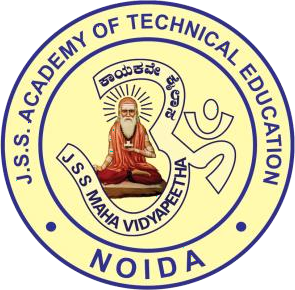
**RSA IMPLEMENTATION USING FPGA**

BY

## NITIN SINGH (2000910310106)

## 2.NISHA CHAUHAN(2000910310105)

**3. RIYA SHARMA (2000910310129)**



## UndertheGuidanceof

MRS.RAJESHWARI BHAT

DEPARTMENTOFELECTRONICSANDCOMMUNICATION ENGINEERING

**JSSACADEMYOFTECHNICALEDUCATIONC-20/1 SECTOR-62,NOIDA**

**March,2023-24**

**Project Report  
 On**

**RSA IMPLEMENTATION USING FPGA**

BY

## 1. NITIN SINGH (2000910310104)

## 2. NISHA CHAUHAN(2000910310105)

**3. RIYA SHARMA (2000910310129)**

## UndertheGuidanceof

MRS. RAJESHWARI BHAT



SubmittedtotheDepartmentofElectronics&CommunicationEngineering inpartial fulfillment oftherequirements

ForthedegreeofBachelorofTechnologyIn

Electronics &Communication Engineering

**JSSAcademyofTechnicalEducation, Noida**

**Dr.A.P.JAbdulKalamTechnicalUniversity,Lucknow**

**JUNE,2024**

# DECLARATION

Weherebydeclarethatthissubmissionisourownworkandthat,tothebest ofourknowledgeandbelief, itcontainsnomaterialpreviouslypublishedorwrittenbyany otherpersonnormaterialwhichto a substantial extent has been accepted for the award of any other degree or diploma of theuniversity or other institute of higher, except where due acknowledgmenthas been made in thetext.

1. NITIN SINGH

Roll No. 2000910310106

1. NISHA CHAUHAN

Roll No. 2000910310105

1. RIYA SHARMA

Roll No. 2000910310129

**PLAGIARISM REPORT**

# 

# CERTIFICATE

This is to certify that Project Report entitled “RSA Implementation Using FPGA”which

issubmittedbyNitin Singh, Nisha Chauhan and Riya Sharma for partial fulfillment of the requirement for the award of B.Techdegree in Electronics and Communication Engineering of Dr. A.P.J. Abdul Kalam TechnicalUniversity,Lucknowisarecord of thecandidate own workcarriedout by him underoursupervision.Thematterembodiedinthisthesisisoriginalandhasnotbeensubmittedfortheawardofanyotherdegree.

### RajeshwariBhat

Assistant Professor

# ACKNOWLEDGEMENT

ItgivesusagreatsenseofpleasuretopresentthereportoftheB.TechProjectundertakenduring B. Tech Final Year. We owe special debt of gratitude to Mrs. RajeshwariBhatAssistant Professor, Department of Electronics and Communication Engineering, J.S.S. Academyof Technical Education, Noida for her constant support and guidance throughout the course of ourwork. Her sincerity, thoroughness and perseverance have been a constant source of inspiration forus.Itisonly her cognizanteffortsthatourendeavorshave seenlightofthe day.

We alsotake the opportunity toacknowledge the contribution of Dr. Arun Kumar G. Head,DepartmentofElectronicsandCommunicationEngineering,J.S.S.AcademyofTechnicalEducation,Noidaforhisfullsupportandassistanceduringthe developmentoftheproject.

Signature:

Name: Nitin Singh

Roll No: 2000910310106

Date:

Signature:

Name: Nisha Chauhan

Roll No: 2000910310105

Date:

Name: Riya Sharma

Signature:

Roll No: 2000910310129

Date:

**ABSTRACT**

This paper presents a scheme for implementation of RSA encryption algorithm on FPGA. A 64 bit cipher text is accepted and using 128 bit public key RSA encryption technique, a 64 bit encrypted message is generated. Each block is coded using Verilog and the code is synthesized and simulated using Xilinx ISE Design Suite 14.7.Unlike previous approaches, we have systematically provided timing, area and power measures for Spartan 3 and Virtex 6 FPGA using Pre and Post synthesis simulations. The design is optimized for either speed or power and a tradeoff is presented between speed, power and space. If the design is optimized for power then fewer resources are consumed but the maximum usable frequency is also reduced. Spartan 3 FPGAs are best suited for low power designs. As a major practical result we show that it is possible to implement RSA algorithm at secure bit lengths on a single commercially available FPGA

# TABLE OF CONTENTSPage

[DECLARATION ii](#_TOC_250016)

[CERTIFICATE iii](#_TOC_250015)

ACKNOWLEDGEMENTS iv

[ABSTRACT vi](#_TOC_250014)

[LISTOFFIGURES vii](#_TOC_250013)

LISTOFTABLES viii

LISTOFSYMBOLS ix

[CHAPTER 1 LITERATURESURVEY 1](#_TOC_250012)

[1.1 Thesis Outline. 5](#_TOC_250011)

[CHAPTER2INTRODUCTION 6](#_TOC_250010)

* 1. [Introduction 6](#_TOC_250009)

[1.2 ProblemStatement 8](#_TOC_250008)

[1.3AimandObjective 8](#_TOC_250007)

1.4 Modules used ……………………………………………………..

[CHAPTER3 KEY GENERATION FUNCTION 9](#_TOC_250006)

* 1. [Introduction 9](#_TOC_250005)
  2. Flow chat 12
  3. Types of keys 12
  4. [Steps of Key generation 13](#_TOC_250004)
  5. Implementation Using Fpga…………………………………….15
  6. [limitation 14](#_TOC_250003)
  7. Application Conclusion……………………………

CHAPTER4 Encryption 17

* 1. [Introduction 17](#_TOC_250002)
  2. Key used 19
  3. [Steps to encrypt 20](#_TOC_250001)
  4. Conclusion …………………………………………………………...

CHAPTER 5 Decryption 17

* 1. [Introduction 17](#_TOC_250002)
  2. Key used 19

5.3 [Steps to decrypt 20](#_TOC_250001)

5.4 Conclusion ……………………………………………………...

CHAPTER 6 Code , output and Test cases 17

* 1. [Code 17](#_TOC_250002)

6.2 Output 19

6.3 [Test case 1 20](#_TOC_250001)

6.4 Test case 2 19

6.5 [Test case 3 20](#_TOC_250001)

6.6 RESULT and conclusion …………………………………….

CHAPTER 7 Conclusion and future work 17

[APPENDIX 17](#_TOC_250002)

REFERENCES 19

# LISTOFFIGURES

Page

# CHAPTER 1

# 1.1 LITERATURESURVEY

FPGA-based implementations offer advantages such as flexibility, parallelism, and reconfigurability, making them suitable for cryptographic applications like RSA. Below are summaries of key papers and studies in this field:

**"High-Performance RSA Implementation on FPGA" by T. K. Melodia et al. (2016):**

This paper presents a high-performance FPGA implementation of the RSA algorithm. The authors focus on optimizing the performance by exploiting parallelism and resource utilization efficiently.

They discuss various optimization techniques such as pipelining, parallel modular exponentiation, and efficient memory management to achieve high throughput and low latency.

The implementation targets resource-constrained environments and demonstrates competitive performance compared to software-based implementations.

**"Efficient FPGA Implementation of RSA Algorithm Using System Generator for DSP" by M. Mirza-Aghatabar et al. (2017):**

This study explores the implementation of the RSA algorithm on FPGA using Xilinx System Generator for DSP. System Generator provides a high-level design flow, enabling rapid prototyping and optimization of signal processing algorithms.

The authors propose efficient architectures for modular exponentiation and modular multiplication, key operations in the RSA algorithm, and implement them using System Generator.

They evaluate the performance in terms of throughput, resource utilization, and power consumption, demonstrating the effectiveness of their approach.

**"Design and Implementation of High-Performance RSA Cryptographic Processor on FPGA" by A. M. Bayoumi et al. (2018):**

This paper presents a detailed design and implementation of an RSA cryptographic processor on FPGA. The authors focus on optimizing critical arithmetic operations such as modular exponentiation and modular multiplication.

They propose novel algorithms and architectures to accelerate these operations, utilizing parallelism and efficient resource utilization.

The implementation is evaluated for performance, throughput, and security considerations, demonstrating its suitability for real-world cryptographic applications.

**"FPGA Implementation of RSA Cryptography Processor for Secure Communication" by H. Kim et al. (2019):**

This study presents an FPGA-based implementation of an RSA cryptography processor targeting secure communication systems.

The authors propose a scalable architecture for modular exponentiation, accommodating various key sizes and achieving high throughput.

They also discuss security considerations such as side-channel attacks and propose countermeasures to enhance the resilience of the implementation against such threats.

**"Efficient Implementation of RSA Algorithm Using Hardware/Software Co-Design" by S. Bhartiya et al. (2020):**

This paper explores a hardware/software co-design approach for implementing the RSA algorithm on FPGA.

The authors partition the algorithm into hardware and software components, leveraging the strengths of both domains.

They optimize the hardware-accelerated modules for performance and resource utilization while offloading certain computations to software running on an embedded processor within the FPGA.

The implementation demonstrates improved efficiency and flexibility compared to purely hardware-based or software-based approaches.

These studies collectively highlight the ongoing efforts in optimizing and implementing the RSA algorithm on FPGA, addressing various challenges such as performance, resource utilization, security, and scalability. Future research in this area may focus on exploring advanced optimization techniques, incorporating hardware security features, and adapting to emerging FPGA architectures and technologies.

**“Cetin Kaya Koc, High-speed RSA algorithm, RSA laboratories, version 2, 1994”**

These include the RSA algorithm, the Die-Hellman key exchange scheme, the ElGamal algorithm, and the recently proposed Digital Signature Standard (DSS) of the National Institute for Standards and Technology. The emphasis of the report is on the underlying mathematics, algorithms, and their running time analyses. This does not include any actual code; however, the algorithms selected are particularly suitable for microprocessor and signal processor implementations. the aim is that the report will close the gap between the mathematics of the modular exponentiation operation and its actual implementation on a general purpose processor.

**“Perovic N.S; Popovic-Bozovic: "FPGA implementation of RSA crypto algorithm using shift and carry algorithm ", IEEE “**

This paper presents a scheme for implementation of RSA encryption algorithm on FPGA. A 64 bit cipher text is accepted and using 128 bit public key RSA encryption technique, a 64 bit encrypted message is generated. Each block is coded using VHDL and the code is synthesized and simulated using Xilinx ISE Design Suite 14.7.Unlike previous approaches we have systematically provided timing, area and power measures for Spartan 3 and Virtex 6 FPGA using Pre and Post synthesis simulations. The design is optimized for either speed or power and a tradeoff is presented between speed, power and space. If the design is optimized for power then fewer resources are consumed but the maximum usable frequency is also reduced. Spartan 3 FPGAs are best suited for low power designs. As a major practical result we show that it is possible to implement RSA algorithm at secure bit lengths on a single commercially available FPGA.

## 

**“Sushanta Kumar Sahu, ManoranjanPradhan, “FPGA Implementation of RSA Encryption System”, International Journal of Computer Applications (0975 – 8887), Volume 19– No.9, April (2011)”**

This paper presents the architecture and modeling of RSA public key encryption/decryption systems. It supports multiple key sizes like 128 bits, 256 bits, 512 bits. Therefore it can easily be fit into the different systems requiring different levels of security. In this paper simple shift and add algorithm is used to implement the blocks. It makes the processing time faster and used comparatively smaller amount of space in the FPGA due to its reusability. Each block is coded with Very High Speed Integrated Circuit Hardware Description Language. The VHDL code is synthesized and simulated using Xilinx-ISE 10.1. It is verified that this architecture support multiple key of 128bits, 256bits, and 512 bits.

**“Lin X Sun L and Qu H, 2017 An efficient RSA-based certificate less public key encryption scheme.”**

In order to resolve the key escrow in identity-based scheme and the significant cost of using a PKI system in traditional public key scheme, the notion of certificate less public key cryptography (CL-PKC) was introduced. The first certificate less public key encryption scheme (CL-PKE) was proposed by Al-Riyami and Paterson, and then further schemes were developed. However, most of them are constructed from the bilinear pairing which is a time costing operation. In this paper, we construct an efficient CL-PKE scheme from RSA since RSA is the de facto Internet standard and is widely used in many applications. The security is based on Kilian–Petrank’s RSA assumption which is a variant of RSA.

**“Meng X and Zheng X, 2015 Cryptanalysis of RSA with a small parameter revisited Inf. Process. Lett. 115, 11 p. 858–862”**

Let 𝑁=𝑝𝑞 be an RSA modulus with balanced primes p and q. Suppose that the public exponent e and private exponent d satisfy ed−1=𝑘𝜙(𝑁). We revisit the birthday attack against short exponent RSA proposed by Meng and Zheng at ACISP 2012. We show that if 𝑒>𝑘(𝑝+𝑞), then N can be factored in both time and space complexity of 𝑂˜(𝑘). This improves the former result. We also give a detail explanation on how the baby-step giant-step method works.

**“Modular Arithmetic for RSA Cryptography. [Courtesy:http://gtk.hopto.org:8089/MODULARRSA.pdf]”**

The RSA system is widely employed and achieves good performance and high security. In this paper, we use Verilog to implement a 16-bit RSA block cipher system. The whole implementation includes three parts: key generation, encryption and decryption process. The key generation stage aims to generate a pair of public key and private key, and then the private key will be distributed to receiver according to certain key distribution schemes. Data security is achieved after the 64-bit input data are block encrypted by RSA public key. The cipher text can be decrypted at receiver side by RSA secret key. These are simulated in NC LAUNCH and hardware is synthesized using RTL Compiler of CADENCE. Netlist generated from RTL Compiler will be used to generate IC.

**“Symeon (Simos) Xenitellis, “A guide to PKIs and Open– source Implementations”, The Open–source PKI Book”**

This document tries to serve as a source of information on Public Key Infrastructures (PKIs) and focuses on both of the theoretic and practical description of PKIs. With relation to specific standards, the work of the PKIX Working Group1 is presented. There is an emphasis on these standards and there is an attempt to classify implementations according to the degree of compliance. This document starts with an introduction on public–key cryptography. Then, it describes several publicly available implementations providing a feature by feature comparison. There is a further discussion on security issues with accordance with PKIs. The implementations are presented as an educational instrument to test the protocols, to provide a source of feedback and to enable the individual to learn more about the wonderfull world of PKIs.

**“SCHNEIER, B., 1996. Applied Cryptography: Protocols, Algorithms, and Source Code in C, John Wiley & Sons”**

The literature of cryptography has a curious history. Secrecy, of course, has always played a central role, but until the First World War, important developments appeared in print in a more or less timely fashion and the field moved forward in much the same way as other specialized disciplines. As late as 1918, one of the most influential cryptanalytic papers of the twentieth century, William F. Friedman’s monograph The Index of Coincidence and Its Applications in Cryptography, appeared as a research report of the private Riverbank Laboratories [577]. And this, despite the fact that the work had been done as part of the war effort. In the same year Edward H. Hebern of Oakland, California filed the first patent for a rotor machine [710], the device destined to be a mainstay of military cryptography for nearly 50 years. After the First World War, however, things began to change. U.S. A

**“Stallings W.2003, Cryptography and Network Security: Principles and Practices.”**

**“Nedjah.N and Mourelle L.2002.Two Hardware Implementation for the Montgomery Modular Multiplication: Sequential versus Parallel. IEEE.”**

This paper describes a hardware architecture for modular multiplication operation which is efficient for bit-lengths suitable for both commonly used types of public key cryptography (PKC) i.e. ECC and RSA cryptosystems. The challenge of current PKC implementations is to deal with long numbers (160-2048 bits) in order to achieve system's efficiency, as well as security. RSA, still the most popular PKC, has at its root the modular exponentiation operation. Modular exponentiation consists of repeated modular multiplications, which is also the basic operation for ECC protocols. The solution proposed in this work uses a systolic array implementation and can be used for arbitrary precisions. We also present modular exponentiation based on Montgomery's Multiplication Method (MMM).

## 1.1THESISOUTLINE

* **Chapter 1** To make this read much easier, this thesis is organized in five chapters.Therestofthethesisisstructuredasfollows.offersabroadsummaryoftheliteraturereview.
* **Chapter2**presentsbasicintroductionofproject,aim&objective&problemstatementofproject. Manual fault detection is a time-consuming process. The problem might berectifiedwithinafewhours,oritmighteventakedays.So,inthischapterintroduceaboutthe project,technologyusedandalgorithm.
* **Chapter3**isrelatedtothedescriptionoftechnologyandalgorithmused.Itcontainsflowchartofproposedalgorithm.Allthe detailsaboutthe algorithm.
* **Chapter 4** is related to the study of result of algorithm used. How current will be at bus1whenfaultoccur.
* **Chapter5** presentsconclusionsandfuturework.

# CHAPTER 2

# INTRODUCTION

## INTRODUCTIONOFPROJECT

Introduction to FPGA-Based Implementation of the RSA Algorithm

In the realm of modern cryptography, the RSA algorithm stands as a cornerstone, offering robust security for digital communication and data protection. As the digital landscape continues to evolve, the demand for efficient and secure implementations of RSA has grown exponentially. Field-Programmable Gate Arrays (FPGAs) have emerged as a promising platform for implementing cryptographic algorithms like RSA due to their inherent flexibility, reconfigurability, and parallel processing capabilities.

This introduction aims to provide an overview of the significance, challenges, and advancements in implementing the RSA algorithm on FPGA platforms. It begins with a brief explanation of the RSA algorithm's importance in securing digital communication, followed by an exploration of FPGA technology and its suitability for cryptographic applications. Subsequently, it highlights the key challenges faced in implementing RSA on FPGAs and introduces recent advancements and research directions in this field.

The RSA algorithm is a widely used public-key encryption technique that has been employed to secure online transactions, communication networks, and digital signatures for over four decades. The algorithm's security relies heavily on the generation of secure keys, which are used for encryption and decryption. Key generation is a critical component of the RSA algorithm, as it directly affects the security of the data being transmitted.

In this project, we aim to implement the key generation function in Verilog for hardware FPGA implementation, demonstrating the feasibility of using FPGAs for RSA key generation. This approach offers several advantages, including high-speed and low-power consumption, making it suitable for a wide range of applications, including secure communication networks and digital signatures.

The RSA algorithm was first introduced in 1978 by Ron Rivest, Adi Shamir, and Leonard Adleman, and it has since become a widely accepted standard for secure data transmission. The algorithm's security is based on the difficulty of factoring large composite numbers, which are used to generate the public and private keys.

The key generation process involves selecting two large prime numbers, p and q, and computing the modulus, n, and the Euler's totient function, φ(n). The public exponent, e, is then chosen, and the private exponent, d, is computed using the extended Euclidean algorithm. Finally, the public and private keys are generated using the computed values.

The security of the RSA algorithm relies on the difficulty of factoring the modulus, n, which is a product of the two prime numbers, p and q. If an attacker can factor n, they can compute the private key, d, and decrypt the encrypted data. Therefore, it is essential to choose large prime numbers, p and q, to ensure the security of the algorithm.

In addition to the security benefits, the RSA algorithm also offers several other advantages, including:

- Key exchange: The RSA algorithm allows for secure key exchange between two parties over an insecure communication channel.

- Digital signatures: The RSA algorithm can be used to create digital signatures, which are used to authenticate the sender of a message and ensure the integrity of the data.

- Authentication: The RSA algorithm can be used for authentication purposes, such as verifying the identity of a user or device.

**1. Importance of RSA Algorithm**

The RSA algorithm, named after its inventors Rivest, Shamir, and Adleman, is a widely used public-key cryptographic system. It relies on the mathematical complexity of factoring large prime numbers to provide encryption, digital signatures, and key exchange mechanisms. RSA's significance lies in its ability to facilitate secure communication over insecure channels, enable digital signatures for authentication, and support secure data storage and transmission in various applications such as e-commerce, secure messaging, and digital identity management.

**2. FPGA Technology for Cryptographic Implementations**

Field-Programmable Gate Arrays (FPGAs) offer a unique blend of hardware and software flexibility, making them well-suited for implementing cryptographic algorithms like RSA. Unlike Application-Specific Integrated Circuits (ASICs), FPGAs can be reprogrammed and customized to adapt to evolving cryptographic standards and requirements. Moreover, FPGAs provide parallel processing capabilities, enabling efficient execution of complex cryptographic operations such as modular exponentiation and modular multiplication inherent in the RSA algorithm. These features make FPGAs an attractive platform for cryptographic implementations, offering high performance, low latency, and scalability.

**3. Challenges in FPGA-Based RSA Implementation**

Despite the advantages offered by FPGAs, implementing the RSA algorithm poses several challenges. The computational complexity of RSA operations, especially modular exponentiation, demands efficient hardware architectures and optimized algorithms to achieve high throughput and low latency. Additionally, resource constraints and power limitations inherent in FPGA platforms require careful design and optimization to balance performance, resource utilization, and energy efficiency. Furthermore, ensuring security against side-channel attacks, fault injection attacks, and other vulnerabilities is paramount in FPGA-based RSA implementations.

**4. Recent Advancements and Research Directions**

In recent years, significant progress has been made in optimizing and accelerating RSA implementations on FPGA platforms. Researchers have explored novel hardware architectures, parallel processing techniques, and algorithmic optimizations to enhance performance and resource utilization. Moreover, advancements in security-aware design methodologies, hardware-based countermeasures, and cryptographic protocols have strengthened the resilience of FPGA-based RSA implementations against various security threats. Future research directions may focus on leveraging emerging FPGA technologies such as high-level synthesis, heterogeneous computing, and hardware-software co-design to further improve the efficiency, security, and scalability of RSA implementations on FPGA platforms.

In conclusion, FPGA-based implementations of the RSA algorithm offer a promising avenue for achieving efficient and secure cryptographic solutions in various applications. By addressing the challenges and leveraging advancements in FPGA technology and cryptographic techniques, researchers continue to push the boundaries of performance, security, and flexibility in RSA implementations on FPGA platforms, contributing to the advancement of secure digital communication and data protection in the digital age.

## Problem statement

## AIMANDOBJECTIVE

* Ourgoalistoaccomplishfaultdetectionandclassificationfortransmissionlineprotection.
* The faultclassificationisestimatedbyK-meansclusteringalgorithm.

**2.4 MODULES USED**

* **Key generation ​**

Key generation is a critical component of cryptography, as it enables secure communication over an insecure channel**.**

* **Encryption**

Encryption is the process by which a readable message is converted to an

unreadable form to prevent unauthorized parties from reading it

* **Decryption**

Decryption is the process of converting an encrypted message back to its original (readable) format.​

​

​

**2.4.1 SUB MODULES**

* Modular multiplication ​
* Divider 16 bit​
* Divider 32 bit ​

**CHAPTER 3**

**KEY GENERATION**

**3.1 INTRODUCTION**

The RSA algorithm is a widely used public-key encryption technique that has been employed to secure online transactions, communication networks, and digital signatures for over four decades. The algorithm's security relies heavily on the generation of secure keys, which are used for encryption and decryption. Key generation is a critical component of the RSA algorithm, as it directly affects the security of the data being transmitted.

In this project, we aim to implement the key generation function in Verilog for hardware FPGA implementation, demonstrating the feasibility of using FPGAs for RSA key generation. This approach offers several advantages, including high-speed and low-power consumption, making it suitable for a wide range of applications, including secure communication networks and digital signatures.

The RSA algorithm was first introduced in 1978 by Ron Rivest, Adi Shamir, and Leonard Adleman, and it has since become a widely accepted standard for secure data transmission. The algorithm's security is based on the difficulty of factoring large composite numbers, which are used to generate the public and private keys.

The key generation process involves selecting two large prime numbers, p and q, and computing the modulus, n, and the Euler's totient function, φ(n). The public exponent, e, is then chosen, and the private exponent, d, is computed using the extended Euclidean algorithm. Finally, the public and private keys are generated using the computed values.

The security of the RSA algorithm relies on the difficulty of factoring the modulus, n, which is a product of the two prime numbers, p and q. If an attacker can factor n, they can compute the private key, d, and decrypt the encrypted data. Therefore, it is essential to choose large prime numbers, p and q, to ensure the security of the algorithm.

In addition to the security benefits, the RSA algorithm also offers several other advantages, including:

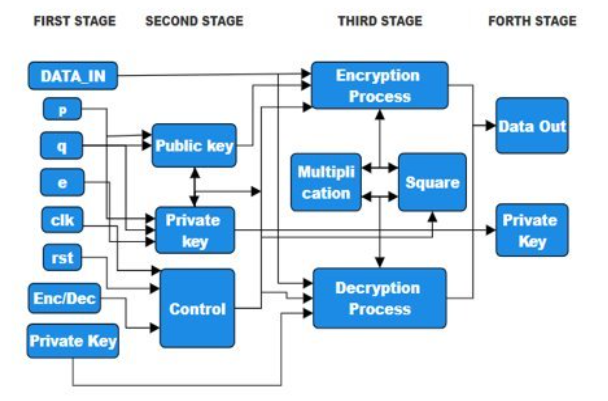
- Key exchange: The RSA algorithm allows for secure key exchange between two parties over an insecure communication channel.

- Digital signatures: The RSA algorithm can be used to create digital signatures, which are used to authenticate the sender of a message and ensure the integrity of the data.

- Authentication: The RSA algorithm can be used for authentication purposes, such as verifying the identity of a user or device.

**3.2 FLOW CHART**

**Conversion Of Plain Text To Cipher Text Using Public Key Encryption**



# 3.3 TYPE OF KEYS :

Key generation is a critical component of cryptography, as it enables secure communication over an insecure channel.

Types of Keys:

There are two types of keys used in key generation:

1. **Public Key** (PK): The public key is used for encryption and is made publicly available. It consists of the modulus, n, and the public exponent, e.

2. **Private Key** (SK): The private key is used for decryption and is kept secret. It consists of the modulus, n, and the private exponent, d.

How Keys Help to Encrypt and Decrypt Messages:

The public and private keys play a crucial role in encrypting and decrypting messages. Here's how it works:

**Encryption:**

1. Alice wants to send a message, M, to Bob.

2. Alice uses Bob's public key, PK, to encrypt the message, M.

3. The encrypted message, C, is computed as C = M^e (mod n).

4. Alice sends the encrypted message, C, to Bob.

**Decryption:**

1. Bob receives the encrypted message, C.

2. Bob uses his private key, SK, to decrypt the message, C.

3. The decrypted message, M, is computed as M = C^d (mod n).

4. Bob recovers the original message, M.

How Keys Help to Decrease the Message:

The RSA algorithm uses large key sizes to ensure security, which can result in increased computational overhead and memory requirements. However, the keys can be used to decrease the message size, making it more efficient for transmission. Here's how:

1. Message compression: The message, M, can be compressed using a compression algorithm, such as gzip or zlib.
2. Encryption: The compressed message, M', is encrypted using the public key, PK.
3. Transmission: The encrypted message, C, is transmitted over the insecure channel.
4. Decryption: The encrypted message, C, is decrypted using the private key, SK.
5. Decompression: The decrypted message, M', is decompressed using the compression algorithm.

By using the keys to encrypt and decrypt messages, the RSA algorithm provides a secure and efficient way to transmit data over an insecure channel. The keys help to decrease the message size, making it more efficient for transmission, while ensuring the security and integrity of the data.

**3.4 STEPS TO GENERATE THE KEY:**

The steps to generate a key pair for the RSA algorithm are:

Step 1: Choose two large prime numbers, p and q

- These prime numbers should be secret and randomly chosen

- They should be of similar size, e.g. both 1024-bit or both 2048-bit

Step 2: Compute the modulus, n = p \* q

- This is the public modulus, which is used to encrypt and decrypt data

Step 3: Compute the Euler's totient function, φ(n) = (p - 1) \* (q - 1)

- This is a value that is used to compute the private exponent, d

Step 4: Choose the public exponent, e

- This should be a small prime number, e.g. 3 or 65537

- It should be coprime with φ(n), meaning that gcd(e, φ(n)) = 1

Step 5: Compute the private exponent, d

- This is computed using the extended Euclidean algorithm

- d is the modular multiplicative inverse of e modulo φ(n), i.e. d \* e ≡ 1 (mod φ(n))

Step 6: Generate the public key, PK = (e, n)

- This is the public key that is used to encrypt data

Step 7: Generate the private key, SK = (d, n)

- This is the private key that is used to decrypt data

Note: These steps are for generating a basic RSA key pair. In practice, additional steps may be taken to ensure security, such as:

- Generating a random seed for the prime numbers

- Using a secure random number generator

- Testing the prime numbers for primality

- Using a secure key storage and management system

It's also important to note that the key size should be sufficient for security, e.g. at least 2048-bit.

**3.5IMPLEMENTATION USING FPGA**

**Hardware Description Language (HDL**)**:**

The implementation of RSA key generation using FPGA typically involves designing and coding hardware circuits using a Hardware Description Language (HDL) such as Verilog or VHDL. HDL allows designers to describe the behavior and structure of digital circuits, including arithmetic operations, logic gates, and memory elements.

1. **Modular Arithmetic Operations**

RSA key generation requires various modular arithmetic operations, such as multiplication, exponentiation, and modular inversion. These operations can be implemented efficiently in hardware using FPGA resources. Parallel processing techniques can be employed to accelerate computation and improve throughput.

1. **Pipelining and Parallelism**

FPGAs offer the advantage of parallelism, allowing multiple operations to be performed simultaneously. Pipelining techniques can be used to overlap the execution of successive operations, reducing latency and increasing throughput. By partitioning the key generation process into stages and implementing each stage in parallel, overall performance can be improved.

1. **Resource Optimization**

Efficient utilization of FPGA resources is critical for achieving high performance and cost-effectiveness. Designers must optimize the allocation of logic elements, memory blocks, and routing resources to minimize resource usage while meeting performance requirements. Techniques such as resource sharing, multiplexing, and algorithmic optimizations can be employed to reduce resource consumption.

1. **Clock Management and Timing Constraints**

FPGA designs require careful consideration of clock management and timing constraints to ensure proper operation and synchronization of digital circuits. Clock signals must be distributed efficiently, and timing requirements must be met to avoid timing violations and ensure reliable operation. Timing analysis tools can be used to verify timing constraints and optimize clock frequency.

1. **Verification and Testing**

Before deploying an FPGA-based RSA key generation system, thorough verification and testing are essential to ensure correctness and reliability. Simulation tools can be used to verify the functionality of the hardware design, while hardware-in-the-loop testing can be performed to validate the design on physical FPGA hardware. Testbenches and test vectors are used to evaluate the performance and robustness of the implementation under various conditions.

**3.6 LIMITATION**

However, the RSA algorithm also has some limitations, including:

**Key size**: The RSA algorithm requires large key sizes to ensure security, which can result in increased computational overhead and memory requirements.

**Computation time**: The RSA algorithm can be computationally intensive, particularly for large key sizes, which can result in increased computation time.

# 3.7 APPLICATION AND CONCLUSION

# Applications and Benefits

**1. Secure Communication**

RSA key generation using FPGA enables secure communication systems, such as encrypted messaging platforms, virtual private networks (VPNs), and secure web protocols. The fast and efficient generation of RSA key pairs ensures the confidentiality and integrity of sensitive data transmitted over insecure networks.

**2. Digital Signatures**

Digital signatures play a crucial role in ensuring the authenticity and integrity of electronic documents and transactions. FPGA-based RSA key generation facilitates the generation and verification of digital signatures with high performance and low latency, making it suitable for applications such as electronic commerce, digital certificates, and document authentication.

**3. Cryptographic Acceleration**

FPGAs provide cryptographic acceleration for computationally intensive algorithms such as RSA, enabling faster key generation and encryption/decryption operations compared to software-based implementations. This acceleration is particularly beneficial for applications requiring real-time cryptographic processing and low-latency response times.

**4. Hardware Security Modules (HSMs)**

FPGA-based RSA key generation can be integrated into hardware security modules (HSMs) to provide tamper-resistant storage and processing of cryptographic keys. HSMs are used to protect sensitive cryptographic operations and key management functions, ensuring compliance with security standards and regulations in industries such as finance, healthcare, and government.

**Conclusion**

RSA key generation is a fundamental component of the RSA encryption algorithm, enabling secure communication and digital signatures in various applications. Implementing RSA key generation using FPGA offers advantages such as high performance, parallel processing, and customizability, making it well-suited for cryptographic applications requiring fast and efficient key generation. By leveraging FPGA technology, organizations can enhance the security and reliability of their cryptographic systems while meeting performance and scalability requirements.

# CHAPTER 4

# ENCRYPTION

## 4.1 INTRODUCTIONOF ENCRYPTION

Encryption is the process of converting plaintext (human-readable data) into ciphertext (unreadable data) using an encryption algorithm and a key. The purpose of encryption is to protect the confidentiality and integrity of data, ensuring that only authorized parties can access and understand the information.

The RSA encryption algorithm is a widely used public-key encryption technique that is based on the principles of asymmetric cryptography. It is used to secure online transactions, communication networks, and digital signatures. The RSA algorithm uses a pair of keys, namely the public key and the private key, to encrypt and decrypt messages. In this section, we will discuss the encryption function of the RSA algorithm, which is used to encrypt messages using the public key.

Implementing encryption using the RSA algorithm on FPGA (Field-Programmable Gate Array) involves several key steps, including key generation, message padding, modular exponentiation, and cipher text generation. Below is a high-level overview of the encryption process along with considerations for FPGA implementation:Here's how encryption works:

**1. Encryption Algorithm:**

An encryption algorithm is a set of mathematical rules and procedures used to transform plaintext into cipher text.

Common encryption algorithms include AES (Advanced Encryption Standard), RSA (Rivest-Shamir-Adleman), and DES (Data Encryption Standard).

**2. Key:**

A key is a piece of information used by the encryption algorithm to control the encryption and decryption processes.

In Symmetric Encryption:

The Same Key Is Used For Both Encryption And Decryption.

In Asymmetric Encryption:

Different Keys Are Used For Encryption And Decryption.

**3. Encryption Process:**

In symmetric encryption, the plaintext and encryption key are inputted into the encryption algorithm, resulting in ciphertext.

In asymmetric encryption, the plaintext is encrypted using the recipient's public key, and the ciphertext can only be decrypted using the recipient's private key.

**4. Ciphertext:**

Ciphertext is the encrypted form of plaintext and appears as a random sequence of characters.

It is unreadable without the corresponding decryption key.

**Importance of Encryption:**

**Confidentiality**: Encryption ensures that only authorized parties can access the plaintext data, protecting it from unauthorized access or interception.

**Integrity**: Encryption helps maintain the integrity of data by detecting any unauthorized modifications or tampering during transmission.

**Authentication**: Encryption can be used to verify the identity of the sender or recipient of encrypted data, ensuring secure communication.

**Types of Encryption:**

**Symmetric Encryption:**

In symmetric encryption, the same key is used for both encryption and decryption.

Examples include AES (Advanced Encryption Standard), DES (Data Encryption Standard), and 3DES (Triple DES).

Symmetric encryption is typically faster and more efficient than asymmetric encryption.

**Asymmetric Encryption:**

In asymmetric encryption, different keys are used for encryption and decryption. A public key is used for encryption, while a private key is used for decryption.

Examples include RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Curve Cryptography).

Hash Functions:

Hash functions are one-way mathematical algorithms that convert input data into a fixed-size string of characters, called a hash value or digest.

They are used for data integrity verification, digital signatures, and password hashing.

**Examples** include SHA-256 (Secure Hash Algorithm 256-bit) and MD5 (Message DigestAlgorithm 5).

**4.2KEY USED**

The RSA encryption function uses the public key, which consists of two components: the modulus (n) and the public exponent (e). The modulus (n) is a product of two large prime numbers (p and q), and the public exponent (e) is a small prime number that is coprime with the Euler's totient function (φ(n)) of the modulus (n). The public key is denoted as (e, n) and is used to encrypt messages.

The **PUBLIC KEY** is used to encrypt messages, and it is typically made available to anyone who wants to send a secure message. The private key, on the other hand, is used to decrypt messages and is kept secret to prevent unauthorized access to the encrypted data.

**4.3 STEPS TO ENCRYPT THE MESSAGE**

The steps to encrypt a message using the RSA encryption function are as follows:

**Step 1**: Convert the message to a numerical value

The message is converted to a numerical value using a padding scheme, such as PKCS#1. This is done to ensure that the message is in a format that can be encrypted by the RSA algorithm.

**Step 2**: Compute the ciphertext

The ciphertext is computed by raising the numerical value of the message to the power of the public exponent (e) modulo the modulus (n). This is represented mathematically as:

**Ciphertext = (Message^e) mod n**

The ciphertext is a numerical value that represents the encrypted message.

**Step 3**: Output the ciphertext

The resulting ciphertext is output as the encrypted message.

**EXAMPLE**

Let's consider an example to illustrate how the RSA encryption function works. Suppose we want to encrypt the message "HELLO" using the RSA algorithm. We first convert the message to a numerical value using a padding scheme, such as PKCS#1. Let's assume that the numerical value of the message is 123456.

Next, we compute the ciphertext by raising the numerical value of the message to the power of the public exponent (e) modulo the modulus (n). Let's assume that the public key is (3, 391), where 3 is the public exponent and 391 is the modulus. Then, the ciphertext is computed as:

Ciphertext = (123456^3) mod 391 = 279841

Finally, the resulting ciphertext is output as the encrypted message.

Decryption

[3:06 PM, 5/9/2024] RiyaJss: Decryption is the process of converting encrypted data back into its original, understandable form. It’s a fundamental aspect of cryptography, the science of secure communication. In the realm of digital security, encryption and decryption play vital roles in safeguarding sensitive information from unauthorized access. At its core encryption involves transforming plaintext into ciphertext using a mathematical algorithm and a cryptographic key.

This process ensures that even if an unauthorized party intercepts the ciphertext, they cannot make sense of it without the proper decryption key. Decryption, therefore, is the reverse process. It takes the ciphertext and using the decryption algorithm and the corresponding key, translates it back into its original plaintext form. Without the correct key, decrypting the data becomes computationally infeasible ensuring the confidentiality of the information.

There are various encryption algorithms and techniques used in modern cryptography, each with its strengths and weaknesses. Some of the most widely used encryption algorithms include:

Symmetric Encryption: In symmetric encryption, the same key is used for both encryption and decryption. This makes the decryption process relatively simple and fast compared to other methods. However, securely distributing the key to all parties involved can be a challenge.

Asymmetric Encryption: Asymmetric encryption, also known as public-key encryption, uses a pair of keys: a public key for encryption and a private key for decryption. This approach addresses the key distribution problem inherent in symmetric encryption. However, asymmetric algorithms are typically slower and require more computational resources.

Block Ciphers: Block ciphers encrypt data in fixed-size blocks, typically 64 or 128 bits at a time. Popular block ciphers include Advanced Encryption Standard (AES) and Data Encryption Standard (DES).

Stream Ciphers: Stream ciphers encrypt data one bit or byte at a time, making them suitable for real-time communication applications. However, they can be susceptible to certain types of attacks if not implemented correctly.

Decryption in cryptography requires not only the correct algorithm and key but also adherence to best practices in key management, algorithm selection, and implementation. Weaknesses in any of these areas can compromise the security of the encrypted data and render the encryption ineffective.

Moreover, the security of encryption systems often depends on the secrecy of the encryption keys. If an attacker gains access to the decryption key, they can decrypt the ciphertext and access the original plaintext, potentially exposing sensitive information. Therefore, protecting the confidentiality of encryption keys is crucial in maintaining the security of encrypted data.

Decryption plays a critical role in various applications of cryptography, including secure communication, data protection, authentication, and digital signatures. From securing sensitive communications over the internet to protecting personal information stored on electronic devices, encryption and decryption are essential components of modern cybersecurity practices.

In conclusion, decryption is the process of converting encrypted data back into its original form using a decryption algorithm and the appropriate key. It is a fundamental aspect of cryptography, ensuring the confidentiality and integrity of sensitive information in digital communication and storage. Effective decryption relies on secure algorithms, robust key management practices, and proper implementation to mitigate the risk of unauthorized access and data breaches.

[3:11 PM, 5/9/2024] RiyaJss: Within the field of cryptography, decryption functions as the opposite of encryption, converting ciphertext back into plaintext. An essential component of decryption is a key, usually referred to as the private key. The intended recipient or other party with access rights to the encrypted data frequently keeps this key confidential. In both symmetric and asymmetric encryption techniques, the private key is an essential element.The same key is used for encryption and decryption in symmetric encryption. To maintain secrecy, communicating parties must safely share this key. In contrast, asymmetric encryption, sometimes referred to as public-key cryptography, makes use of two keys: a private key for decryption and a public key for encryption. The private key is still in place.

[3:12 PM, 5/9/2024] RiyaJss: Steps to Decrypt the Cipher Text into Plain Message Along with Example

Decryption involves a series of steps to revert the ciphertext back into its original plaintext form. These steps typically include obtaining the ciphertext, selecting the appropriate decryption algorithm, applying the private key, and finally, retrieving the plaintext message. Let's explore each step in detail with an illustrative example:

Step 1: Obtain the Ciphertext

The decryption process begins with obtaining the ciphertext, which is the encrypted form of the original message. This ciphertext is generated through an encryption algorithm using the intended recipient's public key in asymmetric encryption or a shared secret key in symmetric encryption. For decryption to occur successfully, the ciphertext must be accessible to the recipient possessing the corresponding private key.

Example:

Suppose Alice wishes to send a confidential message to Bob using asymmetric encryption. Alice encrypts the message using Bob's public key, resulting in the ciphertext. Bob, possessing the corresponding private key, can decrypt the ciphertext to retrieve the original plaintext message.

Step 2: Select the Decryption Algorithm

Once the ciphertext is obtained, the next step involves selecting the appropriate decryption algorithm based on the encryption scheme used. Different encryption algorithms require specific decryption algorithms for reversal. Common encryption algorithms include RSA (Rivest-Shamir-Adleman) for asymmetric encryption and AES (Advanced Encryption Standard) for symmetric encryption.

Example Continuation:

In the scenario involving Alice and Bob, if Alice encrypts the message using RSA encryption, Bob must utilize the RSA decryption algorithm to decipher the ciphertext.

[3:16 PM, 5/9/2024] RiyaJss: Step 3: Utilise the Secret Key

The recipient applies their private key to the ciphertext after choosing the decryption algorithm. The mathematical characteristics of this private key enable it to be used to reverse the encryption process, so disclosing the original plaintext message and unlocking the ciphertext.

Example Continuation: Bob encrypts data using his private key, which matches Alice's public key. Bob is able to decipher the ciphertext and see Alice's original plaintext message by using his private key and the RSA decryption technique.

Step 4: Get the message in plaintext

Getting the plaintext message out of the decrypted ciphertext is the last step once decryption is finished. The accuracy of the decryption procedure should be confirmed by the decrypted plaintext matching the original message.

[3:17 PM, 5/9/2024] RiyaJss: Example Conclusion:

After decrypting the ciphertext using his private key and the RSA decryption algorithm, Bob retrieves the plaintext message, allowing him to read the confidential message sent by Alice securely.