HARDWARE-BASED IMPLEMENTATIONS OF THE RSA ALGORITHM AND THEIR POTENTIAL USE IN PROVIDING DATA SECURITY IN CYBER-PHYSICAL SYSTEMS

A Thesis

by

CHRISTIAN MCLANE LEDGARD

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Garth Crosby
Committee Members,	Rainer Fink
	Ana Goulart
	Katherine Davis
Head of Department,	Reza Langari

August 2023

Major Subject: Engineering Technology

Copyright 2023 Christian Ledgard

ABSTRACT

Cyber-Physical Systems pose unique cybersecurity challenges because of their strict operating constraints, complex interactions, and heterogeneous nature. However, the tight integration of cyber-physical systems and critical infrastructures also makes security paramount to their implementation. Traditional, software-based methods for providing data confidentiality are often not capable of adhering to the strict temporal and spatial requirements of these systems. One proposed solution to providing data security in cyber-physical systems is the use of Field Programmable Gate Arrays to implement traditional cryptographic algorithms at a hardware level. This thesis proposes three different implementations of the RSA algorithm that will be designed, implemented, and evaluated for their feasibility in providing data security to cyber-physical systems.

DEDICATION

To my parents, for always pushing me to strive for excellence

ACKNOWLEDGEMENTS

I would first like to thank my committee chair, Dr. Crosby, for his unwavering support and steadfast guidance throughout the course of this research. I consider myself beyond lucky to have an advisor whose razor-sharp wit and infallible wisdom have guided me through my time at Texas A&M. His belief in my ability to complete this work has been invaluable. I would also like to thank the members of my committee, Dr. Fink, Dr. Goulart, and Dr. Davis, whose critiques and insight have helped to ensure the best possible final product of my research.

I am deeply grateful to all of my friends and family who have accompanied me on this journey. For each of my victories and shortcomings, they were always there.

Finally, I am eternally grateful to my partner, Katie, who spent endless hours on the phone letting me vent. I could never thank you enough.

I am honored to have had this experience.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Dr. Garth Crosby, Dr. Rainer Fink, and Dr. Ana Goulart of the Department of Engineering Technology and Industrial Distribution and Dr. Katherine Davis of the Department of Electrical and Computer Engineering.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

This research was funded by Dr. Garth Crosby through Texas A&M University. Templates for documents associated with this research were provided by Texas A&M University.

NOMENCLATURE

BPS	Bits-per-Second
CPS	Cyber-physical Systems
DSP	Digital Signal Processing
FF	Flip-Flop
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
ICCP	Inter-control Center Communications Protocol
J	Joules
LUT	Look-up-Table
RAM	Random Access Memory
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
W	Watts

TABLE OF CONTENTS

Page

ABSTRACT ... ii

DEDICATION ... iii

ACKNOWLEDGEMENTS ... iv

CONTRIBUTORS AND FUNDING SOURCES ... v

NOMENCLATURE ... vi

TABLE OF CONTENTS ... vii

LIST OF FIGURES ... ix

LIST OF TABLES ... x

CHAPTER I INTRODUCTION ... 1

1.1 Problem ... 1
1.2 Proposal ... 3
1.3 Format ... 3

CHAPTER II BACKGROUND ... 5

2.1 RSA Algorithm ... 5
2.2 Repeated Modular Multiplication ... 6
2.3 Binary Exponentiation (LR Method) ... 7
2.4 Montgomery Multiplication ... 7

CHAPTER III METHODS ... 10

3.1 Hardware Design ... 10
3.2 Metrics ... 12
3.2.1 Frequency ... 12
3.2.2 Area Utilization ... 13
3.2.3 Latency ... 14
3.2.4 Throughput ... 15
3.2.5 Power and Energy ... 15
3.3 Embedded System Performance Benchmarks ... 16
3.3.1 Frequency ... 17

3.3.2 Latency and Throughput 18
3.3.3 Power 19
3.4 Survey of Existing Hardware Designs 20

CHAPTER IV HARDWARE DESIGN 23

4.1 Repeated Modular Multiplication 23
4.2 Binary Exponentiation 24
4.3 Montgomery Exponentiation 25

CHAPTER V RESULTS 30

5.1 Frequency 30
5.2 Utilization 30
5.3 Latency and Throughput 33
5.4 Power and Energy 39

CHAPTER VI ANALYSIS 44

6.1 Strengths and Weaknesses 44
6.2 Comparison to Embedded System Performance 46
6.3 Comparison to Other Hardware Implementations 48

CHAPTER VII CONCLUSION 51

7.1 Future Works 52

REFERENCES 53

APPENDIX A PYTHON SCRIPT FOR GENERATING RSA KEYS 57

LIST OF FIGURES

Page

Figure 1 Artix-7 FPGA Board 10

Figure 2 Vivado Synthesis Configuration.. 11

Figure 3 Vivado Timer Settings... 12

Figure 4 Behavioral Simulation Waveform Output.. 14

Figure 5 Sample Power Report Output 16

Figure 6 Repeated Modular Multiplication Multiplier Block.. 23

Figure 7 Montgomery Multiplier Block 26

Figure 8 Modular Inverse Core Block 27

Figure 9 Distributed Memory Generator Block.. 28

Figure 10 FIFO Generator Block... 29

Figure 11 LUT Utilization at Varying Key Sizes 31

Figure 12 FF Utilization at Varying Key Sizes 32

Figure 13 Total Power Consumption... 39

Figure 14 Total Energy Consumption.. 42

LIST OF TABLES

Page

Table 1 Latency for 1024-bit RSA Encryption and Decryption on MSP430 [16] 18

Table 2 Throughput for 1024-bit RSA Encryption and Decryption on MSP430 [16] 19

Table 3 Maximum Clock Frequency 30

Table 4 LUT Utilization Data 31

Table 5 FF Utilization Data 32

Table 6 Latency and Throughput Data for Repeated Modular Multiplication 33

Table 7 Latency and Throughput Data for Binary Exponentiation 35

Table 8 Minimum, Maximum, and Average Latency for Binary Exponentiation........................ 36

Table 9 Latency and Throughput Data for Montgomery Exponentiation 37

Table 10 Minimum, Maximum, and Average Latency for Montgomery Exponentiation............ 38

Table 11 Repeated Modular Multiplication Power Consumption Data....................................... 40

Table 12 Binary Exponentiation Power Consumption Data.. 40

Table 13 Montgomery Exponentiation Power Consumption Data.. 41

Table 14 Total Energy Consumption... 43

CHAPTER I

INTRODUCTION

1.1 Problem

Cyber-physical systems (CPS) consist of several interconnected, heterogeneous systems with the ability to monitor and control real objects and processes. In these systems, the physical environment is closely integrated with communication networks and computational devices. They are characterized by their ability to operate over large temporal and spatial scales, autonomously perform well-defined tasks, and exchange data in real-time. Additionally, CPS serve as the functional backbone to Industry 4.0, enabling smart applications such as industrial control systems, intelligent vehicles, and smart grid power transmission systems to operate accurately and in real-time [18].

As a result of their heterogeneity, as well as their connection to critical infrastructures, CPS are highly susceptible to various cyber threats and attacks. The vulnerabilities of these systems can be broadly categorized into three categories: communication, software, and privacy vulnerabilities. The communication vulnerabilities arise from CPS reliance on the Transmission Control Protocol/Internet Protocol (TCP/IP) and Inter-Control Center Communications Protocol (ICCP). These protocols were not intended to be secure by design and, in the case of ICCP, lack integrated security measures such as encryption or authentication [15]. Software vulnerabilities arise from spoofing methods such as Structured Query Language (SQL) injections and the use of malicious software programs [18]. Finally, privacy

vulnerabilities in CPS are the result of large amounts of data being transmitted between distinct nodes through unsecure channels, allowing for malicious actors to intercept traffic and make inferences about the performance and operation of the system [15]. Successful attacks on CPS often yield catastrophic results, making robust security paramount to their operation. The complexity of interactions within CPS and their strict operational constraints also poses unique security challenges. Devices within CPS have strict requirements for power consumption, must provide real-time interaction with the physical world, and need to operate using reduced computation and communication budgets. As a result, traditional methods for securing digital systems are often insufficient for providing security in CPS. In the context of data security, traditional, software-based cryptography requires too much network overhead and can result in unacceptable amounts of latency for ideal CPS operation. One solution to this issue is to implement cryptographic methods using hardware platforms such as Field Programmable Gate Arrays (FPGAs). Using FPGAs for cryptography in CPS offers several advantages over traditional embedded systems. Primarily, FPGAs are faster and more resource efficient than embedded systems, allowing for real-time data communication under significant resource constraints. As a result of their higher operational speed, FPGAs would also provide CPS with higher data throughput than traditional embedded computation [21]. Despite their numerous advantages, a concern when using FPGAs in the context of CPS is power consumption, as FPGAs consume far more power than embedded systems. Thus, to be suitable for use in a CPS, an FPGA based cryptography paradigm much be as energy efficient as possible, while still

providing the system with lower latency and higher throughput rates than comparable embedded system paradigms.

1.2 Proposal

To date, much research has been done on the implementation of various cryptographic protocols using FPGAs [7-9, 19-24]. These studies primarily focus on the hardware architecture and design methodology used to implement the algorithm using an FPGA. Currently, no comprehensive study has been done on the practicality of using an FPGA-based cryptography paradigm for providing data confidentiality to CPS. This research seeks to assess the feasibility of implementing a public-key cryptography scheme using FPGAs as a means of securing data in cyber-physical systems. Three hardware architectures of the RSA algorithm will be designed, and their post place and route implementations will be evaluated using timing analysis simulations, HDL synthesis/implementation reports, and FPGA power analysis tools. The simulation data collected from the evaluation of the three designs will be compared with performance benchmarks for embedded systems running the RSA Algorithm.

1.3 Format

The following sections of this paper are formatted in the following manner. Chapter Two introduces relevant background information related to the RSA algorithm, modular exponentiation techniques, and Montgomery multiplication. Chapter Three provides insight to the tools used to conduct this research, discusses the metrics used to

quantify the performance of the RSA Implementations, presents performance benchmarks for the RSA algorithm on embedded systems, and investigates the performance of other hardware designs that have been found in published literature. Chapter Four will review the hardware design for each of the three implementations. Chapter Five presents the data gathered from each of the post place and route implementations for the identified metrics: frequency, utilization, latency, throughput, and power and energy consumption. Chapter Six will discuss the implications of the results presented in Chapter Five and provide a comparison to the performance of the RSA algorithm on embedded systems and other hardware designs. Finally, Chapter Seven will summarize the contents of this paper, provide closing remarks, and discuss future works.

CHAPTER II

BACKGROUND

2.1 RSA Algorithm

The RSA algorithm is one of the most used cryptographic methods for data security. The algorithm takes advantage of the prime factorization trapdoor function: it is much easier to compute the product of two prime numbers than it is to decompose a composite number into a product of two prime integers. The RSA algorithm is comprised of two primary functions, the key generation function, and the encryption/decryption function. The steps to generating RSA public and private keys are listed below [11].

1. Chose two large prime number p and q.
2. Compute the modulus number n = p x q.
3. Calculate the Euler Totient function φ(n) = (p – 1) x (q – 1).
4. Select and integer e to use as a public key. e should be chosen such that the greatest common divisor GCD(e, φ(n)) =1 and e < φ(n)
5. Compute the private key d such that d x e = 1(mod(φ(n))).

The RSA encryption and decryption using the following two equations:

$$\mathrm{C} = \mathrm{M}^{e} \mathrm{mod}(\mathrm{n}) \tag{1}$$

$$\mathrm{M} = \mathrm{C}^{d} \mathrm{mod}(\mathrm{n}) \tag{2}$$

In these equations, M represents the plaintext message, C represents the ciphertext message, e is the public key chosen during key generation, d is the calculated private key, and n is the modulus number.

2.2 Repeated Modular Multiplication

The first approach to efficiently computing the modular exponent is known as the repeated modular multiplication approach. This algorithm for modular exponentiation takes advantage of the identity shown in equation (3) and breaks the operation into a series of repeated multiplication and reductions [21].

$$(\mathrm{a} \times \mathrm{b})\mathrm{mod}(\mathrm{n}) = \left[\left(\mathrm{a}\ \mathrm{mod}(\mathrm{n})\right) \times \left(\mathrm{b}\ \mathrm{mod}(\mathrm{n})\right)\right]\mathrm{mod}(\mathrm{n}) \tag{3}$$

The repeated modular multiplication algorithm is shown below, based on the equation (1).

1. Set C = 1, e' = 0.
2. Increase e' by 1.
3. Set C = (M x C) mod(n)
4. If $e' < e$, go to step 2. Otherwise, C contains the correct solution to equation (1).

This approach to modular exponentiation extends the operation into a series of *x* multiplications, where x is the value of the exponent in equation (1) or (2). While this approach is very memory efficient, it has an obvious shortcoming in its execution time. As the number of bits used in the exponent increases, the worst-case requirement for the number of multiplications involved increases exponentially.

2.3 Binary Exponentiation (LR Method)

The second approach to efficiently implementing modular exponentiation is known as binary exponentiation, or the square-and-multiply technique. This approach works by iterating through the bits of the exponent, either from right-to-left or from left-to-right, and performing a series of operations based on whether each bit contains a 0 or a 1 [7]. Using equation (1) with an integer size of *k*-bits, the algorithm for binary exponentiation is shown below.

0. Set C = 1, then for each bit i of the exponent e from *k-1* down to 0:
1. $C = C^2$ mod(n).
2. If $i = 1$, C = (C x M) mod(n).
3. $i = I - 1$
4. Repeat from step 1 while $i > 0$.

The square-and-multiply method of computing modular exponents is substantially faster than the repeated modular multiplication approach and requires significantly fewer multiplications. However, its hardware implementation can result in overutilization of FPGA resources, as shown in [13].

2.4 Montgomery Multiplication

The third approach to efficiently implementing modular exponentiation is known as Montgomery multiplication. This approach to modular exponentiation takes advantage of a special representation of numbers known as the Montgomery form. When using the Montgomery form, it is possible to efficiently compute the product of (a x b)

mod(N) by avoiding expensive division operations. Unlike classical modular multiplication, which requires dividing the product of (a x b) by N and keeping the remainder, the Montgomery form allows you to divide by a constant R > N, which can be selected to be a power of 2 such that division by R can be accomplished by bit shifting [10]. While Montgomery multiplication is inefficient for computing the product of a single multiplication, it is incredibly efficient for modular exponentiation through repeated multiplications, as the intermediate products can be left in the Montgomery space [19]. To perform Montgomery multiplication, integers must first be converted into their Montgomery form. This process of converting an integer into its Montgomery form is shown below in equation (4).

$$\bar{a} = a \times R \ (mod\ N) \tag{4}$$

To take the product of numbers in their Montgomery forms, and subsequently return them into integer space, two additional terms, R' and N' are needed. These two terms can be computed using the extended Euclid algorithm and must satisfy equation (5).

$$(R \times R') - (N \times N') = 1 \tag{5}$$

The process for computing the Montgomery product of two numbers, $\bar{a} * \bar{b}$, is listed below.

1. $t := \bar{a} \cdot \bar{b} \ (mod\ R)$
2. $m := t \cdot N'(mod\ R)$
3. $u := (t + m \cdot N) \div R$
4. If $u \geq N$ then return u – N, else return u

Finally, to convert a number from the Montgomery space back to its integer representation, the Montgomery product of the number and 1 is taken [10].

Findings from [22] show that hardware-based Montgomery multipliers need fewer clock cycles per multiplication than traditional hardware multipliers. Additionally, [24] suggests that using Montgomery multipliers for modular exponentiation in RSA can allow for significantly higher throughput rates than binary exponentiation or repeated modular multiplication.

CHAPTER III

METHODS

3.1 Hardware Design

Three hardware implementations of the RSA algorithm, each using a different approach to performing modular exponentiation, were designed using the Xilinx Vivado ML 2021.2 design suite. The three implementations use repeated modular multiplication, binary exponentiation, and Montgomery exponentiation, respectively, to compute modular exponents. The designs were written using the Very High-Speed Integrated Circuit Hardware Description Language (VHDL) and target and Artix-7 Family FPGA board, model xc7a100tcsg342-1, shown below in figure 1.

Figure 1 Artix-7 FPGA Board

The operational accuracy of the designs was verified using test bench designs and the Vivado xsim simulation tool. Finally, the designs were synthesized for floor planning using the synthesis configuration shown below in figure 2.

Synth Design (vivado)	
tcl.pre	...
tcl.post	...
-flatten_hierarchy	rebuilt
-gated_clock_conversion	off
-bufg	12
-directive	Default
-retiming	☐
-fsm_extraction	auto
-keep_equivalent_registers	☐
-resource_sharing	auto
-control_set_opt_threshold	auto
-no_lc	☐
-no_srlextract	☐
-shreg_min_size	3
-max_bram	-1
-max_uram	-1
-max_dsp	-1
-max_bram_cascade_height	-1
-max_uram_cascade_height	-1
-cascade_dsp	auto
-assert	☐
-incremental_mode	default

Figure 2 Vivado Synthesis Configuration

This configuration of design synthesis prevents the synthesis tool from converting clock gated logic elements into flip-flop enabled elements and limits the maximum number of global clock buffers to twelve. Additionally, this configuration allows the synthesis tool to incorporate the maximum peripheral hardware units, such as block random access memory (RAM), ultra-RAM, and digital signal processing (DSP) blocks that it deems necessary to optimize the performance of the synthesized design. By allowing the synthesis tool to allocate and configure the peripheral hardware units

through synthesis, the performance of those units in the context of their use is also guaranteed to be optimized.

3.2 Metrics

The performance of the three designs will be quantified according to six key metrics: clock frequency, area utilization, latency, throughput, power, and energy consumption. Data for each of these six metrics will be collected at six key lengths: 32-bits, 64-bits, 128-bits, 256-bits, 512-bits, and 1024-bits.

3.2.1 Frequency

The maximum possible clock frequency for the three designs will be evaluated using the Vivado timing wizard and synthesis reports using the timer settings shown in figure 3.

Settings

Enable Multi Corner Analysis:	Yes
Enable Pessimism Removal:	Yes
Pessimism Removal Resolution:	Nearest Common Node
Enable Input Delay Default Clock:	No
Enable Preset / Clear Arcs:	No
Disable Flight Delays:	No
Ignore I/O Paths:	No
Timing Early Launch at Borrowing Latches:	No
Borrow Time for Max Delay Exceptions:	Yes
Merge Timing Exceptions:	Yes

Figure 3 Vivado Timer Settings

Enabling multi-corner analysis instructs the Vivado timer to evaluate the timing configuration at both the slow and fast operating corners. The slow operating corner occurs when FPGA is experiencing high temperatures and low voltage, whereas the fast operating corner occurs at low temperatures and high voltage. For each of these corners, the clock path and data path are checked for their maximum and minimum values to determine the worst-case timing scenario. Pessimism removal allows the timer to account for this scenario in slack calculations.

3.2.2 Area Utilization

Area utilization refers to the amount of onboard logic elements that are required by a design. The Vivado synthesis process generates a utilization report upon a successful synthesis. This report details the breakdown of logic, memory, DSP, clocking, and primitive elements for all levels of the VHDL design. Logic utilization refers to the number of look-up-tables (LUTs) and slice registers that the design will use. Memory utilization accounts for the number of block RAM and ultra-RAM elements in the design. DSP utilization displays the number and type of different DPS blocks that will be required by the design to perform computationally intense operations. Clocking utilization shows the number clocks, clock buffers, and generated clocks that are routed between the logic elements of the design. The utilization report serves as a benchmark for the overall optimization of the hardware design and is heavily impacted by the design methodologies employed. Higher throughput designs which have been highly parallelized will require more logic elements to synthesize, while designs that are thoroughly pipelined will be more area efficient at the cost of throughput. Additionally,

the overall FPGA utilization has implications for the overall power consumption of the design, as the dynamic power consumption of the hardware is directly tied to the number of logic elements being used.

3.2.3 Latency

The speed at which the designs can perform an encryption or decryption is a key metric for evaluating their suitability for use in CPS. Since these systems operate on complex temporal scales, it is paramount that latency be minimized to ensure that data can be communicated in real time [14]. To evaluate the latency of these designs, testbench behavioral simulations will be used. Testbench simulations emulate the behavior of a hardware design for a given set of inputs using a virtual clock. An example of the waveform output generated from a test bench simulation is shown in figure 4.

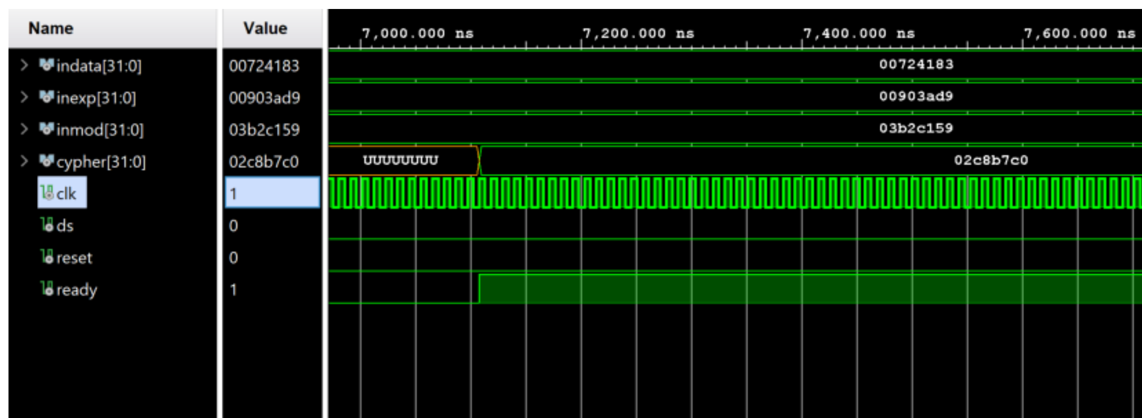

Figure 4 Behavioral Simulation Waveform Output

Using the timing information shown at the top of the waveform output, we can determine the amount of time that was required to perform an encryption or decryption by evaluating how long it takes for the inputs to a system to generate their corresponding

output. In the example above, the system has a latency of slightly more than 7100ns. The behavioral simulation can also be used to determine the approximate number of clock cycles necessary to perform the operation, by taking the product of the latency and the clock frequency as shown in equation (6).

$$Clock\ Cycles = latency \times clock\ frequency \tag{6}$$

3.2.4 Throughput

The throughput for the three implementations will be calculated using equation (7) [9].

$$Throughput = \frac{clock\ frequency \times number\ of\ bits}{number\ of\ clock\ cycles} \tag{7}$$

The number of clock cycles required for each of the three designs will be calculated based on the measured throughput. It is important to note that for binary exponentiation and Montgomery multiplication, the number of clock cycles required to compute and encryption or decryption scales with the size of the key used and not the value of the key. As a result, the encryption throughput and decryption throughput will be relatively identical. However, the number of clock cycles required to perform repeated modular multiplication in implementation one scales with the value of the exponent used and is heavily impacted by the chosen public and private keys.

3.2.5 Power and Energy

Power consumption is one of the strictest constraints placed on devices in CPS. Since they are often battery powered and must be capable of operating for long periods of time, power consumption is another key metric for evaluating if the use of an FPGA-based security paradigm is suitable for CPS, as FPGA designs consume far more power

than traditional embedded systems [4]. The power consumption for the designs will be evaluated using the Vivado power analysis tool. This tool reports the static and dynamic power requirements for the design and breaks down the dynamic power consumption into further categories based on the hardware elements used by the design. A sample of the synthesis power report is shown in figure 5.

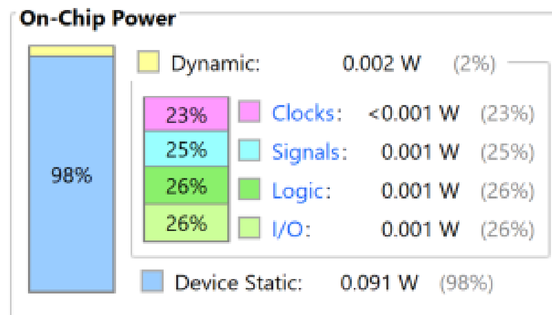

Figure 5 Sample Power Report Output

Energy consumption reflects the total number of joules consumed when each of the designs performs an encryption or decryption. Since the power report returns data in joules per second, faster designs may be reported as having high power consumption since more processes can be performed in any given second. Using data from the power consumption reports and latency measurements, the energy consumption of each design will be calculated to determine which of the designs has the lowest energy consumption per encryption and decryption.

3.3 Embedded System Performance Benchmarks

In order to assess the feasibility of hardware-based RSA implementations for their use in providing data security in CPS, it is necessary to quantify a set of

performance benchmarks for the given metrics. These benchmarks will be set through a comparison to one of the current methods of providing data security to CPS, embedded systems [16]. Area utilization will be omitted from this comparison, as it lacks a relevant equivalent in embedded systems.

3.3.1 Frequency

To parameterize the frequency constraints, it is first helpful to understand the upper and lower bounds that result from the nature of CPS. Since CPS are often used to monitor and control real processes, a lower bound on frequency exists as a consequence of the Nyquist theorem, which states that a sampling frequency must be greater than or equal to twice the maximum analog frequency being measured to ensure no loss of information [14]. However, this only confirms that a lower bound does exist and since this frequency will vary based on what process is being measured, it does not provide any quantitative information. Next, an upper constraint may be placed on frequency by considering the relationship between frequency and dynamic power consumption shown in equation (8) [4].

$$P_{dynamic} = \frac{1}{2} \cdot \alpha \cdot C_L \cdot V_{dd}^2 \cdot f_{clock} \tag{8}$$

Equation (8) shows the dynamic power consumption as a product of α, the average number of transitions per clock cycle, the load capacitance, the supply voltage, and the clock frequency. Thus, since dynamic power and frequency are directly correlated, it is necessary to minimize the clock frequency to help meet the strict power constraints of CPS. Finally, the frequency may be further constrained by considering the importance of clock synchronization between components. In order to mitigate transmission latency

and provide a common time scale for data fusion to CPS, clock synchronization is typically used [14]. This further constrains the lower bound of the frequency benchmark to be capable of running at least as fast as the average microcontroller to ensure that it is possible to achieve a common frequency. When considering the upper frequency constraint arising from power consumption and the lower constraint arising from the need for clock synchronization, it becomes apparent that the ideal frequency for an FPGA would be the same as whichever sensor or embedded system it is connected to. One common ultra-low power microcontroller, the MSP430, can run at speeds up to 25 MHz.

3.3.2 Latency and Throughput

Since the overall temporal goal in CPS is to minimize latency as much as possible [14], a performance standard for the latency of FPGA-based RSA implementations can be made through direct comparison with an embedded system running the RSA algorithm. As of 2019, [16] claimed to be capable of the fastest 1024-bit RSA encryption using an MSP430. Their results are summarized in Table 1.

Table 1 Latency for 1024-bit RSA Encryption and Decryption on MSP430 [16]

Frequency	Encryption Time (ms)	Decryption Time (s)
8 MHz	210	5.42
16 MHz	100	2.5
25 MHz	47	1.14

Using equations (6) and (7), the results from Table 1 can be used to calculate the equivalent throughput for their design. The result of these calculations is shown in Table 2.

Table 2 Throughput for 1024-bit RSA Encryption and Decryption on MSP430 [16]

Frequency	Encryption Throughput (kbps)	Decryption Throughput (bps)
8 MHz	4.876	188.93
16 MHz	10.240	409.6
25 MHz	21.787	898.25

3.3.3 Power

To be suitable for CPS applications, the power consumption of the FPGA hardware needs to be minimized in such a way that it is comparable with the power usage of embedded platforms. Ultra-low-power power microcontrollers, like the MSP430, display huge advantages over FPGAs due to their very low static power consumption. When in its low-power state, the MSP430 can consume as little as 3.9 μW. In active mode, this same device consumes a maximum of 33 mW of power running at 25 MHz [5]. This power consumption, however, only accounts for the power required for the processor to operate and does not account for outputs from the board sourcing or sinking any current. The primary disadvantage of power consumption on FPGAs is that static power consumption typically accounts for a significant portion of the overall power consumption. As shown in figure 5, static power accounts for 98% of the overall power while dynamic power comprises only 2% of the power budget.

3.4 Survey of Existing Hardware Designs

Current research on hardware based implementations of the RSA algorithm is primarily focused on Montgomery modular multiplication, though some studies have also done investigations into binary exponentiation as a method of modular exponentiation. This section will discuss several existing hardware designs present in published literature. These designs will be contextualized using performance metrics discussed in the previous section.

Sahu and Pradhan [7] propose a Montgomery modular multiplication architecture capable of running at 101.06 MHz and performing encryption operations on 32-bit data in as little as 9.895 ns. However, this design required 246% of the available slices and 225% of the available LUTs on their selected FPGA device. As a result of this overutilization, this design is not feasible for use in a real-world setting since it cannot be loaded onto an actual device.

Laracy [8] discusses a hardware-based Montgomery modular RSA implementation with a theoretical worst case latency of 2368 ns for 32-bit data. They also make the claim that by pipelining and load balancing this design, the worst case latency could decrease by a factor of 5 [8]. This paper provides little other insight into its utilization or power consumption.

Kurniasari et al. [9] have developed an ultra-lightweight architecture for Montgomery multiplication based RSA which runs at 133.76MHz and requires only 0.56% of the available flip-flops and 17.66% of the available LUTs on an Artix-7

device. This design takes 10,606ns to decrypt 32-bit ciphertext and has a throughput of 4.37 Mbps [9].

Gnanasekaran et al. [17] propose an architecture for 1024-bit RSA the uses 10% of the available LUTs and 2% of the available flip-flops on a Nexys4 device. This design encrypts data in 16-bit blocks using Montgomery exponentiation in approximately 19.25 μs. They further estimate that their design has a total on-chip power requirement of 0.213 W [17].

Leelavathi et al. [20] discuss two proposed hardware architectures for the RSA algorithm, one using binary exponentiation and the other using Montgomery exponentiation. Both designs can operate with a clock speed of 148.534 MHz. For 128-bit data, the binary exponentiation design can perform an encryption in 33.6 ns and the Montgomery exponentiation architecture can perform an encryption in 20.198 ns. Additionally, the binary exponentiation implementation had an encryption throughput of 3802 Mbps and the Montgomery exponentiation implementation had a throughput of 6338 Mbps [20].

Saini et al. [21] investigate a 1024-bit architecture for RSA using binary exponentiation using a Virtex-5 FPGA. This highly parallelized design operates at 10.149 MHz and has a calculated throughput of 11.105 Gbps. While parallelization does provide this design with exceptionally high throughput, it also requires a total on-chip power of 1.19 W [21].

Parihar and Nakhate [22] propose an ultra-low latency Montgomery exponentiation design that is capable of operating in 1024-bit and 2048-bit modes. For

the 1024-bit key sizes, their design can encrypt data in 850 ns with a throughput rate of 1204.7 Mbps. For 2048-bit keys, this design takes 1.88 μs to perform an encryption with a throughput of 1089.4 Mbps [22].

Varma and Sarawadekar [23] implemented a Montgomery multiplication design with low area utilization and low latency. This design requires 3.36% of the available LUTs and 0.2% of the available flip-flops on a Kintex UltraScale+ FPGA. Additionally, this design performs encryptions on 64-bit inputs in 7.061ns [23].

Xiao et al. [24] propose a high-throughput Montgomery modular multiplier architecture for RSA systems at 256, 512, and 1024-bit key lengths. For 256-bit keys, their design operates at 285.7 MHz, performs an encryption in 165 ns, and has a throughput of 24899.7 Mbps. At 512-bit key-lengths, their design operates at 285.7 MHz, performs an encryption in 588 ns, and has a throughput of 13931.9 Mbps. Finally, for 1024-bit data this design operates at the same frequency as the previous two key sizes, with a latency of 1.208 μs, and a throughput of 13562.9 Mbps [24].

CHAPTER IV

HARDWARE DESIGN

4.1 Repeated Modular Multiplication

To optimize the repeated modular multiplication operation, a proprietary modular multiplication hardware block, shown in figure 6, was designed.

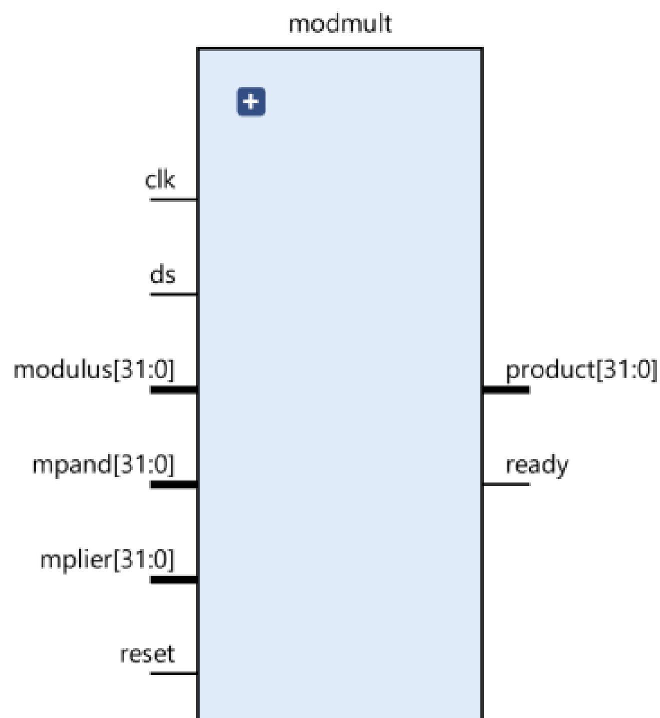

Figure 6 Repeated Modular Multiplication Multiplier Block

This block takes six total inputs, three which are single bit logic inputs and three that are bus inputs. The clk pin is tied to the system clock, the ds pin is an active-high enable, and the reset pin is an active-high reset that will clear the internal registers of the multiplier. The modulus bus is connected to the modulus input buffer and receives the

modulus value, N. The mpand and mplier busses both receive the message value M for the first multiplication, and for every successive multiplication the mpand bus received the value of the previous multiplication. The product buffer outputs the result of the multiplication of the mpand and mplier input buffers with respect to the modulus value N. Finally, the ready pin is used as a flag to signal that a multiplication has been completed and the block is ready to load the next set of values. It is important to note that, while figure 6 shows that the buses are all configured to hold 32-bit values, the hardware block has been configured such that these bus lengths will change to reflect the key size that the system has been synthesized for.

As the number of bits used in the keys increases, the number of successive multiplications that must be performed increases exponentially. To prevent overutilization of logic resources, this design was pipelined to only require two multiplication blocks at all key sizes: one for computing the initial product and a second one for computing the subsequent products.

4.2 Binary Exponentiation

For the second RSA implementation, binary exponentiation, also known as the left-to-right square and multiply method, was used for modular exponentiation. This implementation used two of the multiplication blocks shown in figure 6, one for squaring inputs and the other for multiplying inputs. The appropriate multiplication block is selected using a series of cascaded multiplexers which selectively feed input

values to each of the multipliers depending on if the current binary digit of the key is a 1 or 0.

The binary exponentiation hardware was optimized for area efficiency using pipelining to prevent overutilization at higher key sizes. Additionally, while this implementation requires the same amount of multiplication units as the repeated modular multiplication hardware, the amount of peripheral hardware required to index through the digits of the key and check the values increases the overall area utilization of the design.

4.3 Montgomery Exponentiation

The third RSA implementation was designed to use a combination of Montgomery multipliers and binary exponentiation known as Montgomery Exponentiation. This design, like the previous two, has been pipelined to prevent overutilization and requires only two multiplication units. The block design for the multiplication unit is shown below in figure 7.

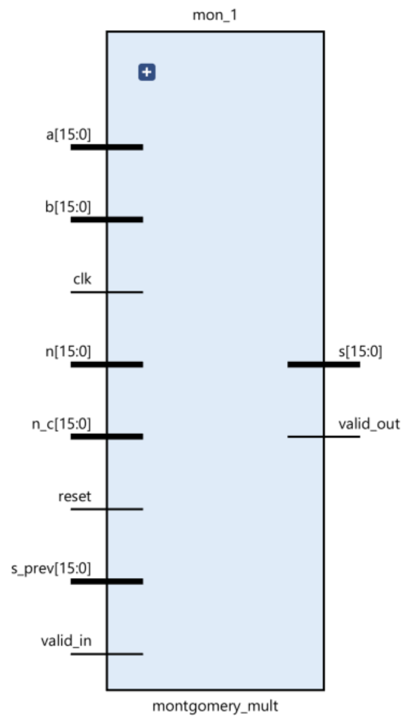

Figure 7 Montgomery Multiplier Block

There are two instances of the Montgomery multiplier used in this design. The first instance of the block is used for the squaring operation. Bus a receives Montgomery form of the number that will be squared, and bus b receives the constant R that is used for Montgomery reduction. Busses n and n_c are tied to the modulus value N and N', respectively. Finally, the s_prev bus is tied to ground. In the second instance of this block, s_prev is tied to the result of the squaring block instead of ground and bus a is connected to the multiplicand value M.

The next hardware unit used in the implementation of Montgomery exponentiation is shown in figure 8.

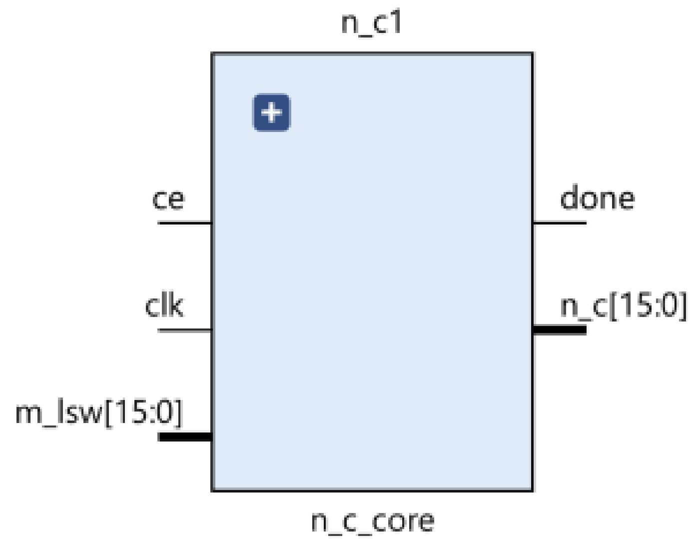

Figure 8 Modular Inverse Core Block

Shown above, the n_c_core is used to compute the modular inverse of the RSA modulus N. Its inputs are a clock enable and clock signal, as well as a bus carrying the RSA modulus number, N. This block outputs the modular inverse of N, N', and a flag signaling completion.

The final three hardware modules used for Montgomery exponentiation are two distributed memory generators and a first-in-first-out (FIFO) generator. The block for the distributed memory generator is shown in figure 9.

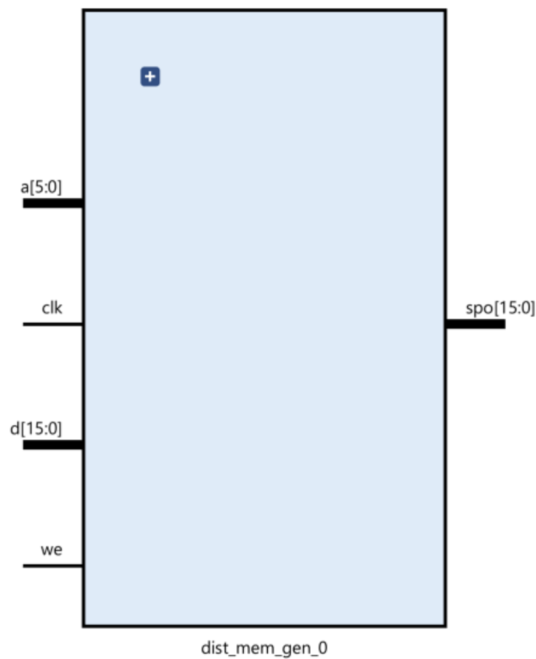

Figure 9 Distributed Memory Generator Block

The distributed memory generator block is a predefined hardware module in Vivado that is used to generate memory. This block has been configured to generate single port RAM for values of the exponent and modulus values.

The final hardware block for the FIFO generator is shown below in figure 10.

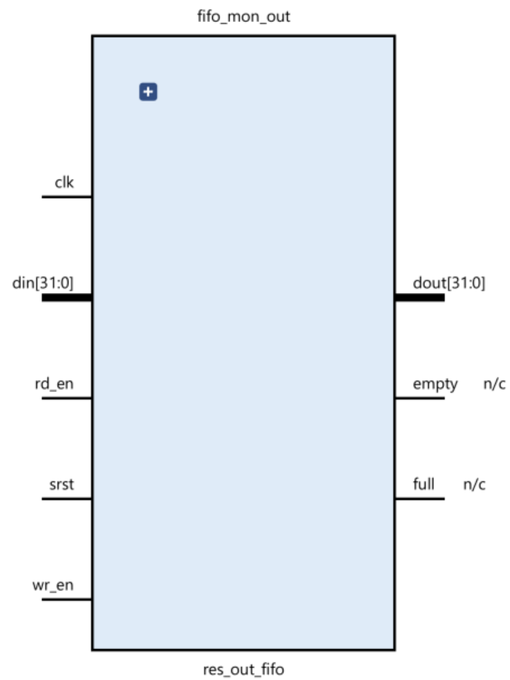

Figure 10 FIFO Generator Block

The FIFO block is used to create a stack of memory for holding the result of the previous round of the square and multiply process. By storing these values in a FIFO register, the process of squaring and multiplying can be parallelized to improve throughput and decrease latency.

CHAPTER V

RESULTS

5.1 Frequency

Table 3 shows the maximum clock frequency for each of the three hardware designs.

Table 3 Maximum Clock Frequency

Implementation	Maximum Clock Frequency
Repeated Modular Multiplication	11.125 MHz
Binary Exponentiation	28.387 MHz
Montgomery Exponentiation	21.512 MHz

The maximum clock speed for each of the designs was found by reviewing the setup, hold, and pulse width slack at a clock speed of 10 MHz. It was discovered that for all three implementations, the critical timing element was the hold slack. The maximum clock frequency was then found by adjusting the clock frequency until the worst hold slack was exactly zero seconds, signifying that the design was capable of exactly meeting the critical path timing constraints at the adjusted frequency.

5.2 Utilization

Figure 11 shows the LUT utilization for each of the three designs.

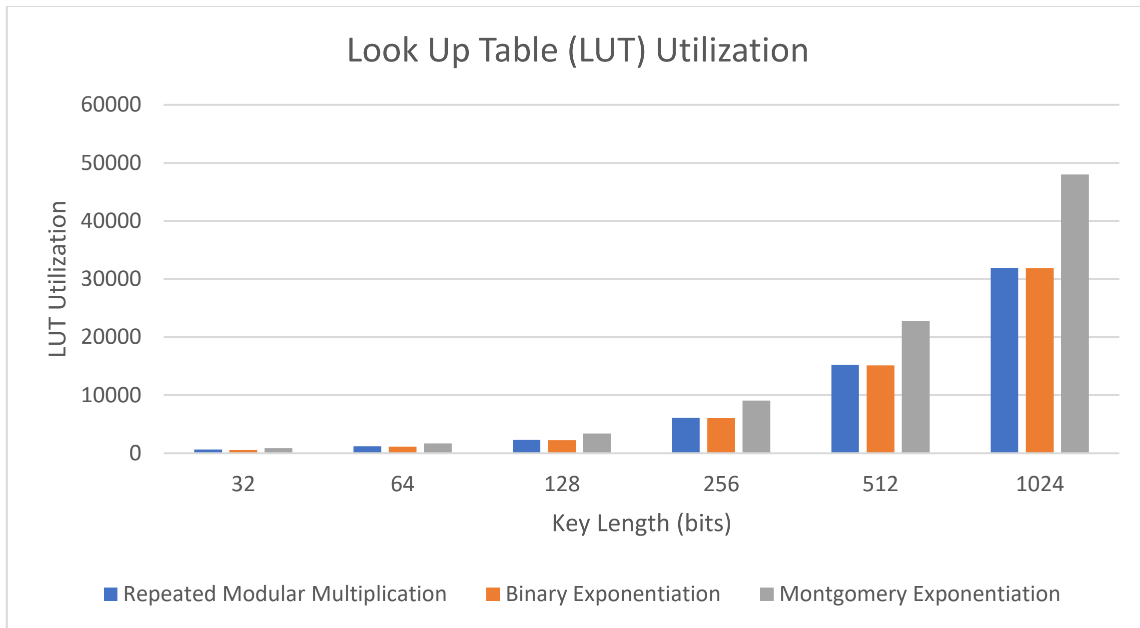

Figure 11 LUT Utilization at Varying Key Sizes

At all key sizes, the repeated modular multiplication hardware and the binary exponentiation hardware required a similar number of LUTs, while the Montgomery exponentiation hardware required approximately 50% more LUTs. The raw data for LUT utilization is shown in Table 4.

Table 4 LUT Utilization Data

Implementation	32-bit	64-bit	128-bit	256-bit	512-bit	1024-bit
Repeated Modular Multiplication	650	1220	2330	6111	15238	31946
Binary Exponentiation	566	1142	2252	6041	15136	31870
Montgomery Exponentiation	862	1703	3385	9083	22766	47991

The FF utilization for the three hardware implementations is shown in figure 12.

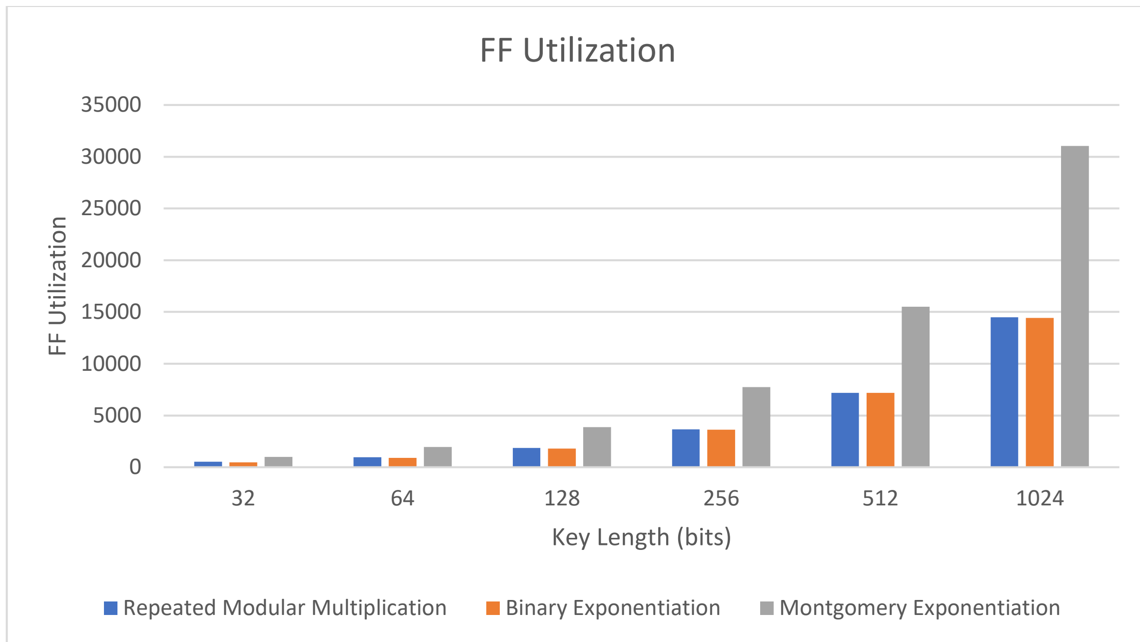

Figure 12 FF Utilization at Varying Key Sizes

The FF utilization requirements for each of the designs displayed similar trends to the LUT utilization. The repeated modular multiplication and binary exponentiation architectures both required a similar number of FFs at each key size, while the Montgomery exponentiation architecture required slightly more than double the amount of FFs as the other two architectures. The raw data for FF utilization is shown in Table 5.

Table 5 FF Utilization Data

Implementation	32-bit	64-bit	128-bit	256-bit	512-bit	1024-bit
Repeated Modular Multiplication	523	971	1867	3667	7187	14498
Binary Exponentiation	459	907	1803	3611	7195	14424
Montgomery Exponentiation	988	1951	3864	7752	15504	31035

5.3 Latency and Throughput

The measured latency and calculated throughput data for the RSA implementation using repeated modular multiplication is shown below in Table 6.

Table 6 Latency and Throughput Data for Repeated Modular Multiplication

	Key Size					
Repeated Modular Multiplication	32	64	128	256	512	1024
Clock Cycles	69,476	256,403,772	495,815,923	754,370,177	1,201,408,492	2,188,254,195
Latency	6.958 ms	25.64 s	49.581 s	75.437 s	120.141 s	218.825 s
Frequency	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz
Throughput (bits/second)	4599.023	2.496	2.582	3.394	4.262	4.680

This latency for this architecture displayed high variability based on the selected public or private key. As a result of the design, repeated multiplications must be performed a number of times equal to the value of the key. This causes the number of required clock cycles to perform an encryption or decryption to diverge exponentially as the number of bits in the key increases.

The calculated throughput shown in Table 6 also displays high variability based on the key values. At sufficiently low key values, the repeated modular multiplication architecture can provide data throughput rates up to 4.599 kilobits per second. However,

at higher key values in the 1024-bit range, the throughput rate converges to a value of 0 bits per second.

The latency and throughput data for the binary exponentiation hardware implementation is shown below in Table 7.

Table 7 Latency and Throughput Data for Binary Exponentiation

	Key Size					
Binary Exponentiation	32	64	128	256	512	1024
Clock Cycles	453	891	1777	3477	6949	13844
Latency	45.3 µs	89.1 µs	178 µs	348 µs	695 µs	1.384 ms
Frequency	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz
Throughput (kilobits/second)	706.401	718.294	719.101	735.632	736.690	739.884

As a result of the key indexing process used to compute modular exponentiation in this architecture, the measured latency displayed far less sensitivity to the selected public or private key than the repeated modular multiplication architecture. Since this design requires far fewer clock cycles to perform a complete encryption or decryption at higher key lengths, it is approximately 158,000 times faster than the repeated modular multiplication architecture.

The calculated throughput for this architecture, shown in Table 7, showed little variability at different key sizes and remained between 706 kilobits per second and 739 kilobits per second. This can be attributed to the direct linear correlation that is displayed between the bits in the key size and the required number of clock cycles required to perform an encryption or decryption.

Table 8 shows the minimum, maximum, and average latency for the binary exponentiation hardware implementation at six key sizes. The minimum and maximum

latency were found by measuring the execution time at each key size using a key consisting of all zeroes or all ones, respectively.

Table 8 Minimum, Maximum, and Average Latency for Binary Exponentiation

Binary Exponentiation	32-bits	64-bits	128-bits	256-bits	512-bits	1024-bits
Minimum	38.8 µs	80.3 µs	162.8 µs	328.5 µs	661.1 µs	1.326 ms
Maximum	61.8 µs	126.2 µs	255 µs	511.2 µs	1.027 ms	2.059 ms
Average	52.253 µs	100.165 µs	199.648 µs	408.715 µs	806.07 µs	1.747 ms

In order to calculate the average latency for each key size, forty latency measurements were taken using randomly generated encryption keys. The statistical average was calculated using equation (9).

$$\bar{x} = \frac{\sum_{i=1}^{40} x_i}{40} \quad (9)$$

As shown above in Table 8, the binary exponentiation implementation required between 38.8 microseconds and 61.8 microseconds to complete an encryption/decryption for 32-bit key sizes, with an average time for completion of 52.253 microseconds. At 1024-bit key sizes, this design took a minimum of 1.326 milliseconds to complete an encryption/decryption, and a maximum of 2.059

milliseconds. The average time to complete a single encryption/decryption for 1024-bit key sizes was 1.747 milliseconds.

The latency and throughput data for the Montgomery exponentiation hardware is shown below in Table 9.

Table 9 Latency and Throughput Data for Montgomery Exponentiation

Montgomery Exponentiation	32	64	128	256	512	1024
Clock Cycles	101	198	384	732	1441	2782
Latency	10.1 µs	19.8 µs	38.4 µs	73.2 µs	144 µs	278 µs
Frequency	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz
Throughput (megabits/second)	3.168	3.232	3.333	3.497	3.555	3.683

The Montgomery exponentiation hardware takes advantage of the same key indexing process used by the binary exponentiation hardware, with the addition of Montgomery form numbers allowing for more efficient modular reduction. The use of Montgomery reduction methods makes this hardware capable of performing an encryption or decryption using 1024-bit keys 4.97 times faster than the binary exponentiation hardware.

The calculated throughput for this architecture, shown in Table 8, displays low sensitivity to the selected keys and remained between 3.168 and 3.683 megabits per second at all key sizes.

Table 10 shows the minimum, maximum, and average latency for the Montgomery exponentiation hardware. The minimum and maximum latency were found

by measuring the execution time at each key size using a key consisting of all zeroes or all ones, respectively.

Table 10 Minimum, Maximum, and Average Latency for Montgomery Exponentiation

Montgomery Exponentiation	32-btis	64-bits	128-bits	256-bits	512-bits	1024-bits
Minimum	7.2 µs	14.8 µs	30.6 µs	63.1 µs	129.6 µs	262.5 µs
Maximum	13.3 µs	27.4 µs	55.1 µs	112.3 µs	226 µs	453.7 µs
Average	10.748 µs	19.936 µs	40.995 µs	85.14 µs	168.933 µs	363.253 µs

In order to calculate the average latency at each key size, forty latency measurements were taken using different encryption keys. The average was calculated using equation (9). The Montgomery exponentiation hardware took a minimum of 7.2 microseconds, and a maximum of 13.3 microseconds, to complete an encryption/decryption for 32-bit keys. At this key size, the average latency was calculated to be 10.748 microseconds. For 1024-bit keys, this hardware requires between 262.5 microseconds and 453.7 microseconds to perform an encryption/decryption. The average latency for the 1024-bit Montgomery exponentiation hardware was 363.253 microseconds.

5.4 Power and Energy

Figure 13 shows a comparison of the total power consumption for the three hardware designs across six different key sizes.

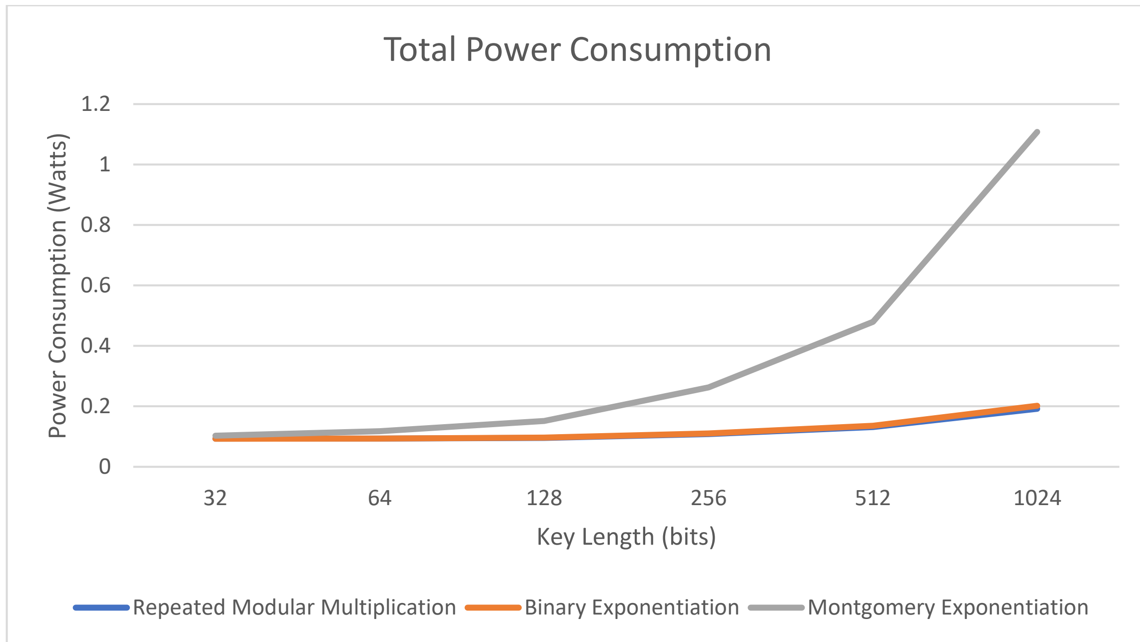

Figure 13 Total Power Consumption

At each of the six key sizes, the repeated modular multiplication hardware and binary exponentiation hardware consumed similar amounts of power, and only began to diverge slightly at the 1024-bit key length. The Montgomery exponentiation hardware, however, consumed more power at all key sizes, with the total power consumed at larger key sizes diverging rapidly from the other two architectures.

The power consumption breakdown data for the repeated modular multiplication hardware is shown below in Table 11.

Table 11 Repeated Modular Multiplication Power Consumption Data

	Key Length (bits)					
Repeated Modular Multiplication	32	64	128	256	512	1024
Static Power (W)	0.091	0.091	0.091	0.091	0.091	0.091
Dynamic Power (W)	0.002	0.003	0.005	0.017	0.041	0.101
Total Power (W)	0.093	0.094	0.096	0.108	0.132	0.192

The static power consumed by this design remained constant at 0.091 W for all key sizes tested. The portion of total power accounted for by the dynamic power consumption increased from 2.15% of the total power at 32-bit key lengths, to 52.6% of the total power at 1024-bit key lengths.

The power consumption breakdown for the binary exponentiation implementation is shown in Table 12.

Table 12 Binary Exponentiation Power Consumption Data

	Key Length (bits)					
Binary Exponentiation	32	64	128	256	512	1024
Static Power (W)	0.091	0.091	0.091	0.091	0.091	0.091
Dynamic Power (W)	0.002	0.003	0.006	0.019	0.045	0.11
Total Power (W)	0.093	0.094	0.097	0.11	0.136	0.202

The binary exponentiation hardware had a static power consumption of 0.091 W for all key sizes tested. The dynamic power consumption for this design displayed similarities to the results shown in Table 9. However, at all key sizes above 64-bits, the binary exponentiation hardware consumed slightly more dynamic power than the

repeated modular multiplication hardware. For 1024-bit keys, the dynamic power consumed by this design accounted for 54.46% of the design's total power consumption.

The power consumption breakdown for the Montgomery exponentiation design is shown below in Table 13.

Table 13 Montgomery Exponentiation Power Consumption Data

	Key Length (bits)					
Montgomery Exponentiation	32	64	128	256	512	1024
Static Power (W)	0.091	0.091	0.091	0.091	0.091	0.091
Dynamic Power (W)	0.012	0.027	0.061	0.172	0.389	1.017
Total Power (W)	0.103	0.118	0.152	0.263	0.48	1.108

This design is consistent with the previous two discussed architectures and requires a static power consumption of 0.091 W for all evaluated key sizes. The Montgomery exponentiation design's dynamic power consumption, however, far exceeded the two other designs at all key sizes. For 32-bit keys, the dynamic power accounted for 11.65% of the design's total power consumption. At a key length of 1024-bits, the dynamic power consumption for this architecture had increased to 91.79% of the total power consumption.

Figure 14 shows the comparison of total energy consumption for the binary exponentiation hardware and Montgomery exponentiation hardware.

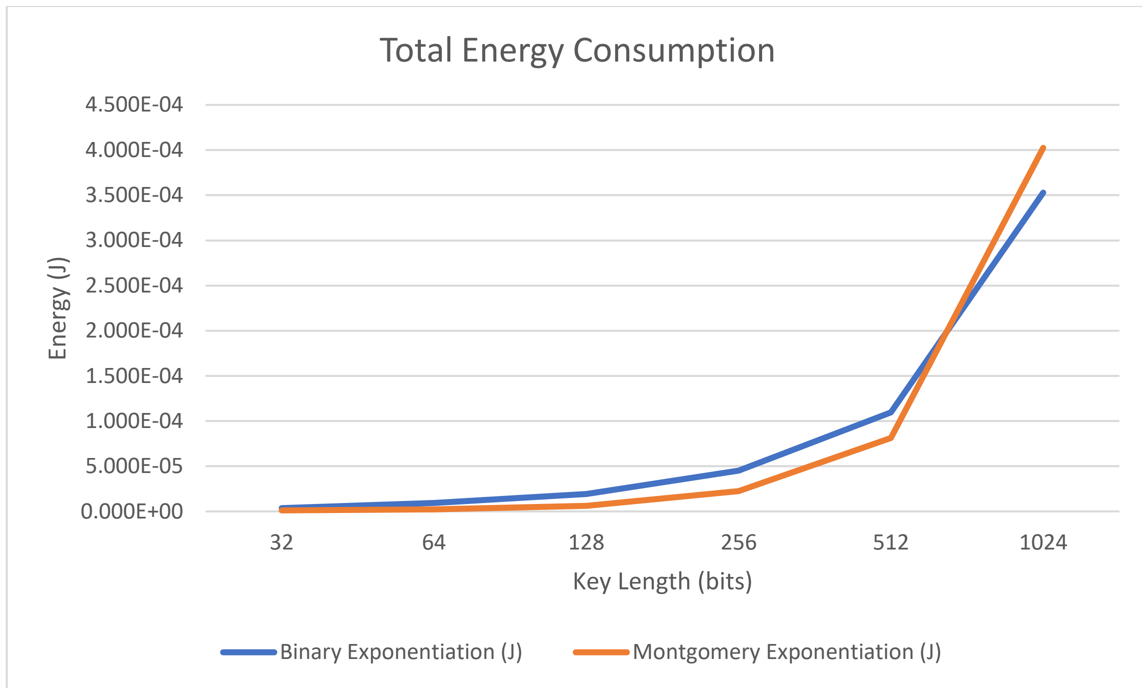

Figure 14 Total Energy Consumption

The energy consumption for these two implementations was calculated according to equation (10), using the power data found in Tables 12 and 13, and the average latency calculated and shown in Tables 8 and 10.

$$Energy = Power \times Time \tag{10}$$

Due to the latency of the repeated modular multiplication hardware being highly variable based on the selected RSA key, a reliable calculation for energy consumption was not possible. Figure 14 shows that, despite the Montgomery exponentiation hardware having higher power consumption at all key lengths, its faster execution time results in lower energy consumption at all key sizes expect for 1024-bits, where it consumes 14% more energy than the binary exponentiation hardware. The energy

consumption data for the binary exponentiation and Montgomery exponentiation implementations is shown below in Table 14.

Table 14 Total Energy Consumption

	Key Length (bits)					
Total Energy	32	64	128	256	512	1024
Binary Exponentiation (J)	3.608E-06	9.416E-06	1.937E-05	4.496E-05	1.096E-04	3.529E-04
Montgomery Exponentiation (J)	1.107E-06	2.352E-06	6.231E-06	2.239E-05	8.109E-05	4.025E-04

CHAPTER VI

ANALYSIS

6.1 Strengths and Weaknesses

Of the three implementations, the binary exponentiation hardware demonstrated the highest possible clock frequency at 28.387 MHz, while the Montgomery exponentiation implementation and repeated modular multiplication design were only capable of operating at a maximum clock frequency of 21.512 MHz and 11.125 MHz, respectively.

The repeated modular multiplication and binary exponentiation designs both displayed similarly low area utilization. At 1024-bits, the repeated modular multiplication design used 49.68% of the available LUTs and 11.43% of the available FFs on the Artix-7, while the binary exponentiation design used 49.56% of the available LUTs and 11.37% of the available FFs. The Montgomery Exponentiation design, however, required 74.64% of the available LUTs and 24.48% of the available FFs on the Artix-7 to perform encryptions and decryptions at a key length of 1024-bits. Since the repeated modular multiplication and binary exponentiation required less than 50% of the available FPGA resources, up to two instances of the designs could be simultaneously loaded onto the same device and perform parallel to one another. Conversely, the Montgomery exponentiation design, which required 74.64% of the available LUTs, uses far too much space to accommodate another parallel process on the same Artix-7 device.

The Montgomery exponentiation architecture exhibited the lowest overall latency of the three designs. At a 1024-bit key length, this design could perform an encryption in 278 microseconds and had a data throughput rate of 3.683 megabits per second. In comparison, the binary exponentiation architecture required 1.348 milliseconds to complete a full encryption at 1024-bit key lengths and had a throughput rate of 739.844 kilobits per second. The latency and throughput of the final implementation using repeated modular multiplication displayed incredibly high sensitivity to the selected public or private key. At sufficiently high key values in the 1024-bit range, this design takes indeterminably long to perform an encryption and its throughput approaches zero bits per second. This indicates that this design is not suitable for use at higher key values and is not suitable for use in providing data security to CPS.

The repeated modular multiplication implementation displayed the lowest total power consumption, requiring 0.192 watts of power when performing operations on 1024-bit data. At this same key size, the binary exponentiation implementation consumed slightly more total power, 0.202 watts, while the Montgomery exponentiation implementation required significantly more power, totaling 1.108 watts.

The Montgomery Exponentiation implementation, despite having a higher power requirement, consumed less energy than the binary exponentiation implementation at all key sizes except 1024-bits. For 512-bit keys, the two implementations had nearly identical energy consumption, with the Montgomery exponentiation architecture consuming slightly less energy overall. At the tested key lengths below 512-bits, The

Montgomery exponentiation architecture has been shown to consume as little as 25% of the energy of the binary exponentiation implementation.

6.2 Comparison to Embedded System Performance

Of the three hardware implementations of the RSA algorithm, only the binary exponentiation design could exceed the full frequency range of the embedded system being used for comparison, the MSP430, which can operate at several modes up to 25 MHz. The Montgomery exponentiation implementation, which is capable of running at a maximum frequency of 21.512 MHz, can meet all of the operating modes of the MSP430, with the exception of the highest operating mode of 25 MHz. The final implementation using repeated modular multiplication had a maximum clock frequency of 11.125 MHz, which is only capable of meeting the lower operating modes of the MSP430 and cannot match the higher MSP430 operating modes of 16 MHZ and 25 MHz.

At a key length of 1024-bits, both the binary exponentiation and Montgomery exponentiation implementations exceeded the latency and throughput of the 1024-bit RSA running on an MSP430 shown in [16]. Of these two implementations, the Montgomery exponentiation design exhibited significantly lower latency and higher throughput than the binary exponentiation hardware and was capable of encrypting 1024-bits of data 169.01 times faster than an embedded system running the RSA algorithm. As discussed previously, the repeated modular multiplication implementation was not capable of meeting the latency benchmark set by the MSP430.

Of the three implementations, the repeated modular multiplication and binary exponentiation implementations both displayed similarly low total power consumption, consuming 0.192 and 0.202 watts, respectively. For 1024-bit data, the Montgomery exponentiation design consumed 1.108 watts of power. All three of these designs, however, fail to meet the benchmark set by the MSP430, which consumes as little as 3.9 μW of power in low-power mode and 33 mW of power in active mode.

Based on these results, the repeated modular multiplication implementation is not sufficient for providing data security to CPS in place of an embedded system for any application. While it does have the lowest power requirement of the three designs, its timing deficiencies and overall sensitivity to the selected public or private key render it unsuitable for use in CPS.

The Montgomery Exponentiation implementation displayed the lowest latency and highest throughput of the three proposed designs, and far surpassed the timing standard of the MSP430. However, the increased power requirement of this design indicates that it is only suitable for use in applications where ultra-low latency and high throughput are prioritized over power efficiency.

The binary exponentiation implementation shows the most promise as an alternative to embedded systems for providing data security to CPS. This design surpassed the timing standard of the MSP430, while maintaining significantly lower power requirements than the Montgomery exponentiation design. However, the power consumption of this implementation is still significantly higher than the power

consumption of an MSP430 in active mode, making it unsuitable for applications with strict power considerations.

6.3 Comparison to Other Hardware Implementations

This section will compare the data collected for the binary exponentiation and Montgomery Modular implementations with other hardware based implementations that were found in published literature.

The binary exponentiation architecture that was designed as a part of this research had a maximum clock frequency of 28.287 MHz, measured latency of 178 us at 128-bit key lengths, calculated throughput of 719.101 kbps and 128-bit key lengths and 739.884 kbps at 1024-bit key lengths, and a total on-chip power requirement of 0.192 W at 1024-bit key lengths. Leelavathi et al. [20] and Saini et al. [21] proposed binary exponentiation architectures for hardware based RSA. The design published by Leelavathi et al had a maximum clock frequency of 148.534 MHz, a latency of 33.67 ns, and a throughput of 3802 Mbps for 128-bit keys [20]. This design is significantly faster than the binary exponentiation architecture designed for this research in all reported metrics. The design shown in the work of Saini et al. had a maximum reported frequency of 10.149 MHz, a calculated throughput of 11.1.05 Gbps, and a total on-chip power requirement of 1.19 W [21]. This implementation's dramatically higher throughput rate is the result of the highly parallelized nature of the architecture. This architecture was intentionally parallelized to require the least number of clock cycles to complete an encryption or decryption. While it is successful in this regard, its power requirement is

6.197 times greater than the architecture designed during this research, making the design of Saini et al. infeasible for use in CPS.

Of the Montgomery multiplication based RSA designs discussed previously, Xiao et al. [24] had the highest clock frequency at 285.7 MHz, notably higher than the design created by this research, which had a maximum frequency of 21.512 MHz. Parihar and Nakhate [22] had the lowest 1024-bit latency of 850 ns, while Xiao et al. [24] achieved the fastest designs for 256-bit and 512-bit, with times of 165 ns and 588 ns, respectively. These designs are significantly faster than the Montgomery exponentiation architecture designed as a part of this research, which had an average latency of 363.253 μs at 1024-bits, 168.933 μs at 512-bits, and 85.14 μs at 256-bits. Sahu and Pradhan [7] displayed the fastest 32-bit Montgomery modular implementation, performing an encryption with 9.895 ns of latency, nearly an entire order of magnitude faster than the minimum 32-bit latency shown in Table 10, 7.2 μs. Varma and Sarawadekar [23] proposed a 64-bit architecture capable of executing an encryption in 7.061 ns, which is nearly 2000 times faster than the minimum 64-bit latency measured in Table 10, 14.8 μs. The design of Xiao et al. [24] had the highest throughput rate at 256, 512, and 1024-bit sizes, with measured rates of 24899.7 Mbps, 13931.9 Mbps, and 13562.9 Mbps, respectively. Comparatively, Table 9 shows that this research's Montgomery exponentiation architecture had a throughput rate of 3.497 Mbps at 256-bit key lengths, 3.555 Mbps and 512-bit key lengths, and 3.683 Mbps for 1024-bit key sizes. The measurements from Xiao et al. are 7120.3, 3918.95, and 3682.57 times higher for 256-bit, 512-bit, and 1024-bit key sizes. However, it should be noted that as the key

length increased, the throughput rate of Xiao et al. fell while the throughput rate of this research's design increased, marginally. The only Montgomery exponentiation design that disclosed power consumption was Gnanasekaran et al. [17], whose design required 0.213 W at 1024-bit key sizes. In comparison, Table 13 shows that the power consumption of this research's Montgomery exponentiation implementation was 1.108 W for 1024-bit key lengths, 5.202 times greater than the design of Gnanasekaran et al.

CHAPTER VII

CONCLUSION

This work proposes three hardware implementations of the RSA algorithm with the potential for providing data security to CPS. The three implementations were evaluated according to their maximum clock frequency, area utilization, latency, throughput, and power consumption. The results of this evaluation were compared to the performance of an embedded system, the MSP430, running the RSA algorithm at 1024-bit key lengths. This comparison shows that the RSA implementation using binary exponentiation has the greatest potential for use in providing data security to CPS, as it provides lower latency and higher throughput than the MSP430, while still consuming sufficiently low amounts of power. However, for applications with incredibly strict power constraints, an embedded system running the RSA algorithm is still more suitable. The Montgomery exponentiation implementation is suitable for providing data security to CPS only in specific applications where ultra-low latency and high throughput need to be prioritized over minimized power consumption. The third RSA implementation, which used repeated modular multiplication to compute modular exponents, is not suitable for use in any CPS application, as the latency of the design exhibited dramatic sensitivity to the selected keys and could not meet the timing standard set by the MSP430.

7.1 Future Works

As this work has shown, public-key encryption methods have the great potential in providing data security to CPS. However, there is still a large amount of work that can be done to improve these implementations by lowering their power usage, decreasing the amount of time necessary to perform an encryption or decryption, and increasing the maximum clock frequency and throughput for each of the designs. Each of these improvements would further lend to the potential that hardware based implementation of the RSA algorithm have in providing data security to CPS. Furthermore, a similar study may be conducted on symmetric key encryption standards to assess their viability for this same purpose. That study may be compared and contrasted with this one in order to assess the most effective method available for providing data security to CPS. Finally, a hardware based hybrid cryptosystem could be developed that utilizes the best approaches from both asymmetric and symmetric cryptography and provides robust security for all aspects of CPS.

REFERENCES

[1] M. A. Faruque, F. Regazzoni, and M. Pajic, “Design methodologies for securing cyber-physical systems,” *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015.

[2] P. Ramesh, “Accelerating RSA public key cryptography via hardware acceleration,” *ScholarWorks@UMass Amherst*, Feb-2020. [Online]. Available: https://scholarworks.umass.edu/masters_theses_2/887/. [Accessed: 13-May-2022].

[3] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, “Energy analysis of public-key cryptography for Wireless Sensor Networks,” *Third IEEE International Conference on Pervasive Computing and Communications*, 2005.

[4] S. S. Almusallam, “Power Consumption in the Embedded System,” *International Journal of Emerging Science and Engineering*, vol. 3, no. 8, Jun. 2015.

[5] “MSP430F552x, MSP430F551x Mixed-Signal Microcontrollers datasheet (Rev. P),” *Texas Instruments*, 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/msp430f5529.pdf. [Accessed: 28-Sep-2022].

[6] R. Duarte, X. Niu, and C. Liu, “RSA cryptography acceleration for embedded system ,” *CAMEL: Computer Architecture and Microprocessor Engineering Lab*, 2010.

[Online]. Available: https://camel.clarkson.edu/Pub/UCAS-6_2010_Final.pdf. [Accessed: 19-Sep-2022].

[7] S. Kumar Sahu and M. Pradhan, "FPGA implementation of RSA encryption system," *International Journal of Computer Applications*, vol. 19, no. 9, pp. 10–12, Apr. 2011.

[8] J. R. Laracy, "An RSA Co-processor Architecture Suitable for a User-Parameterized FPGA Implementation," *Journal of Information Security Research*, vol. 11, no. 2, Jun. 2020.

[9] E. Kurniasari, A. E. Putra, and N. G. Agoestien, "Implementation of the Montgomery Modular based RSA algorithm on FPGA," *2019 5th International Conference on Science and Technology (ICST)*, 2019.

[10] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr. 1985

[11] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," 1978.

[12] L. M. Noordam, "VHDL Implementation of 4096-bit RNS Montgomery Modular Exponentiation for RSA Encryption," *Delft University of Technology*, 18-Jun-2019. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:3174589c-f8a1-4da7-b2a8-79ddb22e7079/datastream/OBJ/download. [Accessed: 17-Nov-2021].

[13] A. S. Tahir, "Design and Implementation of RSA Algorithm using FPGA," International Journal of Computers & Technology, vol. 14, no. 12, pp. 6361-6367, 2015, doi: 10.24297/ijct.v14i12.1737.

[14] A. Shrivastava et al., "Time in cyber-physical systems," presented at the Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, 2016.

[15] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-Physical Systems Security—A Survey," IEEE Internet of Things Journal, vol. 4, no. 6, pp. 1802-1831, 2017, doi: 10.1109/jiot.2017.2703172.

[16] U. Gulen, A. Alkhodary, and S. Baktir, "Implementing RSA for Wireless Sensor Nodes," Sensors (Basel), vol. 19, no. 13, Jun 27 2019, doi: 10.3390/s19132864.

[17] L. Gnanasekaran, A. S. Eddin, H. El Naga, and M. El-Hadedy, "Efficient RSA Crypto Processor Using Montgomery Multiplier in FPGA," in Proceedings of the Future Technologies Conference (FTC) 2019, (Advances in Intelligent Systems and Computing, 2020, ch. Chapter 26, pp. 379-389.

[18] J. A. Yaacoub, O. Salman, H. N. Noura, N. Kaaniche, A. Chehab, and M. Malli, "Cyber-physical systems security: Limitations, issues and future trends," Microprocess Microsyst, vol. 77, p. 103201, Sep 2020, doi: 10.1016/j.micpro.2020.103201.

[19] F. Dang, L. Li, and J. Chen, "xRSA: Construct Larger Bits RSA on Low-Cost Devices," presented at the 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), 2021.

[20] G. Leelavathi, K. Shaila, and K. R. Venugopal, "Hardware performance analysis of RSA cryptosystems on FPGA for wireless sensor nodes," International Journal of Intelligent Networks, vol. 2, pp. 184-194, 2021, doi: 10.1016/j.ijin.2021.09.008.

[21] S. Saini, K. Lata, A. Sharma, and G. R. Sinha, "An FPGA implementation of the RSA algorithm using VHDL and a Xilinx system generator for image applications," in Advances in Image and Data Processing using VLSI Design, Volume 1, 2021.

[22] A. Parihar and S. Nakhate, "Low latency high throughput Montgomery modular multiplier for RSA cryptosystem," Engineering Science and Technology, an International Journal, vol. 30, 2022, doi: 10.1016/j.jestch.2021.08.002.

[23] S. M. K. Varma and K. P. Sarawadekar, "FPGA Implementation of Modular Multiplication for Cryptographic Applications," presented at the 2022 IEEE Delhi Section Conference (DELCON), 2022.

[24] H. Xiao, S. Yu, B. Cheng, and G. Liu, "FPGA-based high-throughput Montgomery modular multipliers for RSA cryptosystems," IEICE Electronics Express, vol. 19, no. 9, pp. 20220101-20220101, 2022, doi: 10.1587/elex.19.20220101.

APPENDIX A

PYTHON SCRIPT FOR GENERATING RSA KEYS

```
Import RSA


def generateKeys():
    (pubKey, privKey) = rsa.newkeys(key_size)
    with open('keys/public.pem', 'wb') as p:
        p.write(pubKey.save_pkcs1('PEM'))
     with open('keys/private.pem', 'wb') as p:
        p.write(privKey.save_pkcs1('PEM'))


def loadKeys():
    with open('keys/public.pem', 'rb') as p:
        public = rsa.PublicKey.load_pkcs1(p.read())
    with open('keys/private.pem', 'rb') as p:
        private = rsa.PrivateKey.load_pkcs1(p.read())
    return private, public


key_size = 1024
generateKeys()
private, public = loadKeys()
```

```
print('Private Key: ', private)

print('Public Key: ', public)
```