AN IMPROVED DEVICE IDENTIFIER COMPOSITION ENGINE ARCHITECTURE TO ENHANCE INTERNET OF THINGS SECURITY

NESNELERİN İNTERNETİ GÜVENLİĞİNİ ARTIRMAK İÇİN GELİŞTİRİLMİŞ BİR CİHAZ TANIMLAYICI BİLEŞİM MOTORU MİMARİSİ

YUSUF YAMAK

ASSOC. PROF. DR. MURAT AYDOS

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

June 2023

ABSTRACT

AN IMPROVED DEVICE IDENTIFIER COMPOSITION ENGINE ARCHITECTURE TO ENHANCE INTERNET OF THINGS SECURITY

Yusuf YAMAK

Master of Science, Computer Engineering

Supervisor: Assoc. Prof. Dr. Murat AYDOS

June 2023, 71 pages

The number of Internet-of-Things (IoT) devices has been increasing rapidly every year. Most of these devices have access to important personal data such as health, daily activities, location, and finance. However, these devices have security problems since they have limited processing power and memory to implement complex security measures. Therefore, they possess weak authentication mechanisms and a lack of encryption. Additionally, there are no widely accepted standards for IoT security. The Device Identifier Composition Engine (DICE) was proposed as a standard that enables adding a security layer to low-cost microcontrollers with minimal silicon overhead. However, DICE-based attestation is vulnerable to some remote attacks such as Time-Of- Check Time-Of-Use (TOCTOU) attacks as shown by previous studies. In this study, we present a novel method to address the security problems of DICE. In order to detect real-time firmware attacks, our design adds an additional security component to DICE that utilizes a hash engine performing periodic memory forensics (PMF). We implemented the enhanced DICE architecture using the open-source RISC-V platform Ibex and the Mbed TLS library for cryptographic operations. We performed the hash operations required by DICE in a hardware-based manner on a

commercial Field Programmable Gate Array (FPGA) platform rather than firmware, which is more vulnerable to attacks. Our test results demonstrate that with minimal cost using the proposed method, a standard microcontroller can detect attacks.

This thesis addresses the urgent need for enhanced security in the rapidly proliferating domain of Internet of Things (IoT) devices, by investigating vulnerabilities associated with the Device Identifier Composition Engine (DICE) and proposing a novel method to mitigate these risks. Through the design and implementation of a hardware-based hash engine that utilizes periodic memory forensics, the thesis offers an innovative solution to the firmware security flaws in DICE. This novel method significantly contributes to the existing body of literature by demonstrating a practical, implementable approach to detect and counteract potential attacks on IoT devices, thereby advancing the understanding of IoT security.

Keywords: Internet-of-Things (IoT), Security, Device Identifier Composition Engine (DICE), Periodic memory forensics (PMF), RISC-V

ÖZET

NESNELERİN İNTERNETİ GÜVENLİĞİNİ ARTIRMAK İÇİN GELİŞTİRİLMİŞ BİR CİHAZ TANIMLAYICI BİLEŞİM MOTORU MİMARİSİ

Yusuf YAMAK

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Assoc. Prof. Dr. Murat AYDOS

June 2023, 71 sayfa

İnternet Nesneleri (IoT) cihazlarının sayısı her yıl hızla artmaktadır. Bu cihazların çoğu, sağlık, günlük aktiviteler, konum ve finans gibi önemli kişisel verilere erişim sağlamaktadır. Ancak, bu cihazlar karmaşık güvenlik önlemlerini uygulamak için yetersiz işlem gücü ve belleğe sahip olduklarından güvenlik sorunları yaşamaktadır. Bu nedenle, zayıf kimlik doğrulama mekanizmalarına ve şifreleme eksikliğine sahiptirler. Ayrıca, IoT güvenliği için yaygın kabul görmüş standartlar bulunmamaktadır. Cihaz Kimlik Belirleyici Bileşim Motoru (DICE), düşük maliyetli mikrodenetleyicilere minimal silikon yükü ile güvenlik katmanı eklemeyi sağlayan bir standart olarak önerilmiştir. Bununla birlikte, DICE tabanlı doğrulama, Zaman-İçinde-Kontrol Zaman-İçinde-Kullanım (TOCTOU) saldırıları gibi bazı uzaktan saldırılara önceki çalışmalarda gösterildiği gibi açıktır. Bu çalışmada, DICE'ın güvenlik sorunlarına çözüm getirmek için yeni bir yöntem sunulmaktadır. Çalışmamızda, gerçek zamanlı yazılım saldırılarını tespit etmek için periyodik bellek taraması (PMF) gerçekleştiren bir hash motoru DICE'a ek bir güvenlik bileşeni olarak eklenmektedir. Geliştirilmiş DICE mimarisinde, kriptografik işlemler için Mbed TLS kitaplığı kullanıldı ve açık kaynaklı RISC-V platformu olan Ibex üzerinde geliştirildi. DICE için gereken

hash işlemleri, saldırılara daha açık olan yazılım tabanlıdan ziyade donanım tabanlı olarak gerçekleştirdi. Test sonuçlarımız sunulan yöntem sayesinde standart bir mikrodenetleyiciye çok az bir maliyet ile saldırıları tespit edebileceğini göstermektedir.

Bu tez, İnternet Nesneleri (IoT) cihazlarının hızla çoğalan alanında gelişmiş güvenlik ihtiyacını gidermek amacıyla Device Identifier Composition Engine (DICE) ile ilgili zafiyetleri araştırarak ve bu riskleri hafifletecek yeni bir yöntem önererek çalışmaktadır. Periyodik bellek taraması için donanım tabanlı bir hash motorunun tasarımını ve uygulanmasını içeren tez, DICE'deki firmware güvenlik açıklarına yenilikçi bir çözüm sunmaktadır. Bu yeni yöntem, IoT cihazlarına yönelik potansiyel saldırıları tespit etme ve karşı koyma konusunda pratik, uygulanabilir bir yaklaşımı göstererek mevcut literatüre önemli bir katkıda bulunmaktadır ve bu şekilde IoT güvenliği hakkındaki anlayışı ilerletmektedir.

Keywords: Internet-of-Things (IoT), Security, Device Identifier Composition Engine (DICE), Periodic memory forensics (PMF), RISC-V

ACKNOWLEDGEMENTS

The completion of this thesis would not have been possible without the unwavering support and encouragement of several important people in my life.

I express my deepest gratitude to my mentors, Prof. Dr. Süleyman Tosun and Assoc. Prof. Dr. Murat Aydos. Their constant guidance, insightful criticisms, and patient encouragement aided the writing of this thesis in innumerable ways. They have broadened my understanding of the subject and set an unparalleled example of dedication and intellectual rigor.

To my wife, Pınar, words cannot adequately express my appreciation. Her love, unwavering support, and enduring patience have been the bedrock upon which this thesis was built. Her belief in my abilities, even when I doubted myself, has been a beacon of hope and a source of strength.

To my wonderful daughter, Sare, I offer my warmest thanks. Her contagious enthusiasm and boundless curiosity have been a constant source of motivation. She has served as a reminder of the joy and wonder that learning and discovery can bring.

In conclusion, this work is as much a product of their faith, support, and love, as it is of my academic endeavors. I owe them a debt of gratitude that goes beyond the words on this page, and it is my hope that this work serves as an embodiment of their belief in me.

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

CISC	:	**C**omplex **I**nstruction **S**et **C**omputing
CDI	:	**C**ompound **D**evice **I**dentifier
DDOS	:	**D**istributed **D**enial **o**f **S**ervice
DICE	:	**D**evice **I**dentifier **C**omposition **E**ngine
DMA	:	**D**irect **M**emory **A**ccess
DLP	:	**D**iscrete **L**ogarithm **P**roblem
ECC	:	**E**lliptic **C**urve **C**Cryptography
ECDH	:	**E**lliptic **C**urve **D**iffie-**H**ellman
ECDLP	:	**E**lliptic **C**urve **D**iscrete **L**ogarithm **P**roblem
ECDSA	:	**E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
EK	:	**E**ndorsement **K**ey
FPGA	:	**F**ield **P**rogrammable **G**ate **A**rray
FSBL	:	**F**irst **S**tage **B**oot **L**oader
GPL	:	**G**eneral **P**ublic **L**icense
GPRs	:	**G**eneral-**P**urpose **R**egister**s**
IMDs	:	**I**mplantable **M**edical **D**evice**s**
IIoT	:	**I**ndustrial **I**nternet **o**f **T**hings
ISA	:	**I**nstruction **S**et **A**rchitecture
IoT	:	**I**nternet-**o**f-**T**hings
LPWANs	:	**L**ow-**P**ower **W**ide-**A**rea **N**etwork**s**
MMIO	:	**M**emory **M**apped **I**nput/**O**utput
MFP	:	**M**emory **F**orensics **P**eripheral
MPU	:	**M**emory **P**rotection **U**nit
MCUs	:	**M**icro**c**ontroller **U**nit**s**
OTP	:	**O**ne-**T**ime Programmable
OTA	:	**O**ver-**T**he-**A**ir

PMF	:	**P**eriodic **M**emory **F**orensics
PC	:	**P**rogram **C**ounter
PCRs	:	**P**latform **C**onfiguration **R**egister**s**
PFS	:	**P**erfect **F**orward **S**ecrecy
PKI	:	**P**ublic **K**ey **I**nfrastructure
ROM	:	**R**ead-**O**nly **M**emory
RISC	:	**R**educed **I**nstruction **S**et **C**omputing
RTL	:	**R**egister-**T**ransfer **L**evel
ROP	:	**R**eturn-**O**riented **P**rogramming
RIoT	:	**R**obust **I**nternet **o**f **T**hings
SoC	:	**S**ystem-**o**n-**C**hip
TOCTOU	:	**T**ime-**O**f-**C**heck **T**ime-**O**f-**U**se
TCB	:	**T**rusted **C**omputing **B**ase
TCG	:	**T**rusted **C**omputing **G**roup
TEE	:	**T**rusted **E**xecution **E**nvironment
TPM	:	**T**rusted **P**latform **M**odule
UDIs	:	**U**nique **D**evice **I**dentifier**s**
UDS	:	**U**nique **D**evice **S**ecret
UVM	:	**U**niversal **V**erification **M**ethodology
V2I	:	**V**ehicle-**T**o-**I**nfrastructure
V2V	:	**V**ehicle-**T**o-**V**ehicle

1. INTRODUCTION

The number of Internet of Things (IoT) devices is rapidly increasing, and these devices play an essential role in our daily lives. IoT devices are used in home automation [3], energy management [4], health monitoring [5], industrial control systems [6], agriculture [7], transportation [8], and many other sectors. However, the rapid growth of IoT devices poses a significant risk to their security [9]. Many of these devices become targets of attackers due to weak passwords or security vulnerabilities. This circumstance has led to the rise of threats targeting IoT devices. Attackers have created various malware to take control of IoT devices and use them for their purposes. This malware can be used for DDoS (Distributed Denial of Service) attacks, spamming, cryptocurrency mining, and other malicious activities.

For example, Mirai Botnet is an IoT botnet that emerged in 2016 [10]. This botnet used weak passwords and security vulnerabilities to take over devices and launch DDoS attacks. This attack caused various websites to crash and greatly impacted internet traffic.

Another example is the malware, BrickerBot, which took over the control of devices and caused them to break permanently [11]. These attacks rendered devices unusable and resulted in significant financial losses for their owners.

Persirai malware is another threat that has emerged for IoT devices. This malware targeted IP cameras and created a malicious botnet that could record video by taking control of the devices [12].

To address these issues, TCG has introduced a specification called DICE [13, 14], which provides a solution for adding a security layer to IoT devices at a low cost and with minimal silicon requirements. This specification relies on a mix of hardware and software security mechanisms designed to guarantee the confidentiality, integrity, and authenticity of device identifiers. DICE provides several benefits for IoT device manufacturers and users. First, it provides each IoT device with a unique and im- mutable identifier, making it resistant to spoofing and tampering. This ensures that only authorized devices can communicate with the network, protecting the network from unauthorized access and attacks. With DICE, even

if the attacker clones the device, the clone will not have the same DICE identifier as the original device, making it easily detectable. Second, DICE utilizes cryptographic techniques to ensure the security of device identification and authentication. This ensures that the device identifier is securely stored and cannot be tampered with, providing a solid defense against attacks. DICE incorporates hardware-based root-of-trust, such as a Trusted Platform Module (TPM) to provide a secure environment for cryptographic operations. However, vulnerabilities, as Hristozov et al. [15] mentioned in Updatable Device Firmware, attackers can take control of devices and even access users' sensitive data through attack techniques such as return-oriented programming. These vulnerabilities demonstrate that no matter how strong the technologies used to provide security may be, they are always susceptible to risk.

In this thesis, we present our design to address the vulnerability of DICE related to firmware security flaws. We utilize the periodic memory forensics technique to detect attacks resulting from security vulnerabilities in firmware. We added a hardware-based hash engine to perform the periodic memory forensics operation. This hash engine cannot be stopped until the next reset when operated in memory forensics mode and periodically calculates the digest of the firmware. If the digest differs from the original value, the system triggers a reset. Thus, if an attacker exploits the system, it can be detected by the memory forensics peripheral(MFP). We also added a one-time programmable memory to the system to prevent the original digest from being modified by attackers. The producer should write the public part of the key pair generated during production to this memory and store the digest in memory along with its signature by the producer. Therefore, any changes made by attackers can be easily detected. We also designed the MFP to be used in general-purpose operations. Thus, we performed the hash operations required by DICE in a hardware-based manner. We present a proof-of-concept design of our study implemented on the Ibex platform. We used the Mbed TLS library for cryptographic operations.

The proposed solution significantly contributes to the existing body of literature by offering an innovative and implementable approach to combat potential security threats to IoT devices. Moreover, it fosters a deeper understanding of IoT security while extending the practical applications of DICE. The successful realization of our design on the Ibex platform,

along with an evaluation of its effectiveness, marks a noteworthy contribution to the broader context of IoT security studies.

The key areas of focus in this thesis include a comprehensive examination of the current state of IoT security, a detailed study of RISC-V and DICE architectures, a review and analysis of existing literature in this domain, the design and implementation of our proposed solution, and a thorough evaluation of its effectiveness. While our focus is on IoT device security and does not cover all aspects of IoT technology, this concentrated approach aids in providing a significant contribution to the field of IoT security. As we continue to advance in this increasingly connected world, research like this is crucial to making our interconnected environment a safer place to live.

1.1. Scope Of The Thesis

The main objective of this thesis is to delve into the realm of IoT security, specifically examining the use of RISC-V and DICE architecture to enhance the security of IoT devices. The key issues addressed in this work include understanding the current security challenges, exploring potential solutions, and proposing a novel method to enhance IoT security.

In particular, the scope of the thesis encompasses the following areas:

1. A comprehensive study of the current state of IoT security, with a particular emphasis on identifying and understanding the main security challenges and vulnerabilities that exist in IoT devices.

2. An in-depth examination of RISC-V and DICE, two key components that are considered vital to enhancing IoT security. These components' functionalities, advantages, and potential drawbacks will be analyzed in detail.

3. A review and analysis of existing studies and literature related to IoT security, RISC-V, and DICE. This will serve to position the proposed research within the broader context of IoT security studies.

4. The design and implementation of a novel method aimed at improving the security of IoT devices. This method involves the application of an enhanced DICE architecture and memory forensics peripherals. This proposed method's methodology, implementation, and validation will be explored in depth.

   An evaluation of the proposed method, including extensive experimentation and analysis to assess its effectiveness in improving IoT security.

It is important to note that while this thesis focuses on IoT security, it does not cover all aspects of IoT technology. The scope specifically focuses on IoT device security, RISC-V, and DICE. As such, other facets of IoT, such as network-level security or application-level security, are beyond the scope of this study. This work also does not include a detailed examination of other security measures or technologies beyond those directly related to the proposed method.

By narrowing down the focus of the research to these key areas, this thesis aims to provide a significant contribution to the field of IoT security.

1.2. Contributions

In this research, we have made the following contributions to the IBEX platform to realize this design:

- A new DICE controller peripheral for controlling the DICE core.
- A memory-mapped register to store the Compound Device Identifier (CDI).
- A memory-mapped MFP that we designed.
- A one-time programmable(OTP) memory to store the public key used for signature verification.
- A reset mechanism to the pipeline to enable the MFP to reset the CPU when it detects a security breach.

- A 3-port RAM instead of 2-port RAM to the Ibex to allow access by the MFP.

We can summarise the contribution of this work as follows:

1. We present a new DICE design that can detect TOCTOU and similar attacks.
2. We give the FPGA implementation of our design that uses a RISC-V processor. We compare our design with the base DICE model. Although our design adds extra area overhead, it is comparably small when considering the overall microcontroller system.

1.3. Organization

The organization of the thesis is as follows:

- Chapter 1 introduced the motivation behind the study, the rapidly increasing number of IoT devices, and the corresponding rise in security risks. It discussed various malware, like the Mirai Botnet, BrickerBot, and Persirai, which exploit weak security measures and pose substantial threats to IoT devices. The chapter also introduces DICE, a specification created by TCG to enhance IoT device security. However, it discusses the vulnerabilities in DICE relating to firmware security flaws, which the thesis aims to address by designing a hardware-based hash engine for periodic memory forensics.
- Chapter 2 delves into various aspects of the Internet of Things (IoT), RISC-V, the Device Identifier Composition Engine (DICE), and various other components and methodologies related to IoT security. This chapter provides a broad theoretical and practical understanding of the technologies and methods that will be discussed in the following chapters.
- Chapter 3 reviews the relevant literature and studies that are related to this thesis.
- Chapter 4 describes in detail the design and implementation of our proposed solution, which is aimed at enhancing the security of IoT devices.

- Chapter 5 presents the results of the experiments conducted to evaluate the performance and effectiveness of the proposed method.
- Chapter 6 provides a conclusion to the thesis, summarizing the main findings, contributions, and potential directions for future research.

2. BACKGROUND OVERVIEW

2.1. IOT

The Internet of Things refers to a vast network of interconnected devices that collect, exchange, and analyze data, enabling smarter and more efficient decision-making. IoT technology has been integrated into a wide range of applications, from smart homes [3] and cities to industrial automation [6] and healthcare [5].

2.1.1. Foundations of IoT

The IoT ecosystem is built upon several key technologies facilitating seamless communication, data processing, and decision-making among connected devices. These technologies include:

Sensors and Actuators: Sensors are devices that detect and measure physical phenomena (e.g., temperature, humidity, pressure) and convert them into digital data. Actuators, on the other hand, perform physical actions based on the processed data. These components serve as the primary interface between the physical world and the digital realm in IoT systems.

Connectivity: IoT devices rely on various communication protocols and networking technologies to exchange data with each other and cloud-based servers. Some of the most commonly used connectivity options in IoT include Wi-Fi, Bluetooth, Zigbee, cellular networks (e.g., 4G, 5G), and Low-Power Wide-Area Networks (LPWANs) [16].

Data Processing and Analytics: IoT devices produce tremendous data that must be processed and examined to glean significant insights. Data processing can occur at the device level (edge computing) [17] or be offloaded to the cloud. Sophisticated analytic methods like

machine learning and artificial intelligence can be utilized to obtain practical insights from the amassed data.

Security and Privacy: Securing IoT devices and protecting user privacy are critical aspects of IoT systems, given the sensitive nature of the data involved. To ensure the confidentiality, integrity, and accessibility of data, it's necessary to implement strong security protocols, such as encryption, authentication, and access control [9].

2.1.2. IoT Applications:

IoT technology has been adopted across various domains, enabling a wide range of applications, including:

Smart Homes Smart homes integrate IoT devices to automate and optimize various household functions, such as lighting, temperature control, and security. This enables greater convenience, energy efficiency, and safety for homeowners.

Smart Cities: Smart cities leverage IoT technology to enhance urban services, such as traffic management, waste management, and public safety. Through the gathering and examination of data from a range of sources, smart cities have the potential to optimize the distribution of resources and enhance the general quality of life [3].

Industrial Automation: The Industrial Internet of Things involves the integration of IoT devices into industrial processes to optimize efficiency, reduce downtime, and enhance safety. This includes applications such as predictive maintenance, remote monitoring, and supply chain optimization [6].

Healthcare IoT technology has been applied in healthcare settings to enable remote patient monitoring, telemedicine, and personalized care. By gathering and examining patient information, healthcare practitioners can make more knowledgeable decisions and provide superior healthcare services [5].

2.1.3. Security Threats in the IoT Ecosystem

The extensive proliferation of IoT devices has enhanced people's quality of life and improved industry productivity. However, the increasing reliance on IoT technology has also raised significant security concerns. IoT security threats can be classified into physical, network, and application-level threats. Physical threats arise from unauthorized access to or tampering with IoT devices, leading to data theft or malfunction. Network threats encompass attacks on communication channels, such as eavesdropping, man-in-the-middle, and distributed DDoS attacks. On the other hand, application-level threats involve exploiting software or firmware vulnerabilities, leading to unauthorized access or data breaches [18].

Key Challenges in Ensuring IoT Security

1. Diversity and scale of IoT devices: The IoT ecosystem is characterized by various devices with varying capabilities, processing power, and operating systems. This diversity complicates the implementation of standardized security measures and necessitates tailored solutions for different types of devices.

2. Resource constraints: Many IoT devices are designed to be low-cost and energy-efficient, resulting in limited computing power and memory. These constraints make it challenging to implement robust security mechanisms, such as encryption and intrusion detection systems, which require significant computational resources.

3. Lack of security awareness: Many IoT device manufacturers and end-users are unaware of or underestimate the security risks associated with IoT technology. This

lack of awareness often results in inadequate security measures, leaving devices vulnerable to cyberattacks.

4. Fragmented regulatory landscape: IoT security regulations vary across different jurisdictions, making it difficult to establish a unified security framework that can be applied globally. The absence of comprehensive security standards and guidelines can lead to inconsistent security practices and hinder effective stakeholder collaboration.

5. Data privacy concerns: IoT devices generate a large amount of sensitive data, including personal and financial information. It's essential to guarantee the privacy and security of this data to sustain user trust and avoid data breaches. However, the vast amount of data generated and shared among IoT devices makes it challenging to implement effective privacy measures.

2.2. RISC-V

2.2.1. History of RISC-V

The roots of RISC-V can be traced back to the 1980s when computer scientists such as John Hennessy and David Patterson began exploring the potential benefits of a reduced instruction set for improving processing efficiency. Their efforts paved the way for the emergence of RISC architectures, prioritizing a smaller set of simple instructions over a larger set of complex instructions found in Complex Instruction Set Computing (CISC) architectures.

RISC-V emerged from the Computer Science Division at the University of California, Berkeley, in 2010. A research project led by Professors Krste Asanović and David Patterson aimed to create a free and open ISA for research and education. The project's primary goal was to foster innovation and collaboration in the computer architecture community by providing an open-source alternative to proprietary ISAs.

Recognizing the potential impact of the RISC-V ISA on the computing industry, the RISC-V Foundation was established in 2015. The foundation's mission was to manage and promote

the RISC-V ISA, ensuring its growth and long-term sustainability. In 2020, the foundation transitioned into the RISC-V International Association, a non-profit organization dedicated to promoting the global adoption and execution of the RISC-V ISA.

RISC-V's open-source nature and focus on education and research made it an attractive option for academic institutions. Universities worldwide have adopted RISC-V for teaching computer architecture and as a platform for research projects, including processor design, operating systems, and compiler optimizations. This widespread adoption in academia has led to a growing pool of expertise in RISC-V development, contributing to the architecture's continuous improvement. Over the years, RISC-V has garnered attention from various industries due to its open-source model, which allows companies to develop custom processors without licensing fees. This flexibility and cost-saving potential have led several prominent organizations to adopt RISC-V, including NVIDIA, Western Digital, and Google. As a result, RISC-V has found its way into numerous applications, such as data centers, IoT devices, and edge computing.

2.2.2. The Significance of RISC-V for the Internet of Things

The Internet of Things is characterized by a vast network of interconnected devices that collect, exchange, and analyze data. As the IoT ecosystem expands, the need for energy-efficient, versatile, and scalable processing architectures has become increasingly critical. RISC-V, an open standard Instruction Set Architecture (ISA) based on Reduced Instruction Set Computing (RISC) principles, offers a promising solution to address these challenges.

Open-Source and Scalability: One of the key benefits of RISC-V is its open-source nature, which enables companies to design custom processor implementations without incurring licensing fees. This fosters innovation and reduces costs, which is essential for IoT devices where tight budget constraints are often a significant concern. Furthermore,

RISC-V offers a scalable architecture that can be tailored to specific applications, making it suitable for the diverse range of IoT devices with varying performance requirements.

Energy Efficiency: Energy efficiency is a crucial consideration for IoT devices, which are battery-powered and require long operational lifetimes between recharges [19]. RISC-V enables energy-efficient designs by utilizing a reduced instruction set, which simplifies processor pipelines and minimizes power consumption. This results in extended battery life and reduced overall energy usage, making RISC-V an attractive option for IoT devices with strict power constraints.

Modular and Extensible: The modular design of RISC-V allows for the addition of custom extensions without affecting the base ISA. This extensibility enables developers to optimize their processors for specific IoT applications, catering to the unique requirements of different devices and use cases. This flexibility ensures that RISC-V processors can be designed to meet the specific needs of IoT devices rather than relying on generic, one-size-fits-all solutions.

2.2.3. Application Areas

Smart Homes: Smart homes are an increasingly popular application of IoT technology, with connected devices such as thermostats, lighting systems, and security cameras providing greater convenience and control for homeowners. RISC-V's energy efficiency and customizability make it well-suited for these devices, enabling designers to optimize processors for the specific needs of each application while minimizing power consumption.

Industrial Automation: Industrial IoT applications often involve complex systems that require robust, reliable, and secure processing solutions. RISC-V's scalability and extensibility make it an excellent choice for IIoT devices, allowing designers to create

processors that cater to the varied requirements of different industrial applications, such as manufacturing, logistics, and energy management.

Wearable Technology: Wearable devices, such as fitness trackers and smartwatches, present unique challenges for processor design, including strict power constraints and the need for compact, lightweight components. RISC-V's energy efficiency and modular design make it an ideal choice for wearable technology, enabling the creation of processors that balance performance with low power consumption and small form factors.

2.2.4. Technical Details of RISC-V

2.2.4.1. Base Instruction Set The RISC-V ISA comprises a modular design featuring a base instruction set and several optional extensions. The base instruction set is classified into four variants based on the address space width: RV32I (32-bit), RV64I (64-bit), RV128I (128-bit), and RV16I (16-bit, not yet formally standardized). The 32-bit base ISA (RV32I) is the foundation for the RISC-V architecture and includes 47 instructions, such as arithmetic, logical, memory access, control flow, and system instructions.

2.2.4.2. RISC-V Extensions To accommodate a diverse range of applications, RISC-V includes a set of optional extensions that can be added to the base ISA. Some of the most notable extensions are:

Standard Extensions

- M: Multiplication and division instructions for integer arithmetic
- A: Atomic memory access instructions for concurrent programming
- F: Single-precision floating-point instructions
- D: Double-precision floating-point instructions

- C: Compressed instructions for reducing code size
- V: Vector instructions for high-performance computing

Custom Extensions RISC-V's open and modular architecture allows for the development of custom extensions, enabling designers to optimize processors for specific applications. This flexibility is particularly valuable in IoT and other specialized computing domains, where unique requirements may necessitate specialized instructions.

2.2.4.3. Register File RISC-V employs a register file with 32 general-purpose registers (GPRs) labeled x0 to x31. The zeroth register (x0) is hardwired to zero, while the other registers can be used for various purposes, such as holding values, pointers, and temporary data. This register-based design enhances RISC-V's simplicity and efficiency.

2.2.4.4. Memory Model The RISC-V memory model supports both little-endian and big-endian byte orders, making it compatible with different software and hardware platforms. The model employs a flat virtual address space, enabling flexible memory management for various applications.

2.2.4.5. Privilege Levels RISC-V defines three privilege levels to control access to system resources and manage the execution of software.

Machine Level (M) The highest privilege level, the Machine level, has unrestricted access to all system resources and is responsible for managing hardware resources, configuring the system, and handling exceptions.

Supervisor Level (S) The Supervisor level is typically used by operating systems and has limited access to system resources. It is responsible for managing memory protection, handling system calls, and controlling the execution of user-level applications.

User Level (U) The User level is the least privileged level and is intended for running application software. It has restricted access to system resources and relies on the Supervisor level to manage system resources and perform privileged operations.

2.3. Device Identifier Composition Engine(DICE)

The Device Identifier Composition Engine, commonly known as DICE, is a security framework by the Trusted Computing Group (TCG) [13, 14]. Its primary function is to serve as a reliable yet light cryptographic backbone for devices that belong to the Internet of Things (IoT). As the digital realm expands, connected devices' security is becoming a top priority. IoT devices are quite susceptible to cyber threats, data theft, and unsanctioned access because of their limited resources and the potential risks connected to their use in various sectors. DICE is engineered to overcome these issues by producing unique identifiers for each device, setting up secure device identities, and facilitating safe boot processes.

2.3.1. DICE Architecture and Components

The architecture of DICE is made up of a number of critical elements and processes which collectively establish a secure foundation for IoT devices. The first among these is the Unique Device Secret (UDS), which is a device-specific confidential key. This key could be ingrained during manufacturing or generated and securely stored within the device. The UDS is fundamental to the cryptographic procedures within DICE and guarantees that every device can be uniquely identified.

Next, we have the Compound Device Identifier (CDI), which is an exclusive identifier created by merging the device's UDS with a measurement of the device's firmware. The measurement is generally carried out using a cryptographic hash function such as SHA-256. The CDI forms a cryptographic basis for secure device attestation and authentication, enabling other entities to verify the device's identity and ensure its integrity.

DICE employs a hierarchical method for authentication, which is often referred to as Layered Authentication. In this system, each layer's trust is built upon the trust established in the previous layer. This layered authentication strategy allows the implementation of comprehensive access control and secure boot processes. It typically encompasses the device manufacturer, the device itself, and the applications running on it.

Additionally, DICE relies on a hardware-based root of trust. This is a secure, tamper-proof component that is responsible for generating and safeguarding the device's secret keys. This root of trust fortifies the security and integrity of the DICE architecture by providing a secure anchor for cryptographic operations.

Lastly, attestation in DICE pertains to the procedure of validating the device's identity and integrity. This is achieved by exchanging CDIs and other relevant information, such as firmware measurement and associated metadata. Attestation enables other parties, like service providers or device owners, to trust the device and establish secure communication channels.

2.3.2. Technical Details and Implementation

DICE can be implemented using a combination of hardware and software components, depending on the specific requirements and constraints of the target IoT device. The hardware root of trust can be realized using dedicated security modules, such as Trusted Platform Modules (TPMs), or integrated directly into the device's System-on-Chip (SoC).

The secure boot process requires the storage of hash values, which can be achieved using non-volatile memory, such as flash memory or read-only memory (ROM). These stored values are used during the boot process to validate the integrity of the firmware.

The layered authentication model in DICE allows for separating duties and establishing trust boundaries between different system components. This model requires the implementation of appropriate key management and access control mechanisms to ensure that only

authorized entities can access the device's sensitive information. This layered architecture is as in Figure 2.1.

In addition to the core DICE components, the architecture can be extended with additional security features, such as secure storage for sensitive data, encrypted communication channels, and hardware-accelerated cryptographic operations. These extensions can further enhance the overall security of the IoT device and protect it against a wide range of threats.

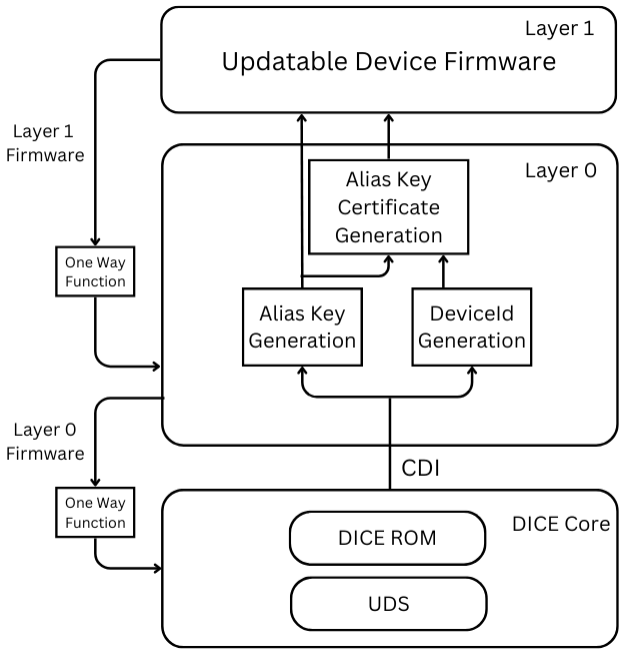

Figure 2.1 DICE Architecture

2.4. Elliptic Curve Cryptography

Elliptic Curve Cryptography, or ECC, has become a strong contender to traditional public-key cryptographic methods. ECC finds its foundation in the mathematics of elliptic curves, and its benefits include efficiency and security. These are particularly advantageous when it comes to embedded systems. Fundamentally, ECC is a public-key cryptography technique that uses the math behind elliptic curves to deliver potent encryption. The curves are delineated by a mathematical formula, and the points that lie on the curve become the bedrock of the cryptographic system. The challenge of finding the solution to the discrete

logarithm problem (DLP) within the elliptic curve group, called the Elliptic Curve Discrete Logarithm Problem (ECDLP), forms the basis for ECC's security [20].

ECC comes with several benefits when employed within embedded systems. One of the leading advantages is its efficiency, a critical factor for embedded systems as they often have restricted processing power, memory, and power supply. ECC offers the same security as conventional cryptographic methods such as RSA, but it does so with smaller key lengths. Shorter keys translate to faster computations and lower power usage, which is essential for batteries-powered devices.

ECC's merits also extend to enhanced security. The security strength of ECC is anchored in the ECDLP, which is generally considered harder to solve than the factoring problem that RSA relies on. Consequently, an elliptic curve system provides significantly higher security than an RSA equivalent for the same key length. However, as ECC can achieve equal security with shorter keys, it typically uses smaller keys than RSA for a comparable security level, further boosting its efficiency.

With the consistent expansion of Internet of Things (IoT) devices, the ability to scale up is also crucial. Thanks to ECC's minimal computational needs, it is highly scalable, a critical feature in an era with ever-increasing IoT networks.

Moreover, ECC can be applied to design protocols that provide Perfect Forward Secrecy (PFS), such as the Diffie-Hellman key exchange on elliptic curves (ECDH). PFS assures that, even if a private key is compromised, all past communications encrypted with that key will remain secure.

Given its benefits like reduced key size, lower computational demands, and enhanced security, ECC provides a persuasive solution to the cryptographic challenges faced by modern embedded systems. Thus, it is anticipated that ECC will take on a more significant role in securing embedded systems in the future.

2.4.1. Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm, or ECDSA, is a critical tool in digital communication to produce a digital signature [21]. It's a cryptographic algorithm that helps confirm the authenticity and intactness of the information. ECDSA is grounded in Elliptic Curve Cryptography (ECC) principles and the Digital Signature Algorithm (DSA).

When creating a digital signature using ECDSA, the process entails producing a pair of large numbers. These numbers originate from the signed message, the private key, and a random number generated for each signature. On the other hand, the verification process employs the signature and the signer's public key to confirm the message's authenticity.

ECDSA comes with several benefits. First and foremost, it provides a mechanism to check data integrity. In an embedded system, this capability can be incredibly beneficial to ascertain that the transmitted and received data has not been tampered with.

ECDSA signatures also offer a means to confirm the sender's identity, which is essential in embedded systems. Verifying the origin of a command or data can hinder unauthorized control or access to the system.

Another essential feature of ECDSA is non-repudiation, which ensures an entity cannot deny its actions. With ECDSA, when a party signs something, they cannot later reject that they signed it. This characteristic is especially significant in scenarios where accountability is necessary.

Moreover, like ECC, ECDSA offers these benefits more efficiently than its RSA equivalent. Considering the limited resources of embedded systems, this efficiency can make a substantial difference. Therefore, with its capability to ensure data integrity, authenticate the sender, provide non-repudiation, and deliver these benefits efficiently, ECDSA becomes an invaluable tool in the realm of digital communication and embedded systems.

2.5. SHA-256

SHA-256, or Secure Hash Algorithm 256, is a member of the Secure Hash Algorithm 2 (SHA-2) family, which is integral to many cybersecurity protocols in use today. Developed by the National Security Agency (NSA) in the United States, SHA-256 plays a critical role in ensuring the integrity and security of digital information.

SHA-256 is a cryptographic hash function that produces a 256-bit (32-byte) hash value. This is a unique representation of data, much like a digital fingerprint. It is widely used in cryptographic applications and protocols, including TLS and SSL, PGP, SSH, IPsec, and Bitcoin. Its primary role is to ensure data integrity.

The primary characteristic of any hash function, including SHA-256, is that it is deterministic. This means that the same input will always produce the same output. In contrast, even a tiny change in input data results in a drastically different output hash, a property known as the ”avalanche effect”. This makes it virtually impossible to infer the original data from the hash, which is crucial for maintaining data security.

Moreover, SHA-256 is designed to be a one-way function: it’s computationally infeasible to retrieve the original data from the hash value, making it resistant to hacking attempts. This is particularly important when storing sensitive information like passwords. Instead of storing the password itself, the system stores the SHA-256 hash. When a user enters their password, it is hashed again, and the hashes are compared. This ensures the password is never stored in a way that could be useful to a potential attacker.

However, no cryptographic hash function is completely immune to attacks, and SHA-256 is no exception. A hash collision occurs when two different inputs produce the same hash output. SHA-256, theoretically, is susceptible to a birthday attack, a type of collision attack [22]. However, the chances of such a collision are astronomically low, making this more of a theoretical concern than a practical one.

SHA-256 is also used in the process of mining in the Bitcoin network. In Bitcoin mining, the miners solve computationally difficult puzzles, which primarily involve SHA-256 hash

functions. The miner who first solves the puzzle gets the reward of newly minted bitcoins and transaction fees.

In conclusion, SHA-256 is a robust and reliable cryptographic hash function that forms the backbone of many digital security systems. Despite potential vulnerabilities, its widespread use in various applications attests to its reliability and security. As the volume of digital data grows, SHA-256's role in ensuring data integrity and security is likely to become even more vital.

2.6. Periodic Memory Forensics

While the DICE architecture provides a strong foundation for IoT device security, it does not inherently address malware detection within user applications. To enhance the DICE security model, this thesis proposes the integration of a hash engine for periodic memory forensics, which aims to detect and mitigate malware threats within user applications.

The hash engine periodically scans the device's memory, computing hash values for different memory segments and comparing them against a set of known-good values. If any discrepancies are found, the system can raise an alert and take appropriate action, such as quarantining the affected application or initiating a reboot.

This enhancement requires modifications to the existing DICE architecture and the underlying hardware, such as adding a dedicated memory forensics module and integrating the hash engine with the device's memory subsystem. The implementation of the proposed system is conducted on a modified version of the Ibex processor, an open-source RISC-V processor core.

2.7. Ibex Architecture

The Ibex processor is based on the RISC-V Instruction Set Architecture (ISA), a modular and open-source ISA designed for scalability and extensibility [23]. The RISC-V ISA supports

various base integer instruction sets and optional extensions for floating-point arithmetic, atomic operations, and other specialized instructions.

The Ibex processor implements the RV32IMC subset of the RISC-V ISA, which includes:

1. RV32I: The base integer instruction set for 32-bit processors, containing core instructions for arithmetic, logical, and control operations.
2. M: The integer multiplication and division extension, providing instructions for signed and unsigned integer multiplication and division.
3. C: The compressed instruction extension, which enables more compact code by defining 16-bit versions of commonly used RISC-V instructions. The Ibex processor also features a two-stage pipeline, which separates instruction fetch and decodes from execution, allowing for improved instruction throughput and reduced power consumption. This simple pipeline design is well-suited for resource-constrained environments and helps to minimize the processor's area footprint.

The Ibex processor platform has a variety of attractive qualities that make it a favorable choice for Internet of Things (IoT) devices and embedded systems.

One of the standout aspects of the Ibex processor is its high configurability. It can be customized to include or exclude different features such as the M and C instruction set extensions, debug support, and interrupt handling, all based on the unique application requirements it's being used for. This level of adaptability gives designers the freedom to shape the processor in a way that suits their needs best, optimizing it for power, area, or performance as needed.

Another compelling feature of the Ibex processor is its low power consumption. The processor is designed to emphasize power efficiency, utilizing methods like clock gating and power domain isolation to keep power consumption at a minimum while in operation. This is especially significant for IoT devices that operate on battery power or energy harvesting.

The Ibex processor also boasts a small area footprint. Its compact design allows easy integration into Systems-on-Chip (SoCs) and other environments with limited resources. This small footprint results from a simple pipeline, efficient instruction encoding, and careful resource sharing.

Lastly, the open-source nature of the Ibex processor gives it a unique edge. It's released under the permissive Apache 2.0 license, which permits free use, modification, and distribution of the processor core. The open-source nature encourages a broad community of developers and researchers to contribute to the project, promoting innovation and collaboration in processor design.

The Ibex processor platform's high configurability, low power consumption, small footprint, and open-source nature make it an appealing choice for IoT devices and embedded systems.

2.7.1. Technical Details and Implementation

The Ibex processor is written in SystemVerilog, a hardware description language widely used for the design and verification of digital systems. The processor can be synthesized for various target technologies, including FPGA, ASIC, and custom fabric, depending on the specific requirements of the target application.

In addition to the core processor functionality, the Ibex platform includes several optional components that can be integrated to provide additional capabilities, such as:

1. Debug Support: The Ibex processor can be configured with optional debug support, enabling developers to inspect and control the processor's operation during development and testing. This debug support complies with the RISC-V debug specification and includes hardware breakpoints, watchpoints, and single-stepping features.

2. Interrupt Controller: The Ibex processor can be configured with an optional RISC-V compliant interrupt controller, providing support for handling local and external

interrupts. This controller enables the processor to respond to asynchronous events and prioritize the handling of multiple interrupt sources.

3. Memory Interfaces: The Ibex processor includes flexible memory interfaces, allowing for the integrating of various memory types and hierarchies, such as on-chip SRAM, off-chip DRAM, and non-volatile memory. These interfaces can be tailored to the specific memory requirements of the target application, optimizing for access latency, bandwidth, or power consumption.

4. Peripheral Interfaces: The Ibex processor can be connected to a wide range of peripheral devices through standard interfaces, such as SPI, I2C, UART, and GPIO. These interfaces enable the processor to interact with sensors, actuators, and other components commonly found in IoT devices and embedded systems.

2.8. Verilator

With the increasing intricacy of digital designs, the requirement for strong and efficient verification tools is becoming vital. Verilator, a high-performance open-source simulator, has gained popularity as a preferred tool among hardware designers. Developed by Wilson Snyder, Verilator is licensed under the GNU General Public License. It translates Verilog and SystemVerilog Hardware Description Language (HDL) into C++ or SystemC code. Verilator supports the IEEE 1364-2005 Verilog standard and is a part of the IEEE 1800-2017 SystemVerilog standard. It is mainly used to verify Register-Transfer Level (RTL) designs, concentrating on simulation speed and the capability to interface with C++ or SystemC testbenches [24].

Verilator's primary benefit lies in its high-performance simulation. It transforms HDL designs into C++ or SystemC code, leveraging the compiled languages' speed to accomplish faster simulation times than traditional event-driven simulators.

As an open-source tool, Verilator allows users to access and modify its source code, promoting a community-driven development model that encourages continuous enhancement and adaptability to new design challenges.

Another crucial feature of Verilator is its compatibility with C++ and SystemC testbenches. This compatibility empowers designers to use the complete strength of these high-level languages for verification. It also allows advanced testbench techniques like constrained-random verification, functional coverage, and assertions, significantly improving verification productivity and quality.

Verilator supports multi-threading, which allows users to utilize multi-core processors to enhance simulation performance further. This feature is especially beneficial when simulating large, complex designs.

Lastly, Verilator easily integrates with other hardware design tools and verification methodologies, such as Universal Verification Methodology (UVM) and hardware acceleration platforms. This integration allows for the creation of a comprehensive and efficient verification environment.

2.9. Mbed TLS

Mbed TLS is a cryptographic library developed by ARM, available under the Apache 2.0 license. This library is written in C and provides a wide selection of cryptographic primitives and secure communication protocols [25]. Its focus is on keeping resource consumption minimal while ensuring high security, making it an excellent fit for embedded systems with limited resources. Mbed TLS supports an array of cryptographic algorithms, including symmetric and asymmetric encryption, hashing, and digital signatures, as well as secure communication protocols like SSL/TLS and DTLS.

One of the main strengths of Mbed TLS is its lightweight and modular design. It allows developers to include only the components required for their application, thus reducing

memory footprint and computational overhead. This is particularly valuable for embedded systems that have limited resources.

The platform independence of Mbed TLS, which stems from it being written in ANSI C, is another crucial feature. It enables easy porting to various platforms, ranging from microcontrollers and operating systems to bare-metal environments. This makes Mbed TLS integrate into a broad range of embedded systems.

Additionally, Mbed TLS supports an extensive variety of cryptographic algorithms and protocols. This wide-ranging support allows developers to choose the security mechanisms that best fit their application's needs.

Mbed TLS also allows developers to configure its features with a high level of granularity. This feature enables developers to tweak the library to optimize it for their specific use case, balancing their security needs with their system's resource limitations.

Finally, the active development and support of Mbed TLS by ARM ensure that the library stays current with the latest security standards and vulnerabilities. The Mbed TLS community also provides additional support and resources for developers using the library in their projects, making it an even more attractive choice.

2.9.0.1. Applications of Mbed TLS in Embedded Systems Mbed TLS is particularly well-suited for securing communication and data in embedded systems, such as:

1. IoT devices: Mbed TLS can be used to secure communication between IoT devices and cloud services, ensuring data privacy and integrity.

2. Industrial control systems: Mbed TLS can provide encryption and authentication for industrial control systems, protecting sensitive data and preventing unauthorized access.

3. Medical devices: Mbed TLS can secure communication between medical devices and data storage or processing systems, safeguarding patient information and ensuring regulatory compliance.

4. Automotive systems: Mbed TLS can be integrated into automotive systems to protect vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication, enhancing safety and privacy.

2.10. GTKWave

GTKWave, developed by Tony Bybell, is an open-source and cross-platform tool that provides a platform for users to visualize and interact with simulation outputs, aiding in the analysis and debugging of digital designs [26]. It is highly valued by digital design engineers and students as it enhances their understanding of digital circuits and systems. GTKWave supports various file formats such as VCD, LXT, FST, and GHW, making it versatile and compatible with various digital design applications.

One of GTKWave's primary functionalities is waveform viewing and navigation. It presents users with a detailed representation of signal values, transitions, and timing data through complex waveform diagrams. Users have the flexibility to zoom, pan, scroll, and customize their view to focus on particular signals or time frames, offering an in-depth perspective on their designs.

GTKWave also enables signal manipulation and analysis. Users can modify signals and perform logical and arithmetic operations, creating new signals from existing waveforms. The tool also allows users to search for specific signal patterns or transitions, making pinpointing and analyzing crucial events in the design easier.

The tool provides measurement and annotation capabilities. Users can measure time intervals, signal values, and the occurrences of events, providing essential insights into the performance and behavior of their digital designs. Users can also annotate waveforms to document their observations and findings, promoting collaboration and knowledge sharing.

Scripting and automation are supported in GTKWave through the Tool Command Language (Tcl). This feature allows users to automate repetitive tasks and customize the tool's functionality to fit their specific requirements.

Overall, GTKWave offers an exhaustive visualization of digital designs, enabling engineers to identify and debug issues more efficiently. This leads to a reduction in development time and an improvement in design quality. As a learning tool, it allows students to understand complex digital circuits and systems better. Being open-source and cross-platform, GTKWave is flexible and compatible with various operating systems and digital design tools, making it a significant asset in any digital design workflow.

3. RELATED WORK

IoT security has been a growing concern in recent years, and several solutions have been proposed to address the vulnerabilities of IoT devices. One approach is to use secure boot mechanisms to ensure that only trusted firmware is loaded onto the device [27, 28]. This approach can prevent unauthorized firmware modifications and protect the device against malicious attacks. However, secure boot mechanisms can be bypassed through hardware or software vulnerabilities, which makes them vulnerable to attacks [29, 30].

Another approach is to use hardware-based root-of-trust mechanisms, such as Trusted Platform Modules, to ensure the security of IoT devices [31, 32]. The Trusted Platform Module is a microcontroller designed to provide hardware-based security functions for computer systems. Developed by the Trusted Computing Group (TCG), a non-profit organization, TPM is now an international standard for secure computing. The primary goal of TPM is to establish a root of trust, enabling secure storage, integrity measurement, and attestation for software and data on a computing platform. The TPM consists of several components: a cryptographic processor, secure non-volatile storage, platform configuration registers (PCRs), and an endorsement key (EK). These components work together to provide essential security services such as secure key storage, cryptographic operations, and platform integrity measurement. The primary functions of the TPM include secure storage of cryptographic keys and other sensitive data in a tamper-resistant manner, integrity measurement and recording of software and firmware during the boot process to generate a unique value known as the platform configuration register (PCR) value, and attestation, which allows a system to prove its integrity and identity to remote parties using signed statements called ”quotes” that include PCR values and other platform-specific information. The main advantages of employing TPM in computing systems include improved security through hardware-based protection against unauthorized access and tampering, enhanced trust by enabling systems to establish a root of trust, and support for advanced security features such as full-disk encryption, secure boot, and remote attestation. However, TPMs can be expensive and unsuitable for all IoT devices. One of the primary reasons TPM is not

widely implemented in low-cost devices is the additional cost associated with its integration. The TPM chip, as a specialized hardware component, adds to the overall production cost of a device. In highly competitive markets with low-cost devices, manufacturers may exclude TPM to minimize production costs and offer more affordable products. Low-cost devices often have limited processing power and memory resources. Integrating TPM requires additional processing capabilities and memory to handle cryptographic operations and secure storage. These resource requirements can be challenging for low-cost devices that prioritize minimizing resource usage to maintain affordability. Implementing TPM in low-cost devices may introduce complexity and compatibility issues. TPM's integration into a device's hardware and software requires a compatible platform and additional development effort. This added complexity may pose challenges for manufacturers prioritizing simplicity and ease of use in their low-cost devices. Low-cost devices cater to specific market segments where price and affordability are the primary concerns of users. In these markets, security features like TPM may be seen as less important, causing manufacturers to focus on other features that appeal to their target customers. This results in TPM being overlooked in the design and production of low-cost devices. Some low-cost device manufacturers may opt for alternative security solutions that are less expensive or better suited to their product's specific requirements. Software-based security mechanisms or lightweight hardware security modules may be chosen instead of TPM to balance security needs with cost and resource limitations.

In recent years, a new approach called Device Identifier Composition Engine (DICE) has been proposed to address the security vulnerabilities of IoT devices. DICE is designed to be a cost-effective alternative to TPM, requiring minimal additional hardware components. This reduced hardware requirement makes DICE more affordable and easily integrated into low-cost devices. Additionally, DICE's lightweight design demands fewer processing and memory resources, making it more suitable for devices with limited capabilities. DICE's simplicity and modular architecture allow easier integration into a device's hardware and software. By leveraging existing hardware components, such as unique device identifiers (UDIs), DICE reduces complexity and development efforts compared to TPM. DICE is

designed to be scalable, enabling it to be tailored to a device's specific security requirements. This flexibility allows manufacturers to choose the appropriate level of security for their low-cost devices without incurring unnecessary costs or complexity. However, DICE is susceptible to vulnerabilities in Updatable Device Firmware, allowing attackers to take control of devices and access users' sensitive data. Attackers can exploit vulnerabilities in the firmware update process to install malicious firmware, which may compromise the security of the entire device.

Seshadri et al. [1] introduce Pioneer, an innovative software-driven solution devised to confirm the legitimacy of code execution on untrusted, outdated computing systems. The problem stems from the risk of these systems tampering with code execution in various ways, such as changing the code before execution, substituting it with different code, or altering execution states during the operation. Pioneer confronts this issue through a challenge-response method, engaging an external trusted agent (the dispatcher) and the untrusted computing system. Upon the successful implementation of Pioneer, the dispatcher is reassured that the code executed on the untrusted system remains unaltered, that this untampered code has been initiated for execution, and that this execution proceeds without any interference, even if malicious software is present on the system. The paper further discusses the establishment of a "dynamic root of trust" on the untrusted system by the dispatcher using Pioneer. This root comprises code that is guaranteed to remain unchanged and is assured to execute in a secure environment. Pioneer also serves as a foundation for the creation of security applications. This is exemplified in the paper by a kernel rootkit detector, which uses a software-driven kernel integrity monitor to periodically compute hashes of kernel code segments and static data structures, identifying any unauthorized alterations to the kernel. Another essential aspect of Pioneer is its support for software-driven code attestation. Unlike hardware-dependent attestation systems like the Trusted Computing Group's (TCG) Trusted Platform Module (TPM), Pioneer supports code attestation without any need for hardware augmentations. This is especially useful for outdated systems. Unlike the TPM, which uses an SHA-1 hash function that could be compromised, Pioneer can be updated if the underlying primitives are found to be compromised and provide runtime

attestation, allowing verification of software integrity at any point. This offers a stronger guarantee than TPM's load-time attestation and can better handle the threat of dynamic attacks post-loading. Pioneer's overview is as in Figure 3.1.

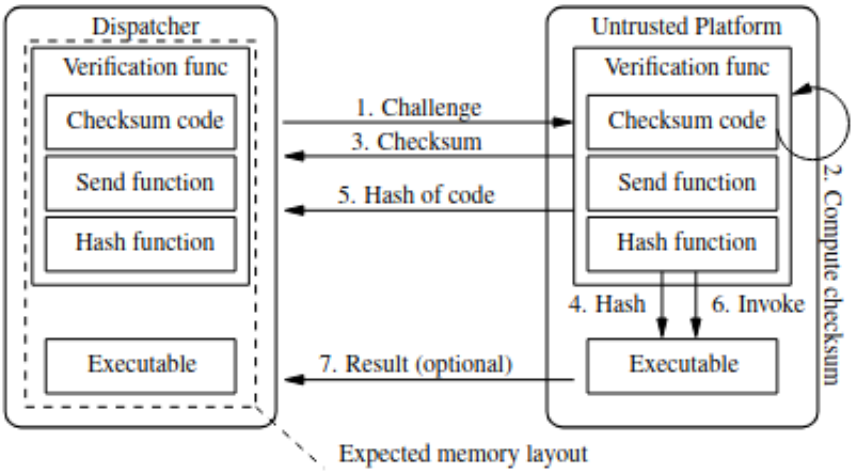

Figure 3.1 Overview of Pioneer [1]

.

In their research, Jäger et al. [33] present a specialized hardware architecture designed to implement the Device Identifier Composition Engine (DICE). DICE is an element of the Trusted Computing Group's strategy for devices with limited resources that require only a subset of Trusted Platform Module capabilities. The goal of the proposed architecture is to cause minimal disruption to the original hardware design. It utilizes the open-source RISC-V implementation, VexRiscV, synthesized for a Field-Programmable Gate Array (FPGA). The paper briefly overviews DICE and its role within the Robust Internet of Things (RIoT) framework. It further examines existing Trusted Computing solutions and DICE implementations tailored for devices with limited resources, assessing their strengths and weaknesses. The authors discuss the current RISC-V implementations and their enhanced version of the VexRiscV core, named Dairo. They elaborate on their proposed implementation architecture based on Dairo, detailing the attacker model, design principles, and the hardware and firmware implementation specifics. The aim of the paper is to establish a minimal Root of Trust for resource-constrained microcontrollers. This could serve as a useful reference for your research on Trusted Computing, DICE, and hardware-driven

security solutions. However, it's important to note that the study focuses only on the implementation of DICE, with security precautions being taken at the user application layer.

Hristozov et al. [34] presented a practical runtime attestation technique for small IoT devices. The authors focus on generating attestation evidence during runtime, specifically on a DICE-based hardware/software method that requires only standard on-chip hardware. The proposed method does not require external components, trusted execution environments, or non-standard hardware features. Instead, it utilizes a small, privileged, write-protected software layer that manages access to secrets at runtime using the Memory Protection Unit (MPU) and generates attestation evidence upon request from a verifier. The method targets inexpensive off-the-shelf microcontrollers with limited resources. The authors demonstrate the feasibility and practicality of their approach by implementing the STM32L476, an ARM Cortex-M4 microcontroller. However, this study cannot provide a countermeasure against attacks such as TOCTOU.

In their work, Suzaki et al.[2] propose a solution aligned with PMF to tackle the increasing challenges of security, life cycle management, and disposability for Internet of Things (IoT) devices. These devices are progressively being employed for AI-Edge applications in fields such as smart cities and smart farming. IoT devices operate on Linux and are typically updated via Over-The-Air (OTA) updates. However, these updates are often delayed, leaving the devices exposed to potential attacks. The authors introduce Reboot-Oriented IoT (RO-IoT), an extension of existing system-reset architectures designed to handle IoT security and life cycle management. RO-IoT operates by rebooting and reinstalling the OS as a primary strategy for system recovery after a compromise. It connects the life cycles of the device, software, and service to Public Key Infrastructure (PKI) based TLS certificates, which ensures that devices can only boot up when a secure network connection is in place. Notable features of RO-IoT include a secure network bootloader safeguarded by Trusted Execution Environment (TEE), TEE-secured live memory forensics, and IoT life cycle management employing TEE and PKI-based certificates. The implementation makes use of a small Linux image as the bootloader in conjunction with OP-TEE, a trusted OS for ARM TrustZone. It alternates between two types of Linux images, one dedicated to the

network bootloader and life cycle management and the other for executing IoT applications. You can see the system's design in Figure 3.2.

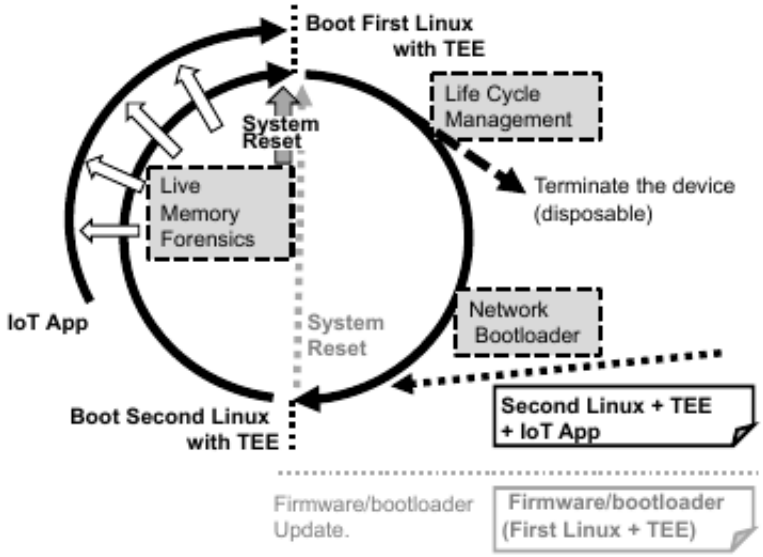

Figure 3.2 RO-IoT design overview [2]

.

RO-IoT ensures secure network booting with TEE, allows for the utilization of downloaded Linux images and facilitates firmware/bootloader updates. It also incorporates live memory forensics to protect the integrity of the forensics process and enforce application whitelisting. RO-IoT's life cycle management encompasses four stakeholders: the device factory, device supplier, software vendor, and service provider. The life cycles of IoT hardware devices, software, and services are connected to the expiration of three certificates used in TLS connections. By checking these certificates at boot time, RO-IoT can deactivate the device to avert misuse, if required. Notably, unlike the study in focus, RO-IoT needs costly hardware to function.

Noorman et al. [35] introduce Sancus, a security framework designed for networked embedded systems with constrained resources. This architecture aims to tackle the escalating security risks associated with increased network connectivity and software extensibility. Sancus endeavors to offer remote attestation, robust integrity, and authenticity assurances with a hardware-based Trusted Computing Base (TCB) that's as minimal as possible.

The authors have incorporated the necessary hardware as an extension of a conventional microprocessor, demonstrating that this comes with a minimal impact on performance, area, and power consumption. In addition to this, they have created a C compiler that is tailored for Sancus-enabled devices. This allows software modules to be constructed using straightforward annotations on standard C files, thereby keeping software development costs manageable.

4. PROPOSED METHOD

4.1. Design

In this section, we describe our design's software and hardware blocks. This design uses the DICE architecture for cryptographic attestation and device identity. However, the DICE architecture does not provide a solution for runtime malware detection. An attacker could inject malware into the firmware by exploiting any security vulnerabilities in the user firmware and obtaining the Alias Key. In this case, the attacker can authenticate with the IoT server and obtain encrypted assets. DICE suggests a firmware update as a solution to this issue. However, detecting this vulnerability in the firmware can take a long time. During this period, the attacker can manipulate the IoT network. To prevent this situation, we propose an enhanced Device Identifier Composition Engine with an integrated hash engine for periodic memory forensics, allowing malware detection within user applications. The implementation of this proposed system is conducted on a modified version of the Ibex processor, which includes the integration of both the DICE and the memory forensics module. To integrate the enhanced DICE and the memory forensics module with the Ibex processor, several modifications to the processor platform are required, such as:

1. Integration of the DICE: The DICE architecture is added to the processor platform, providing support for generating unique device identifiers, establishing secure device identity, and enabling secure boot processes.

2. Memory forensics module: A memory forensics module is added to the processor platform, responsible for periodically scanning the device's memory, computing hash values for different memory segments, and comparing them against a set of known-good values. If discrepancies are detected, the module raises an alert and takes appropriate action, such as quarantining the affected application or initiating a system reboot.

3. Modifications to memory subsystem: The memory subsystem of the Ibex processor must be modified to support the periodic scanning and hashing of memory segments by the memory forensics module, as well as the secure boot process and device attestation required by the DICE implementation. This may involve adding new memory access mechanisms, such as direct memory access (DMA), or modifying existing memory controllers to facilitate efficient and secure memory forensics and DICE operations.

4. Integration of layered authentication and attestation: The Ibex processor platform must be modified to support the layered authentication model employed by DICE, which includes the device manufacturer, the device itself, and the applications running on the device. This requires the implementation of appropriate key management and access control mechanisms to ensure that only authorized entities can access the device's sensitive information. Additionally, the platform should support attestation processes, allowing other parties to verify the device's identity and integrity.

We incorporate an MFP to the original DICE architecture as shown in Figure 4.1. This figure shows our additions to the original DICE given in Figure 2.1. MFP is a hash engine that periodically calculates the digest of the user firmware. It uses the sha256 algorithm [36] to calculate the digest periodically and compares it with the original value. If the computed digest differs, it resets the system like a watchdog timer. After the reset, the system can be forced to update the firmware using the firmware update software stored in the read-only memory, or this situation can be reported to the IoT server.

4.1.1. Enhanced DICE Architecture Design

The DICE architecture is divided into two parts, hardware and software. We use a memory-mapped ROM for the DICE core to function in the hardware part. We also add a Unique Device Secret (UDS) register and a hardware-based register to transfer CDI to Layer 0. Layer 0 reads the CDI through this register.

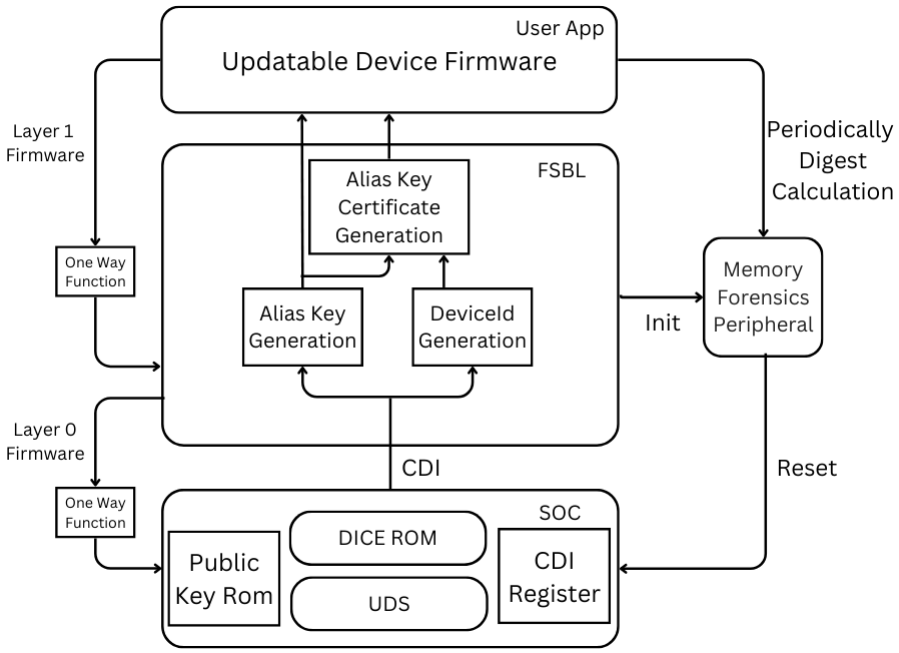

Figure 4.1 System Architecture

The developed firmware combines the UDS with Layer 0 and calculates digest to generate the CDI. We use MFP in the general-purpose mode for hash operation. By performing the hash operation in hardware, we can avoid the need for an enormous DICE ROM size. Otherwise, if we perform the hash operation as software-based, the ROM size increases significantly, and CDI generation takes longer computation times. We use the elliptic curve algorithm in Layer 0 since it requires less processing power than RSA for key generation [37]. Layer 0 first generates a deterministic 256-bit ECDSA key pair for the DeviceID using the CDI received from the DICE core. Then, Layer 0 generates a deterministic EC P-256 key pair for the Alias Key, which is later used for authentication. After generating the key pairs, we use the private part of the DeviceID to produce an X.509 certificate like 4.1 for the Alias Key. In this certificate, we used the hash of the DeviceId's public part as the issuer name in order to understand the identity of the device. To understand the version of the firmware running on the device, we used the FWID (hash of the firmware) as the subject name. Once generating the certificate, we clear the DeviceID and CDI from memory as a security measure. Configuring and running the MFP is also one of the tasks of Layer 0. After completing this process, it sends the addresses of the X.509 certificate and Alias Key

to Layer 1 and initiates Layer 1.

Table 4.1 X.509 Certificate Attributes.

Attribute	**Value**
Issuer Key	DeviceId
Subject Key	AliasKey
Issuer Name	$H(DeviceId_{public})$
Subject Name	FWID
Signature Algorithm	ECDSA with SHA256

4.1.2. Memory Forensics Peripheral Design

The MFP is a hardware-based memory-mapped hash engine that uses the cryptographic hash function sha256. It accesses the memory directly so that memory forensics can be done without interrupting the CPU. It is designed in such as way that it can also be used in general-purpose hash calculations. Once started, the peripheral scans the memory at the specified interval and calculates the digest. It resets the system if the computed digest differs from the one written to the configuration registers. Once the peripheral enters the memory forensics mode, it cannot be stopped until the next reset, thus preventing its operation from being blocked by attackers. We utilize digital signatures to prevent tampering with the digest that the peripheral will compare. To enable this process, we add a one-time programmable (OTP) memory to store the public part of an ECDSA key pair. This feature ensures that attackers cannot manipulate the public key. The digest value that the MFP compares is stored along with a signature created using the private key corresponding to this public key. Before running the MFP, Layer 0 verifies this signature. If the signature is invalid, Layer 0 reports this to the server and does not run Layer 1. You can see the overall flow chart of the system in Figure 4.2.

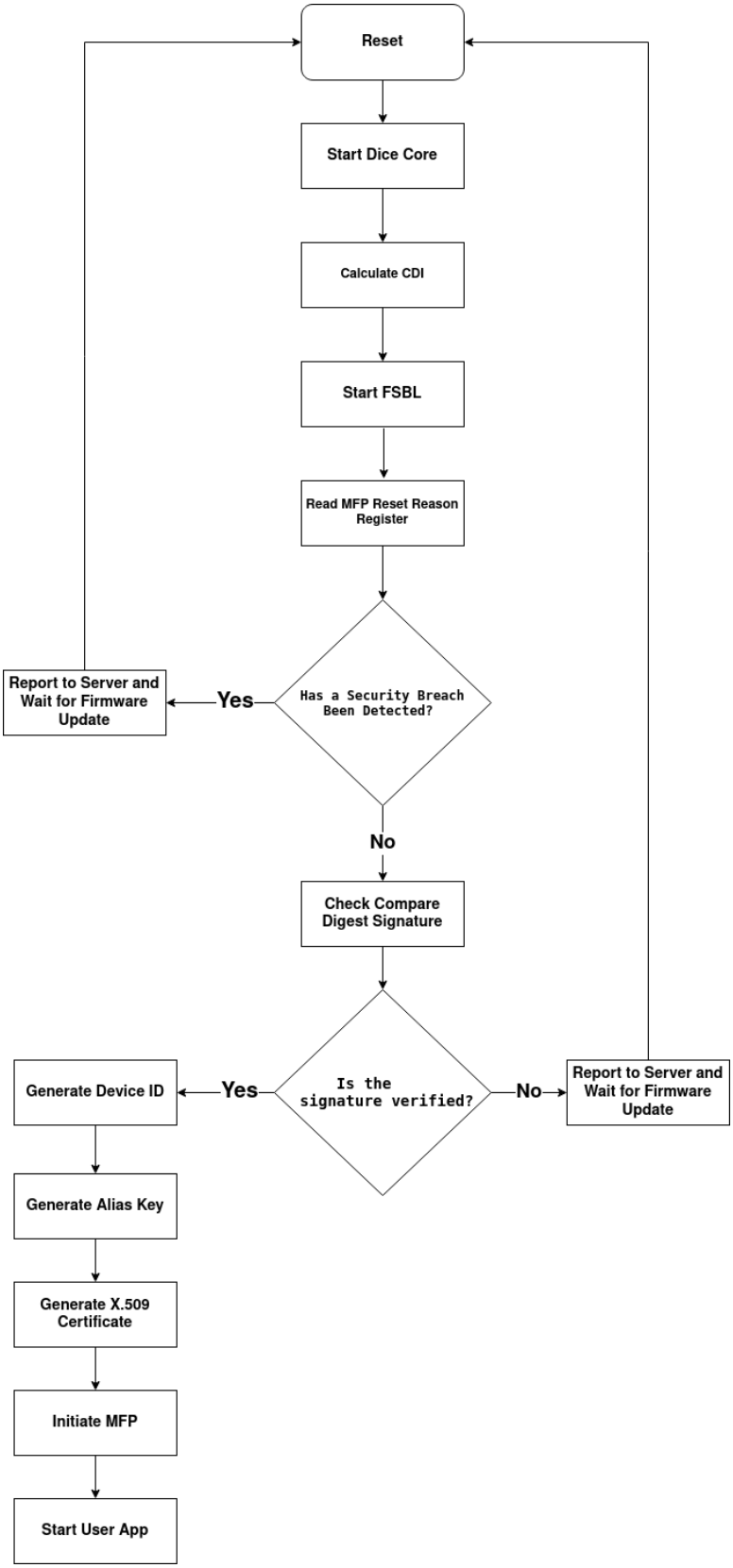

Figure 4.2 Overall Flow Chart

4.2. Implementation

An important step of IoT system design is selecting a suitable processor. In our study, we use RISC-V as our processor. RISC-V is becoming increasingly popular for IoT applications due to its open-source nature, low power consumption, and flexibility. Its modular architecture allows designers to customize the instruction set for their specific needs. This makes it ideal for IoT devices with specific power, performance, and memory requirements. RISC-V also has a small footprint and low power consumption, which is important for devices that need to operate on limited battery power. Additionally, RISC-V is an open-source architecture, which means that it can be freely used and modified by anyone. This makes it easier for device manufacturers to develop customized solutions for their specific IoT applications without the licensing fees associated with proprietary architectures.

There are several open-source implementations of RISC-V. In this study, we use the Ibex platform, which is an open-source RISC-V processor core developed by lowRISC and written in SystemVerilog [23]. To validate our design, we created a simulator using Verilator [38]. This allows us to test our system in a simulation environment and ensure it functions as intended. Once satisfied with the simulation results, we synthesize our system on FPGAs. The Ibex project includes an instance called Artya7, developed explicitly for Xilinx's Artix-7 platform [39]. For comparison purposes, we chose Artix-7 as our target FPGA. Overall, the Ibex platform provided a reliable and efficient solution for our proof of concept design. Its open-source nature and high configurability feature allowed us to tailor our system to meet the specific requirements of an IoT device.

4.2.1. Implementation of DICE

The original Ibex processor uses a single RAM for both data and instructions. We add a ROM to the system to operate the DICE Core firmware. We also add a mux to direct specific signals from the pipeline to RAM or the DICE-ROM, allowing the CPU to execute from ROM after reset. We connect the selection input of this mux to a signal called 'diceEn',

which determines whether the pipeline signals are directed to DICE-ROM or RAM, as shown in Figure 4.3. When 'diceEn' is high, the CPU executes the instructions in DICE-ROM; otherwise, it executes the instructions in RAM. To prevent attackers from accessing UDS, we disable access to the DICE-ROM while 'diceEn' is low. This means that once the DICE Core completes its operation, it cannot be read.

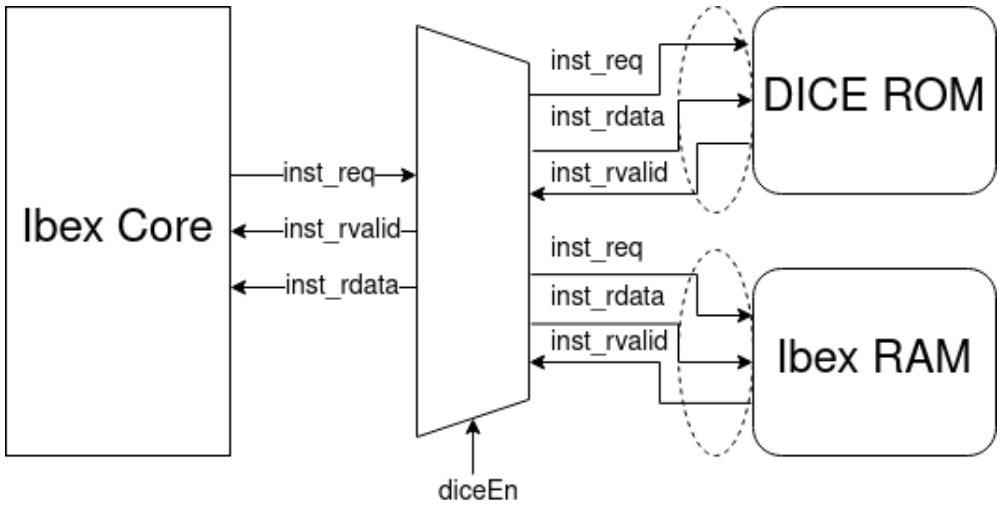

Figure 4.3 Ibex (RISC-V) core and its modified memory structure.

We add some extra functionalities to the firmware to ensure that the CPU executes the code from the DICE-ROM after the reset. While we place the DICE-ROM at the memory address 0x1000000, we set the Ibex's boot address to 0x100000 since there is no way to change the boot address at runtime. As the selected ROM size in our design is less than 1 megabyte, we mask the instruction and data addresses generated by the 'ibex_top' core with the address 0xFFFFF. This allows us to avoid issues with the DICE-ROM's address being 0x1000000. However, to prevent malfunctioning the pc(program counter)-relative instructions, we modify the reset vector so that the instruction pointer continues from the base address of the DICE Core, which is 0x1000000. Ibex has a simple bus structure. The top core generates a 32-bit address called 'host_addr' based on the addresses accessed in load-store instructions. The 'bus', which takes 'host_addr' as an input, directs it to the necessary MMIO (Memory Mapped IO) peripheral through the 'device_addr_xxx' signal. If we don't implement the change we're discussing, an incorrect 'host_addr' will be generated due to program counter-relative instructions, and we will not be able to access the DICE

ROM at all. You can see the GTKWave outputs related to this situation in Figure 4.4 and Figure 4.5. Figure 4.4 shows the output when it works incorrectly, while Figure 4.5 shows the correct output.

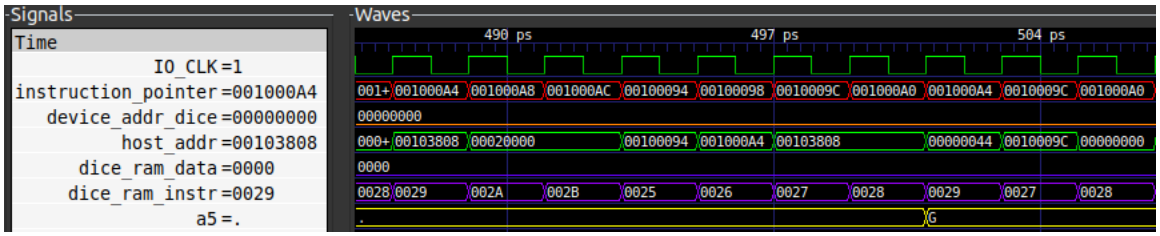

Figure 4.4 Incorrect Host Address

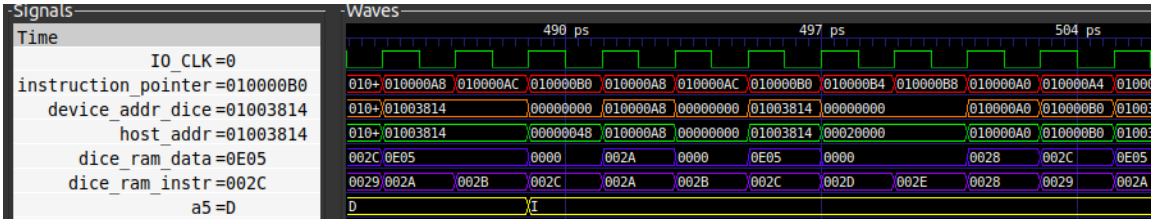

Figure 4.5 Correct Host Address

As seen from the waveform, in Figure 4.4, the 'host_addr' should be 0x1003808, but since it is 0x103808, the 'device_addr_dice' has taken the value of 0x00000000. Therefore, the system cannot access the DICE ROM. In Figure 4.5, it can be seen that 'device_addr_dice' follows the 'host_addr'. You can see the instruction block causing this situation in Figure 4.6. The 'auipc' instruction is a pc-relative instruction. This instruction offsets the current value of the program counter with the specified value and loads it into the specified register. When the value of the program counter is 0x103388 instead of 0x1003388, the value 0x103888 is loaded into the a0 register, and in the next instruction, the value 0x47c is added to this value, and the a0 register ends up storing the address of the <<<<<DICE_CORE>>>>> string as 0x103808. Of course, since this value is incorrect, incorrect characters are printed on the screen instead of <<<<<DICE_CORE>>>>>.

```
01003388 17 05 00 00     auipc      a0,0x0
0100338c 13 05 c5 47     addi       a0=>s_<<<<<DICE_CORE>>>>>_01003804,a0,0x47c
01003390 86 cf           c.swsp     ra,0xdc(sp)
01003392 ef c0 ff cf     jal        ra,puts
```

Figure 4.6 PC-Relative Instruction

We add a memory-mapped CDI register of 32 bytes in length to the system to store the CDI. After calculating the CDI, the DICE Core writes it to this register. Additionally, we also add a memory-mapped DICE Controller peripheral to the system to terminate the DICE Core. When the finish command is given to this peripheral, it sets the diceEn signal to low. You can see the waveform related to the 'diceEn' signal and the CDI registers at the moment when the Dice Core completes its operation in Figure 4.7.

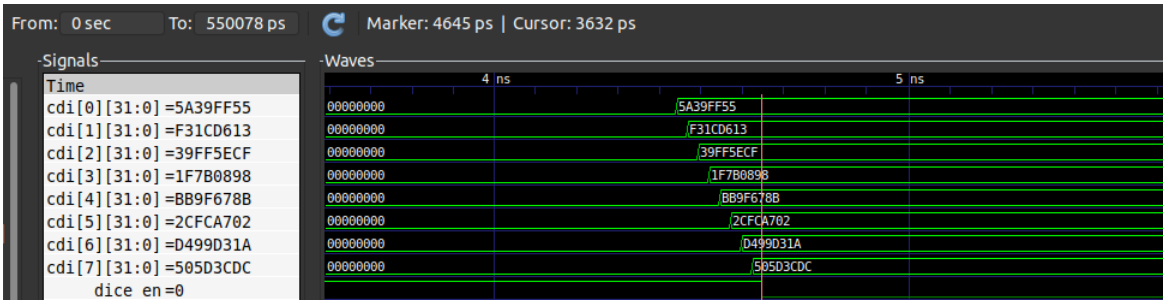

Figure 4.7 Snapshot of the moment when the Dice Core finishes.

In the startup file, we make the necessary changes to direct the instruction pointer to the start of Layer 0 and to give the finish command to the DICE Controller once the main function is completed. The memory footprint of the DICE Core is as shown in Table 4.2.

Table 4.2 Memory Footprint of DICE Core

Segment	***Size(byte)***
.text	424
.data	4
.bss	0
.stack	16

Since the hash operation is performed through MFP in the general-purpose mode in hardware rather than in software, as can be seen from this table, the code size is significantly smaller. In addition, our use of the hash engine significantly shortened the operation time of the DICE Core. You can see this difference in Figure 4.8 and Figure 4.9. The waveform in Figure 4.8 belongs to the scenario where we used the hash engine for hash operations. On the other hand, figure 4.9 belongs to the scenario where we performed hash operations using Mbed TLS.

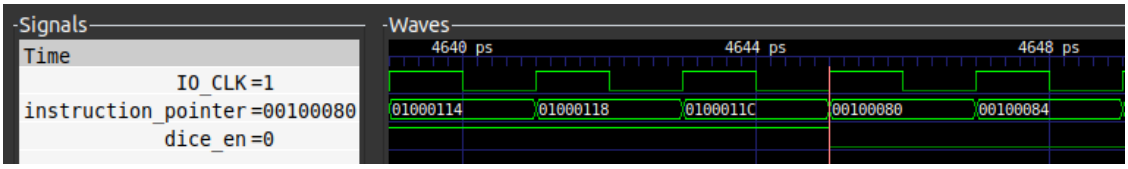

Figure 4.8 Dice Core uses MFP for hash operations

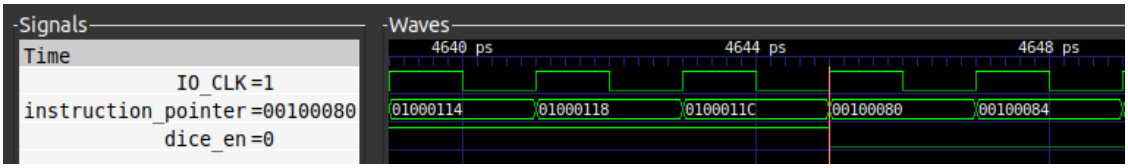

Figure 4.9 Dice Core uses Mbed TLS for hash operations

In our implementation, we name Layer 0 as the First Stage Boot Loader (FSBL) and Layer 1 as the user application. We place the FSBL at address 0x100000 and the user application at address 0x160000. We use the Mbed TLS library [25] for cryptographic functions used in the FSBL. To keep the attack surface small, we minimize the code size of the FSBL. Firstly, FSBL reads the status register of the MFP to determine if there was a security breach in the previous operation. If there is, it reports this to the server and does not run the user application. Otherwise, FSBL first reads the CDI from the CDI Register and then generates the DeviceId using the ECDSA algorithm and the Alias Key using the EC P-256 algorithm. The Alias Key and DeviceId must be generated deterministically in the DICE architecture. To achieve this, we add an entropy source that outputs the digest of the CDI. As a result, the random number generators used in the ECDSA and EC-P key generation functions produce the same result every time. This makes the generated keys deterministic. After the key generation process, an X.509 certificate signed with the private part of the DeviceId is generated for the Alias Key. Then, the MFP starts to execute. For this operation, the signature of the digest, start address, and end address belonging to the user application are verified with the public key stored in the Public Key ROM that we added to the system. If the verification is successful, the memory forensic peripheral is activated. Finally, the CDI and DeviceId are erased from memory as a security measure. The Alias Key and X.509 certificate are passed as parameters to the main function of the user application and executed. Additionally, the vector table must be offset to enable interrupts to function correctly in the

user application. The MFP periodically scans and compares the memory with the entered digest. If the comparison process yields an incorrect output, it resets the system. Once the MFP is initiated, the attacker cannot stop it until the reset.

We designed a user application to verify the operation of the MFP. This application exploits itself by changing a piece of data in its own address space 5 seconds after it starts. Shortly after this exploit operation, we observed that the MFP detected this and reset the system. In addition, to ensure the proper operation of interrupts, we offset the vector table using Ibex's 'mtvec' register to 0x160000, which is the base address of the user application.

4.2.2. Implementation of Memory Forensics Peripheral

The memory-mapped MFP detects exploits resulting from security vulnerabilities in user applications. This peripheral scans the designated memory region at specified intervals, calculates its digest using the sha256 algorithm and resets the system if the result differs from the input digest. We use secworks's sha256 IP core [40] for the sha256 algorithm and change the two-port Ram used in Ibex to a three-port Ram, which we connect to the peripheral to allow direct access to RAM. This change enables the MFP to calculate digests while the CPU continues to operate. The peripheral has two modes: memory forensics and general purpose. In the general-purpose mode, it is used for hash operations by DICE Core and FSBL.

Before processing the input data, SHA-256 requires that the input be padded to a certain length. This is necessary because the SHA-256 algorithm processes data in 512-bit blocks, and any remaining data that doesn't fit into these blocks must be padded.

Here is the step-by-step process for padding an input message for SHA-256:

1. Append a single '1' bit to the end of the input.

   Let's take the message "abc" as an example. In binary, the ASCII representation is:

   a: 01100001

b: 01100010

c: 01100011

After appending a '1' bit, the message becomes:

01100001 01100010 01100011 1

2. Append '0' bits until the length of the input is 448 mod 512. This is done because we need to leave space for the bit length of the original message, which is 64 bits long and will be appended at the end. Hence, we need the message length to be 448 bits (512 - 64 bits) modulo 512.

   In our example, the message is currently 25 bits long. To make the length 448 modulo 512, we append 423 '0' bits (448 - 25).

3. Append a 64-bit representation of the original input length.

   The last 64 bits of the padded message is a binary representation of the length of the original message (before the padding). This is in bits, not bytes.

   In our example, the original message was 24 bits long (3 bytes x 8). So we append a 64-bit binary representation of 24, which is 0000000000000000000000000000000000000000000000000000000000011000

   So, the final padded message in the case of "abc" would be: 01100001 01100010 01100011 1 + 423 zeroes + 0000000000000000000000000000000000000000000000000000000000011000

The MFP we have designed accepts the input as 512 bits, with padding applied. Therefore, we have designed a logic that performs this operation as described above. In our design, you can see an example of this process in the waveform in Figure 4.10. In the design, the 512-bit block is stored in an array, each element of which is 32 bits. The snapshot you see here belongs to the last 64 bytes of a 1316 byte code block. As you will also see in hash_block_15, 0x2920 indicates the number of bits in the 1316 bytes (0x2920 bits = 10528 bits = 1316 bytes).

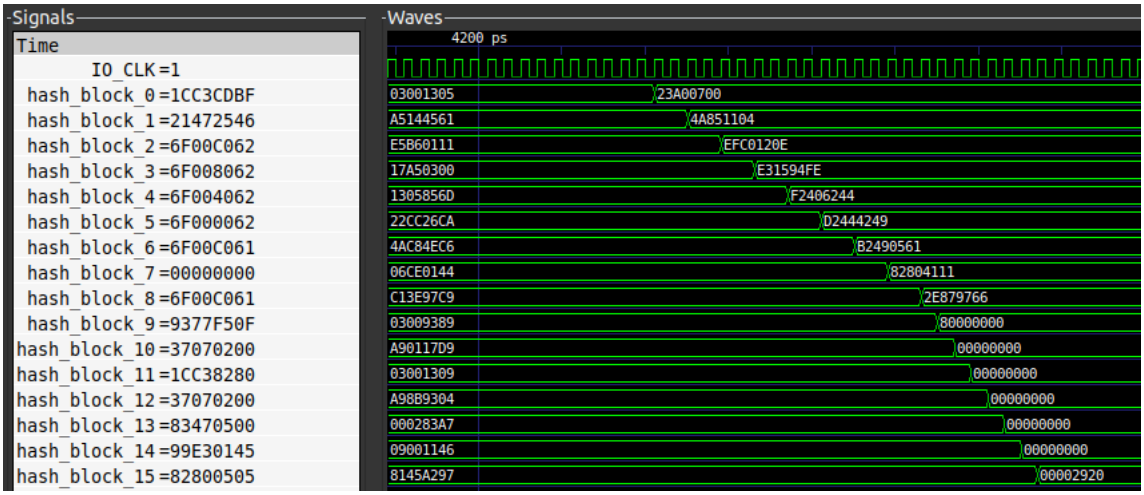

Figure 4.10 SHA256 Padding

We also added a 32-byte Initial Value register to the peripheral. This feature allows the DICE Core to write UDS to this register and to calculate the digest combined with FSBL. We also incorporate the status register of this peripheral not to be reset after a soft reset. This way, the FSBL can read this register and determine whether there was a security breach in the previous run. All registers of this peripheral and its memory addresses are listed in Table 4.3.

Table 4.3 Registers and their memory addresses of MFP.

Address Offset	***Register***
0x00	CTRL
0x04	Status
0x08	Start Address
0x0C	End Address
0x10	Scan Period
0x14 - 0x30	Compare Digest
0x34 - 0x50	Initial Data
0x80 - 0x9C	Digest

Since the digest value, the start address, and the end address of the application provided to the peripheral can be altered by an attacker, we store these three values in memory along with an ECDSA signature. Since the public key, which will be used to verify this signature, can also be changed by the attacker, it must be stored in read-only memory. Therefore, we add one-time programmable (OTP) memory to the system to store the public key. Once

programmed during production, this memory cannot be modified. This feature ensures that an attacker cannot manipulate the public key to verify the signature. In our proof of concept design, we use ROM instead of OTP memory, which is programmed during synthesis and cannot be modified after that. The memory map of the completed design is given in Table 4.4.

Table 4.4 Memory map of the design components.

Address Offset	***Components***
0x30000	Timer
0x40000	CDI Register
0x50000	DICE Controller
0x60000	MFP
0x70000	Public Key ROM
0x100000	RAM
0x1000000	DICE ROM

5. EXPERIMENTAL RESULTS

We evaluated our design in terms of hardware overhead, power consumption, and latency and compared it with the base DICE architecture. The RISC-V architecture Ibex we used in our implementation includes an instance called Artya7, developed explicitly for Xilinx's Artix-7 platform. For comparison purposes, we chose Artix-7 as our target FPGA. We modified the RAM in the design to be 3-ported so that the MFP could directly access it. However, block RAMs in Xilinx FPGAs can have a maximum of 2 ports. One solution can be using register-based RAM; however, this consumes too many resources on the FPGA. Therefore, we used two 2-ported block RAMs to mimic the 3-ported RAM, as shown in Figure 5.1. In this design, we connect the data signals from the 'ibex_top' core to both A ports of the two block RAMs. We connect the instruction signals from the 'ibex_top' core to the B port of the first block RAM and the MFP to the B port of the second block RAM. Additionally, we program both block RAMs with the same firmware during synthesis. Therefore, if an attacker wants to modify an instruction, this change can be reflected in both block RAMs and can be detected by the MFP.

In the first set of experiments, we compared the execution times of using the hash engine, our design choice, and Mbed TLS, a commonly used open-source cryptography library. Note that our design is hardware-based, while the alternative one is software-based, running on the Ibex core. We give the execution times of both designs in terms of the clock cycle in Table 5.1. We evaluated both designs on the system running at 100 MHz. As can be seen from the execution times of both designs, our design is able to improve the execution time 67 times for DICE against a software-based base DICE design.

Table 5.1 Execution times of our design (hardware) and the base design that uses Mbed TLS (software).

	Our Design	**Base design**	**Improvement (Times)**
Clock cycle	2323	156764	67.48

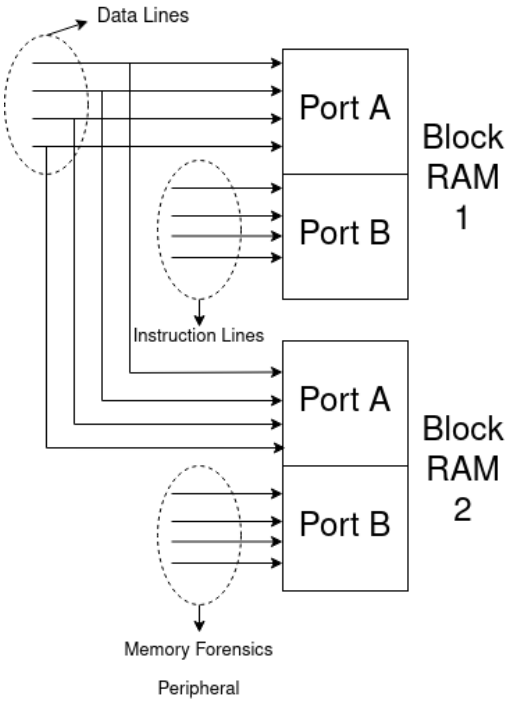

Figure 5.1 Using two 2-port RAMs to construct of 3-port block RAM

As expected, the performance improvement comes with an extra hardware cost. We implemented the original Ibex with no DICE support, Ibex with DICE, and our design. We give the FPGA utilization report for different parts of each design in Table 5.2. In this table, the first column shows the type of site for different components. While columns two, four, and six show the utilization amount, the others show the percentage utilized (Util% in the table) in our selected FPGA. When we look at the slice LUTS, which consumes the highest area in the design, the original DICE increase the Ibex only 1.18 times. However, our design almost doubles it. A fair comparison can be between the original DICE and our design. Our design area is 1.77 times higher than the original DICE implementation. The main reason for this area increase comes from the two 2-ported memory shown in Figure 5.1. Although this area increase seems very high, it can be tolerable in commercial microcontrollers considering the improvement in performance and security.

In order to see the area increase of a microcontroller system with our design, we implemented an open-source microcontroller system named BlackParrot [41]. The BlackParrot platform on the ArtyA7 FPGA utilizes 30,264 LUTs. Integrating our design into this system will

only occupy 8% of the total area, which is tolerable considering the additional security and performance increase.

Table 5.2 FPGA area utilization for three designs.

Site Type	**Original Ibex**		**Ibex+DICE**		**Our Design**	
	Used	***Util%***	***Used***	***Util%***	***Used***	***Util%***
Slice Luts	2457	3.88	2906	4.58	5143	8.11
Slice Registers	909	0.72	1438	1.13	3932	3.10
F7 Muxes	0	0	38	0.12	134	0.42
Block RAM Tile	16	11.85	17.5	12.96	33.5	24.81

In the last set of experiments, we compare three designs in terms of power consumption on our FPGA platform. We give each design's power consumption and temperature values in Table 5.3. Our design increases the on-chip power by 5%, which is an acceptable increase for this component. Considering the overall microcontroller system, this overhead is negligible. The temperature values coming from the FPGA report are the same for the two DICE implementations.

Table 5.3 Power consumption report for three designs.

	Original Ibex	**Ibex+DICE**	**Our Design**
Total On-Chip Power (W)	0.271	0.277	0.290
Dynamic (W)	0.173	0.179	0.191
Device Static (W)	0.098	0.098	0.099
Effective TJA (C/W)	4.6	4.6	4.6
Max Ambient (C)	83.8	83.7	83.7
Junction Temperature (C)	26.2	26.3	26.3

Finally, in order to verify our design in terms of its success in detecting TOCTOU and similar attacks, we developed a user application that can run in the simulator we designed using Verilator. In the simulations, we modify instructions in the firmware after a certain period. We run the FSBL and this application onto the RAM in our design. After a certain period from the start of the application, we observe that the instruction is modified, and the MFP detects it and resets the system.

6. CONCLUSION

The rise of the Internet of Things (IoT) has led to an exponential increase in the number of connected devices, resulting in an increased focus on ensuring their security. These devices, which extend into multiple sectors including healthcare, energy management, home automation, and industrial control systems, pose significant security risks, as demonstrated by several malware attacks such as the Mirai Botnet, BrickerBot, and Persirai. While specifications like Device Identifier Composition Engine (DICE) have emerged as potential security solutions, they too have shown vulnerabilities, particularly with firmware security flaws. This thesis has endeavored to address these vulnerabilities by proposing a novel method to enhance IoT security, focusing on firmware security in the context of DICE architecture.

In the course of this thesis, we examined the fundamental issues of IoT device security, especially the vulnerabilities of DICE related to firmware security flaws. Our proposed solution, which uses a periodic memory forensics technique to detect attacks due to these vulnerabilities, has shown promising results. We've developed a hardware-based hash engine to perform the periodic memory forensics operation, which ensures detection of malicious activities and triggers system reset if necessary.

We designed a Memory Forensics Peripheral (MFP) and integrated it with a one-time programmable memory to safeguard the original digest from potential modifications by attackers. This design allows us to detect any changes made by intruders, thereby enhancing the overall security of the IoT device. The application of the MFP in general-purpose operations and performing hash operations required by DICE in a hardware-based manner have further fortified the security.

Our approach was realized by implementing a proof-of-concept design on the Ibex platform. While our design added a minor area overhead, the benefits of enhanced security significantly outweigh this increase. Our experiments and evaluation have shown that our design has

effectively enhanced the security of IoT devices, thereby confirming the efficacy of our method.

In addition to proposing a novel method for enhancing the security of IoT devices, this thesis also contributes to a deeper understanding of the security challenges in IoT, the functionalities, advantages, and potential drawbacks of key components like RISC-V and DICE, and the positioning of this research within the broader IoT security studies. However, it is crucial to mention that the study was scoped to address IoT device security and did not delve into other facets such as network-level or application-level security.

Looking forward, as IoT devices continue to permeate every aspect of human life, securing these devices will remain a paramount challenge. Although our proposed design presents a strong defense against current attacks, the dynamic nature of security threats necessitates continuous research and advancements. Further studies could explore enhancements to our design to make it even more resilient, such as applying machine learning techniques for more sophisticated detection of anomalies. Furthermore, research could investigate how our design can be made even more efficient and scalable for large IoT networks.

In conclusion, this thesis provides significant contributions to the field of IoT security by enhancing the security of IoT devices using DICE architecture. As we continue to live in an increasingly connected world, the importance of research in this area cannot be overstated. It is our hope that this work will inspire future research and innovation in the realm of IoT security, thereby making our interconnected world a safer place to live.

REFERENCES

[1] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 1–16. **2005**.

[2] Kuniyasu Suzaki, Akira Tsukamoto, Andy Green, and Mohammad Mannan. Reboot-oriented iot: Life cycle management in trusted execution environment for disposable iot devices. In *Annual Computer Security Applications Conference*, pages 428–441. **2020**.

[3] D Pavithra and Ranjith Balakrishnan. Iot based monitoring and control system for home automation. In *2015 global conference on communication technologies (GCCT)*, pages 169–173. IEEE, **2015**.

[4] Yi Liu, Chao Yang, Li Jiang, Shengli Xie, and Yan Zhang. Intelligent edge computing for iot-based energy management in smart cities. *IEEE network*, 33(2):111–117, **2019**.

[5] Shwetank Dattatraya Mamdiwar, Zainab Shakruwala, Utkarsh Chadha, Kathiravan Srinivasan, and Chuan-Yu Chang. Recent advances on iot-assisted wearable sensor systems for healthcare monitoring. *Biosensors*, 11(10):372, **2021**.

[6] Marouane Salhaoui, Antonio Guerrero-González, Mounir Arioua, Francisco J Ortiz, Ahmed El Oualkadi, and Carlos Luis Torregrosa. Smart industrial iot monitoring and control system based on uav and cloud computing applied to a concrete plant. *Sensors*, 19(15):3316, **2019**.

[7] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Tariq Umer, and Yousaf Bin Zikria. Role of iot technology in agriculture: A systematic literature review. *Electronics*, 9(2):319, **2020**.

[8] S Muthuramalingam, A Bharathi, S Rakesh Kumar, N Gayathri, R Sathiyaraj, and B Balamurugan. Iot based intelligent transportation system (iot-its) for global perspective: A case study. *Internet of Things and Big Data Analytics for Smart Generation*, pages 279–300, **2019**.

[9] Wan Haslina Hassan et al. Current research on internet of things (iot) security: A survey. *Computer networks*, 148:283–294, **2019**.

[10] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, **2017**.

[11] M Shobana and S Rathi. Iot malware: An analysis of iot device hijacking. *International Journal of Scientific Research in Computer Science, Computer Engineering, and Information Technology*, 3(5):2456–3307, **2018**.

[12] Tim Yeh, Dove Chiu, and Kenney Lu. Persirai: New iot botnet targets ip cameras. *Trend Micro*, 9, **2017**.

[13] Trusted computing group.hardware requirements for a device identifier composition engine. family 2.0, level 00, revision 78.

[14] Trusted computing group.trusted computing group: Implicit identity based device attestation. version 1.0, revision 0.93.

[15] Stefan Hristozov, Moritz Wettermann, and Manuel Huber. A toctou attack on dice attestation. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*, pages 226–235. **2022**.

[16] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. Internet of things (iot) communication protocols. In *2017 8th International conference on information technology (ICIT)*, pages 685–690. IEEE, **2017**.

[17] Marica Amadeo, Claudia Campolo, Antonella Molinaro, and Giuseppe Ruggeri. Iot data processing at the edge with named data networking. In *European Wireless 2018; 24th European Wireless Conference*, pages 1–6. VDE, **2018**.

[18] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, and Biplab Sikdar. A survey on iot security: application areas, security threats, and solution architectures. *IEEE Access*, 7:82721–82743, **2019**.

[19] Navroop Kaur and Sandeep K Sood. An energy-efficient architecture for the internet of things (iot). *IEEE Systems Journal*, 11(2):796–805, **2015**.

[20] Neal Koblitz, Alfred Menezes, and Scott Vanstone. The state of elliptic curve cryptography. *Designs, codes and cryptography*, 19:173–193, **2000**.

[21] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, **2001**.

[22] Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 401–418. Springer, **2004**.

[23] Ibex risc-v core. `https://github.com/lowRISC/ibex`. Last accessed: Dec 11, 2022.

[24] Verilator. `https://www.veripool.org/verilator/`. Last accessed: Jan 14, 2023.

[25] Mbed-tls library. `https://github.com/Mbed-TLS/mbedtls`. Last accessed: May 3, 2023.

[26] Gtkwave. `https://github.com/gtkwave/gtkwave`. Last accessed: May 12, 2023.

[27] Steffen Sanwald, Liron Kaneti, Marc Stöttinger, and Martin Böhner. Secure boot revisited: challenges for secure implementations in the automotive domain. *17th Escar Europe: Embedded Security in Cars*, pages 113–127, **2020**.

[28] Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. Patterns for secure boot and secure storage in computer systems. In *2010 International Conference on Availability, Reliability and Security*, pages 569–573. IEEE, **2010**.

[29] Nisha Jacob, Johann Heyszl, Andreas Zankl, Carsten Rolfes, and Georg Sigl. How to break secure boot on fpga socs through malicious hardware. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 425–442. Springer, **2017**.

[30] Corey Kallenberg, Sam Cornwell, Xeno Kovah, and John Butterworth. Setup for failure: defeating secure boot. In *The Symposium on Security for Asia Network (SyScan)(April 2014)*. **2014**.

[31] Shijun Zhao, Qianying Zhang, Guangyao Hu, Yu Qin, and Dengguo Feng. Providing root of trust for arm trustzone using on-chip sram. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, pages 25–36. **2014**.

[32] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. Smart: secure and minimal architecture for (establishing dynamic) root of trust. In *Ndss*, volume 12, pages 1–15. **2012**.

[33] Lukas Jäger and Richard Petri. Dice harder: a hardware implementation of the device identifier composition engine. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–8. **2020**.

[34] Stefan Hristozov, Johann Heyszl, Steffen Wagner, and Georg Sigl. Practical runtime attestation for tiny iot devices. In *NDSS Workshop on Decentralized IoT Security and Standards (DISS)*, volume 18. **2018**.

[35] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. Sancus: Low-cost trustworthy extensible networked devices with

a zero-software trusted computing base. In *USENIX Security Symposium*, pages 479–494. **2013**.

[36] Manel Kammoun, Manel Elleuchi, Mohamed Abid, and Mohammed S BenSaleh. Fpga-based implementation of the sha-256 hash algorithm. In *2020 IEEE international conference on design & test of integrated micro & nano-systems (DTS)*, pages 1–6. IEEE, **2020**.

[37] Dindayal Mahto and Dilip Kumar Yadav. Performance analysis of rsa and elliptic curve cryptography. *Int. J. Netw. Secur.*, 20(4):625–635, **2018**.

[38] Wilson Snyder. Verilator and systemperl. In *North American SystemC Users' Group, Design Automation Conference*. **2004**.

[39] Brent Przybus. Xilinx redefines power, performance, and design productivity with three new 28 nm fpga families: Virtex-7, kintex-7, and artix-7 devices. *Xilinx White Paper*, **2010**.

[40] Sha-256 ip core. `https://github.com/secworks/sha256`. Last accessed: May 3, 2023.

[41] Blackparrot platform guide. `https://github.com/black-parrot/black-parrot/blob/master/docs/platform_guide.md`. Last accessed: May 3, 2023.