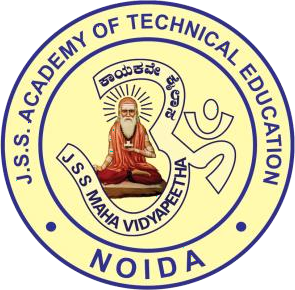
**RSA IMPLEMENTATION USING FPGA**

BY

## NITIN SINGH (2000910310104)

## 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

# TABLEOFCONTENTS

[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. 5](#_TOC_250011)

[CHAPTER2INTRODUCTION,AIM&OBJECTIVE 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. [Application and Conclusion 14](#_TOC_250003)

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.

RSA (rivest, shamir, adleman) is an Asymmetric Encryption algorithm that provides data authentication and security. Two sets of keys are used i.e. Public key and private key on encryption and decryption side ​

(Public key [e] on encryption and private key[d] on decryption side). ​

Plain text is converted / encrypted into cipher text using public key on the encryption side or senders’s side while this cipher text is again converted/decrypted using private key on decryption side i.e. receiver’s side.​

​

**TWO MAIN COMPONENTS:​**

KEY GENERATION ​

ENCRYPTION / DECRYPTION FUNCTIONS​

## 

## 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 AIM & OBJECTIVE

## 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.

## PROBLEMSTATEMENT

* + - Manualfaultdetectionisatime-consumingprocess.Theproblemmightberectifiedwithinafewhoursoritmighteventake days.
    - Itmayleadtomonetarylossesandmayevenrequireextramanuallabor.
    - Theprocessesthatarebeingcurrentlyusedaresoaccuratetodetectandclassifyfault.
    - sometimesmightnotbehappenedtoclassifythefaultmanual.

## 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**

RSA (Rivest-Shamir-Adleman) is a widely used asymmetric encryption algorithm, named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. It relies on the computational complexity of factoring large prime numbers for its security. The RSA algorithm involves generating a public key and a private key pair, where the public key is used for encryption and the private key is used for decryption.

Field-Programmable Gate Arrays (FPGAs) are semiconductor devices that can be programmed and reconfigured after manufacturing. They consist of an array of programmable logic blocks and interconnects, allowing for the implementation of custom digital circuits. FPGAs offer advantages such as high performance, parallel processing capabilities, and flexibility, making them well-suited for cryptographic applications like RSA.

**Key Generation Process**

1. Prime Number Generation

The RSA key generation process begins with the selection of two large prime numbers, typically denoted as p and q. These prime numbers are randomly generated within a specified range. Generating large prime numbers efficiently is crucial for the security of the RSA algorithm.

**2. Modulus Calculation**

Once the prime numbers p and 𝑞 are generated, they are multiplied to obtain the modulus n, which is part of both the public and private keys. The modulus is calculated as n=p×q. The modulus n represents the size of the keyspace and determines the maximum size of the plaintext that can be encrypted.

**3. Euler's Totient Function**

The next step involves calculating Euler's totient function ϕ(n), where ϕ(n)=(p−1)×(q−1). Euler's totient function is crucial for deriving the private key exponent and ensuring the security of the RSA algorithm.

**4. Public Key Generation**

The public key consists of two components: the modulus 𝑛 and the public exponent 𝑒. The public exponent is typically a small prime number, commonly chosen as e=65537 due to its efficiency in encryption operations. The public key is represented as (n,e).

**5. Private Key Calculation**

The private key consists of the modulus 𝑛 and the private exponent 𝑑. The private exponent is derived using the modular multiplicative inverse of the public exponent modulo ϕ(n). In other words, 𝑑 satisfies the equation e×d≡1(modϕ(n)). The private key is represented as (n,d).

**6. Key Storage and Management**

Once the public and private keys are generated, they need to be securely stored and managed. The private key must be kept confidential and protected from unauthorized access, while the public key can be freely distributed to parties wishing to communicate securely.

**Implementation 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**

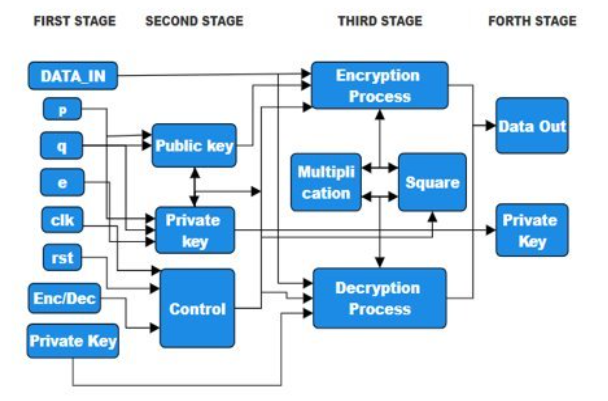
FPGA designs require careful consideration of clock management and timing constraints to ensure proper operation and synchronization of digital circuits. Clock signals must be distributed efficiently, and timing requirements must be met to avoid timing violations and ensure reliable operation. Timing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

**FLOW CHART**

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



**TYPES OF KEYS**

**Types of Keys in RSA Key Generation Using FPGA**

RSA (Rivest-Shamir-Adleman) is a widely-used asymmetric encryption algorithm that relies on the generation of public and private key pairs. The key generation process involves several types of keys, each with its own significance and role in ensuring the security and efficiency of the RSA algorithm. When implementing RSA key generation using FPGA (Field-Programmable Gate Array), it's crucial to understand the types of keys involved and their respective characteristics.

1. **Public Key**

The public key in RSA encryption is used for encrypting plaintext data. It consists of two components: the modulus 𝑛 and the public exponent 𝑒. These values are derived during the key generation process and are made publicly available to anyone who wishes to send encrypted messages to the owner of the corresponding private key. The public key is typically represented as (n,e).The security of the RSA algorithm relies on the computational difficulty of factoring the modulus n into its prime factors p and q, which makes it infeasible for an attacker to derive the private key from the public key alone.

**Characteristics:**

Modulus (n): The modulus 𝑛 is the product of two large prime numbers, p and q, where

n=p×q. It determines the size of the keyspace and the maximum size of the plaintext that can be encrypted.

Public Exponent (e): The public exponent e is a small prime number chosen for its efficiency in encryption operations. The most common choice is e=65537 due to its fast modular exponentiation properties.

1. **Private Key**

The private key in RSA encryption is used for decrypting ciphertext data encrypted with the corresponding public key. It consists of two components: the modulus n (which is the same as in the public key) and the private exponent d. The private key must be kept confidential and securely stored by its owner to prevent unauthorized access.

The private key in RSA encryption plays a critical role in the security and functionality of the cryptographic system. While the public key is used for encrypting plaintext data, the private key is required for decrypting ciphertext and recovering the original plaintext. The private key enables the owner to securely access and decode encrypted messages sent using their corresponding public key

**Characteristics:**

Modulus (n): The modulus n is the same as in the public key and is used for decrypting ciphertext using modular exponentiation.

Private Exponent (d): The private exponent d is derived during the key generation process and satisfies the equation e×d≡1(modϕ(n)), where ϕ(n) is Euler's totient function of n. The private exponent is used for decrypting ciphertext using modular exponentiation.

**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.

# 

# STEPS OF KEY GENERATION

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.

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.

# 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.

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 Digest Algorithm 5).

**Encryption Modes:**

**Electronic Codebook (ECB):**

Each block of plaintext is encrypted independently, resulting in identical blocks of ciphertext for identical plaintext blocks.

Vulnerable to patterns in the plaintext and not recommended for secure communications.

**Cipher Block Chaining (CBC):**

Each plaintext block is XORed with the previous ciphertext block before encryption.

Requires an initialization vector (IV) to randomize the first block and prevent patterns in the plaintext.

Provides better security than ECB mode.

**Counter (CTR) Mode:**

Uses a counter value as input to generate a key stream, which is then XORed with the plaintext to produce ciphertext.

Allows for parallel encryption and decryption, making it suitable for high-speed data processing.

**Encryption Key Management:**

**Key Generation**: Keys must be generated securely using random or pseudo-random number generators.

**Key Distribution**: Secure distribution of keys to authorized parties is crucial to ensure the confidentiality and integrity of encrypted data.

**Key Storage:** Keys should be securely stored and managed to prevent unauthorized access or compromise.

**Key Rotation**: Periodic rotation of encryption keys helps mitigate the risk of key compromise and enhances security.

Challenges and Considerations:

Performance: Encryption and decryption operations can impact system performance, especially in high-throughput applications.

**Key Length**: Longer key lengths generally provide greater security but may impact performance and resource consumption.

**Compliance**: Compliance with regulatory requirements and industry standards (e.g., GDPR, HIPAA) may dictate encryption practices for data protection.

Cryptographic Agility: Systems should be designed with cryptographic agility to support the transition to stronger encryption algorithms and key lengths as needed.