F \leftarrow$ empty string
$i \leftarrow 1$ to 32 $F[i] \leftarrow$ bit$(key, i + 16)$
F
FnFunction: LFSR $nextBit \leftarrow (LFSR[13] \oplus LFSR[16] \oplus LFSR[17] \oplus LFSR[18])$
$LFSR \leftarrow (LFSR) \oplus nextBit$
$LFSR$
FnFunction: F, z $nextBit \leftarrow (F[28] \oplus F[31])$
$F \leftarrow (F) \oplus (z \oplus nextBit)$
$F[1]$
FnFunction: bits $shiftedBits \leftarrow$ empty string
$shiftedBits \leftarrow$ bits$[2:32]||$bits$[1]$
$shiftedBits$

updating the contents of registers R2 and R3. Register R1 is updated using a combination of XOR and Modulo Addition operations. This combination of registers, S-boxes, and operations forms the core of the SNOW 3G cipher's finite state machine.

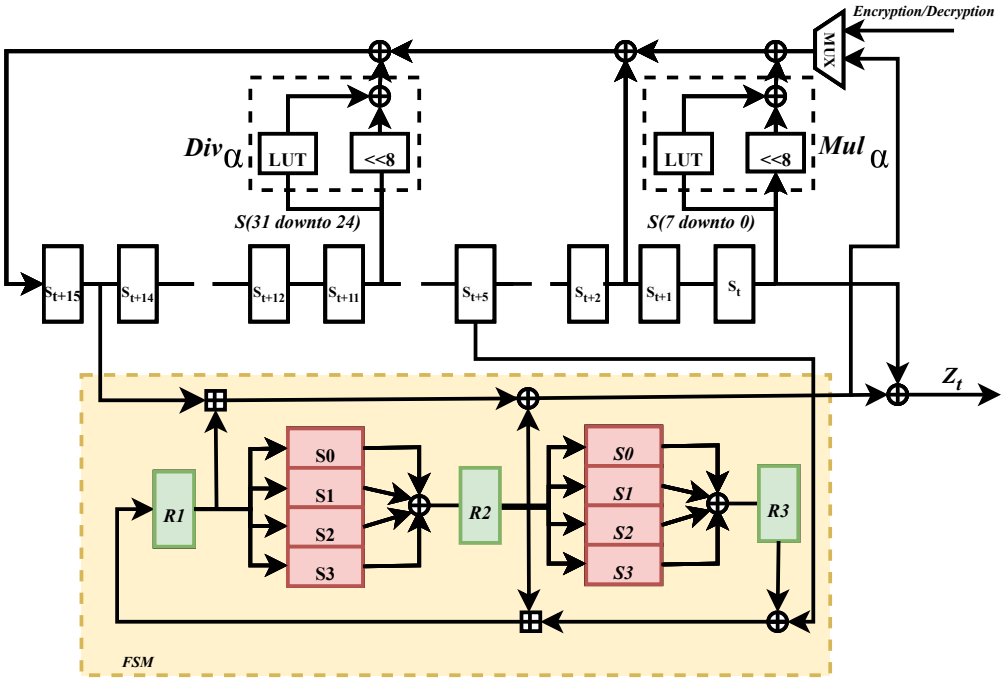


Figure 3.8: Hardware Architecture of Modified SNOW 3G Stream cipher

3.4.4 The 32×32 S-Box

The Substitution box (S-box) in the SNOW stream cipher requires significant hardware resources, especially in storage capacity. S-boxes often involve multilevel logic, making them resource-intensive and unsuitable for devices with low storage capacity. However, researchers have proposed various hardware implementations of S-boxes to address this challenge.

One standard implementation approach involves using Look-Up Tables (LUTs) or Block RAM (BRAM) for S-box implementation. However, these methods offer little room for optimization, leaving little opportunity to reduce resource usage. On the other hand, implementing S-boxes using logic gates provides further possibilities for area optimization, as demonstrated by S. Ashaq et al. They utilized Karnaugh maps to produce minimized Sum-of-Products (SOP) expressions for the S-box. By applying Boolean theorems, these expressions were further reduced to gate-level logic. Standard functions found in the expressions were shared, effectively minimizing the number of gates required for S-box implementation.

For instance, Table 3.1 and Table 3.2 present the common subexpressions and final expressions for the gate-level S-box implementation. This implementation takes inputs A, B, C, and D, with A as the most significant bit (MSB) and D as the least significant bit (LSB). The outputs of the S-box are denoted as $S3, S2, S1$, and $S0$, with $S3$ as the MSB and $S0$ as the LSB.

Table 3.7: FPGA Implementation of Proposed SNOW 3G Architectures (FPGA Zynq ZC702)

Resources	Total Available	SNOW 3G (LUT)		Proposed Modified SNOW 3G	
		Resources Utilized	Percentage Utilization (%)	Resources Utilized	Percentage Utilization (%)
Slice LUT	53200	317	0.6	262	0.49
Slice Reg	106400	204	0.19	150	0.14
F7MUX	26600	80	0.30	48	0.18
F8MUX	13300	24	0.18	24	0.18
BRAM	140	1.5	1.07	0	0
RAMB18E1	280	3	1.07	0	0
IOB	200	144	72	144	72
BUFGCTRL	32	1	3.13	1	3.13
Total Resources	200152	774.5	0.38	629	0.31

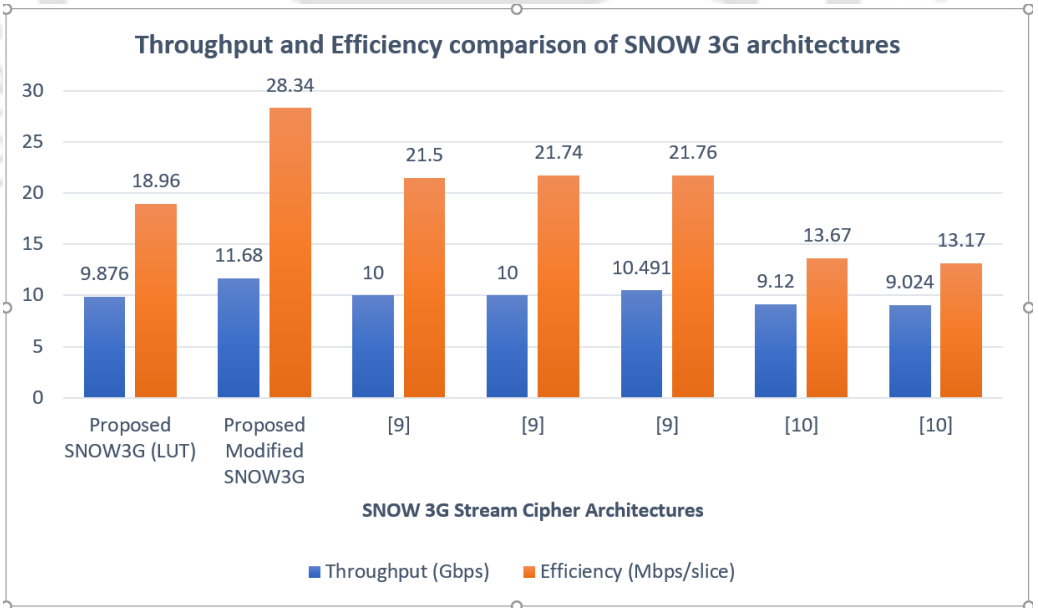


Figure 3.9: Throughput and Efficiency comparison of SNOW 3G architectures

This chapter presents a novel approach to implement the SNOW 3G stream cipher architecture on an FPGA platform, where instead of using LUTs for the S-Box implementation, the proposed architecture utilizes boolean functions as mentioned in Table. 3.1 and Table. 3.2 to realize the S-Box. These boolean functions exhibit a new S-Box with a similar security level provided by the

LUT-based S-Box implementations. By adopting this approach, the hardware implementation of the S-box requires only 9 AND gates, 7 OR gates, 6 XOR gates, and 4 NOT gates. This is a significant reduction compared to the large memory space or resources required for LUT or BRAM-based S-box implementations. Overall, this optimized gate-level logic implementation of the S-box offers a more efficient use of hardware resources, allowing for compact and low-cost implementations of the SNOW stream cipher or other cryptographic systems that employ S-boxes.

Table 3.8: Comparison with Existing FPGA Implementations

Work	Platform	Number of Resources	Comparative Resource Utilization	Power (mW)	Frequency (MHz)	Throughput (Gbps)	Efficiency (Mbps/slice)	Comparative Efficiency
Modified SNOW 3G (Proposed)	**Zynq ZC702**	**412**	**X**	**0.142**	**365**	**11.68**	**28.34**	**Y**
SNOW 3G (LUT) (In-house Design)		**521**	**1.26 X**	**0.154**	**308.64**	**9.876**	**18.96**	**0.67 Y**
[42]	Stratix III C2	464	1.13 X	-	312.5	10	21.5	0.76 Y
	Stratix IV C2	460	1.12 X	-	312.5	10	21.74	0.77 Y
	Stratix V D8	482	1.17 X	-	327.5	10.491	21.76	0.77 Y
[41]	Virtex-5	1713	4.16 X	-	28.84	0.923	-	-
[23]	Spartan-XC3S700A	3559	8.63 X	-	104	3.328	-	-
[43]	Stratix III C2	667	1.62 X	-	285	9.120	13.67	0.48 Y
	Stratix IV C2	685	1.66 X	-	282	9.024	13.17	0.46 Y

3.4.5 Simulation Results and Performance Analysis

In this chapter, along with the traditional SNOW 3G (LUT) architecture, the modified SNOW 3G stream cipher based on a fast resource-efficient S-box is designed and implemented on the Zynq ZC702 FPGA platform. Although the traditional SNOW 3G (LUT) architecture is already available in the literature, it is realized in-house for a fair comparison with the modified SNOW 3G stream cipher scheme. Since other contemporary works implemented on the different FPGA platforms, for comparing the proposed design, we depict the comparative resource utilization of all the designs, taking the resource utilization of the proposed design as a reference.

The proposed implementation is simulated using the test vectors provided by the cipher specification [31] for checking the correct operations. The resource utilization of the proposed architectures on the FPGA Zynq ZC702 is shown in the table. 3.7, which shows that 0.38% of the total available resources are being utilized by the traditional SNOW 3G (LUT) architecture, whereas the proposed modified SNOW 3G architecture uses 0.31% of total available resources making it resource-efficient. Similarly, the traditional SNOW3G (LUT) architecture consumes $0.154mW$ power while the modified logic-gate-based SNOW 3G architecture consumes $0.142mW$ power for FPGA implementation, which is 7.79% less than the traditional one. Performance characteristics like resource utilization in

several slices, frequency, throughput and efficiency of the proposed modified SNOW 3G architecture, and traditional SNOW 3G stream ciphers are compared with the existing architectures as shown in the table. 3.8. As shown in the table. 3.8, the throughput of the traditional SNOW 3G (LUT) is comparable with the proposed modified SNOW 3G (LUT) architecture, but the throughput of the proposed modified SNOW 3G design is 11.3% more than the design presented in [42]. The proposed modified SNOW 3G architecture operates in $365MHz$ frequency. The proposed design utilizes 11.2% fewer resources than the design presented in [42]. The efficiency of SNOW 3G architecture by [42] in Stratix III C2, Stratix IV C2 and Stratix V D8 are 21.5, 21.74 and 21.76 respectively, while the efficiency of proposed modified SNOW 3G architecture is 28.34. From table. 3.8, it can be inferred that the modified SNOW 3G utilizes 20.92% less resource than the traditional SNOW 3G stream cipher. Also, the modified SNOW 3G stream cipher is 49.47% more efficient than the traditional one. The throughput-to-area consumption metric can determine the efficiency of each cipher in hardware implementation. This metric quantifies the ratio of throughput to the area of hardware resources used. A higher value of this ratio indicates a hardware implementation with greater throughput and minimal usage of hardware resources. Figure. 3.9 illustrates the throughput and the efficiency metric for the proposed SNOW 3G stream cipher and the existing stream cipher architectures.

3.4.6 Summary

This section presents an enhanced version of the SNOW 2.0 stream cipher architecture. Our modified design incorporates a gate-level S-Box, resulting in improved resource utilization compared to existing designs and a significant increase in throughput. To assess its performance, we implement the proposed architecture on an FPGA platform and compare it with contemporary SNOW 2.0 architectures, demonstrating superior efficiency. Our evaluation reveals several advantages of the modified SNOW 2.0 stream cipher. Firstly, it exhibits a 28% increase in throughput and a 3% better efficiency index. These results highlight the enhanced performance and efficiency of our proposed architecture. This characteristic renders it particularly suitable for resource-constrained systems such as mobile devices with limited hardware resources. When implemented on ASIC, the modified SNOW 2.0 utilizes 20% fewer resources than the traditional SNOW 2.0 stream cipher. Also, the modified SNOW 2.0 stream cipher is 7% more power efficient than the traditional one. Overall, our findings underscore the efficacy of the modified SNOW 2.0 stream cipher architecture, showcasing its superior resource

utilization, increased throughput, improved efficiency index, and reduced power consumption. These attributes position it as a promising solution for various applications, especially those with limited hardware resources.

In this chapter, we also introduce a modified SNOW 3G stream cipher architecture that utilizes a gate-level S-Box to achieve greater resource efficiency compared to existing designs. The proposed architecture was implemented on an FPGA to evaluate its performance and compare it against existing SNOW 3G architectures. Results indicate that the modified architecture uses fewer slices, reducing critical path delay and increasing throughput. Furthermore, the proposed design demonstrates higher efficiency in the throughput per slice ratio value. The modified SNOW 3G hardware architecture is optimized for efficient implementation in hardware, with a low resource count and high throughput. This makes it well-suited for mobile devices and other resource-constrained systems with limited hardware resources.

4

Secret Sharing Schemes

Contents

Secret sharing is a method of encoding a secret by distributing it amongst a group of participants. The secret can only be decoded when a sufficient number of participants coordinate. Secure Multi-Party Computation (SMPC) is a computationally intensive application of secret sharing, where various functions over the input data are computed by keeping it private. An effective way of addressing this problem is to implement secret sharing algorithms on hardware. Significant performance gains, reduced power consumption, and better resource utilization can be achieved by designing application-specific hardware. This chapter presents a resource and delay-efficient hardware realization of Shamir's linear secret sharing and Renval-Ding's nonlinear secret sharing schemes on FPGA ZedBoard. These schemes are scalable with respect to secret size and the number of participants.

4.1 Introduction

Secret sharing refers to all methods that seek to secure sensitive information by distributing it among a group of participants. Secret sharing has numerous applications, like distributing cryptographic encryption keys among multiple people to enhance security. Secret sharing is also used for distributed backup storage of cloud data to improve data security and prevent data loss in cases of failure of multiple nodes. When a pre-set number of participants combine their shares, it is possible to reconstruct the secret using mathematical formulae of the corresponding secret sharing method. This pre-set number is called the threshold, k. Most secret sharing schemes are (k, n) threshold schemes where the secret can be recovered from any set of k shares. There are two classes of secret sharing schemes: linear and nonlinear. Most secret sharing schemes like Shamir's scheme [48] that are being used today are linear because of their algorithmic simplicity. However, unlike nonlinear secret sharing schemes, linear schemes are susceptible to Tompa-Woll [49] attacks. Despite many improvements to these algorithms over the years, they are still vulnerable to these attacks. Secret sharing schemes can also be classified into perfect and nonperfect schemes. A perfect scheme is one in which unauthorized or any number of shares less than the threshold will reveal no information about the secret. On the other hand, nonperfect schemes theoretically allow unauthorized shares to give some information about the secret, but it is computationally hard to do so.

4.1.1 Previous work

There have been a few prior implementations of secret sharing. An implementation of Shamir's secret sharing by Jakob Stangl et al. [50] parallelizes the generation of shares. The architecture for

share generation contains parallel polynomial evaluation units (PEU), which compute the value of Shamir's polynomial for each participant. Each PEU contains a multiplier, adder, register each, and a 'reduce' block to compute the modulus of the result from the preceding block since operations must be performed in a Galois field of size 2^{32}. By using the matrix form of the share generation equations, the secret reconstruction is done by a matrix inversion and parallel modular multiplication of each row.

The implementation of Shamir's secret sharing by Pei Luo et al. [26] includes an error detecting module which is used to identify cheaters in the group and their corresponding erroneous shares. The share reconstruction module is followed by an error correction module which outputs the secret if no error is detected. In this implementation, a redundancy of 32 bits is added to the 96-bit secret to facilitate error detection. The above implementations are highly parallelized. Since each parallel block has a multiplier, increasing the number of participants will increase the number of multipliers used, and consequently, resource utilization will increase. Hence, such a parallel design is not scalable. Adding redundancy to the secret, as done by Pei Luo, is also not scalable since a larger multiplier will be needed, leading to an increase in resource utilization.

The scheme for computational secret sharing published by H. Krawczyk [51], firstly encrypts the data by a seccure encryption function ENC and a randomly generated key. Shamir's polynomial concept of sharing is used to share the data by replacing random numbers in all the coefficients of the polynomial with other secrets, while the key is shared using a perfectly secure secret sharing scheme (PSS).

Information dispersal algorithm (IDA) is the way of using polynomials to encode data is introduced by M. Rabin [52], which is used for applications to secure and reliable storage of information in computer networks.

The performance of secret sharing schemes were analysed according to the threshold parameters n and k in [53]. For generating 10 shares a throughput of about $9Mbps$ could be achieved with Shamir's secret sharing scheme and about $18Mbps$ with combinational secret sharing.

A secret sharing-based countermeasure for AES S-box, which has the ability to defend against both HODPA (Higher-Order Differential power attack) and glitch attack, was implemented by Yi Wang et. Al [54]. The input byte of their S-box is divided into two shares; it is difficult for the attacker to retrieve the real intermediate values. Share1 is independent of the first part of the input to the affine

transformation. Similarly, share2 is also independent of the second part of the input, which shows the non-completeness property of the secret dividing function. The attacker can only get two independent intermediate values by applying HODPA, which cannot recover the original intermediate values.

The implementation of an (n, n) threshold secret image sharing (SIS) scheme with equal size in shares by T. Bhattacharjee et al. [55] represents share creation by two sets of bit patterns formed from the fixed and the variable bit positions of the affine Boolean classification. Their SIS scheme gives robustness against random gain scaling and different operations.

In [13] the throughput is increased by using the Graphics Processing Unit (GPU). It was seen that a throughput of $48Mbps$ was achieved for a threshold of 4 for calculation of one share. In [56] a $5/slash5$ threshold scheme achieves a speed of $40 - 160Mbps$. Here the implementation of secret sharing is based on cellular automata on a GPU.

4.1.2 Motivation

Cryptographic methods are not suitable for ensuring both confidentiality and reliability. We can achieve the highest level of confidentiality by storing the encryption key in a single location which does not ensure reliability. If we want to achieve reliability, we can store multiple copies of the key in different locations, albeit at the cost of decreased confidentiality. Secret sharing algorithms allow for achieving both simultaneously. Secret sharing has applications in cloud computing, sensor networks, and several protocols like SMPC (Secure Multi-Party Computation). Secret sharing refers to all methods that seek to secure sensitive information by distributing it among participants. Nonlinear secret sharing schemes are relatively more computationally intensive than their linear counterparts, like Shamir's secret sharing scheme. Consequently, such schemes take significantly longer to work when implemented on software. A hardware implementation can be greatly optimized for both performance and resource utilization. A quick example would be a divide by a power of 2 operations. We would need to perform a standard division operation in a software implementation. However, in hardware, it can be done using a simple bit-shifting operation. Also, there is a high scope for pipelining and parallelizing processes in hardware.

Novelty of the chapter can be discussed as follows:

- The chapter proposes a hardware design technique for nonlinear secret sharing schemes that can be used in dynamic environments where the number of players is constantly changing.

- The proposed design technique uses a modular arithmetic block as the primary computational block, and the Barret reduction and Montgomery reduction techniques are used to reduce the computation time.
- The chapter also proposes a block design for modular inverse using the Gauss-Jordan elimination algorithm, which can be parallelized to bring down the time complexity.
- The proposed design technique is scalable and can be implemented on FPGA platforms.
- The chapter concludes that the proposed design technique can be used to implement secure and efficient nonlinear secret sharing schemes in dynamic environments.

Here implementations of both Shamir's secret sharing (proposal 1) and of a nonlinear secret sharing scheme developed by Renval and Ding (proposal 2) are proposed. In both implementations, unique methods for computing the basic operations like square root [57] and modular inverse have been used to optimize the design. Unlike the implementation by Jakob Stangl [50], which uses multiple PEUs, proposal 1 only uses a single PEU and hence a single multiplier to generate all the shares. Both implementations accept these parameters as inputs to make the design scalable and work in dynamic environments where the number of players and threshold changes. Despite Shamir's secret sharing scheme being perfect, it is vulnerable to Tompa-Woll attacks. proposal 2 implements a non-perfect nonlinear secret sharing scheme immune to such attacks. In this chapter, we propose implementations of both Shamir's secret sharing (Proposal 1) and a nonlinear secret sharing scheme developed by Renval and Ding [58] (Proposal 2).

4.2 Secret Sharing Schemes

Secret sharing is a family of algorithms used to secure sensitive data. These algorithms can be used to secure encryption keys and missile launch codes. These methods distribute the secret (sensitive information that needs to be secured) to each one of the members of a group. The catch is that the original secret can only be reconstructed when a certain number of members put together their shares (their part of the secret). Secret sharing involves one dealer (who creates and distributes the shares) and n players. The secret can only be recovered when a fixed number of players come together [48].

4.2.1 Shamir's Linear Secret Sharing

Shamir's secret sharing scheme is part of a more prominent family of "linear" secret sharing schemes. Adi Shamir [48], in his chapter "How to share a secret," presents a sharing scheme that uses properties of a polynomial to create the shares of the secret. A 't-1' degree polynomial can be identified if we know any 't' points lying on it. A (t, n) Shamir's secret sharing scheme involves a dealer and n players. The dealer is a trusted third party. The dealer generates a 't-1' degree polynomial $f(x)$ with the secret as the constant term and computes the values of this polynomial at 'n' values of x. The pair $(x_j, f(x_j))$, j= 1, 2, $\cdots$, n is said to be the secret share, and each player P_j gets a share. The value 't' is called the threshold, the minimum number of shares needed to reconstruct the original polynomial. Shamir's secret sharing scheme is information-theoretically secure because knowing less than the threshold number of schemes does not reveal any information about the secret. It also allows us to increase the number of players/shares without changing the threshold 't'.

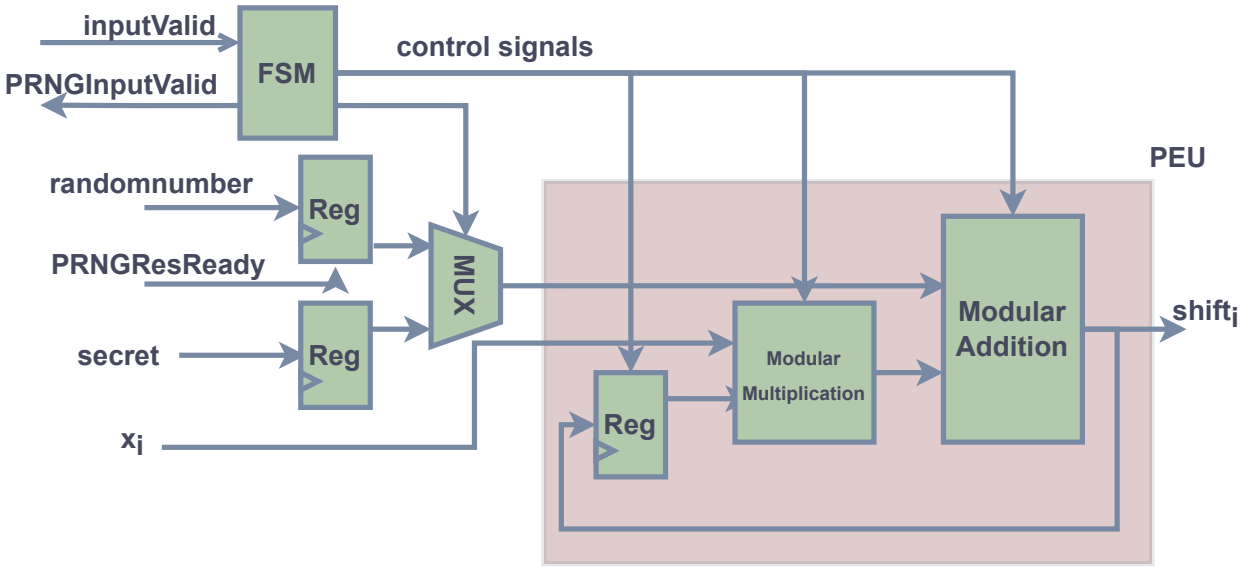


Figure 4.1: Block design of linear share generation unit

Share Generation:

To generate shares in Shamir's scheme, we need to construct a 't-1' degree polynomial with the secret as the zero-degree term. We must select 't-1' random numbers to be the coefficients of the terms in the polynomial. Let a_i, $i = 1, 2, \cdots, t$ be the coefficients of this polynomial chosen from a finite field.

$$f(x) = \left(\sum_{0}^{t-1} a_i x^i \right)_{modp} \tag{4.1}$$

a_0 is the secret that must be secured. The value p is the size of the finite field. We assume that the secret is lesser than p. The use of finite field modular arithmetic is vital as there is a security issue in using integer arithmetic. In a practical implementation of integer arithmetic, there are cases where the number of possibilities of the secret is significantly reduced depending on the share that has been compromised, thus giving away unintended information even when less than k shares have been compromised by the attacker. After this, the dealer computes values of $f(x)$ for 'n' values of x usually from x=1 to $x = n$ to get 'n' pairs(shares) of the form $(x_j, f(x_j))$ corresponding to player P_j. These shares are distributed among the n players. Our design of the linear share generation block is given in Figure. 4.1. The time complexity of the implementation is $O(nk)$ clock cycles, where n is the number of players, and k is the threshold. As shown in Figure. 4.1,the linear share generation block takes secret, threshold and number of players as input and generates the required number of shares. Since one of our main aim is to design a system that is suitable for dynamic environments where the number of players is changing like already discussed in the previous section, all the stages in the process must be executed variable number of times because of which most of the algorithm can't be parallelized. Once all the random numbers have been generated, the direct implementation of polynomial evaluation involves multiplications and n additions. However, by using the below method, this can be reduced to multiplications and additions.

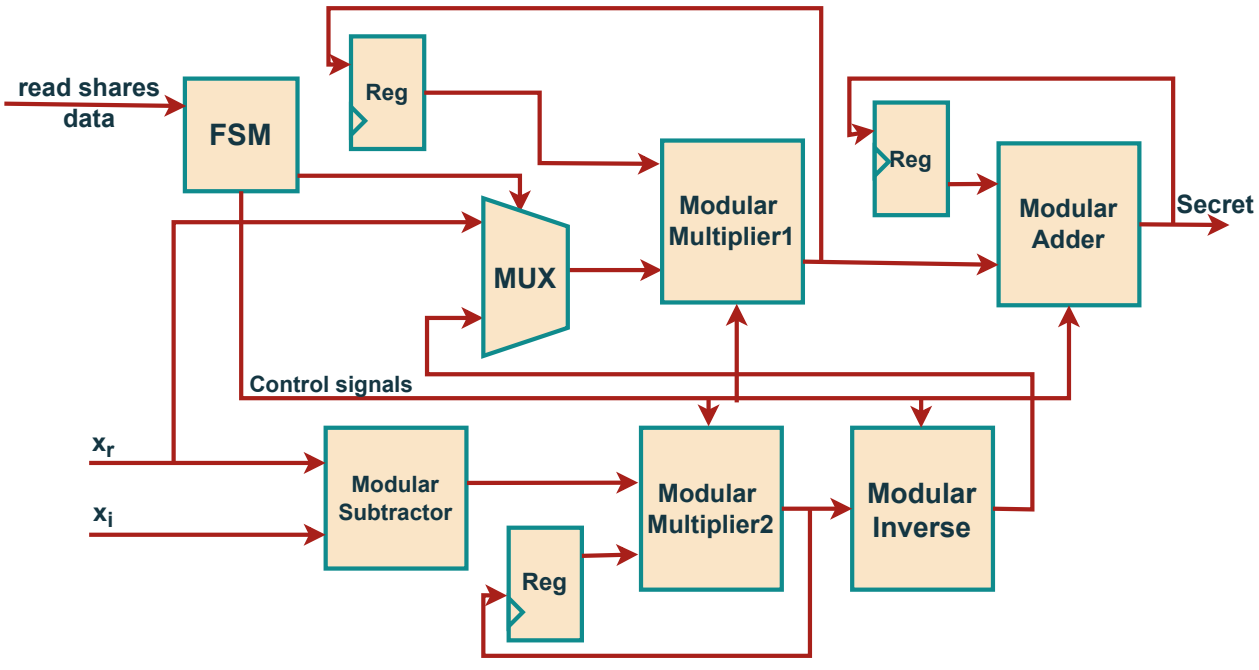


Figure 4.2: Block design of linear secret reconstruction unit

Secret Reconstruction:

The secret can be reconstructed from any subset of t player shares out of a total of n shares by interpolating the t 2-dimensional points, formed by combining the player shares with the player ids, to form a unique polynomial of degree t-1 in 2-dimensional space. Our linear secret reconstruction block design is given in Figure. 4.2. The time complexity of the implementation is $O(k)$ clock cycles, where n is the number of players, and k is the threshold. The linear secret reconstruction block takes threshold and all the player shares as input and reconstructs the secret using Lagrange interpolation. Even this stage can't be completely parallelized because of the same reasons of making the system dynamic as mentioned above. But some stages can be pipelined as shown in the Fig 4.2 improving the performance slightly. Let the subset be denoted as $(x_i, f(x_i))$, i= 1, 2, $\cdots$,t Lagrange interpolation can be used to compute the unique polynomial $f(x)$ of degree t-1 that passes through the t points mentioned above.

$$f(x) = \sum_{i=1}^{t} f(x_i) L_i(x) \tag{4.2}$$

Where $L_i(x)$ is the Lagrange polynomial.

$$L_i(x) = \prod_{r=1, r\neq i}^{t} \left(\frac{x - x_r}{x_i - x_r}\right) modp \tag{4.3}$$

Now the original secret s can be obtained by decoding the zero-degree term in the obtained polynomial $f(x)$.

$$s = f(0) = \sum_{i=1}^{t} f(x_i) \prod_{r=1, r\neq i}^{t} \left(\frac{x_r}{x_r - x_i}\right) modp \tag{4.4}$$

Vulnerabilities, and Improvements:

Shamir's linear secret sharing is prone to a few vulnerabilities when implemented in practice though it is information-theoretically secure. Many improvements have been made over time to this scheme to secure it further. Shamir's secret sharing is vulnerable to Tompa and Woll's [49] attacks, where a few shareholders are dishonest and provide the wrong share during reconstruction. This gives away unintended information to dishonest shareholders, reducing the possible value of the secret, hence making the secret less secure. A slight modification that can be made to fix the issue of cheating among shareholders involves signing every share that is distributed with an unforgeable signature prior to

distribution so that all the shares can be validated during the secret reconstruction process. Tartary and Wang [49] provide a solution to eavesdropping when secret shares are transmitted over insecure networks. It describes a computationally secure approach that dynamically modifies the threshold based on the capabilities of the eavesdropper.

Another vulnerability of Shamir's secret sharing scheme arises when player's shares are not stored securely, which uses proactive secret sharing [48] to fix this. It uses an essential property of Shamir's secret sharing scheme. This property is that by keeping the secret unchanged, we can create an entirely different polynomial with new coefficients. We only need to distribute the updated shares. When a player is compromised, the dealer creates a new polynomial $f_1(x)$ with the constant term zero and computes the value of this polynomial for each player for the exact value of x. The players update their original share by adding the new value $f_1(x_i)$. Because of this, an attacker will not be able to acquire the required number of old shares to reconstruct the secret, nor can he get any information from the updates because they are just random values. However, these improvements have yet to make the scheme completely secure. A family of secret sharing schemes known as nonlinear secret sharing can be used as an alternative to solve this issue at the cost of increased computational power.

4.2.2 Nonlinear Secret Sharing

Every linear secret sharing scheme is associated with a Maximum Distance Separable (MDS) error-correcting code. However, very few MDS codes limit the number of linear secret sharing schemes that can be developed. Linear schemes are also vulnerable to Tompa-Woll [49] attacks. Due to these drawbacks, nonlinear schemes are preferred.

The algorithm proposed by Renvall and Ding is an (n, k)nonlinear scheme where the reconstruction function is quadratic. Unlike Shamir's secret sharing scheme, this scheme is non-perfect, meaning any number of shares less than k-1 may give a limited amount of information about the secret. However, it is computationally intractable to compute the secret. The major advantage of this trade-off is that we can make these schemes as computationally efficient as linear schemes. k-1 or more shares are enough to reconstruct the secret efficiently. Another advantage of this scheme is that it has error-correcting capabilities, which can be useful when there are cheaters among the players. The algorithm is explained in detail below.

Designing the Parameters:

All the arithmetic is done in $GF(p)$ where p is a large prime number which satisfies $p3mod4$. The secret lies between 0, and $((p-1))/2$. For a given scheme, there exists a generator matrix M which is of the following form.

$$\mathrm{M}(i_1, i_2, ...i_k) = \begin{bmatrix} 1 & a_{i1} & (a_{i1})^2 & ... & (a_{i1})^{k-1} \\ 1 & a_{i2} & (a_{i2})^2 & ... & (a_{i2})^{k-1} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & ... & . \\ 1 & a_{ik} & (a_{ik})^2 & & (a_{ik})^{k-1} \end{bmatrix}$$

where $1 \leq i_1 \leq \cdots \leq i_k \leq n$, and $1 \leq k \leq n$. Let $N(i_1, i_2, \cdots, i_k) = [n(i_1, i_2, \ldots i_k)_{(u,v)}]$ is the inverse of matrix M. Parameters $a_i : i = 1, 2, \cdots n$ are chosen such that they satisfy the following conditions:

(i) $C1 : a_1, \ldots a_n$ are distinct non-zero elements of $GF(p)$.

(ii) $C2 : \forall 1 \leq i_1 \leq \cdots \leq i_k \leq n$,

$$1 + \sum_{u=1}^{k-1} \left[\sum_{v=1}^{k-2} n\left(i_1, ...i_k - 2\right)_{u,v} \right]^2 \neq 0 \tag{4.5}$$

(iii) $C3 : \forall 1 \leq i_1 \leq \cdots \leq i_k \leq n$, one of the following conditions hold:

- $\delta(i_1, \ldots i_{(k-2)}) \neq 0$, and $\delta(i_1, \cdots, i_{(k-2)})^2 - 4\beta_2(i_1, \cdots, i_{(k-2)})\gamma_2(i_1, \cdots i_{(k-2)})$ is not a quadratic non-residue.

- $\delta(i_1, \cdots i_{(k-2)}) = 0, \gamma_2(i_1, \cdots, i_{(k-2)}) \neq 0$, and $-(\beta_2(i_1, \cdots, i_{(k-2)}))/(\gamma_2(i_1, \cdots, i_{(k-2)}))$ is not a quadratic non-residue.

Where

$$\beta_2\left(i_1, ...i_{k-2}\right) = 1 + \sum_{u=1}^{k-1} \left[\sum_{v=1}^{k-2} n\left(i_1, ..., i_{k-2}\right)_{u,v} \right]^2 \tag{4.6}$$

$$\gamma_2\left(i_1,...i_{k-2}\right)=1+\sum_{u=1}^{k-1}\left[\sum_{v=1}^{k-2}n\left(i_1,...,i_{k-2}\right)_{u,v}a_{iv}^{k-1}\right]^2 \tag{4.7}$$

$$\begin{aligned}\delta\left(i_1,...i_{k-2}\right)=2\sum_{u=1}^{k-2}\left[\sum_{v=1}^{k-2}n\left(i_1,...,i_{k-2}\right)_{u,v}\right]\\ \left[\sum_{v=1}^{k-2}n\left(i_1,...,i_{k-2}\right)_{u,v}a_{iv}^{k-1}\right]\end{aligned} \tag{4.8}$$

The above conditions are necessary to ensure that the amount of information obtained about the secret by any number of shares less than k-1 is minimal, so that it is computationally intractable to compute the secret. The parameters $a_i : i = 1, 2, \cdots, n$ can be chosen randomly in $GF(p)$ as there is a high probability of these parameters satisfying the above conditions when p is large.

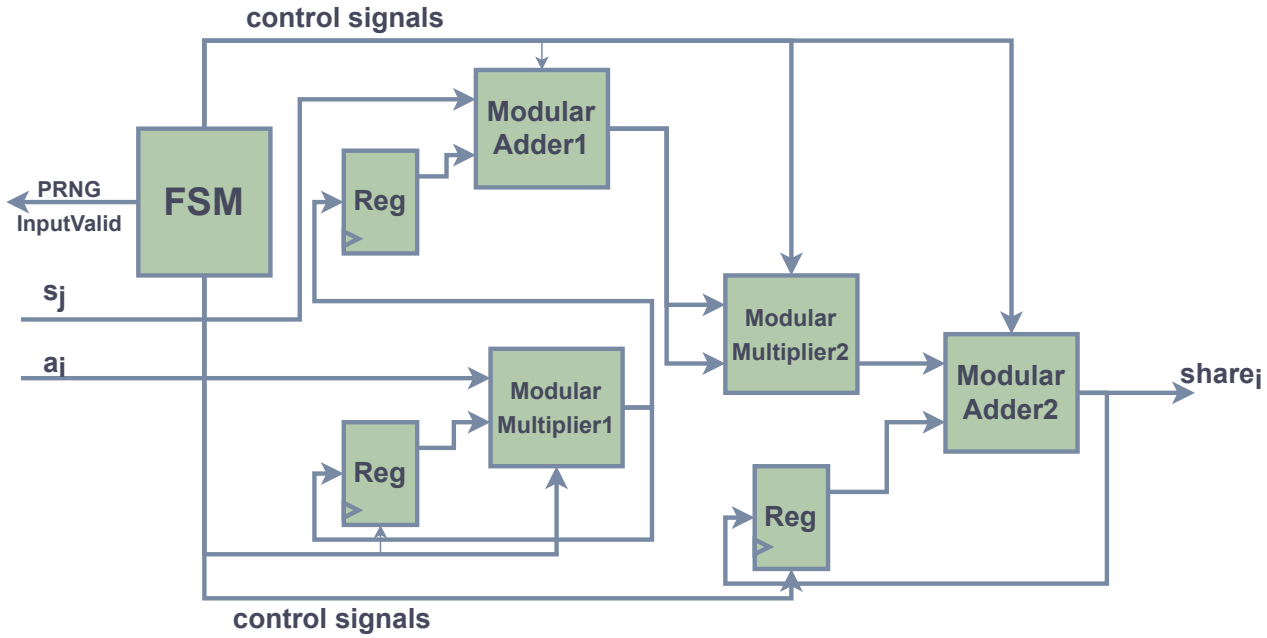


Figure 4.3: Block design of nonlinear share generation unit

Share Generation:

Let s_1, where $0 \leq s_1 \leq (p-1)$ be the secret, and let $s_2, s_3, \cdots, s_k$ be k-1 random numbers in $GF(p)$. Let $s = (s_1, s_2, \cdots, s_k)$, and let $f(x) = xx^T$ over $GF(p)$. Let $\alpha_i = (1, a_i, {a_i}^2, \ldots {a_i}^(k-1)); i = 1, 2, \ldots n$ $t_0 = f(s)$, and $t_i = f(s+_i); i = 1, 2, \cdots, n$ The pair (t_0, t_i) is the share that is given to a player P_i. Our design of the share generation block is given in Figure. 4.3. The time complexity of the implementation is $O(nk)$ clock cycles, where n is the number of players, and k is the threshold.

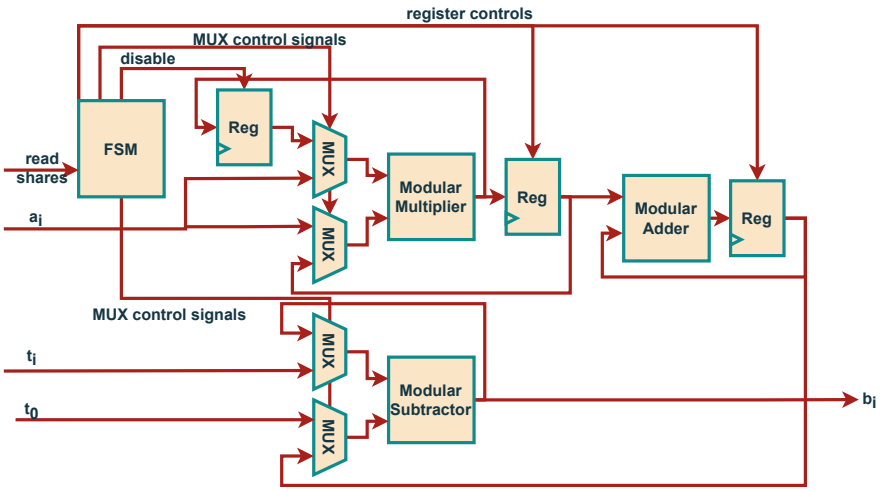


Figure 4.4: Stage1 of Nonlinear secret reconstruction unit

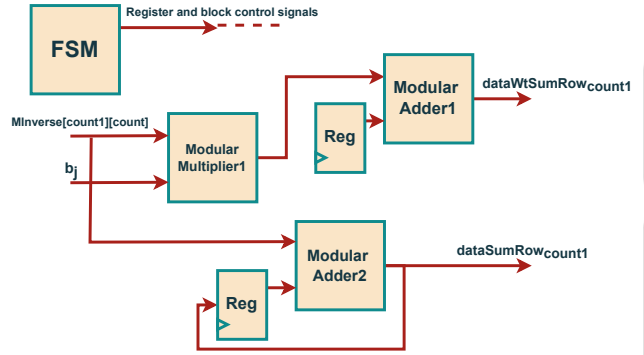


Figure 4.5: Stage2 of Nonlinear secret reconstruction unit

Secret Reconstruction:

When there are exactly k-1 shares, the secret can be reconstructed nonlinearly as shown below

$$M(i_1, i_2, ...i_{k-1}) \begin{bmatrix} s_2 \\ s_3 \\ . \\ . \\ s_k \end{bmatrix} = \begin{bmatrix} b_1 - 2s_1 \\ b_2 - 2s_1 \\ . \\ . \\ b_{k-1} - 2s_1 \end{bmatrix}$$

Where $b_i = t_i - t_0 - \alpha_i \alpha_i^T, i = 1, 2, \cdots, n$

Solving the above set of equation along with the equation for t_0, we get a quadratic equation in s_1 of the form $as_1^2 + bs_1 + c = 0$. Solving this quadratic equation, we get two solutions for s_1, and only one of them lies between 0, and $((p-1))/2$ which is the secret.

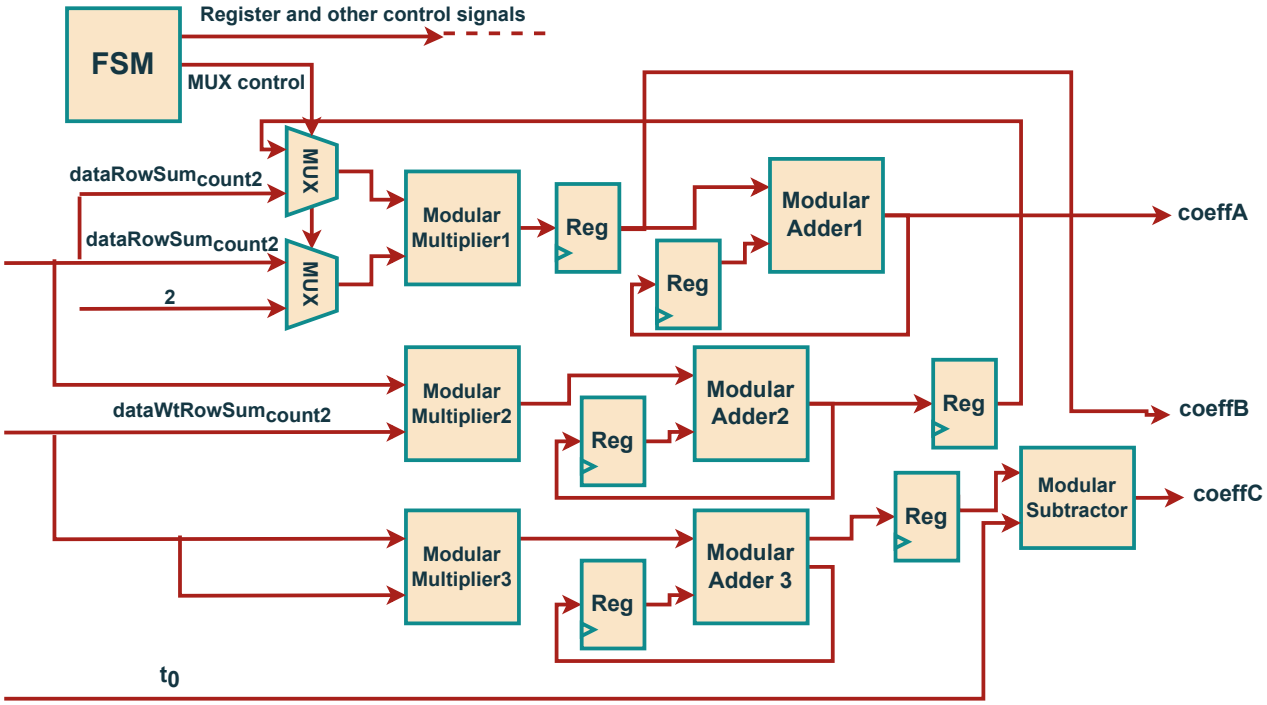


Figure 4.6: Stage3 of Nonlinear secret reconstruction unit

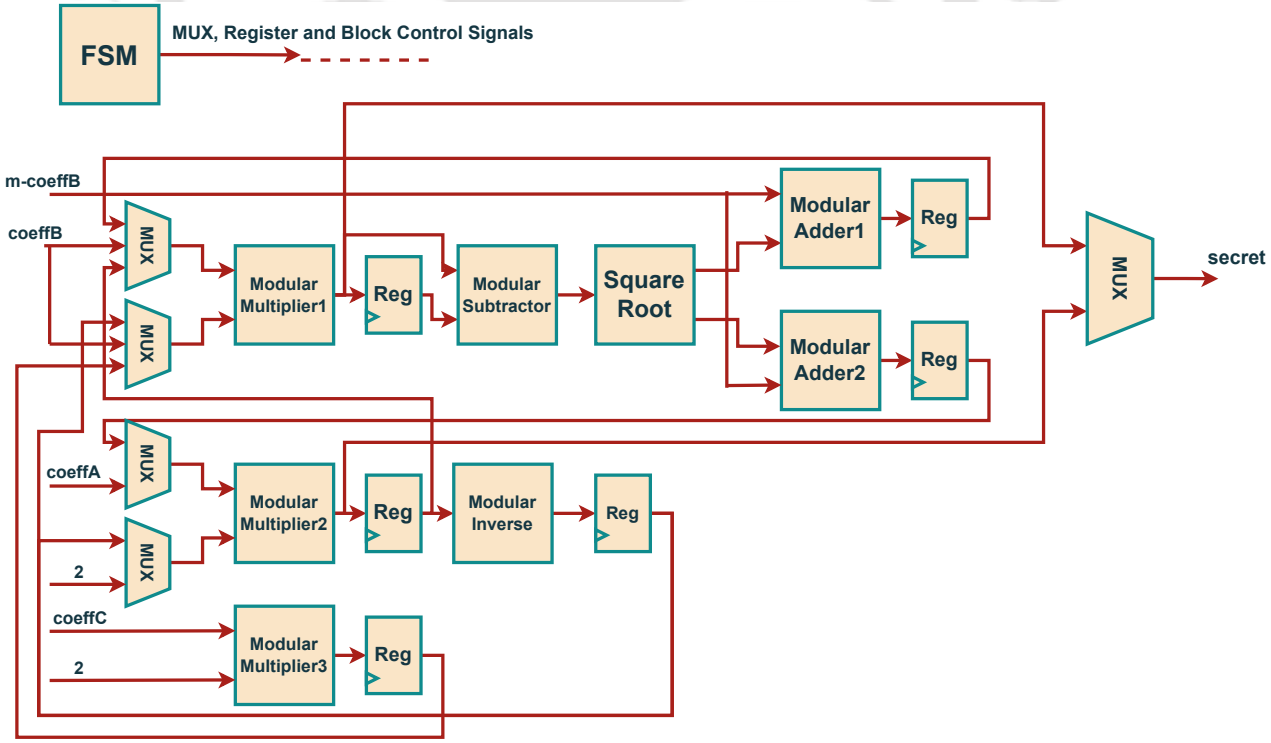


Figure 4.7: Stage4 of Nonlinear secret reconstruction unit

When there are more than k-1 shares available for reconstruction, the secret can be reconstructed linearly by solving set of linear equations

$$\alpha_i s^T = b_i for i = 1, 2, ..., k \tag{4.9}$$

This nonlinear reconstruction allows for error correction when there are cheaters in the network. This kind of reconstruction can correct up to $(n-k)2+k+j-n$ cheater when $k+j$ parties come together.The nonlinear secret reconstruction block can be divided into four stages, as shown in Figures 4.4 to 4.7. stage 1 computes b_i in the equation x, and even this stage can't be parallelized because of the same reasons for making the system dynamic, as mentioned above. stage 2 computes the row sum of the matrix inverse, stage 3 computes the coefficients of the quadratic form, and stage 4 computes the roots of the equation. These stages can be partially parallelized for better performance. The time complexity of the implementation is $O(k)$ clock cycles, where n is the number of players, and k is the threshold.

4.3 FPGA Implementation of Secret Sharing

4.3.1 Linear Secret Sharing

Generally, in most secret sharing applications (including linear secret sharing schemes), arithmetic is performed in a Galois Field of size equal to a power of 2, i.e., $GF(2^n)$. Implementation of the arithmetic becomes very simple and computationally efficient, as the arithmetic can be performed using bitwise manipulations. However, in the nonlinear scheme proposed by Renvall and Ding, the arithmetic must be performed in a Galois Field of size p, where p is a large prime number satisfying the condition $p = 3 \bmod 4$. The arithmetic in such fields is realized by performing the everyday operations and then reducing the result to modulo p. Implementing the scheme becomes challenging because designing the modulus block uses popular methods like The Barrett Reduction or Montgomery Reduction. They take a significant amount of clock cycles to compute the result since they are iterative algorithms (also considering the time to perform arithmetic operations).

Modular Arithmetic:

The modulus block is the primary computational block, and it is used multiple times in the design. Therefore, even small reductions in the computation time of this block can result in notable improvement in the total time taken to reconstruct the secret [59]. The Barret reduction and Montgomery

reduction can be parallelized to a great extent to reduce the computation time.

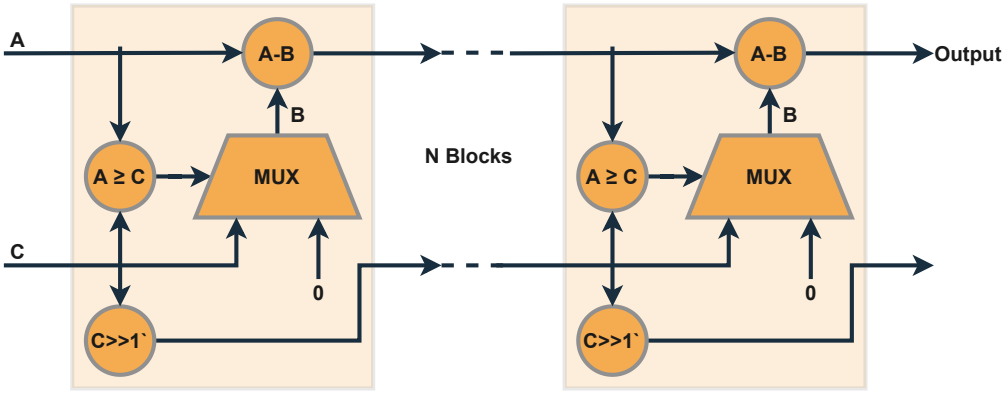


Figure 4.8: Block design of Modulus operation

Surprisingly, there needs to be more work on efficient hardware implementations of the modular inverse. Butler and Sasao [60] presented a high-speed hardware implementation of modulo operation. It considers the computation of $x \mod z$ as a modulo reduction process, where, at each stage, the magnitude of x is reduced. However, the residue remains the same, and this continues until only the residue remains. The circuit can be made entirely combinational using this approach. But even this method utilizes a lot of resources when the input size is large. To avoid this, they also present a pipelined architecture for larger input sizes which utilizes fewer resources but at the cost of the increase in computation time. We introduced a few modifications to this implementation to suit our requirements and the block design of modulus is shown in Figure.4.8. The design is purely combinational, making the design significantly fast. It utilizes fewer resources compared to the implementation of algorithms like the Barret reduction, which computes the modulus by finding the reminder using the long division method. The exact number of stages required in the design depends on the value of p (input C in the block design), and in our implementation for $p = 2^{32} - 5$, the required number of stages is 33.

An alternate method of computing modulus is Newton's method of dividing. The Newton-Raphson method is a way of computing a good approximation [52] for the root of a real-valued function. It works on the principle that a straight-line tangent can approximate a continuous and differentiable function. This approach has successfully been extended for performing floating point division efficiently with significant performance improvement, especially where high output precision is required. In this implementation of modulus, we first calculate the quotient using Newton's method and then

the remainder, which is the required output. Since calculating the quotient does not require high precision computation, we use fixed point arithmetic operations with 32 bits of precision after the decimal point, which is approximately equivalent to 10 decimal places of precision, instead of using floating point operations in order to optimize the design. Newton's method is an iterative $O(log(N))$ algorithm whose hardware implementation requires $O(log(N))$ additions and one multiplication, where N is the size of the secret. This method is comparable to the previously mentioned algorithm [60] in terms of resources utilized only with a minimal increase in computational time. This method is beneficial in some secret-sharing algorithms where both the quotient and remainder are required while computing the modulus. In this chapter, we use the implementation by Butler and Sasao for computing modulus. However, it can be replaced by newton's implementation method based on the application's requirements.

All other operations like multiplication, addition, and subtraction can be implemented using the modulus block. The multiplication block first performs standard multiplication on the operands, then reduces the result to modulo p. Addition and subtraction could be performed using a similar method. However, we took a different approach by considering that the prime number p is very close to a power of two. This is a highly application-specific method that is well suited to our requirements to reduce the resource utilization of the design. The algorithm is based on the idea that when p is very close to a power of two, the modular addition of two numbers can be split into three simple cases. When the regular addition of inputs is less than p, this value is the required result, and when the regular addition of the inputs is greater than the nearest power of 2, then p subtracted from this value is the required result. Finally, when the regular addition lies between p and the nearest power of two, the required result can be obtained by subtracting this value from p. However, when using this method, it is essential to ensure that $p > k/2$, where k is the nearest power of 2 greater than p. The modular multiplication, modular addition, and modular subtraction blocks are all combinational and produce the results immediately.

Since the nonlinear secret sharing scheme being implemented here reduces to a quadratic expression, a modular inverse block and a modular square root block are required to compute the roots of the equation. The most commonly used algorithm for computing the modular inverse is the Extended Euclid's Algorithm, an iterative algorithm that takes $O(log(N))$ time to compute the result. However, a drawback of using this algorithm to implement the modular inverse is that it performs a modulus

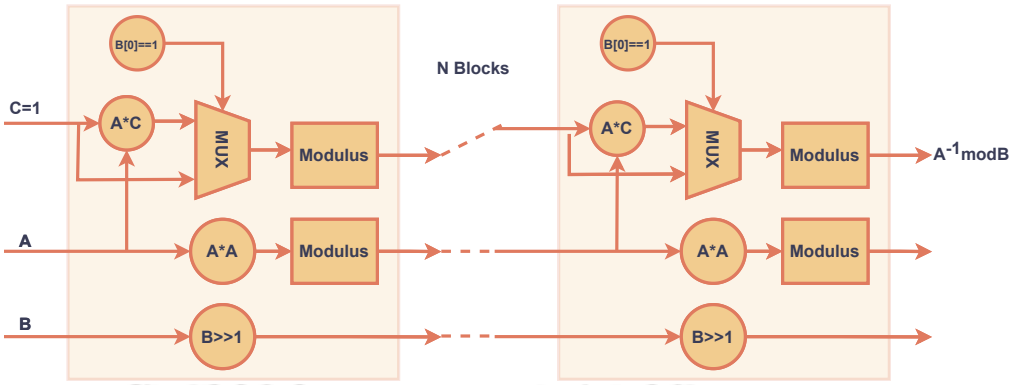


Figure 4.9: Block design of Modular inverse operation

and a division operation in each iteration. It utilizes more resources and requires more computational time. The number of resources utilized can be reduced by using newton's method of dividing. However, a more efficient method to compute the inverse is to use Fermat's little theorem, considering that p is a- prime number. $a^p = a \mod p \xrightarrow{a} a^{(p-2)} a^(- 1) \mod p$ where p is a prime number Even this is an iterative algorithm that computes the result using exponentiation by squaring method in $O(log(N))$ time. Since the p value is fixed throughout the algorithm, the number of steps required to compute the modulus is fixed. Hence the circuit can be made purely combinational, which reduces the computational time drastically, only at the expense of a few extra resources. The exact number of stages depends on the value of p (B in our block design), and in our implementation for $p = 2^{32} - 5$, the required number of stages is 33. The block deign of modular inverse is shown in Figure.4.9.

An efficient algorithm for computing the modular square root is proposed in [61], which has a worst-case complexity of $O((log(p))^4)$. The algorithm has two stages. The first stage involves the computation of $r = a^{(s+1)/2}$, and $b = n^s$ where $p = 2^\alpha s$, s is odd. The second stage involves computing the value of j using an iterative approach such that $b^j r$ is the desired value of the square root. The second stage is the main bottleneck of the algorithm. However, this stage can be completely avoided by choosing an appropriate value of p, which is of form $4n$+3. Thus, by completely avoiding the second stage, the circuit can be made purely combinational, optimizing resource utilization and computational time greatly. The design is purely combinational, and the exact number of stages required again depends on the value of p. In our implementation for $p = 2^{32} - 5$, the required number of stages is 32.

A prime number p close to a power of two (required for the addition module) and satisfying

the condition $p \equiv 3 \mod 4$ can be found by using primality tests like Miller-Rabi, and Baillie-PSW primality test combined with the fact that every prime number is of the form $6m1$. Finding such prime numbers is not easy for large powers of two, but this is a one-time computation and can be afforded. Values of p for commonly used word sizes are listed in Table 4.1.

Table 4.1: Prime numbers closest to large powers of 2

Word Size (in bits)	p
16	$2^{16} - 17$
32	$2^{32} - 5$
64	$2^{64} - 189$
128	$2^{128} - 173$
256	$2^{256} - 189$

Random Number Generation:

A Random Number Generator (RNG) is required to generate the random numbers in the share generation stage of the algorithm. It is tough to design True Random Number Generators (TRNG) [62], and they have limited applications because of their complexity and slow speed of operation. Therefore, in most applications like cryptography, Pseudo Random Number Generators (PRNG) [61] are used as an alternative even though they are deterministic to some extent. PRNGs are very simple to implement on an FPGA and produce random numbers significantly faster than TRNGs. However, robust PRNGs that are specially used for cryptographic applications are complex to design. LFSRs are the simplest type of PRNGs that can be implemented on an FPGA. However, it has notable drawbacks due to its simplicity. The LFSRs can be improved by using a good nonlinear feedback function. However, a more cryptographically secure PRNG can go a long way in increasing the security of the design. No PRNGs are available that satisfy all the conditions required for randomness and cryptographic security. Hence the choice of PRNG depends solely on the application. A 32-bit LFSR-based PRNG is used in this design because of its ease of implementation, but it can be changed without difficulty based on the application's requirements. The most widely used general-purpose PRNG is a Mersenne-Twister which satisfies most of the required conditions. Most of our design of the LFSR [56] is based on this chapter, as discussed in the previous section. However, we made a slight improvement in the security by sampling the LFSR output at every 32^{nd} clock cycle so that there are no overlapping bits between consecutive random numbers and still maintaining the maximum cycle.

Matrix Inverse:

A matrix inversion block is required to solve the system of linear equations in the share reconstruction stage of the nonlinear secret sharing scheme. Even though there are a lot of advanced and efficient methods available for the computation of matrix inverse, especially when the matrix is of a particular type, such approaches are not suitable for computing inverse using field arithmetic. Hence, we must use the standard method of computing the inverse using the Gauss-Jordan elimination algorithm, an iterative algorithm whose software implementation has an asymptotic time complexity of $O(n^3)$. However, the algorithm can be parallelized greatly, thus bringing down the time complexity to $O(n)$. However, even such an implementation poses problems to the system's scalability. Since one of our main aim of the chapter is to make the system suitable for dynamic environments where the number of players is constantly changing, such parallel implementation of a matrix inverse is not suitable because of the varying size of the matrix. We could fix an upper limit on the number of players and design the system for the worst possible case, but such an implementation would contain a lot of unutilized resources in most cases. This problem cannot be completely solved, but the number of unutilized resources can be bought down by parallelizing the algorithm to $O(n^2)$ rather than $O(n)$ but at the cost of increased computation time. In such an algorithm, the subtractions in each row are performed in parallel.

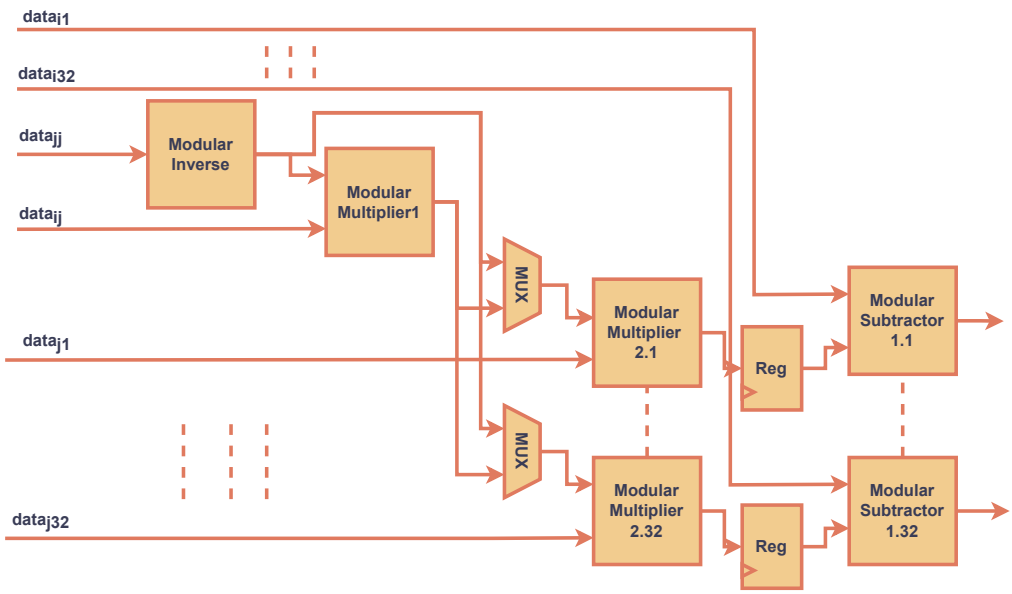


Figure 4.10: Block design of Matrix inverse operation

The block design of matrix inverse block is shown in Figure. 4.10. In this chapter, we choose

the upper limit of 32 for implementation, which can easily be extended without much effort. This implementation performs $k^2(k-1)$modular subtractions, $k(k+1)$modular inverse, $k(k+1)$ modular multiplications in k^2 clock cycles, where k is the size of the matrix. This implementation is significantly better than the software counterpart, which takes k^3 flops to compute the inverse, thus making our implementation scalable.

Parameter Design:

The design of parameters $a_i : i = 1, 2, \cdots n$, is an important step in the algorithm to ensure that the terminal quadratic expression has a good solution. As mentioned in section 3, conditions $C1$ through $C3$ must be satisfied by the above parameters. Verifying these conditions for a particular value of n is a computationally intensive task whose worst-case time complexity is of the order $\mathcal{O}\left(\sum_k \binom{n}{k} k^3\right)$.

Even for the small values of n, and k, it takes a few seconds to verify these conditions because verifying them on the FPGA is not feasible to design a scalable system, especially for applications where n and k are dynamic. In order to solve this problem, the values $a_i : i = 1, 2, \cdots n$ can be pre-computed beforehand using the following method.

- Step 1: Three different random numbers are generated that satisfy the conditions mentioned above for n=3. Note: Since p is a large number, the probability that these randomly generated numbers satisfy the given conditions is very high; hence, most of the steps' verifications can be done in one go.
- Step 2: This is extended to n=4 by generating another random number such that this random number, along with the three previously generated random numbers, again satisfy the mentioned conditions.
- Step 3: The previous step is repeated until the required number (say n) of such random numbers is generated. When any $n_0 < n$ number of these values is required, a subset $a_i : i = 1, 2, \cdots n_0$ is chosen. This method is computationally intensive but needs to be done only once, and these numbers can be stored on an FPGA for easy access.

The linear SGU block takes number of players as input and generates the required number of shares as explained in Algorithm 1. Since one of our main aims is to design a system that is suitable for dynamic environments where the number of players is changing, as already discussed in the previous

Algorithm 2 Linear Share Generation

Input: secret a_0, number of players n, threshold t
Output: shares
Initialize: p = number of polynomial Coefficients a_i, i=1, 2, $\cdots$, t
$(x = 1 \text{ to } n)$ $f(x) = \left(\sum_0^{t-1} a_i x^i\right)_{mod\ p}$
s

Algorithm 3 Linear Secret Reconstruction

Input: $f(x_1, x_2, \cdots, x_n)$, number of players n, threshold t
Output: secret s
$(i = 1 \text{ to } n)$ $((r = 1 \text{ to } n), \text{and } r \neq i)$ $L_i(x) = \prod_{r=1, r\neq i}^{t} \left(\frac{x - x_r}{x_i - x_r}\right) mod\ p$ $s = f(0) = \sum_{i=1}^{t} f(x_i) L_i(x)$
$f(x)$

section, all the stages in the process must be executed a variable number of times because of which most of the algorithm can't be parallelized. But the process can be pipelined for better performance. Once all the random numbers have been generated, the direct implementation of polynomial evaluation involves $n(n+1)/2$ multiplications and n additions. However, by using the below method, this can be reduced to n multiplications and n additions.

For a secret a_0, and randomly generated polynomial coefficients $a_i : i = 1, 2, \cdots, t$, the share is computed as follows:

$f(x) = ((\cdots(a_n x + a_{(n-1)})x + \cdots + a_2)x + a_1)x + a_0)$

The linear SRU block takes the threshold, and all the player shares as input and reconstructs the secret using Lagrange interpolation as discussed in Algorithm 2. Even this stage can't be completely parallelized because of the same reasons for making the system dynamic, as mentioned above.

4.3.2 Non-linear Secret Sharing

All the basic blocks used in the nonlinear secret sharing scheme are the same as the ones used in the linear scheme. The implementation of these blocks has already been discussed in the linear part of this section. The share generation and reconstruction of nonlinear secret sharing are implemented using all the blocks and techniques discussed in previous parts of the chapter.

The nonlinear share generation block takes secret, threshold, and number of players as input and generates the required number of shares as discussed in Algorithm 3. Since one of our main aims is to design a system that is suitable for dynamic environments where the number of players is changing, all the stages in the process must be executed a variable number of times, because of which parallelization of most of the algorithms is not possible.The nonlinear secret reconstruction block takes in all the player

Algorithm 4 Non-linear Share Generation

Input: $s = (s_1, s_2, \cdots, s_k)$, number of players n, threshold t
Output: (t_0, t_i)
Initialize: $(s_i;\ i$=1, 2, $\cdots$, $k)$, $(a_j$;j= $1, 2, \cdots, k)$
$(i = 1$ to $n)$ $j = 1$ to k $\alpha_j = (1, a_j, {a_j}^2, \ldots a_j{}^(k-1))$ $f(x) = xx^T$
$t_0 = f(s)$
$t_i = f(s + \alpha_i)$
(t_0, t_i)

Algorithm 5 Non-linear Secret Reconstruction

Input: $s = (s_2, s_3, \cdots, s_k)$, number of players n, threshold t
Output: s_1
Initialize: $(s_i;\ i$=1, 2, $\cdots$, $k)$, $(a_j;\ j$=1, 2, $\cdots$, $k)$

$(i = 1$ to $k)$ $(j = 1$ to $n)$ $b_j = t_j - t_0 - \alpha_j \alpha_j^T$ $M(i_1, i_2, \ldots i_{k-1})$ $\begin{bmatrix} s_2 \\ s_3 \\ . \\ . \\ s_k \end{bmatrix} = \begin{bmatrix} b_1 - 2s_1 \\ b_2 - 2s_1 \\ . \\ . \\ b_{k-1} - 2s_1 \end{bmatrix}$

$(0 < s_1 <$ (P−1)) s1

shares and thresholds as input and reconstructs the secret as discussed in Algorithm 4. This block can be divided into four stages. stage 1 computes bi in the equation x, and even this stage can't be parallelized because of the same reasons for making the system dynamic, as mentioned above. stage 2 computes the row sum of the matrix inverse, stage 3 computes the coefficients of the quadratic form, and stage 4 computes the roots of the equation. These stages can be partially parallelized for better performance. The time complexity of the implementation is $O(k)$ clock cycles, where n is the number of players, and k is the threshold.

4.4 Results and Performance Analysis

The time taken for the generation of shares for some pairs of values (n, k) when the clock speed is $100MHz$ is presented. In case of linear share generation, a fixed value of k, increasing n by 5 increases the time taken by approximately $0.45us$, and follows this linear relationship. On the other hand, for a fixed value of n, increasing k by 5 increases time taken by approximately $5.7us$, and follows this linear relationship. Similarly, in case of nonlinear share generation, for a fixed value of k, increasing the value of n by 5 increases the time taken by approximately $0.6us$. On the other hand, for a fixed value of n, increasing k by 5 increases the time taken by approximately $5.9us$.

The implementation of the nonlinear scheme involves computationally heavy modules like matrix

Table 4.2: Computation time for share generation unit of linear, and nonlinear secret sharing

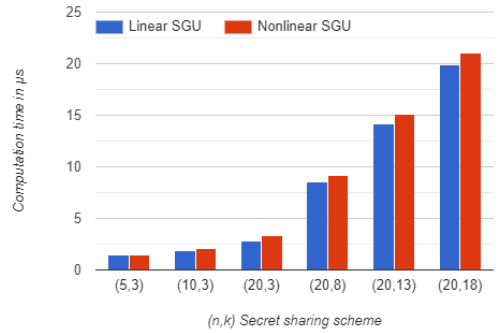
		Linear Secret Sharing		Nonlinear Secret Sharing	
n	k	Time taken by FPGA(μs)	Time taken by Software(μs)	Time taken by FPGA(μs)	Time taken by Software(μs)
5	3	1.46	356.5	1.5	372.2
10	3	1.91	464.8	2.1	476.4
20	3	2.81	550.3	3.3	539.8
20	8	8.51	645.8	9.2	639.3
20	13	14.21	825.4	15.1	830.9
20	18	19.91	901.2	21	906.5

Table 4.3: Computation time for secret reconstruction unit of linear, and nonlinear secret sharing

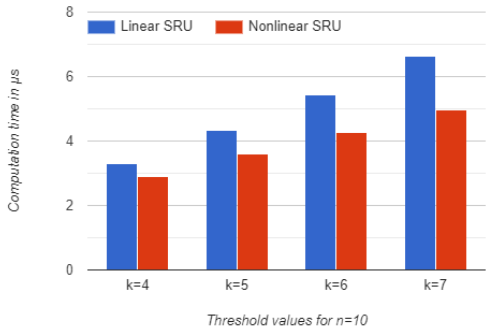
	Linear Secret Sharing		Nonlinear Secret Sharing	
k	Time taken by FPGA(μs)	Time taken by Software(μs)	Time taken by FPGA(μs)	Time taken by Software(μs)
4	3.31	1248.7	2.9	1024.6
5	4.34	1401.8	3.59	1181.1
6	5.45	1549.3	4.28	1357.2
7	6.64	1705.9	4.96	1489.5

inverse and square root, which makes the algorithm more complex than the linear counterpart. The time taken for the reconstruction of the secret for some values of threshold, k when the clock speed is 100MHz is presented in Figure.4.11, which shows that the time taken by both implementations to generate shares and reconstruct secrets are comparable. All the results in this section have been measured using a secret size of 32 bits. For linear secret reconstruction case, increasing the value of k by 1 increases the time taken by approximately $1.1us$, and follows the linear relationship. Similarly, in case of nonlinear secret reconstruction, increasing the value of k by 1 increases the time taken by approximately $0.69us$. When these algorithms were implemented on a computer of clock speed $2GHz$ in python, the time taken by the program was of the order of a few milliseconds. The design implemented in this chapter is significantly faster compared to implementation on a computer, even though the computer is running on a clock that is 20 times faster.

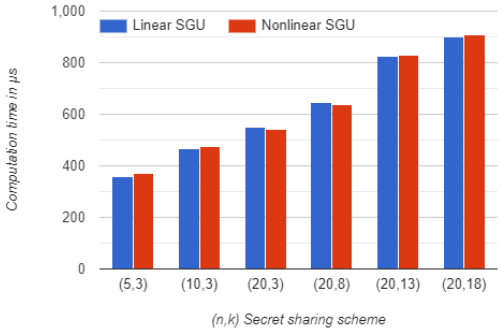
Further, the designs are implemented as an ASIC using SCL $180nm$ PDK. The functionality of the proposed architecture is tested with the Synopsis VCS tool. To obtain a gate-level netlist Synopsis Design Compiler is used, and from the generated netlist, placement and routing are done using ICC compiler. Figure. 4.12 represents the post-placement and routing layout of the proposed linear and



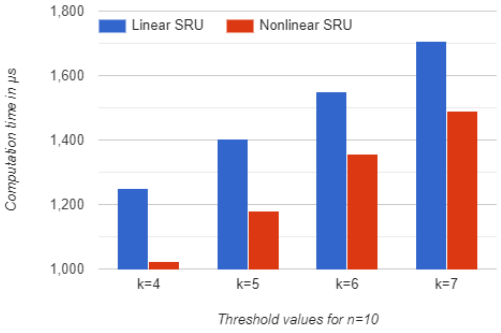
(a) FPGA implementaion of SGU



(b) Softwarechaos implementation of SGU



(c) FPGA implementation of SRU



(d) Software implementation of SRU

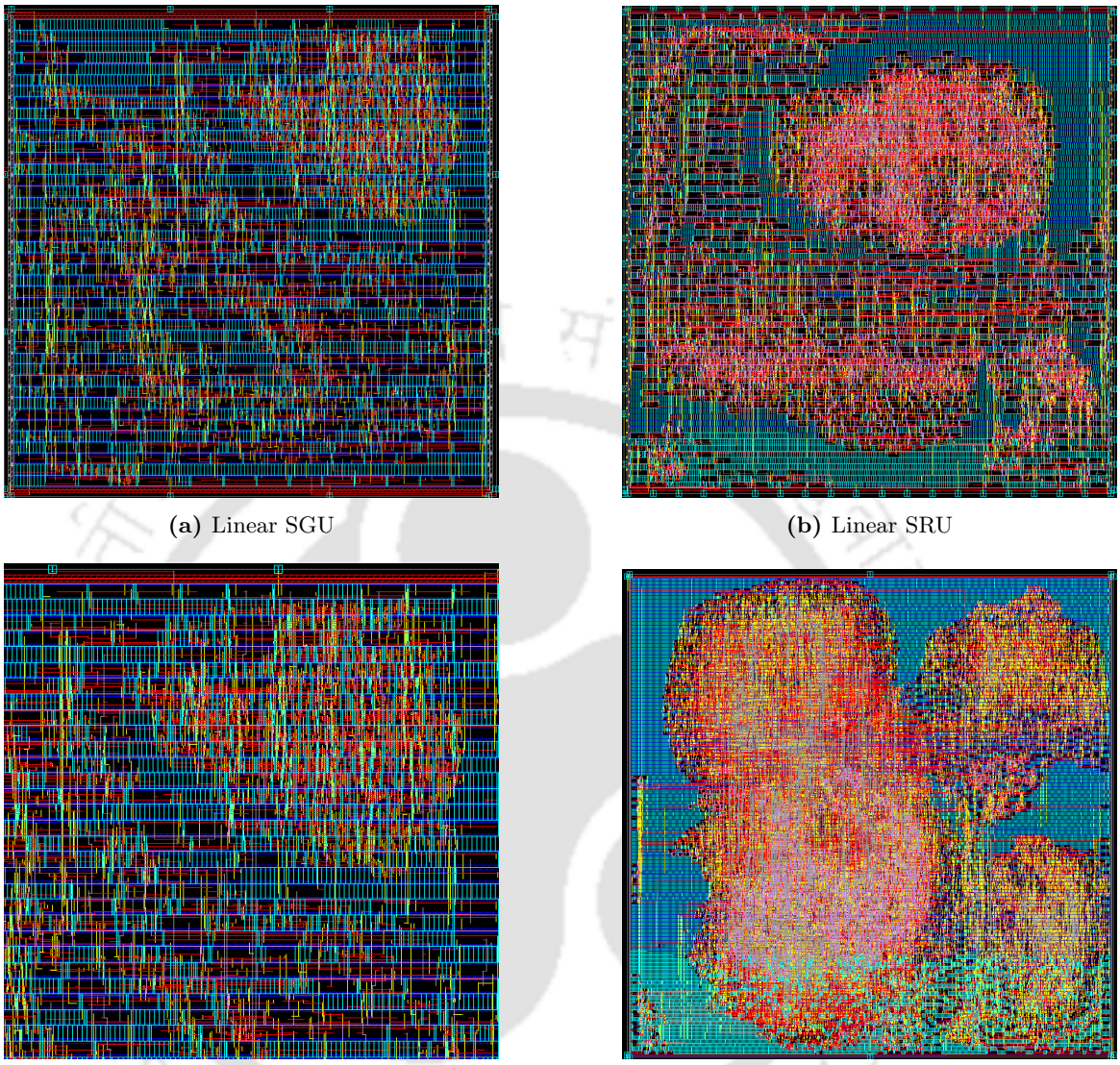
Figure 4.11: Computation time taken by for secret sharing scheme

Table 4.4: Comparison with existing implementations

Secret sharing schemes	Operations	FPGA Implemantations		ASIC Implementations			
		LUT/FF/Adder/XOR/Reg/Multiplier/RAM/MUX	Power (W)	Power (mW)	Area (mm^2)	Delay (ns)	Throughput ($Gbps$)
Jakob Stangl [50]	SGU	1638/1095/ – / – / – / – / – /–	0.033	-	-	-	-
	SRU	2942/7467/ – / – / – / – / – /–	0.3	-	-	-	-
Linear secret sharing	SGU	47/28/43/1/23/1/1/63	0.001	1.72	0.036	7.11	8.38
	SRU	64/59/144/ – /67/2/ – /158	0.001	7.48	0.311	9.5	6.27
nonlinear secret sharing	SGU	47/34/77/1/68/2/ – /104	0.002	4.69	0.09	7.7	7.74
	SRU	141/118/259/ – /185/3/3/399	0.002	9.7	1.72	8.38	7.12

nonlinear secret sharing scheme architectures. The layout results of the proposed schemes like power, area, delay and throughput are discussed as mentioned in table **??**. The resource area for SGU and SRU of the linear secret sharing scheme are $0.036mm^2$, and $0.311mm^2$, respectively, while for the nonlinear scheme, the resource area for SGU and SRU are $0.09mm^2$ and $1.72mm^2$ respectively.

In the design, and implementation [50] done by Jakob Stangl et al., the share generation module in Shamir's mode computes shares of all players in parallel, assuming a fixed number of players for



(a) Linear SGU

(b) Linear SRU

(c) Nonlinear SGU

(d) Nonlinear SRU

Figure 4.12: Layout of secret sharing schemes

the design. We must adopt a different kind of parallelism in our design since one of our aims is to make a system for dynamic input that can be used in applications like SMPC. The method used to compute a single share is the same as this implementation. The above implementation, we are using a pseudo-random number generator which is simpler and faster than the TRNG. There are no existing implementations of Renvall and Ding's nonlinear secret sharing scheme on an FPGA.

For both share generation and reconstruction, the number of resources utilized in both schemes is comparable because of similarities in the implementation of the algorithms, even though they fundamentally differ. Table 4.5 shows the resource utilized in our secret sharing design units while

Table 4.5: FPGA Resource utilization of secret sharing schemes

Resources	Linear SGU	Linear SRU	Nonlinear SGU	Nonlinear SRU
LUT	47	64	47	141
FF	28	59	34	118
Adder	43	144	77	259
XOR	1	-	1	-
REG	23	67	68	185
Multiplier	1	2	2	3
RAM	1	-	-	3
MUX	63	158	104	399

Table 4.6: Comparison with existing secret sharing scheme architectures on FPGA

Schemes	Functional unit	LUT	Percentage utilization (%)	FF	Percentage utilization (%)	Power (W)
Proposed	Linear SGU	47	0.08	28	0.02	0.001
	Linear SRU	64	0.12	59	0.05	0.001
	Nonlinear SGU	47	0.08	34	0.03	0.002
	Nonlinear SRU	141	0.26	118	0.11	0.002
[50]	Linear SGU	1638	6	1095	5.3	0.033
	Linear SRU	2942	11	7467	36	0.3

implemented on Xilinx Zedboard. The power utilized for the linear SGU block in Jakob Stangl [50] is $33mW$, whereas in our design of linear SGU block, $1mW$ power is utilized. Similarly, Jakob Stangl's design uses $300mW$ power for secret reconstruction while our design uses only $1mW$ power. For nonlinear secret sharing, the share generation and secret reconstruction architecture consumes $2mW$ of power individually. Table 4.6 represents the performance comparison of the proposed secret sharing schemes with the existing schemes. By analysing the results, it can be found that the proposed linear SGU block is 97.24% more resource efficint and consumes 96.96% less power than the design presented in [50]. Similarly, the proposed linear SRU block is 98.81% more resource efficint and consumes 99.67% less power than the existing design.

4.5 Summary

This chapter focuses on the development of scalable and dynamically efficient hardware implementations for Shamir's linear secret sharing scheme [48] and Renvall-Ding's nonlinear secret sharing scheme [58] on FPGA (Field-Programmable Gate Array). The objective was to optimize the funda-

mental design components and strategically parallelize the algorithm to minimize computational time, resource usage, and power consumption within the design. The results of this implementation reveal a substantial improvement in the algorithm's performance when it is executed on an FPGA as compared to a software-based implementation. The proposed hardware design for Shamir's linear secret sharing scheme demonstrates a remarkable 98.49% improvement in resource efficiency when compared to its contemporary counterpart. Specifically, the power consumption of the Share Generation Unit (SGU) and Secret Reconstruction Unit (SRU), which are the primary components of Shamir's linear secret sharing scheme, are reduced by approximately 96.96% and 99.67%, respectively, compared to the existing method documented in the literature. In the case of both Shamir's linear secret sharing and Renvall-Ding's nonlinear secret sharing schemes, their hardware implementations achieve an impressive speed increase of approximately 300× compared to their software-based counterparts. Additionally, when implemented as an ASIC using SCL $180nm$ BULK CMOS PDKs, Renvall-Ding's nonlinear secret sharing scheme consumes only $9.7mW$ of power and occupies an area of $1.72mm^2$. Notably, this marks the first instance of a nonlinear scheme being successfully realized in hardware. These accomplishments pave the way for practical, high-speed solutions rooted in secret sharing, opening up new possibilities for applications such as secure cloud backups and Secure Multi-Party Computation (SMPC).

5

Blockchain Mining Architecture

Contents

The security and integrity of any blockchain are hinged on an enumerative algorithm for mining, which blocks the multiple expenditures of bitcoins and meddling with established transactions due to asymmetric encryption, hashing and other algorithms used in the blockchain. This 'proof-of-work' algorithm requires high energy for its execution on the hardware. The present energy consumption is in the range of 100–500 MW, which increases the realization cost. This is because of the high energy dissipation in the form of heat, and the quality hardware is required for the temperature control. Therefore, an efficient ASIC for blockchain technology realization is necessary to reduce the energy consumption, cost and the computational resources.

5.1 Introduction

The work on Hashing, i.e. SHA-256 hash generator, was presented in [63]. It utilized a pipelined architecture to reduce the resources and critical paths. To reduce the memory access and to overcome the performance issues, the nonce generator and nonce validator are proposed as MSA, i.e. Multimode SHA Accelerator, which was validated using Xilinx Alveo U280 FPGA. The works proposed by Zhang et al. [64] and Martino et al. [65] showcase the implementations of various SHA algorithms on FPGA. The proposed work provides a hash with the generation rate by different methods.

Intel Technologies developed an ASIC for cryptocurrency mining. This ASIC provides high performance and efficiency and can be used at various frequencies. It can sense temperature and voltages [66] and has SHA-256 cores, supporting Uni-cast, multi-cast & broadcast capabilities. It also exhibits 10Mbps UART Baud Rate and incorporates a voltage and temperature sensing system. Please note that up to 256 ASICs per chain can be supported to implement cryptocurrency mining. The specifications of this ASIC are given in Table 5.1.

Table 5.1: Specifications of Intel Blockscale Chip

Hash Rate (GH/s)	Upto 580GH/s
Power Consumtion	4.8-22.7 W
Power Efficiency	Up to 26J/TH
Operating Temperature (Ti)	50-80 degree Celsius
Package Flip Chip LGA	7.5 X 7mm

5.2 Flow Chart

Blockchain mining is essential to operating blockchain networks such as Bitcoin and Ethereum. Mining involves validating and adding new transactions to the blockchain by solving computationally intensive puzzles. Miners compete to solve the puzzle, which requires significant computational power and electricity consumption. Once the miners find the solutions, they are rewarded with cryptocurrency and the new blocks of transactions are added to the blockchain. Mining ensures the security, decentralization, and immutability of blockchain transactions. The aim of the proposed work is to design an ASIC with the blockchain technology to perform Blockchain Mining efficiently. The ASIC is designed using the following algorithms, as shown in Figure. 5.1;

(i) Asymmetric Encryption

(ii) Peer-to-Peer (Decentralization)

(iii) SHA (Secured Hash Algorithm)

(iv) Merkel Root Tree

(v) Proof of Work

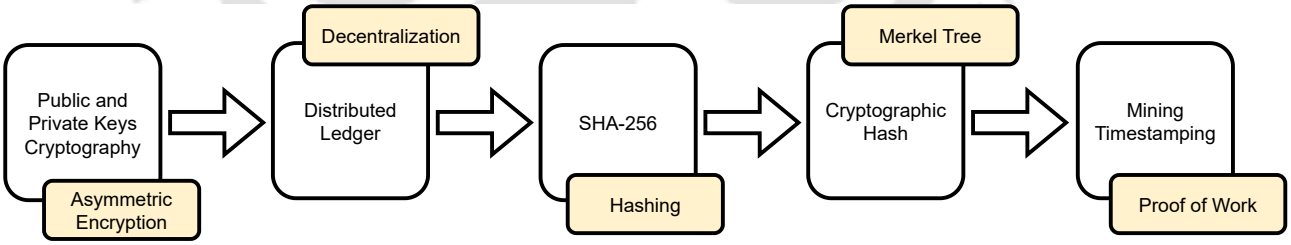


Figure 5.1: Flowchart of Blockchain mining

5.2.1 Asymmetric Encryption and Decryption

In asymmetric encryption, two keys are used instead of one to encrypt and decrypt the data. The keys are known as the private and the public keys. The public keys are known to others in the network, and private keys are only known to the individual [67]. Let's say that there are two users, A and B. Therefore, during an asymmetric encryption, when A sends the data to B, A signs (encrypts) the data with B's public key and sends the data to B. Now, the data can only be decrypted using

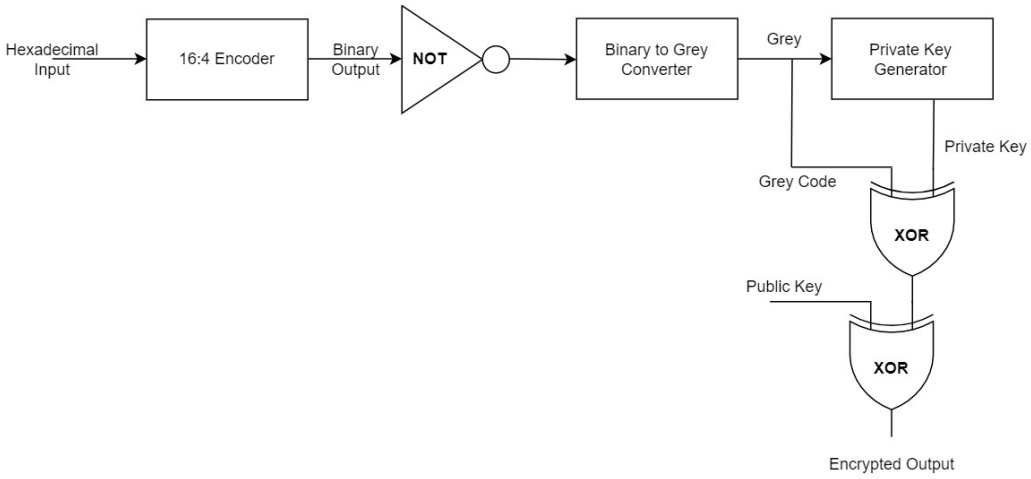


Figure 5.2: Asymmetric Encryption

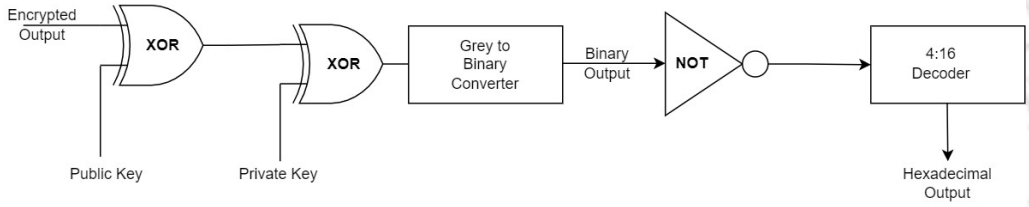


Figure 5.3: Asymmetric Decryption

B's private and public keys together [68]. Thus, when C tries to snoop in and breaks the public key and C somehow manages to break the public key, C is unable to get the plain text because it is again ciphered with a private key, which is known only to B [69].

The approach shown above in the circuit diagram is stated as asymmetric encryption. In this, a hexadecimal input is passed through a 16:4 encoder to generate a binary output, which is further passed through an inverter and later through a binary-to-grey converter. The binary-to-grey converter generates the grey code, which is then passed through a private key generator to generate the private keys. The private key and the grey code are passed through an XOR gate. The XOR gate's output is further passed through an XOR Gate along with the public key (of the receiver) as another input to generate the asymmetric encrypted output, as shown in Figure. 5.2.

The above circuit shown in Figure.5.3 depicts the circuit of asymmetric decryption. It is the reverse of the asymmetric encryption process. The encrypted output is initially passed along with the receiver's public key through an XOR gate. This partially decrypts the message. Now, the output of the XOR gate is again passed along with the receiver's private key as another input through another

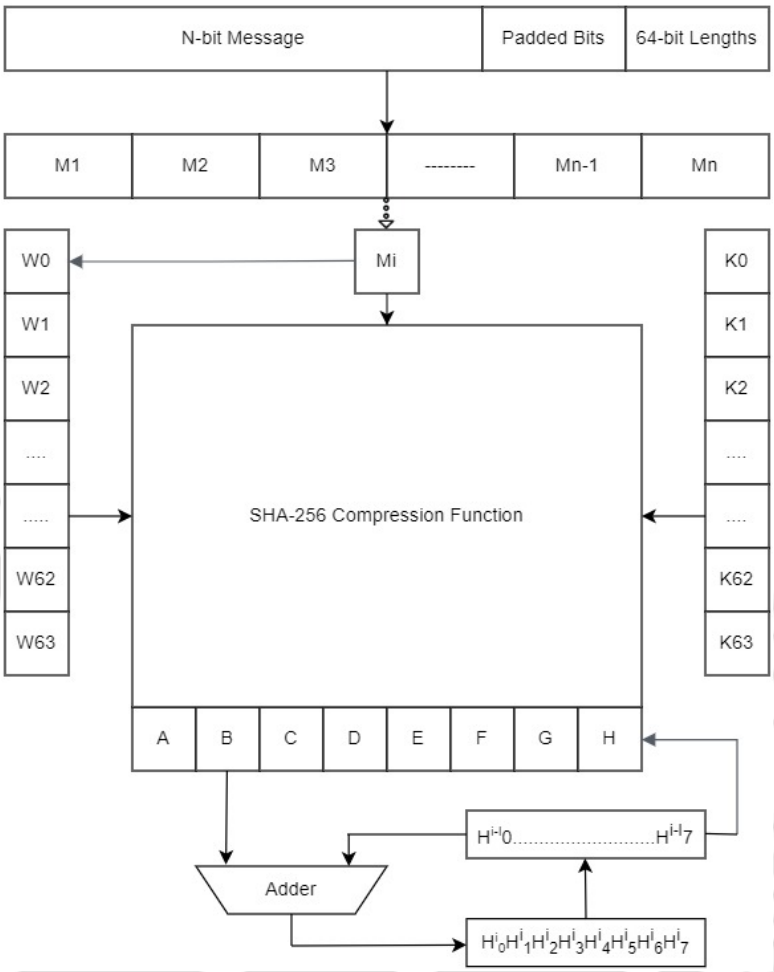


Figure 5.4: SHA-256

XOR gate. The output is further passed through grey-to-binary converter to generate the binary output, which is later passed through an inverter and then a 4:16 decoder to get the hexadecimal output, i.e., the decrypted output.

5.2.2 SHA (Secured Hash Algorithm)-256

SHA-256 stands for Secured Hash Algorithm. Hashing is a process of reducing the length of an N-bit message to a message of particular length bits. This is an irreversible process, i.e., we can produce a hash from the input data, but we can't produce data from the hash value, as shown in Figure. 5.4.

To generate a hash using SHA-256 algorithm, we use an SHA-256 compression function. Bit 1 and zero bits are added to the N-bit message according to the following equation [70]:

$$N + 1 + K = 448 mod 512 \tag{5.1}$$

Where k depicts the number of zero bits to be added [71].

SHA-256 compressor function requires six logical functions to operate on the 32-bit words, creating a 32-bit word as an output.

5.2.3 Merkel Tree

Once the hashes are generated for the data, they must be linked with the blocks in a blockchain network. The previous block contains the next block's hash, and hence, the data is secured since, if there is a change in the transaction, all the blocks change [72].

The Merkel root tree allows us to identify the hash with respect to the data so that we can backtrack which data has been modified. To create a Merkel root tree, let's say there are raw data r1, r2. . . ..r8, each having hashes h1, h2. . . . . h8, respectively, [73]. Now the hash h1 and h2 are appended to generate hash H12=H(h1—h2). Similarly, H34=H(h3—h4)

H56=H(h5—h6)

H78=H(h7—h8)

This process continues until we reach the Merkel tree's top, known as the Merkel root [74].

H1234=H(h12—h34)

H5678=H(h56—h78)

Finally,

H12345678=H(h1234—h5678)

Now, let's consider a case where data r3 is modified. Thus, all the chains containing H3 will be changed, and hence, the malicious activity can be identified, i.e. change in r3 will change H3, H34, H1234, H12345678. Thus, the Merkel root tree validates the data integrity [75], as shown in Figure. 5.6.

5.2.4 Proof of Work

In a blockchain network, since the previous block contains the hash address of the next block, when there is a new transaction, the hash of that particular block changes, which further changes the hashes of all other blocks linked to that block. Therefore, to validate the transaction and add it to a blockchain network, there is a process called mining [76]. To perform mining in a blockchain, the

Proof of Work (PoW) algorithm is required to solve a mathematical puzzle, which in turn creates new blocks in a blockchain network. The Proof of Work system used by Bitcoin is known as the hashcash proof of work consensus. The mathematical puzzle can be defined as:

Given data A, find a number x such that the hash of x appended to A results is a number less than B.

The miner should find a number lesser than the header of the hashed block, i.e., if the target hash is lower or equal to the header as shown in 5.5, then the block is accepted and the miner, who finds it first, is rewarded and all the other miners verify the block mined by the winning miner [77].

This PoW algorithm requires high energy to perform the computations in the range of 100–500 MW. Thus, an alternate efficient solution, such as ASIC realizing blockchain technology, is necessary to reduce energy consumption, cost and more computational resources. As we know, the blockchain can be demarcated as a distributed, decentralized ledger that records the source of a digital transactions [78]. It keeps a permanent record of all the transactions in a blockchain network in a secure, chronological and immutable way.

5.3 Blockchain network architecture

The terms employed in the blockchain network architecture can be defined as follows:

- **Ledger:** A continuously rising file
- **Permanent:** Once a transaction is made, it is recorded permanently in a blockchain network
- **Secure:** It uses asymmetric encryption and advanced cryptographic algorithms to secure the information
- **Chronological:** The transactions are stored in a manner that they can be backtracked, i.e., every transaction happened after the previous one.
- **Immutable:** The transactions can never be altered

Every block in a blockchain contains data. Once a transaction is completed and validated, it stores the data as a block in a blockchain network and hence, a new block is generated as shown in Figure. 5.6. Every block in a blockchain is connected to its previous block, which makes it almost impossible to tamper with the data.

(i) **Header Hash:** It is required to indicate the location of a block in a blockchain. The encrypted message is formulated with the block header's hash as the input once a new block is created in the chain. The information stored inside the block is different from this data.

(ii) **Previous Block Hash:** As discussed above, to maintain the integrity of a blockchain, every next block is connected to the previous block and the medium to connect the two blocks is the hash address of the blocks, i.e., the header of n^{th} block contains the hash of $(n-1)^{th}$ block.

(iii) **Nonce:** It is a 32-bit random value that is used to increase the complexity of the mathematical puzzle to be solved by the miner.

(iv) **Block Version:** It is a 4-byte number which stipulates a new version of the blockchain.

(v) **Timestamp:** It specifies the real-time at which the present block is formed.

(vi) **Merkle Root Tree Hash:** It contains the hash addresses of the transactions in a specific block. It formulates the hash of the transaction and the top hash, known as the Merkle root, and is derived by formulating the hash of all the intermediate hashes.

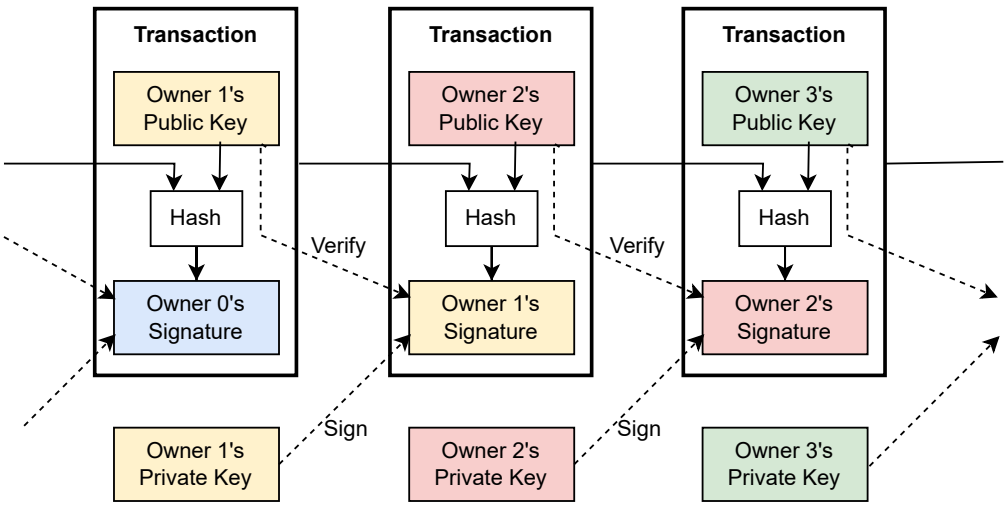


Figure 5.5: Peer-to-peer decentralized system

5.4 Implementation and performance analysis

The proposed work is implemented using Verilog HDL. It is followed by the logic simulation and equivalence check. The details of the entire implementation is given below.

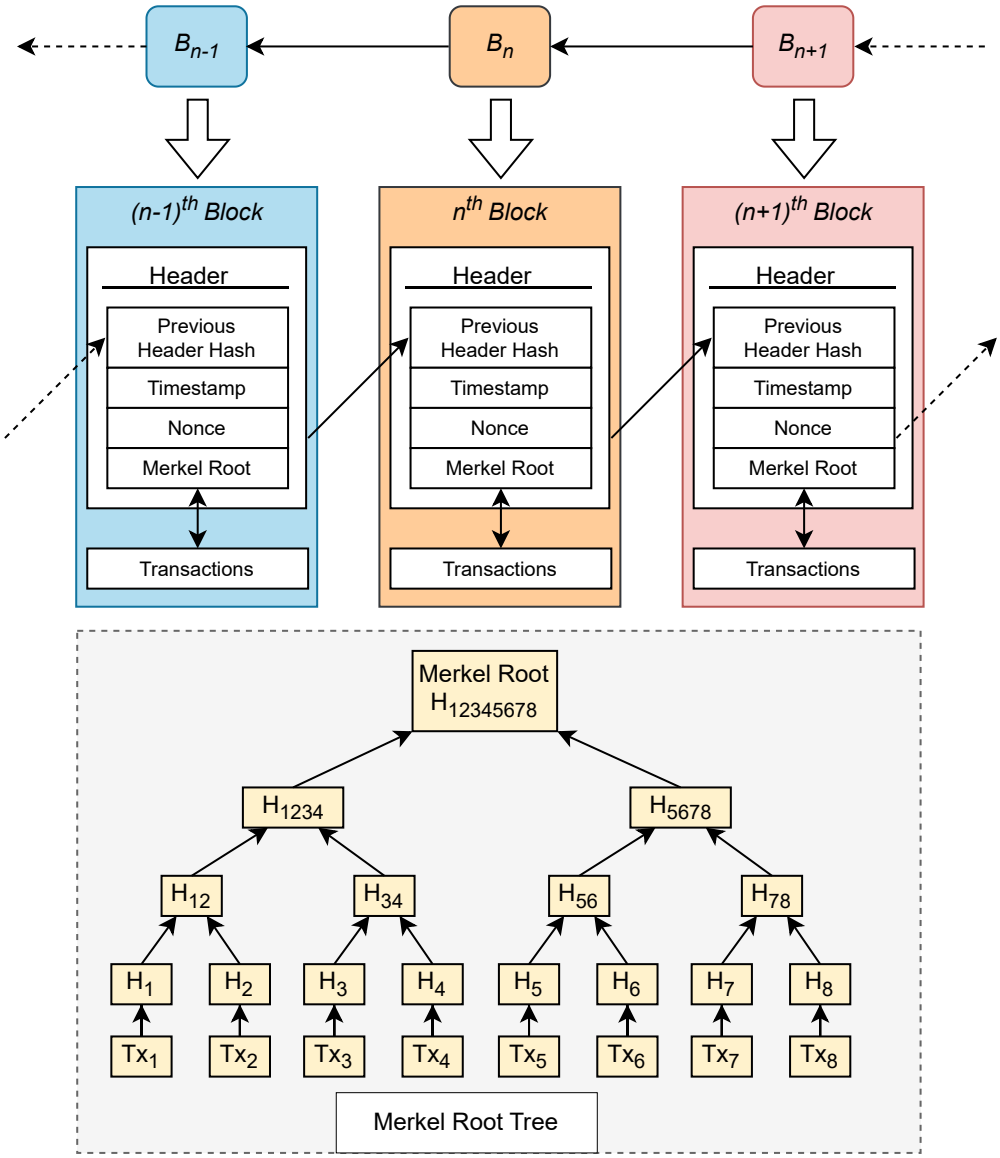


Figure 5.6: Blockchain network architecture

- **Verilog Simulation:** The RTL design of the algorithm of the blockchain mining is analyzed using ModelSim and Xilinx ISE-Design Suite Software. We obtained the output of the hash generated after the mining process using this environment.

- **Logic Synthesis:** Once the analysis is performed, we need to map our design for the actual hardware implementation, i.e., the conversion of RTL Code to the gate level netlist after mapping the logic according to the hardware. This is achieved using Synopsys Design Compiler (Synopsys DC) tool. This step also generates a Synopsys Design Constraints (.SDC) file.

- **Logic Equivalence Check:** Once our logic synthesis is performed, we can move towards a logic equivalence check using the Verilog design and the gate level netlist. The design and the netlist are compared using Synopsys Formality tool.

5.4.1 Synopsys IC Compiler:

After the Logic Equivalence Check, we input our design to the Synopsys IC-Compiler. In order to achieve it, we set up TLU+ files, create a library and set up a link and target library. The floorplan of the proposed design prepared using ICC after importing the required information. Once the floorplan is ready, we can start the pre-routing process by adding the VDD and VSS nets, followed by creating Rings and adding the Nets. After that, we fill the empty rows and legalize and optimize the floorplan for placement and routing. Then, we perform placement and routing optimizations, followed by static timing analysis and clock optimization. Finally, after getting the GDS-II layout, various checks like DRC, LVS, Short Circuit, floating points, etc., are performed to determine correctness of the design.

5.4.2 FPGA Design Flow and Testing

The design is ported on an FPGA to validate its correctness. We have employed a ZYNQ-ZC702 FPGA Board for the testing. After implementing the design, we perform the synthesis and checks it using various inputs for the correctness.

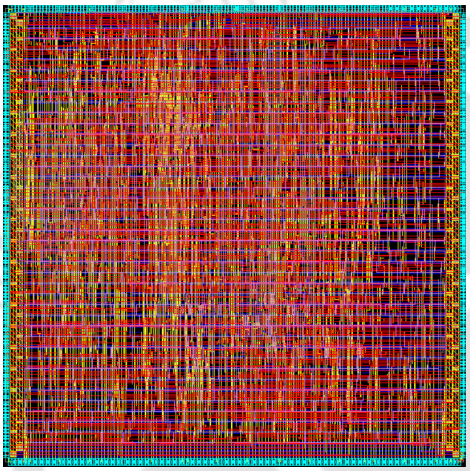
The test results and implementation details are presented in Table 5.2 and Table 5.3. Details of the ASIC implementation are stated in Figure. 5.7. The proposed design is realized using SCL 180nm Bulk CMOS technology. The design is also compared with the implementation details provided by Intel as shown in Table 5.4, exhibiting its superioriy.

Table 5.2: FPGA Resource Utilization of Blockchain Mining Architectures (Zynq ZC702)

Resources	Total Avalable	Resources Utilized	Percentage Utilization (%)
Slice LUTs	53200	4547	8.60
Slice Registers	106400	5048	4.74
Slice	13300	1576	11.85
LUT as Logic	53200	4485	8.43
LUT as Memory	17400	62	0.35
BRAM	140	6	4.28
DSPs	220	6	2.72
BUFIO	16	1	6.25
Total Resources	**243876**	**15731**	**6.45**

Table 5.3: Comparison with Existing Architectures

Work	[65]	[64]	[63]	Proposed Chip
FPGA	$KCU116$	$KCU105$	$AlveoU280$	$ZYNQZC702$
Throughput (Mhps)	5.05	200	1280	3825.92
Slice	-	4870	44769	1576
Slice LUTs	5385	23270	210880	4547
FF	4314	45312	366208	5048
Power (W)	0.43	2.56	2.56	1.493
Area Efficiency (Mhps/1kLUT)	0.94	8.59	6.07	841.41
Energy Efficiency (Mhps/W)	11.74	78.13	78.13	2562.57



CMOS process	SCL $180nm$
Area	$0.034mm^2$
Voltage	$1.98V$
Power	$3.1132mW$

Figure 5.7: ASIC layout Design and Specification of Blockchain Mining chip

Table 5.4: Comparison with Intel Blockscale ASIC

Work	**Intel Blockscale Chip [66]**		**Proposed Chip**
	Worst Case	**Best Case**	
Power Consumption (in $Watt$)	22.7	4.8	3.1132×10^{-3}
Hash Efficiency (in $GH/s/W$)	25.5	120.83	1221.8

5.5 Summary

The various blocks of blockchain mining algorithms are designed, and their validations are performed using Verilog HDL. It can be seen that an ASIC having blockchain technology is the need of the hour to reduce the energy consumption, cost, and to optimize the computational resources. In the proposed work, an ASIC is successfully designed and tested on the Hardware (FPGA), providing better results than the existing ones. The proposed design is power efficient,

consuming power in the range of milli Watts. The hardware design of the blockchain mining, i.e., the SHA-256 algorithm, is better than its contemporary designs. However, our proposed design has lower hash rate compared to Intel Blockscale Chip, but the latter one consumes more power than the design proposed in this thesis. To justify our design, we have also compared its hashing efficiency with Intel's best case and worst-case scenarios, and our ASIC comes out to be more than 10× efficient than the Intel Blockscale chip.

6

Approximate Multipliers

Contents

6.1 Introduction

In the cryptography systems, a large number of multipliers are employed, which compute intensive algorithms. They contribute mainly in the power consumption and speed, thus the efficient multipliers are the need of the hour. Approximate computing has added a unique dimension in the area of digital design by reducing area, power and delay. It can be employed to design approximate multipliers, which can be utilized in the cryptographic primitives and applications. The demand of efficient approximate multipliers is enhancing due to the high speed and fault tolerance as well as its power efficiency. This chapter presents comparison of few recent approximate multiplier designs. Experimental results based on the accuracy and circuit parameters are presented. The accuracy parameters are amplitude data accuracy (ACC_amp), information data accuracy (ACC_inf), error rate (ER) and mean error distance (MED). The circuit parameters are delay, power consumption, area. Based on these parameters, the best design in terms of power consumption and area is datapath complexity reduction approximate multiplier, which exhibits 58% less power and 61% less area as compared to broken array multiplier. A descriptive view of the accuracy parameters are proposed in [79] and [80], which are:

- Accuracy for amplitude data (ACC_amp): It is used to quantify the errors in the amplitude, and is represented as $ACC\_amp = 1 - (R_c – R_e)/R_e$, where R_c is the correct result and R_e is the result of the approximate multiplier.
- Accuracy for information data (ACC_inf): It measures error significance and can be represented as $ACC\_inf = (1 - B_e)/B_w$, where B_e is the number of error bits and B_w is the bit width of the data. The error rate is defined as the probability of occurrence of an error.
- Mean Error Distance: It is the value of the error distances of all possible outputs for each input of a circuit is called the mean error distance.

A descriptive view of the circuit parameters are given below.

- *Delay:* The propagation delay or delay is the time delay between the 50% transition of the rising/falling input voltage and the 50% transition of the falling/rising output voltage.
- *Power Consumption:* When the transistor makes a transition from one state to another, there is a consumption of power in the process, which is calculated in this analysis.

– *Area:* Area is measured according to three factors, area covered (μm^2), number of gates and number of transistors.

Following sections depict details of the approximate multipliers and their comparison for the completeness.

6.2 Categories of Approximate Multipliers

This section presents various approximate multipliers in detail.

6.2.1 Approximate Error Tolerant Multipliers:

2×2 Under designed Multiplier:

This design is proposed in [81]. It introduces error into the multiplier using the 2×2 multiplier as a building block. The motivation behind this design is to introduce a 4-bit result in 3-bits. So, the multiplication of 3×3 is 7 instead of 9. Figure. 6.1 shows the gate level circuit of the multiplier. Partial products are produced by using the inaccurate 2×2 block and then adding the shifted partial products larger multipliers are built. Figure.6.2 shows the description of the design.

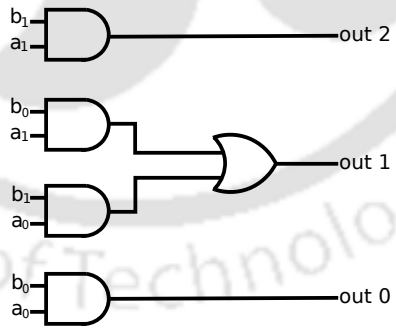


Figure 6.1: Gate level circuit of the multiplier

Datapath complexity reduction approximate multiplier:

This multiplier design is proposed in [82]. Two changes have been carried out in [81] to design this approximate multiplier. First, by approximating out_0 to 0, we further approximate the 2×2 multiplier building block . Though the critical path of the resulting multiplier remains unchanged, the area has reduced. Figure. 6.3 shows the logic functions of the approximate 2×2 multiplier.

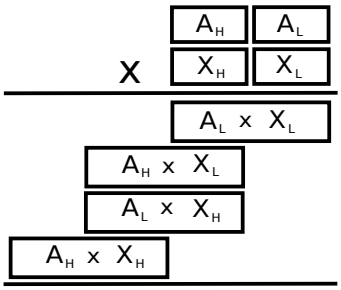


Figure 6.2: Description of a large multiplier

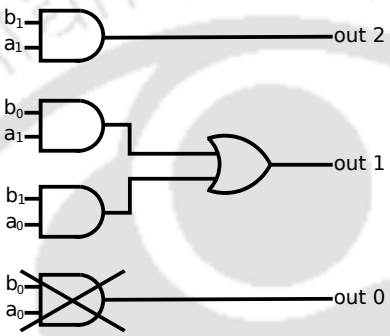


Figure 6.3: Logic functions of the approx. 2×2 multiplier

The second improvement that we propose is the manner in which a bigger multiplier is assembled. The multiplier in [81] utilizes the equivalent 2×2 surmised multiplier to figure every single incomplete item. Rather, three degrees of guess are presented inside the bigger multiplier. The least huge halfway item with greatest level of guess, the center incomplete items with medium estimation and the most huge fractional item with no guess are determined. The inspiration driving presenting three degrees of estimate is when two huge numbers are duplicated; just the lower fractional items are figured with guess. This improves the exactness of the multiplier contrasted with the reference multiplier.

Error Tolerant Multiplier:

The above mentioned approximate multiplier design is introduced in [83]. The multiplication algorithm can be explained as shown in Figure. 6.4. To begin with, the information operands are part into two sections: a duplication part in which various higher request bits are incorporated and a non-increase part which is comprised of the rest of the lower request bits. The duplication part experiences precise augmentation process. For the non-augmentation part, checking accomplished for each piece position from left to right (MSB to LSB) and the checking procedure is finished when either or both of the two operand bits are "1" and all the bit positions are set to

"1" from that bit onwards. The situation when both operand bits are "0", the comparing item bit is set to "0".

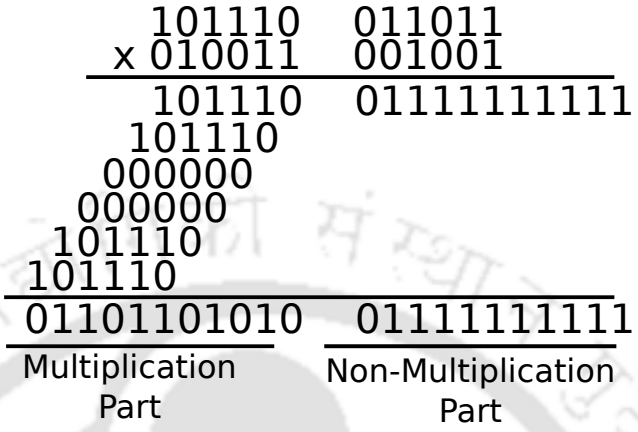


Figure 6.4: Example of the approx. multiplication algorithm

Accuracy Configurable Multiplier(ACMA) :

The ACMA design is proposed in [84]. It improvises the recursive multiplication architecture and designs a new approximate pipelined architecture. The recursive multiplication builds a recursive tree. Let A be the multiplicand and X be the multiplier and both are assumed to be of $2b$ bits each. A and X can be written as $A = AHAL$ and $X = XHXL$, where each of them are of b bits each. The most critical $2b$ bits (out of an aggregate of $4b$ bits) considered as exact to a huge degree. Further, out of the least noteworthy $2b$ bits remaining, lower $b/2$ bits are exact and upper $3b/2$ bitsare incorrect. The lower $b/2$ bits are kept exact in light of the fact that it doesn't require a great deal of equipment and simultaneously exactness increments. As it were, in the last item, out of $4b$ bits least huge $b/2$ bits are exact and most huge $2b$ bits are precise to a huge degree and remaining $3b/2$ bits are off base. The approximate multiplication algorithm is shown in Figure. 6.5.

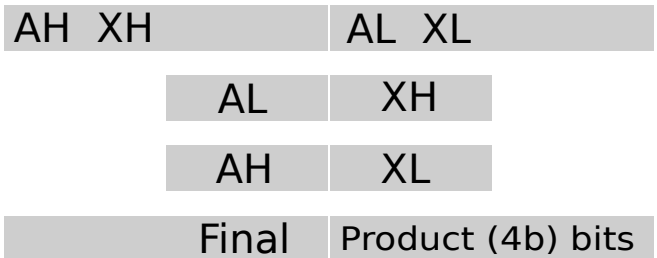


Figure 6.5: The approx. multiplication algorithm of ACMA

6.2.2 Truncated Multipliers :

In [85], it is mentioned that multiplication of two numbers brings about an product with twice the original bit width. It is required to truncate the product bits to the desired accuracy to reduce area cost, leading to the structure of truncated multipliers, or fixed-width multipliers. By and large, there are two truncated multiplier structure techniques, in particular, steady and variable remedies, contingent upon how to repay the mistake acquainted due with the end of the least noteworthy halfway item (PP) bits. Consistent amendment structures diagnostically process the normal truncation mistake of the truncated PP bits and blunder is remunerated by including a line of steady into the lattice of the PP bits. Variable amendment plans diminish the all out truncation mistake by thinking about the least noteworthy piece of the PP bits.

Fixed-width multipliers is a subset of truncated multipliers which figure just n significant bits (MSBs) out of the $2n$-bit product for $n \times n$ increase and additional revision/remuneration circuits are utilized to diminish truncation errors. Fixed-width multipliers are generally utilized for advanced signal processor activities, for example, filtering, convolution, and fast Fourier or discrete cosine change. Along these lines, in this discussion, just the basic fixed width multiplier is examined with low error.

Low error fixed-width multiplier:

The low error fixed-width multiplier is proposed in [86]. Parallel multiplier circuits consist of two parts: one for the higher order partial products (MP) and one for the lower order partial products (LP). The most basic fixed width multiplier removed the LP and substituted with a 0 to be carried to the residual part to generate MP'. This reduced the area to half, but also generated a large amount of error. So, as to reduce the error, a circuit C_g is designed and is placed in place of 0. It consists of $n-2$ AO cells and one 2- input AND gate. Each AO cell consists of one 2-input AND gate and one 2-input OR gate. Each input of C_g is fed with a proper product term X_iY_j i.e. the partial products.

6.2.3 Bio-inspired imprecise multiplier

Instead of giving the precise value of the result, Bio-inspired Imprecise Computational blocks (BICs) [87] provide an applicable estimation of the same at a lower cost. In terms of area, speed, and power consumption, these novel structures are more efficient with respect to their precise

counterparts. Here the 'Broken Array Multiplier' or BAM is discussed which is a bio-inspired imprecise multiplier structure.

The Broken Array Multiplier (BAM)

The broken-array multiplier (BAM), proposed in [87], is fundamentally the same as the design of an array multiplier. An Array multiplier with an m-bit multiplicand and n-bit multiplier comprises of mxn comparative cells called carry-save adder (CSA), trailed by a m-bit vector merging adder which converts carry-save redundant form to regular binary form. Every cell contains a 2-input AND gate to produce partial product and a full-adder (CSA) to add the partial products into the running sum.

A BAM breaks the CSA array and cells giving smaller and faster circuit while providing approximate results are eliminated. Depending on two introduced parameters namely Horizontal Break Level (HBL) and Vertical Break Level (VBL), the number and position of the eliminated cells are decided. The HBL=0 means that there is no horizontally eliminated CSA cell. If HBL increases, the horizontal dashed line moves downward and all cells falling above the dashed line are eliminated. So also the VBL=0 does not eliminate any CSA cells but as the VBL increases, the vertical dashed line moves left and eliminates all separated right-side cells. The respective outcome of the all eliminated cells are considered to be null. Figure. 6.6(a) shows the hardware structure of BAM, Figure. 6.6(b) shows the CSA cell and Figure. 6.6(c) shows the vector merging adder cell.

6.2.4 Inaccurate [4:2] Compressor based multipliers

A $[4:2]$ compressor accepts four equal weights as input and produces two outputs. It may be designed by using two $(3,2)$ counters. An intermediate carry, t_i is fed into the next column and accepts a carry, t_{i-1}, from the previous column, [88].

Inexact (or approximate) computing is an appealing paradigm for digital processing at nanometric scales. Approximate computing is especially interesting for computer arithmetic designs. The difference between an inaccurate computer and an ordinary counter is that an ordinary counter gives the outcome 1002 when all the 4 inputs are 1, but an inaccurate computer simplifies the outcome by representing the 3-bit outcome in 2-bits, [80]. This classification manages

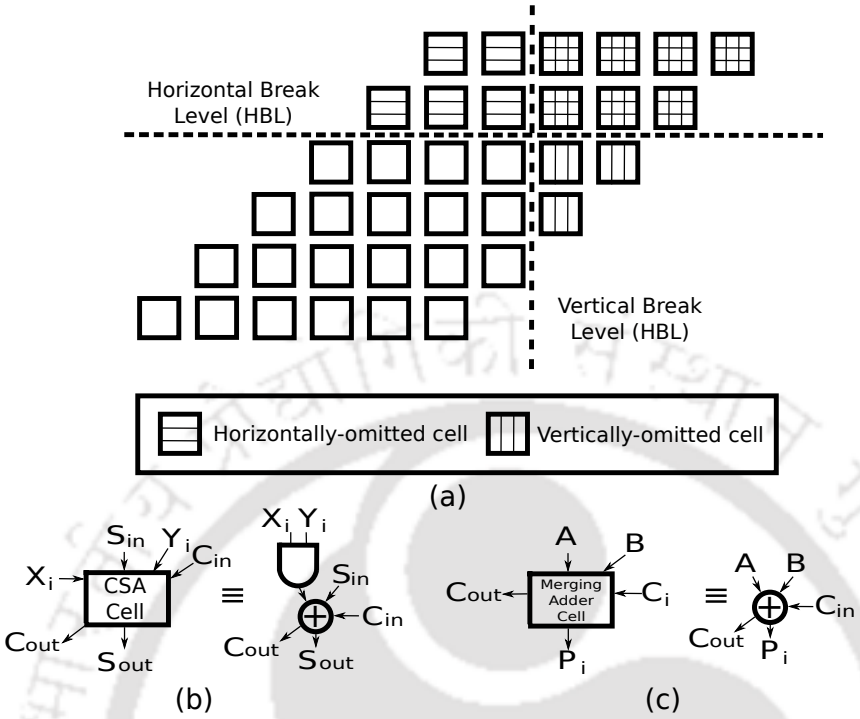


Figure 6.6: (a) BAM (b) CSA array (c) Vector merging Adder

three inexact designs of inexact compressors to be applied on a Dadda Multiplier.

Design 1:

In an accurate compressor, out of 32 states, the carry output has the same value of the input C_{in} in 24 states. In Design 1, proposed in [89], by changing the value of the other eight outputs, the carry is simplified to:

$$C_{in} \times Carry = C_{in} \tag{6.1}$$

Being the higher weight of a binary bit, an erroneous value of carry output signal creates a distinction estimation of two in the output. For example, if "01001" is given as the input pattern, then the right output is generated that is "010" which is equivalent to 2. By simplifying the carry output to 'Cin', the inexact compressor produces the "000" pattern as output which is equivalent to value 0. This significant contrast can not be adequate. However, it has to be redressed or decreased by rearranging the 'C_{out}' and 'Sum' signals. Specifically, the difference between the approximate and the exact outputs can be reduced by simplifying the sum to a value of 0 and the the complexity of its design is also reduced. Additionally, as due to the presence of certain errors in the sum signal, the delay of creating the inexact sum reduces resulting in

reduction of the overall delay of the design. Figure.6.7 shows the gate level structure of Design 1.

$$Sum = \overline{C_{in}}(\overline{(x_1 \oplus x_2)} + \overline{(x_3 \oplus x_4)}) \tag{6.2}$$

$$Cout = \overline{(\overline{(x_1 x_2)} + \overline{(x_3 x_4)})} \tag{6.3}$$

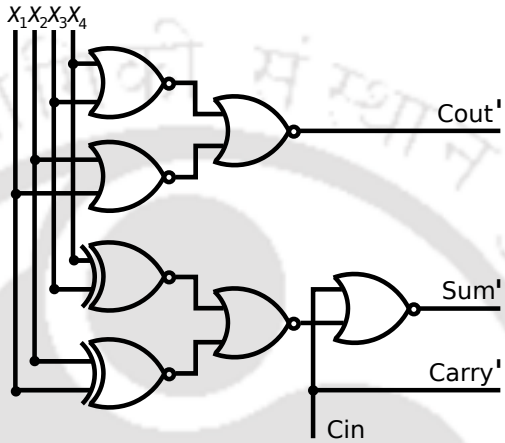


Figure 6.7: The gate level structure of Design1

Design 2:

To additionally increase performance as well as to decrease the error rate, a subsequent design of an inexact compressor is also proposed in [89]. In the previous part, the proposed equations for the approximate $Carry$ and C_{out} can be interchanged as the '$Carry$' and 'C_{out}' outputs have the same weight,. In this new design, C_{out} is constantly equal to 'C_{in}' and carry utilizes the right hand side of (3). As C_{in} is zero in the first stage, in rest of the stages, C_{out} and C_{in} are zero. Therefore, in the hardware design, C_{in} and C_{out} can be overlooked. Figure. 6.8 shows the gate level structure of this inexact $4-2$ compressor and the expressions below gives its outputs.

$$Sum = \overline{C_{in}}(\overline{(x_1 \oplus x_2)} + \overline{(x_3 \oplus x_4)}) \tag{6.4}$$

$$Carry = \overline{(\overline{(x_1 x_2)} + \overline{(x_3 x_4)})} \tag{6.5}$$

Design 3:

In Design 3, proposed in [80], the main idea is to use a $2:1$ MUX in place of a XOR gate to reduce the delay and the power consumed by the multiplier. Let x_1, x_2, x_3 and x_4 are the four inputs and 'Sum' and '$Carry$' are the two outputs. The error occurs when all the four

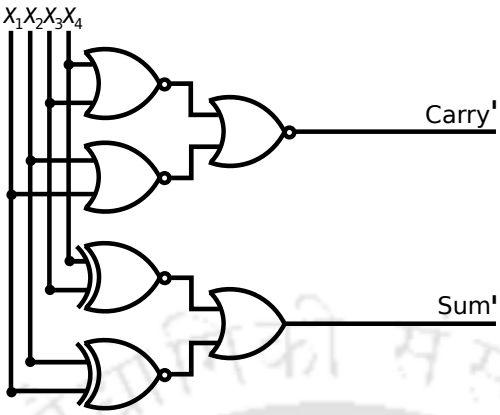


Figure 6.8: The gate level structure of Design 2

summands are 1 and the output 1112 is reduced to 102. It can be used to build an inaccurate 4×4 multiplier. It helps to reduce the adding stages from four to two to reduce the delay and power.

6.2.5 Approximate Logarithmic Multiplier :

A strategy of computer multiplication and division is proposed in [90], which utilizes binary logarithms. The logarithm of a binary number may be found approximately from the number itself by simple shifting and counting. A basic add or subtract and shift operation is needed to multiply or divide. Since the logarithms used are inexact, there can be errors in the outcome.

Simple Mitchell based logarithmic multiplier:

The simple Mitchell based logarithmic multiplier is proposed in [90]. It uses the concept of finding the multiplication of two numbers using binary logarithms. The arithmetic used is an approximation to the actual logarithm. To find the binary logarithm of a binary number, avoid the position of the most significant 1, and consider the number as a binary fraction. For example, approximate lg_{13} is 3.625 decimal and 11.101, where the largest characteristic is 3, since 13 consists of 4-bits. A step wise process is explained below.

Let A and B be two registers containing the two numbers, whose word size is of 8 bits. So, the largest characteristic is 7.

Step 1: Shift A and B left until their most significant "one" bits are in the left-most positions. After the shifting is completed, the counters contain the characteristics of the logarithms of A

and B.

Step 2: Shift bits $0-6$ of A and B into bit positions $0-6$ of registers C and D. C and D now contain the logarithms of the original numbers.

Step 3: Add $C + D \rightarrow E$. The result is now stored in register E.

6.3 Simulation Results

The experimental results are divided in two parts:

(i) **Error Characteristics:** It discusses the accuracy parameters of all the aforesaid approximate multiplier designs architectures.

(ii) **Circuit Characteristics:** It discusses the circuit parameters of the five aforesaid approximate multiplier designs architectures from four of the five categories.

6.3.1 Error Characteristics:

The accuracy parameters are evaluated using MATLAB R2013a software for the approximate multiplication algorithms. It is observed that with the increase of bit width, the accuracy of the multiplier decreases. The approximate functions are simulated with 10000 random inputs. Some of the characteristic observations in the accuracy parameters are discussed here.

Table 6.1: 2×2 Under Designed Multiplier

Design Name	2×2 Under Designed Multiplier							
Parameters	2	4	6	8	10	12	14	16
Delay (ns)	10.0595	20.119	30.141	40.238	50.2975	60.357	70.4165	80.476
Power (μW)	2.6574	5.3148	7.9721	10.6295	13.2869	15.9443	18.6016	21.259
Area (μm^2)	15.6528	31.3055	46.9583	62.611	78.2638	93.9165	109.5693	125.222
Area (gates)	13	25	38	50	62	75	88	100
Area (transistors)	84	168	252	336	420	504	588	672
ER	0.054	0.882	0.972	0.991	0.999	1	1	1
MED	0.108	46.96	968.35	1.63E+04	2.63E+05	4.21E+06	6.70E+07	1.07E+09
Acc_amp	0.98	0.296	0.082	0.082	0.009	0.004	0.0022	0.0011
Acc_inf (E-04)	9.59	6.79	6.21	5.9	5.69	5.6	5.52	5.48

(i) **Acc_amp:** The value of Acc_amp < 1. The maximum value is observed in 2×2 Under Designed Multiplier, of 0.98 for its 2-bit multiplier, while the minimum value is observed in Simple Mitchell based Logarithmic multiplier, of -27.018 for its 16-bit multiplier. It reaches a negative value for two designs, Datapath complexity reduction approximate multiplier and

Table 6.2: Datapath complexity reduction

Design Name	Datapath complexity reduction approximate multiplier							
Parameters	2	4	6	8	10	12	14	16
Delay(ns)	10.0595	20.119	30.141	40.238	50.2975	60.357	70.4165	80.476
Power(μW)	2.6139	5.2278	7.8413	10.4555	13.0694	15.6832	18.2971	20.911
Area (μm^2)	15.4109	30.8219	46.2328	61.6437	77.0546	92.4656	107.8765	123.2874
Area (gates)	12	24	37	49	61	74	87	99
Area (transistors)	83	165	247	330	412	494	576	658
ER	0.553	0.842	0.974	0.992	0.998	1	1	1
MED	1.72	95.22	327.23	2.62E+04	1.61E+05	1.29E+07	6.08E+07	6.70E+09
Acc_amp	0.528	-0.0781	0.53	-0.156	0.287	-1.23	0.117	-3.187
Acc_inf(E-04)	7.82	6.19	6.15	5.66	5.61	5.41	5.45	5.28

Table 6.3: Error tolerant Multiplier

Design Name	Error tolerant Multiplier							
Parameters	2	4	6	8	10	12	14	16
ER	0.702	0.928	0.98	0.994	1	0.999	1	1
MED	41.42	115.65	253.68	1.29E+04	2.41E+05	4.21E+06	6.74E+07	1.03E+09
Acc_amp					0.052	0.015	0.0055	0.0024
Acc_inf(E-04)	2.81	5.34	5.89	5.33	5.4	5.57	5.41	5.41

Table 6.4: ACMA multiplier

Design Name	Data for ACMA multiplier							
Parameters	2	4	6	8	10	12	14	16
ER	0.262	0.855	0.972	0.993	0.996	1	1	1
MED	0.524	49.17	920.53	1.65E+04	2.57E+05	4.11E+06	6.84E+07	1.04E+09
Acc_amp	0.905	0.271	0.0937	0.331	0.0168	0.0063	0.0033	0.0015
Acc_inf(E-04)	8.89	6.77	6.24	5.89	5.8	5.64	5.5	5.52

Table 6.5: Simple Mitchell based Logarithmic multiplier

Design Name	Data for Simple Mitchell based Logarithmic multiplier							
Parameters	2	4	6	8	10	12	14	16
ER	0.949	0.996	1	1	1	1	1	1
MED	2.89	45.68	710.46	1.17E+04	1.98E+05	2.98E+06	4.80E+07	7.29E+08
Acc_amp						-12.78	-7.61	-27.018
Acc_inf(E-04)	6.04	6.06	5.83	5.57	5.47	5.37	5.31	5.29

Table 6.6: Low error fixed width multiplier

Design Name	Data for Low error fixed width multiplier							
Parameters	2	4	6	8	10	12	14	16
Delay(ns)	10.0695	20.139	30.2085	40.278	50.3475	60.417	70.4865	80.556
Power(μW)	4.5469	9.0938	13.6406	18.1875	22.7344	27.2813	31.8281	36.375
Area (μm^2)	27.898	55.7959	83.6939	111.5918	139.4898	167.3877	195.2857	223.1836
Area (gates)	20	40	60	80	100	120	140	160
Area (transistors)	146	291	437	582	728	873	1019	1164
ER	0.508	0.888	0.964	0.989	1	1	0.999	1
MED	1.39	43.92	783.96	1.17E+04	1.87E+05	2.98E+06	4.68E+07	7.62E+08
Acc_amp	0.662	0.306	0.189	0.188	0.192	0.2174	0.223	0.2438
Acc_inf(E-04)	8	6.52	6.01	6.02	5.81	5.81	5.71	5.66

error tolerant multiplier. An interesting observation can be seen in this multiplier is that the value becomes negative after an interval of 4-bits. The value of Acc_amp is countable

Table 6.7: Broken Array Multiplier

Design Name	Data for Broken Array Multiplier							
Parameters	2	4	6	8	10	12	14	16
Delay(ns)	10.0788	20.1575	30.2363	40.315	50.3938	60.4725	70.5513	80.63
Power(μW)	6.2617	12.5234	18.7851	25.0468	31.3085	37.5702	43.8319	50.0936
Area (μm^2)	43.5068	87.0135	130.5203	174.027	271.5338	261.0405	304.5473	348.054
Area (gates)	32	63	95	126	158	189	221	252
Area (transistors)	221	441	661	881	1101	1321	1541	1761
ER	0.421	0.873	0.97	0.986	0.999	1	1	1
MED	1.27	52.58	1.03E+03	1.67E+04	2.49E+05	4.29E+06	6.86E+07	1.09E+09
Acc_amp	0.729	0.192	0.042	0.016	0.0033	6.60E-05	1.19E-05	2.52E-06
Acc_inf(E-04)	8.78	6.54	6.02	5.78	5.7	5.54	5.46	5.4

Table 6.8: Inaccurate [4:2] compressor multiplier(1)

Design Name	Data for Inaccurate [4:2] compressor based multiplier (Design 1)						
Parameters	2	4	6	8	10	12	14
ER	0.58	0.893	0.96	0.995	0.999	1	1
MED	2.31	59.55	971.93	1.62E+04	2.59E+05	4.35E+06	6.78E+07
Acc_amp	0.412	0.117	0.044	0.0065	0.0014	1.16E-04	2.72E-05
Acc_inf(E-04)	7.7	6.59	6.08	5.1	5.56	5.43	5.36

Table 6.9: Inaccurate [4:2] compressor multiplier(2)

Design Name	Data for Inaccurate [4:2] compressor based multiplier (Design 2)						
Parameters	2	4	6	8	10	12	14
Delay(ns)	10.0588	20.1175	30.1763	40.235	50.2938	60.3525	70.4113
Power(μW)	4.5469	9.0938	13.6407	18.1875	22.7343	27.2813	31.8281
Area (μm^2)	30.9978	61.9955	92.99318	123.9909	154.9886	185.9863	216.984
Area (gates)	24	48	72	96	120	144	168
Area (transistors)	161	322	483	644	805	966	1127
ER	0.56	0.875	0.972	0.993	0.999	1	1
MED	2.26	52.1	967.3	1.57E+04	2.59E+05	4.27E+06	6.37E+07
Acc_amp	0.44	0.13	0.0319	0.0085	1.40E-03	1.26E-04	2.88E-05
Acc_inf(E-04)	7.81	6.71	6.04	5.72	5.58	5.38	5.43

Table 6.10: Inaccurate [4:2] compressor multiplier(3)

Design Name	Data for Inaccurate [4 : 2] compressor based multiplier (Design 3)						
Parameters	2	4	6	8	10	12	14
ER	0.55	0.881	0.977	0.991	1	1	1
MED	2.2	58.42	970.36	1.63E+04	2.56E+05	4.23E+06	6.89E+07
Acc_amp	0.45	0.125	0.02741	0.0104	4.13E-04	1.37E-04	2.84E-05
Acc_inf(E-04)	7.8	6.55	6.01	5.72	5.55	5.48	5.43

for this multiplier only from 10-bits onward and for Simple Mitchell based Logarithmic multiplier after 12-bits. This phenomenon occurs due to large amount of error present for the lower bits. For the approx. compressor based multiplier designs, all the designs show almost the same results. The results follow the order of Design 3, 2 & 1.

(ii) **Acc_inf:** The values of Acc_inf lie in $\times 10^{-4}$ range. All the designs follow the same trend

and even almost the same values for various bit widths. The maximum value is 9.59×10^{-4} for 2×2 Under Designed Multiplier for its 2-bit multiplier, while the minimum value is 5.28×10^{-4} for Datapath complexity reduction approximate multiplier, for its 16-bit multiplier. For the approx. compressor based multiplier designs, all the designs show almost the same results. The results follow the order of Design 3, Design 2 and Design 1.

(iii) **ER:** The value of ER decreases as the bit width increases and the value saturates to 1 at higher bit widths. The least values of ERs are 0.054 for 2×2 Under Designed Multiplier and 0.262 for ACMA Multiplier for their respective 2-bit multipliers. For the approx. compressor based multiplier designs, all the designs show almost the same results. Although, Design 3 shows better results than Design 2 and Design 1, its ER reaches 1 for a 10×10 multiplier and the ERs of other two reach 1 only at 11×11 multiplier.

(iv) **MED:** For all the designs, the values of MED lie in a very large interval. It is seen that for higher bits (> 10), for each increment of 1 bit, the value of MED increases by 10 times. Two unusual observations are observed are:

- For error-tolerant multiplier, at a 6×6 bit multiplier, MED reaches to a very low value of 253.68 compared to other designs.
- In broken array multiplier, also for 6×6 bit multiplier, MED is rather larger value of 1020, compared to other designs. For the approx. compressor based multiplier designs, all the designs show almost the same results. Design 3 shows the best result, followed by Design 2 and 1.

6.3.2 Circuit Characteristics:

Among the five categories presented, circuit characteristics of five approximate multipliers from four categories have been discussed. It is to be notified that the circuit characteristics are evaluated by using the TANNER EDA, software.

(i) **Delay:** The delays of the designs are of the order of (ns). Considering a 15-bit multiplier, largest delay is observed in Broken Array Multiplier of $75.5906ns$, while the smallest delay is observed in Design 2 of approximate compressor based multipliers, of $75.446ns$. Since, Datapath complexity reduction approximate multiplier is designed from 2×2 Under Designed Multiplier, both have the same delay, of $80.476ns$ for a 16-bit multiplier.

(ii) **Power:** The power consumptions are in the order of (μW). Considering a 16-bit multiplier, largest power consumption is observed in Broken Array Multiplier of $50.0936\mu W$, while the smallest power consumption is observed in Datapath complexity reduction approximate multiplier of $20.911\mu W$. Power consumed in Design 2 of approximate compressor based multipliers is also very high of $34.1016\mu W$.

(iii) **Area :** Area is measured according to three factors, area covered in (μm^2), number of gates and the number of transistors. For a 16-bit multiplier, the largest number of gates and transistors are used in Broken Array Multiplier, 252 and 1761 respectively. Similarly, for a 16-bit multiplier, the smallest number of gates and transistors are used in Datapath complexity reduction approximate multiplier, 99 and 658 respectively. In terms of area covered, Datapath complexity reduction approximate multiplier covers the least area of $123.2874\mu m^2$ for a 16-bit multiplier, followed by 2×2 Under Designed Multiplier with an area of $125.222\mu m^2$, while the largest area covered by Broken Array Multiplier, with an area of $348.054\mu m^2$ for a 16-bit multiplier.

6.4 Summary

In this chapter, approximate multipliers are reviewed; their error and circuit characteristics are evaluated. Based on accuracy, the category of approx. error tolerant multipliers provide the best results; the two best designs are Error tolerant Multiplier and 2×2 Under Designed Multiplier. For the lower bit multipliers, the design provided by 2×2 Under Designed Multiplier is very effective. But for higher bit multiplier structure, Error tolerant Multiplier provides a better performance. Based on circuit characteristics, the smallest delay is observed in the Design 2 of approximate compressor based multipliers, but its power consumption and area covered is pretty high. The least power consumed and area covered design is in Datapath complexity reduction approximate multiplier. Its delay is also not that large, so it is the best design based on circuit characteristics.

7

Conclusion and Future Work

Contents

In this chapter, we summarize the work proposed in the thesis, highlight the contributions, and suggest directions for the possible future work.

7.1 Summary of Contributions

In this thesis, we have proposed various cryptographic primitive algorithms for applications like IoT, cloud computing, wireless communication, secure multi-party computation (SMPC) and blockchain mining. In particular, we have proposed and implemented the following algorithms.

(i) **ECCDH**. It is a cryptographic algorithm used for the blockchain technology.

(ii) **Modified SNOW 2.0**. A modified version of the SNOW 2.0 stream cipher architecture.

(iii) **Modified SNOW 3G**. It is a modified version of the SNOW 3G stream cipher architecture.

(iv) **Nonlinear secret sharing scheme**. A secure and resource-efficient secret sharing scheme.

(v) **Blockchain mining architecture**. An energy-efficient blockchain mining architecture.

In the second chapter, we designed and implemented various cryptographic algorithms on the Xilinx FPGA ZedBoard to find suitable algorithms for IoT security applications. It was found that the elliptical curve cryptography using the Diffie-Hellman algorithm (ECCDH) used less power compared to AES and ECC, making it fast and resource-efficient algorithm suitable for the IoT applications.

In the third chapter, we studied the stream cipher architectures, discussed their drawbacks and proposed modified resource-efficient architecture for SNOW 2.0 and SNOW 3G stream cipher architectures. We implemented the substitution box of the designs by using boolean functions instead of LUTs, reducing the implementation area of the whole architecture and the power consumption.

In the fourth chapter, we presented the secret sharing schemes for the secure multi-party computation. The linear secret sharing scheme was implemented on FPGA and compared with the existing architecture. Please note that the nonlinear secret sharing scheme was realized on FPGA for the first time, exhibiting comparable results in terms of power and delay performances.

In the fifth chapter, the various blocks of blockchain mining algorithms were developed, and all the algorithms used in a blockchain were implemented as an ASIC, which reduced energy consumption, cost and computational resources.

Finally in the sixth chapter, various approximate multipliers are implemented and compared to find multipliers for use in specific applications, such as cryptography to optimize area, power and delay. Approximate computing reduces the resources utilized and power consumed for any architecture provided it is error resilient.

7.2 Directions for future work

In this thesis, several cryptographic algorithms and their low-power resource-efficient VLSI architectures are proposed, which are suitable for implementation in the hardware devices. The proposed lightweight cryptographic algorithms, like the Elliptical Curve Cryptography encryption scheme and Deffie-Hellmann key exchange methods, are suitable for application in IoT networks. Similarly, the proposed modified SNOW stream cipher schemes are used to secure wireless, IoT or Bluetooth devices where maximizing the trade-off between security and performance on the hardware platforms is essential. Also, a scalable and hardware-efficient implementation of Shamir's linear secret sharing and Renvall-Ding's nonlinear secret sharing schemes on FPGA were implemented. This was achieved by optimizing the basic design blocks and appropriate parallelization of the algorithm to reduce the design's computational time, resource utilization, and power consumption to make them suitable for applications like secure cloud backups and SMPC. Please note that this is the first time the nonlinear scheme has been implemented on an FPGA. Further, we have implemented Blockscale ASIC, a SHA-256 (Secure Hash Algorithm-256) hardware accelerator for blockchain proof-of-work consensus applications, optimized for the energy-efficient hashing. The accelerator enables customized cryptocurrency mining, lowering the total cost of the ownership by allowing system designs to be tailored to end-user requirements.

Approximate computing can be used to design cryptographic primitives like block ciphers, stream ciphers, and digital signature implementations. Similarly, secret sharing schemes can be utilized in blockchain protocols for secure multi-party computation and key management. They can help distribute key management responsibility among multiple participants, enhancing security and reducing the risk of single points of failure. Additionally, secret sharing schemes can be used to distribute private keys among participants to enable secure and decentralized control over blockchain transactions. The use of secret sharing schemes in blockchain protocols aligns with

the goals of maintaining security, decentralization, and trust in blockchain networks. Overall secret sharing schemes, Snow Stream Cipher, and blockchain mining all contribute to the broader field of cryptography and play important roles in ensuring security, confidentiality, and trust in various contexts.

List of Publications

Journal Publications

1. **S. S. P. Goswami** and G. Trivedi, "FPGA Implementation of Modified SNOW 3G Stream Ciphers Using Fast and Resource Efficient Substitution Box," in IEEE Embedded Systems Letters, doi: 10.1109/LES.2023.3298743.

2. **S. S. P. Goswami** and G. Trivedi, "Implementation of Modified SNOW 2.0 Stream Ciphers using Fast and Resource Efficient Substitution Box," 2023, TCASII (Under review)

3. **S. S. P. Goswami** and G. Trivedi, "FPGA Implementation of Low Power Resource Efficient Secret Sharing Schemes for Cloud Computing Applications," 2023 (Under review)

4. **S. S. P. Goswami**, S. Garg, K. Walia and G. Trivedi, "ASIC Implementation of Secure Hash Algorithm hardware accelerator for Blockchain Mining applications," 2023 (Under Review)

Conference Publications

1. **S. S. P. Goswami**, B. Paul, S. Dutt and G. Trivedi, "Comparative Review of Approximate Multipliers," 2020 30th International Conference Radioelektronika (RADIOELEKTRONIKA), Bratislava, Slovakia, 2020.

2. **S. S. P. Goswami** and G. Trivedi, "Comparison of Hardware Implementations of Cryptographic Algorithms for IoT Applications," 2023 33rd International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 2023.

3. B. Paul, A. Khobragade, S. Javvaji Sai, **S. S. P. Goswami**, S. Dutt and G. Trivedi, "Design and Implementation of Low-Power High-throughput PRNGs for Security Applications," 2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID), Delhi, India, 2019, pp. 535-536, doi: 10.1109/VL-

SID.2019.00123.

4. S. Garg, K. Walia, **S. S. P. Goswami** and G. Trivedi, "Design of Blockchain Mining Algorithms for ASIC Implementation," 2023 3rd INTERNATIONAL CONFERENCE ON EMERGING TRENDS AND TECHNOLOGIES ON INTELLIGENT SYSTEMS (ETTIS 2023).

Bibliography

[1] E. K. Subramanian and L. Tamilselvan, "Elliptic curve diffie–hellman cryptosystem in big data cloud security," *Cluster Computing*, vol. 23, pp. 1–11, 12 2020.

[2] P. Ekdahl and T. Johansson, "A new version of the stream cipher snow," in *Selected Areas in Cryptography*, K. Nyberg and H. Heys, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 47–61.

[3] T. A. Pham, M. S. Hasan, and H. Yu, "Area and power optimisation for aes encryption module implementation on fpga," *18th International Conference on Automation and Computing (ICAC)*, pp. 1–6, 2012.

[4] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "Aes implementation on a grain of sand," *Information Security, IEE Proceedings*, vol. 152, pp. 13– 20, 11 2005.

[5] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, "Design and implementation of low-area and low-power aes encryption hardware core," *9th EUROMICRO Conference on Digital System Design (DSD'06)*, pp. 577–583, 2006.

[6] A. Ricci, M. Grisanti, I. De Munari, and P. Ciampolini, "Design of a 2 µw rfid baseband processor featuring an aes cryptography primitive," 10 2008, pp. 376 – 379.

[7] K. Rahimunnisa, P. Karthigaikumar, S. Rasheed, J. Jayakumar, and S. SureshKumar, "Fpga implementation of aes algorithm for high throughput using folded parallel architecture," *Security and Communication Networks*, vol. 7, no. 11, pp. 2225–2236.

[8] S. S. S. Priya, P. Karthigaikumar, and N. R. Teja, "Fpga implementation of aes algorithm for high speed applications," pp. 115–125, Jul 2022.

[9] J. Van Dyken and J. G. Delgado-Frias, "Fpga schemes for minimizing the power-throughput trade-off in executing the advanced encryption standard algorithm," vol. 56, no. 2–3, 2010.

[10] T. M. Kumar, K. S. Reddy, S. Rinaldi, B. D. Parameshachari, and K. Arunachalam, "A low area high speed fpga implementation of aes architecture for cryptography application," *Electronics*, vol. 10, no. 16, 2021.

[11] R. Farashahi, B. Rashidi, and S. Sayedi, "Fpga based fast and high-throughput 2-slow retiming 128-bit aes encryption algorithm," *Microelectronics Journal*, vol. 45, p. 1014–1025, 08 2014.

[12] R. Naseer and J. Draper, "Dec ecc design to improve memory reliability in sub-100nm technologies," in *2008 15th IEEE International Conference on Electronics, Circuits and Systems*, 2008, pp. 586–589.

[13] S. R. Chatterjee, S. Majumder, B. Pramanik, and M. Chakraborty, "Fpga implementation of pipelined blowfish algorithm," in *2014 Fifth International Symposium on Electronic System Design*, 2014, pp. 208–209.

[14] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 198–205, 2008.

[15] M. S. Hossain, E. Saeedi, and Y. Kong, "High-speed, area-efficient, fpga-based elliptic curve cryptographic processor over nist binary fields," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*, 2015, pp. 175–181.

[16] G. D. Sutter, J.-P. Deschamps, and J. L. Imana, "Efficient elliptic curve point multiplication using digit-serial binary field operations," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 1, pp. 217–225, 2013.

[17] V. Kamalakannan and V. Kamalakannan, "Fpga implementation of elliptic curve cryptoprocessor for perceptual layer of the internet of things," *ICST Transactions on Security and Safety*, vol. 5, p. 155739, 10 2018.

[18] M. Morales-Sandoval, L. A. R. Flores, R. Cumplido, J. J. Garcia-Hernandez, C. Feregrino, and I. Algredo, "A compact fpga-based accelerator for curve-based cryptography in wireless sensor networks," *Journal of Sensors*, vol. 2021, Jan 2021.

[19] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical modeling of elliptic curve scalar multiplier on lut-based fpgas for area and speed," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 901–909, 2013.

[20] H. Marzouqi, M. Al-Qutayri, and K. Salah, "Review of elliptic curve cryptography processor designs," *Microprocessors and Microsystems*, vol. 39, no. 2, pp. 97–112, 2015.

[21] E. Venugopal and T. Hailu, "Fpga based architecture of elliptic curve scalar multiplication for iot," in *2018 Conference on Emerging Devices and Smart Systems (ICEDSS)*, 2018, pp. 178–182.

[22] R. Gurunath, M. Agarwal, A. Nandi, and D. Samanta, "An overview: Security issue in iot network," in *2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2018, pp. 104–107.

[23] P. Kitsos, "Hardware implementations for the iso/iec 18033-4:2005 standard for stream ciphers," *Signal Processing*, 01 2006.

[24] C. Boura and A. Canteaut, "On the influence of the algebraic degree of f-1 on the algebraic degree of g circle f," *IACR Cryptology ePrint Archive*, vol. 2011, p. 503, 01 2011.

[25] M. D. Galanis, G. Kostopoulos, O. G. Koufopavlou, and C. E. Goutis, "Comparison of the performance of stream ciphers for wireless communications," 2004.

[26] P. Léglise, F.-X. Standaert, G. Rouvroy, and J.-J. Quisquater, "Efficient implementation of recent stream ciphers on reconfigurable hardware devices," 01 2005.

[27] H. Sekine, T. Nosaka, Y. Hatano, M. Takeda, and T. Kaneko, "A strength evaluation of a pseudo-random number generator mugi against linear cryptanalysis," *IEICE Transactions*, vol. 88-A, pp. 16–24, 01 2005.

[28] S. Ashaq, M. Nazish, M. Ali, I. Sultan, and M. Tariq Banday, "Fpga implementation of present block cypher with optimised substitution box," in *Smart Technologies, Communication and Robotics (STCR)*, 2022, pp. 1–6.

[29] P. Ekdahl and T. Johansson, "Snow a new stream cipher," 2000, first open Nessie Workshop ; Conference date: 02-01-0001.

[30] J. Massey, "Shift-register synthesis and bch decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969.

[31] 3GPP, "Specification of the 3gpp confidentiality and integrity algorithms uea2 & uia2," 2006.

[32] P. Ekdahl, T. Johansson, A. Maximov, and J. Yang, "A new snow stream cipher called snow-v," *Cryptology ePrint Archive*, 2018.

[33] P. Hawkes and G. Rose, "Guess-and-determine attacks on snow," vol. 2595, 08 2002, pp. 37–46.

[34] Y. Zhu, H. Zhang, and Y. Bao, "Study of the aes realization method on the reconfigurable hardware," *2013 International Conference on Computer Sciences and Applications*, pp. 72–76, 2013.

[35] T. Good and M. Benaissa, "692 – nw advanced encryption standard (aes) on a 0.13 – μm cmos," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 1753–1757, 2010.

[36] T. Johansson *et al.*, "An overview of the snow 3g cipher, including its design and security analysis," *Proceedings of the 8th international conference on Security for information technology and communications*, pp. 93–108, 2011.

[37] G. Leurent *et al.*, "Linearity of snow 3g and snow 3g+," in *International Workshop on Fast Software Encryption*. Springer, 2012, pp. 47–62.

[38] V. Conti *et al.*, "Efficient fpga implementation of the snow 3g cipher for 4g/lte systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 2, pp. 310–319, 2014.

[39] J. Xu *et al.*, "Analysis and improvement of snow 3g cipher in 4g/lte networks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1604–1616, 2015.

[40] L. Ding *et al.*, "On the security of snow 3g in 5g nr," *IEEE Communications Magazine*, vol. 57, no. 2, pp. 82–87, 2019.

[41] M. Madani, I. Benkhaddra, C. Tanougast, S. Chitroub, and L. Sieler, "Digital implementation of an improved lte stream cipher snow-3g based on hyperchaotic prng," *Security and Communication Networks*, vol. 2017, p. 5746976, Nov 2017. [Online]. Available: https://doi.org/10.1155/2017/5746976

[42] C. Shen, X. Yan, Q. Liu, J. Kang, Y. Liu, X. Zheng, X. Zhang, and Y. Liu, "An fpga design and implementation of 4g network security algorithm based on snow3g," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2019, pp. 387–392.

[43] H. T. Ltd, *SNOW3G Stream Cipher Core for Altera FPGA[EB/OL]*, 2009. [Online]. Available: http://www.heliontech.com/downloads/3gpp_snow3g_altera.p

[44] H. Liang and L. Li, "Efficient implementation of snow 3g on gpu," in *2012 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2012, pp. 794–798.

[45] D. Coppersmith, S. Halevi, and C. Jutla, "Cryptanalysis of stream ciphers with linear masking." *IACR Cryptology ePrint Archive*, vol. 2002, p. 20, 01 2002.

[46] K. Alexander, R. Karri, I. Minkin, K. Wu, P. Mishra, and X. Li, "Towards 10-100 gbps cryptographic architectures," in *Proc. of CATT/WICAT Annual Research Review*, 2003. [Online]. Available: http://wicat.poly.edu/techreport/tr/02-005.pdf

[47] A. Tsohou, S. Kokolakis, C. Lambrinoudakis, and S. Gritzalis, "Information systems security management: A review and a classification of the iso standards," A. B. Sideridis and C. Z. Patrikakis, Eds.

[48] A. Shamir, " How to share a secret örper. (German) [On the electrodynamics of moving bodies]," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[49] M. Tompa and H. Woll, "How to share a secret with cheaters," in *Proceedings on Advances in Cryptology—CRYPTO '86*. Springer-Verlag, 1987, p. 261–265.

[50] J. Stangl, T. Lorünser, and S. M. P D, "A fast and resource efficient fpga implementation of secret sharing for storage applications," 03 2018, pp. 654–659.

[51] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology — CRYPTO' 93*, D. R. Stinson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 136–146.

[52] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, p. 335–348, 1989.

[53] A. Abdallah and M. Salleh, "Secret sharing scheme security and performance analysis," in *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, 2015, pp. 173–180.

[54] Y. Wang, Z. Yuan, Z. Li, and L. Renfa, "Secret sharing based countermeasure for aes s-box," 12 2011.

[55] T. Bhattacharjee, R. K. Rout, and S. P. Maity, "Affine boolean classification in secret image sharing for progressive quality access control," *J. Inf. Secur. Appl.*, vol. 33, no. C, p. 16–29, apr 2017.

[56] R. Hernandez-Becerril, A. Bucio Ramirez, M. Nakano-Miyatake, H. Perez-Meana, and M. Ramirez-Tachiquin, "A gpu implementation of secret sharing scheme based on cellular automata," *The Journal of Supercomputing*, vol. 72, 04 2016.

[57] C. Tartary and H. Wang, "Dynamic threshold and cheater resistance for shamir secret sharing scheme," in *Information Security and Cryptology*, H. Lipmaa, M. Yung, and D. Lin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 103–117.

[58] A. Renvall and C. Ding, "A nonlinear secret sharing scheme," in *Information Security and Privacy*, J. Pieprzyk and J. Seberry, Eds. Springer Berlin Heidelberg, 1996, pp. 56–66.

[59] M. E. Kaihara, "Studies on modular arithmetic hardware algorithms for public-key cryptography," 2006.

[60] J. Butler and T. Sasao, "Fast hardware computation of x mod z," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 294–297.

[61] Z. Tan, W. Guo, G. Gong, and H. Lu, "A new pseudo-random number generator based on the leap-ahead lfsr architecture," 11 2018, pp. 57–58.

[62] K. Wold and C. H. Tan, "Analysis and enhancement of random number generator in fpga based on oscillator rings," in *2008 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 2008, pp. 385–390.

[63] H. L. Pham, T. H. Tran, V. T. Duong Le, and Y. Nakashima, "A high-efficiency fpga-based multimode sha2 accelerator," *IEEE Access*, vol. 10, pp. 11 830–11 845, 2022.

[64] Y. Zhang, Z. He, M. Wan, M. Zhan, M. Zhang, K. Peng, M. Song, and H. Gu, "A new message expansion structure for full pipeline sha-2," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 4, pp. 1553–1566, April 2021.

[65] R. Martino and A. Cilardo, "A flexible framework for exploring, evaluating, and comparing sha-2 designs," *IEEE Access*, vol. 7, pp. 72 443–72 456, 2019.

[66] Intel, "Custom ASIC Product Brief," https://www.intel.in/content/www/in/en/products/docs/blockchain/asic-product-brief.html, 2023.

[67] B. M. Krishna, D. S. Gopinath, M. Kiran, and S. Javid, "Reconfigurable asymmetric lightweight cryptosystem," *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 5, pp. 1678–1684, May 2020.

[68] L. Rodriguez-Flores, M. Morales-Sandoval, R. Cumplido, C. Feregrino-Uribe, and I. Algredo-Badillo, "A compact fpga-based microcoded coprocessor for exponentiation in asymmetric encryption," in *2017 IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS)*, 2017, pp. 1–4.

[69] L. Xu, L. Chen, Z. Gao, H. Kim, T. Suh, and W. Shi, "Fpga based blockchain system for industrial iot," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 876–883.

[70] D. N. Kumar and R. Bhakthavatchalu, "Parameterizable fpga implementation of sha-256 using blockchain concept," in *2019 International Conference on Communication and Signal Processing (ICCSP)*, 2019, pp. 370–374.

[71] S. Ghimire and H. Selvaraj, "A survey on bitcoin cryptocurrency and its mining," 2018.

[72] Ethicomp, "Proceedings of Ethicomp, 20th International Conference on the Ethical and Social issues in Information and Communication Technologies," 2022, pp. 578–582.

[73] G. Cui, K. Shi, Y. Qin, L. Liu, B. Qi, and B. Li, "Application of blockchain in multi-level demand response reliable mechanism," in *2017 3rd International Conference on Information Management (ICIM)*, 2017, pp. 337–341.

[74] D. Hwang, J. Choi, and K.-H. Kim, "Dynamic access control scheme for iot devices using blockchain," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 713–715.

[75] Y. Sakakibara, K. Nakamura, and H. Matsutani, "An fpga nic based hardware caching for blockchain," in *2017*, pp. 1–6.

[76] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, 2003, pp. 295–306.

[77] I. Gemeliarana and R. Sari, "Evaluation of proof of work (pow) blockchains security network on selfish mining," 2019.

[78] H. Vranken, "Sustainability of bitcoin and blockchains," *Current Opinion in Environmental Sustainability*, vol. 28, pp. 1–9, 2017.

[79] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on computers*, vol. 62, no. 9, pp. 1760–1771, 2012.

[80] C.-H. Lin and C. Lin, "High accuracy approximate multiplier with error correction," in *2013 IEEE 31st international conference on computer design (ICCD)*. IEEE, 2013, pp. 33–38.

[81] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*. IEEE, 2011, pp. 346–351.

[82] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, 2014.

[83] M. Vasudevan and C. Chakrabarti, "Image processing using approximate datapath units," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2014, pp. 1544–1547.

[84] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *2010 IEEE international conference of electron devices and solid-state circuits (EDSSC)*. IEEE, 2010, pp. 1–4.

[85] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[86] K. Bhardwaj and P. S. Mane, "Acma: Accuracy-configurable multiplier architecture for error-resilient system-on-chip," in *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2013, pp. 1–6.

[87] H.-J. Ko and S.-F. Hsiao, "Design and application of faithfully rounded and truncated multipliers with combined deletion, reduction, truncation, and rounding," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 5, pp. 304–308, 2011.

[88] J. M. Jou and S. R. Kuang, "Design of low-error fixed-width multiplier for dsp applications," *Electronics Letters*, vol. 33, no. 19, pp. 1597–1598, 1997.

[89] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, 2009.

[90] D. Harris and N. Weste, "Cmos vlsi design," *ed: Pearson Education, Inc*, 2010.