INTEGRATING PHYSICAL UNCLONABLE FUNCTIONS WITH MACHINE LEARNING FOR THE AUTHENTICATION OF EDGE DEVICES IN IOT NETWORKS

BY

SHEIKH ABDUL MANAN

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

2026

INTEGRATING PHYSICAL UNCLONABLE FUNCTIONS WITH MACHINE LEARNING FOR THE AUTHENTICATION OF EDGE DEVICES IN IOT NETWORKS

BY

SHEIKH ABDUL MANAN

A thesis submitted in fulfilment of the requirement for the degree of Doctor of Philosophy in Engineering

Kulliyyah of Engineering
International Islamic University Malaysia

JUNE 2026

ABSTRACT

The rapid growth of the Internet of Thing (IoT) has increased the demand for lightweight and robust hardware security mechanisms. Physical Unclonable Functions (PUFs) have emerged as promising hardware security primitives for device authentication, cryptographic key generation, and counterfeit prevention by leveraging intrinsic manufacturing process variations. Their capability for on demand cryptographic key generation eliminates the requirement for persistent key storage, thereby enhancing security against key extraction and tampering. However, PUF responses are often affected by environmental noise, voltage fluctuations, and device aging, which can compromise their reliability and randomness. Furthermore, recent advances in Machine Learning (ML) have demonstrated the ability to model and predict Challenge–Response Pairs (CRPs) of conventional strong PUFs, posing significant security risks. PUFs can be implemented in Application-Specific Integrated Circuits (ASICs) or, at lower cost, in Field-Programmable Gate Arrays (FPGAs). FPGAs are widely used in IoT and embedded systems due to their programmability, partial reconfigurability, and faster time-to-market. This research presents the design, implementation, calibration, and evaluation of FPGA-based PUFs aimed at improving robustness against environmental variations and ML modeling attacks. Both Arbiter PUFs (APUFs) and Ring Oscillator (RO) PUFs were implemented on an Intel Altera Cyclone IV-E FPGA, with CRPs extracted in real time using a logic analyzer. To mitigate noise and improve consistency, PUF CRPs were processed using whitening and hashing techniques. The performance of the proposed PUFs was evaluated using key metrics including reliability, uniqueness, randomness, and correctness. Randomness was validated through the NIST Statistical Test Suite (STS), while reliability was assessed under variations in temperature, voltage, and device aging. The study also explores the synergy between Artificial Intelligence (AI) and PUFs for IoT security. AI enhances edge computing security through advanced threat detection, automated responses, and optimized resource management. More than 900,000 CRPs were collected for both APUF and RO PUF and evaluated using key metrics including randomness, reliability, and uniqueness. The APUF demonstrated near ideal characteristics with high entropy, reliability (97.99%), and uniqueness (49.99%), while the RO PUF showed similarly strong statistical quality and robustness with reliability of 98.01% and uniqueness of 49.99%. Machine learning resistance was assessed using Logistic Regression (LR), Random Forest (RF), Gradient Boosting (GB), and MultiLayer Perceptron (MLP) models. For APUF, LR and GB achieved low prediction performance (around 61%), whereas RF and MLP attained high accuracies up to 97%, reflecting their ability to model complex PUF behaviour. The higher prediction accuracy observed for complex models such as MLP and RF can be advantageously interpreted from an anomaly detection perspective, rather than purely as a weakness. In contrast, RO PUF prediction accuracies remained close to random guessing (below 50%), confirming strong resistance against learning based attacks and high robustness of the CRP data. Moreover, when integrating PUFs into existing FPGA architectures, design compatibility, required modifications, and interactions with other system functions must be carefully considered. These factors highlight the importance of robust design strategies to ensure reliable and scalable PUF implementations for practical hardware security applications.

خلاصة البحث

زيادةً في الطلب على آليات أمنية عتادية خفيفة الوزن (IoT) يشهد النمو السريع لـ إنترنت الأشياء كعناصر واعدة للأمن العتادي (PUFs) وموثوقة. وقد برزت الدوال الفيزيائية غير القابلة للاستنساخ من أجل مصادقة الأجهزة، وتوليد المفاتيح التشفيرية، ومنع التزوير، وذلك من خلال الاستفادة من الاختلافات الجوهرية الناتجة عن عمليات التصنيع. إن قدرتها على توليد المفاتيح التشفيرية عند الطلب تلغي الحاجة إلى التخزين الدائم للمفاتيح، مما يعزز الحماية ضد استخراج المفاتيح أو العبث بها. ومع ذلك، تتأثر غالباً بالضوضاء البيئية، وتقلبات الجهد الكهربائي، وتقادم الأجهزة، مما قد PUF فإن استجابات (ML) يؤثر سلباً في موثوقيتها وعشوائيتها. علاوةً على ذلك، أظهرت التطورات الحديثة في التعلم الآلي القوية التقليدية، مما PUF الخاصة بـ (CRPs) القدرة على نمذجة والتنبؤ بأزواج التحدي والاستجابة (ASICs) باستخدام الدوائر المتكاملة الخاصة بالتطبيقات PUF يمكن تنفيذ .يشكل مخاطر أمنية كبيرة FPGAs وتُستخدم .(FPGAs) أو بتكلفة أقل باستخدام المصفوفات المنطقية القابلة للبرمجة ميدانياً على نطاق واسع في أنظمة إنترنت الأشياء والأنظمة المضمنة بسبب قابليتها للبرمجة، وإمكانية إعادة التهيئة المعتمدة على PUF الجزئية، وسرعة الوصول إلى السوق. يقدم هذا البحث تصميم وتنفيذ ومعايرة وتقييم تم تنفيذ .بهدف تحسين المتانة ضد التغيرات البيئية وهجمات النمذجة القائمة على التعلم الآلي FPGA Ring Oscillator PUFs (RO PUFs) و Arbiter PUFs (APUFs) كل من ، مع استخراج أزواج التحدي والاستجابةIntel Altera Cyclone IV-E FPGA على لوحة في الزمن الحقيقي باستخدام محلل منطقي. ولتقليل الضوضاء وتحسين الاتساق، تمت معالجة (CRPs) كما تم تقييم .(Hashing) والتجزئة (Whitening) باستخدام تقنيات التبييض CRPs بيانات المقترحة باستخدام مؤشرات رئيسية تشمل الموثوقية، والتفرد، والعشوائية، والصحة PUF أداء NIST وتم التحقق من العشوائية باستخدام حزمة الاختبارات الإحصائية .(Correctness) ، بينما تم تقييم الموثوقية تحت تأثير تغيرات درجة الحرارة، والجهد Statistical Test Suite (STS) (AI) تستكشف هذه الدراسة أيضاً التكامل بين الذكاء الاصطناعي .الكهربائي، وتقادم الأجهزة لتعزيز أمن إنترنت الأشياء. إذ يسهم الذكاء الاصطناعي في تحسين أمن الحوسبة الطرفية من PUFsو

خلال الكشف المتقدم عن التهديدات، والاستجابة الآلية، وإدارة الموارد بشكل أمثل. وقد تم جمع أكثر وتقييمها RO PUFو APUF لكل من (CRPs) من 900,000 زوج من التحدي والاستجابة APUF أظهرت نتائج .باستخدام مؤشرات الأداء الأساسية، بما في ذلك العشوائية، والموثوقية، والتفرد خصائص شبه مثالية من حيث الإنتروبيا العالية، والموثوقية (97.99%)، والتفرد (49.99%)، بينما جودة إحصائية قوية ومتانة عالية مع موثوقية بلغت (98.01%) وتفرد RO PUF أظهر ، والغابة (LR) (49.99%). كما تم تقييم مقاومة التعلم الآلي باستخدام نماذج الانحدار اللوجستي بالنسبة .(MLP) ، والشبكات العصبية متعددة الطبقات(GB) ، والتعزيز التدريجي(RF) العشوائية أداءً منخفضاً في التنبؤ (حوالي 61%)، في حين حقق كل GBو LR ، حقق كل منAPUF إلى دقة عالية وصلت إلى 97%، مما يعكس قدرتهما على نمذجة السلوك المعقد لـ MLPو RF من بصورة إيجابية من منظور RFو MLP ويمكن تفسير دقة التنبؤ المرتفعة للنماذج المعقدة مثل .PUF في المقابل، ظلت .بدلاً من اعتبارها نقطة ضعف فقط (Anomaly Detection) كشف الشذوذ قريبة من التخمين العشوائي (أقل من 50%)، مما يؤكد مقاومتها RO PUFدقة التنبؤ الخاصة بـ العالية. علاوة على ذلك، عند CRPsالقوية ضد الهجمات القائمة على التعلم الآلي ومتانة بيانات ، يجب مراعاة توافق التصميم، والتعديلات FPGA ضمن البنى الحالية المعتمدة على PUFsدمج المطلوبة، والتفاعلات مع وظائف النظام الأخرى بعناية. وتبرز هذه العوامل أهمية استراتيجيات التصميم في تطبيقات الأمن العتادي العملية.PUFالقوية لضمان تنفيذ موثوق وقابل للتوسع لـ

APPROVAL PAGE

The thesis of Sheikh Abdul Manan has been approved by the following:

_____
Md. Rafiqul Islam
Supervisor

_____
Mohamed Hadi Habaebi
Co-Supervisor

_____
Suriza Ahmad Zabidi
Co-Supervisor

_____
Athaur Rahman Bin Najeeb
Co-Supervisor

_____
Khaizuran Bin Abdullah
Internal Examiner

_____
S. M. A. Motakabber
Internal Examiner

_____
Chebil Jalel
External Examiner

_____
Asadullah Shah
Chairman

DECLARATION

I hereby declare that this dissertation is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted as a whole for any other degrees at IIUM or other institutions.

Sheikh Abdul Manan

Signature ..... Date: June 04th, 2026

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH

INTEGRATING PHYSICAL UNCLONABLE FUNCTIONS WITH MACHINE LEARNING FOR THE AUTHENTICATION OF EDGE DEVICES IN IOT NETWORKS

I declare that the copyright holders of this dissertation are jointly owned by the student and IIUM.

1. Any material contained in or derived from this unpublished research may be used by others in their writing with due acknowledgement.

2. IIUM or its library will have the right to make and transmit copies (print or electronic) for institutional and academic purposes.

3. The IIUM library will have the right to make, store in a retrieval system, and supply copies of this unpublished research if requested by other universities and research libraries.

By signing this form, I acknowledge that I have read and understand the IIUM Intellectual Property Rights and Commercialization policy.

Affirmed by Sheikh Abdul Manan

……..……..	June 4th, 2026 ……..
Signature	Date

This thesis is dedicated to my parents, whose unwavering support, encouragement, and sacrifices have been instrumental in my academic and personal journey.

It is also dedicated to my daughters, Ms. Manha Sheikh, Ms. Muizzah Sheikh, and my wife Mrs. Afreen Sheikh, whose presence has been a constant source of inspiration and motivation throughout this work.

ACKNOWLEDGEMENTS

All praise and glory are due to Allah, the Almighty, whose infinite Mercy and Grace have guided and sustained me throughout the course of this research work. Though the journey has been demanding, His boundless blessings eased the challenges and enabled me to complete this thesis.

I owe profound gratitude to my supervisor, Prof. Dr. Md. Rafiqul Islam, for his patience, encouragement, and invaluable guidance. His dedication, insightful comments, constructive suggestions, and thought-provoking questions greatly enhanced the quality of this work. Despite his many commitments, he always found time to listen and provide prompt support whenever needed. His moral support and mentorship played a vital role in shaping and strengthening this research. I also wish to extend my sincere thanks to my co-supervisors, Prof. Dr. Mohamed Hadi Habaebi, Assoc. Prof. Dr. Suriza Ahmad Zabidi and Dr. Athaur Rahman Bin Najeeb, whose constant support, cooperation, and helpful feedback have contributed significantly throughout the journey.

My heartfelt appreciation also goes to my field supervisor, Dr. Adnan Kabbani, whose practical guidance, constructive feedback, and continuous support during the fieldwork phase were invaluable. My heartfelt gratitude to Assoc. Prof. Dr. Moumen Idres for his inputs on Arabic translation. Special appreciation is due to Prof. Dr. Sher Afghan Khan for his valuable advice, encouragement, and support throughout the course of my studies. I am equally grateful to the various departments of the International Islamic University Malaysia for providing the academic resources, facilities, and enabling environment that made this research possible.

My most profound appreciation goes to my beloved parents, whose prayers, sacrifices, and unwavering belief in me have been the foundation of my success. Their guidance, encouragement, and endless support throughout my life have given me the strength to persevere. I am also grateful to my immediate family members for their understanding, patience, and constant motivation, which sustained me during this challenging journey.

Above all, I return all thanks and praise to Allah, the Most Merciful, for granting me the health, wisdom, and perseverance to bring this work to a successful completion.

Alhamdulillah.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interfaces
APUF	Arbiter PUF
AR	Augmented Reality
ASIC	Application-Specific Integrated Circuits
AUC	Area Under the Curve
BER	Bit Error Rate
CLB	Configurable Logic Blocks
CoAP	Constrained Application Protocol
CRP	Challenge–Response Pairs
CSL	Cloud Server Layer
DDoS	Distributed Denial of Service
DDS	Data Distribution Service
DL	Deep Learning
DRL	Deep Reinforcement Learning
DUT	Device Under Test
EC	Edge Computing
ECC	Elliptic Curve Cryptography
EDA	Electronic Design Automation
EDL	Edge Device Layer
EH	Energy Harvesting
ESL	Edge Server Layer
FL	Federated Learning
FPGA	Field-Programmable Gate Arrays
FSM	Finite State Machine
GB	Gradient Boosting
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
GPU	Graphics Processing Unit
HD	Hamming Distance
HLS	High Level Synthesis
ICS	Industrial Control Systems
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IDS	Intrusion Detection Systems
IoT	Internet of Things
IP	Intellectual Property
ITU	International Telecommunication Union
LE	Logic Elements

LFSR	Linear Feedback Shift Registers
LR	Logistic Regression
LUT	Lookup Tables
LWBC	Lightweight Block Ciphers
LWSC	Lightweight Stream Ciphers
MAE	Mean Absolute Error
MDC	Micro Data Centres
MEC	Mobile Edge Computing
ML	Machine Learning
MLP	Multi-Layer Perceptron
MQTT	Message Queuing Telemetry Transport
MSE	Mean Squared Error
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
PKI	Public Key Infrastructure
PMU	Power Management Unit
PRNG	Pseudo-Random Number Generators
PUF	Physical Unclonable Functions
QoS	Quality of Service
RAN	Radio Access Network
RF	Random Forest
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
RO	Ring Oscillator
RToS	Real-Time Operating Systems
RTL	Register Transfer Level
SaaS	Software as a Service
SDF	Standard Delay Format
SDN	Software-Defined Networking
SHA	Secure Hashing Algorithm
SoA	Service-Oriented Architecture
STS	Statistical Test Suite
SVM	Support Vector Machine
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
TTP	Trusted Third Party
VM	Virtual Machines
ZTA	Zero Trust Architecture

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND

The Internet of Things (IoT) refers to a network of interconnected devices, including smart objects, sensors, and actuators, that communicate by transmitting and receiving application-specific data over the internet. These devices are increasingly equipped with advanced sensing, computational, processing, and decision-making capabilities, enabling their deployment in diverse and complex environments (Alwarafy et al., 2021). According to Statista's projections, the global number of IoT devices in operation is expected to surpass 50 billion by 2030. This massive proliferation of heterogeneous devices will generate vast amounts of data that must be processed in the cloud or data centres, leading to network congestion and bandwidth limitations (Singh et al., 2024). The rapid expansion of the IoT ecosystem has led to an increase in data security threats and challenges, which demand a comprehensive understanding and effective mitigation strategies. Many IoT devices inherently suffer from vulnerabilities such as weak default configurations, obsolete components, and insecure firmware update mechanisms. Attackers can exploit these weaknesses to gain unauthorized access, compromise sensitive information, or disrupt large-scale network operations. As a result, IoT systems are becoming increasingly susceptible to various cybersecurity threats, including Distributed Denial of Service (DDoS) attacks, malware infections, and insecure communication channels that undermine device integrity and data confidentiality. Figure 1.1 illustrates the hierarchical relationship between IoT devices, edge computing, edge cloud, and cloud computing, emphasizing their processing location, latency, bandwidth use, and security characteristics. IoT devices operate at the edge of the network, with limited power and processing capability, and are more vulnerable to cybersecurity threats. Edge computing provides localized data processing with the lowest latency. The edge cloud serves as a bridge between edge and cloud, balancing processing efficiency and scalability. At the top, cloud computing offers centralized, high-performance computation, extensive storage, and advanced security, though with higher latency.

Also, traditional cloud computing cannot fully meet the requirements of IoT and mobile services due to several limitations, including location unawareness, constrained bandwidth, high operational costs, lack of real-time responsiveness, and insufficient attention to data privacy (Zhou et al., 2017). These shortcomings have paved the way for the emergence of edge computing (EC), which is designed to address the growing demands of IoT and mobile devices by providing localized processing, reduced latency, and enhanced security (Xiao et al., 2019). EC is a distributed architecture, conceived to minimize both bandwidth and time response in an IoT system (Alrowaily & Lu, 2018). EC architecture is a federated network structure that extends cloud services to the network's edge by introducing edge devices between terminal devices and cloud computing. Data exchange between end devices and EC nodes takes place through either wired or wireless connections.

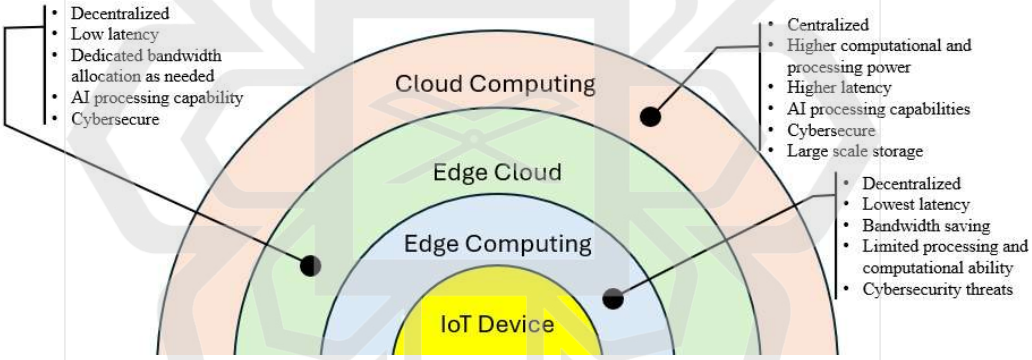


Figure 1.1 IoT architecture layers

In contrast, communication between EC nodes and the cloud occurs over public or private networks. However, EC-assisted systems remain susceptible to various security and privacy threats and attacks (Alwarafy et al., 2021). Additionally, the distributed nature of EC architectures makes them vulnerable to surface attacks and difficult to secure physically. Attackers can exploit weakly secured edge devices to gain access to an organization’s network, as these devices often connect to multiple systems. Key security concerns include ensuring secure authentication, enforcing minimal permissions, and implementing strong encryption for data both in transit and at rest. Traditional security and privacy mechanisms, such as certificates and Public Key

Infrastructure (PKI) authentication, may not be well-suited for edge environments. In edge computing based IoT systems, the large scale deployment of heterogeneous and geographically distributed devices introduces significant scalability challenges for certificate based PKI, as centralized certificate authorities become bottlenecks for identity management, issuance, and verification across multiple stakeholders. These issues are further exacerbated by intermittent connectivity, strict latency constraints, and limited computational resources at edge devices, rendering traditional PKI mechanisms inefficient for real time authentication and large scale IoT deployment scenarios (Singh et al., 2019). Significant challenges in securing edge architectures include limited or intermittent network access, which hampers updates and visibility; susceptibility to physical tampering and supply chain attacks; risks introduced by human error due to the absence of on-site IT expertise; and the complexity of managing and patching large-scale deployments of edge devices.

A wide range of strategies has been proposed to address the security challenges associated with EC. Software-based security approaches rely on encryption techniques that are not entirely foolproof. Moreover, they demand substantial computational resources and energy, which creates a bottleneck for IoT devices due to their limited capabilities. Software-based security approaches are cost-effective and easy to implement or update, but their efficacy is often restricted to vulnerabilities in the operating systems. In contrast, hardware-based approaches are effective against both physical and software attacks, making reverse engineering and data leakage significantly harder. With advancements in semiconductor technology, hardware-based security is increasingly favored for developing low-cost, high-performance, and more secure products (Lee et al., 2022). Hardware-Assisted Security (HAS) can be implemented in several ways, such as (i) dedicated hardware primitives designed for security, (ii) chip-level security-oriented design modifications, and (iii) system-level security monitors and architectural adjustments. A key advantage of HAS is its ability to integrate security directly into the system design flow, thereby avoiding the need to retrofit security features after the IoT system has already been productized (Mohanty et al., 2021).

In contrast to software-based security methods that are vulnerable to reverse engineering or malware, HAS exploits intrinsic physical properties of devices to provide stronger, tamper-resistant platforms called hardware root of trust. Beyond

conventional approaches, advanced technologies such as artificial intelligence (AI) and machine learning (ML) enable real-time anomaly detection and adaptive responses to emerging threats. The AI-based HAS integrates AI/ML with hardware roots of trust to achieve adaptive, intelligent, and tamper-resistant protection (Gopika Rajan & Ganesh, 2022). It includes deploying intrusion detection systems (IDS) to monitor and detect malicious activities, and enforcing robust access control to manage user rights and prevent unauthorized access. Additional measures involve applying data encryption to secure information both at rest and in transit, as well as implementing strong authentication mechanisms to verify user and device identities. Blockchain further enhances security by providing decentralized, tamper-resistant mechanisms that ensure trust, secure data sharing, and transaction integrity (Alnuaimi et al., 2023).

Due to the limited computational resources and heterogeneous architectures of IoT devices, hardware-based security solutions offer a more reliable foundation than software-only methods for safeguarding critical operations such as authentication, encryption, and key management. Trusted hardware components, such as Trusted Platform Modules (TPMs), Hardware Security Modules (HSMs), Secure Enclaves, and Physical Unclonable Functions (PUFs), can establish device identity, protect cryptographic keys, and ensure data integrity even if software layers are compromised. PUFs utilize inherent manufacturing variations in silicon to generate unique and unclonable hardware fingerprints for secure authentication (Khan et al., 2024). By shifting essential security functions to hardware, IoT systems achieve greater resistance to tampering and side-channel attacks, thereby strengthening the overall trust framework across networks. PUFs have gained significant attention for their ability to provide strong security primitives such as device authentication, secure key generation, and support for cryptographic protocols. By exploiting inherent manufacturing variations in hardware components, PUFs generate unique device-specific identifiers, thereby adding a layer of protection against a wide range of attacks (Balan et al., 2020). Field-Programmable Gate Arrays (FPGAs) significantly enhance hardware security in IoT systems and enable customized security architectures that can evolve even after deployment. Unlike application-specific integrated circuits (ASICs), FPGAs allow security mechanisms to be designed, updated, or modified in the field to adapt to new threats. They are well-suited for implementing PUF-based security architectures by leveraging manufacturing process variations to generate intrinsic, unclonable

"fingerprints" that serve as hardware identifiers. These identifiers can be used for authentication, secure key generation, and protection against cloning or spoofing. Thus, FPGA-based IoT devices improve resilience against physical tampering and offer more adaptive security than is typically possible with static hardware (Babaei et al., 2022).

1.2 PROBLEM STATEMENT AND ITS SIGNIFICANCE

The rapid proliferation of IoT devices has significantly expanded the digital attack surface, creating an urgent need for lightweight and reliable authentication mechanisms that do not rely on traditional secret key storage. PUFs have emerged as a promising solution due to their unique hardware signatures derived from inherent manufacturing variations. However, PUF implementations on FPGAs face several critical challenges. Environmental factors such as temperature, supply voltage fluctuations, and device aging reduce the stability of PUF responses, while noise introduced during response generation further undermines their reliability. Moreover, the increasing sophistication of ML modelling attacks allows adversaries to accurately predict PUF responses when large sets of Challenge Response Pairs (CRPs) are intercepted, threatening the security of PUF-based systems. Additionally, FPGA-based implementations are susceptible to timing misalignments and acquisition noise when CRPs are extracted using tools such as logic analysers, which complicates data processing and degrades authentication accuracy (Lata & Cenkeramaddi, 2023).

Therefore, the core research problem is how to design and implement a PUF-based authentication system on FPGAs that is synchronized with the on-board clock, incorporates effective CRP preprocessing techniques, and achieves high reliability, uniqueness, and resistance against ML modeling attacks, while still meeting the resource and performance constraints of IoT deployments. The significance of this research lies in addressing these challenges by proposing a robust calibration and preprocessing framework for FPGA-based PUFs (Roy et al., 2024). By improving the quality of CRP data, the system can achieve higher levels of reliability, uniqueness, and resistance against ML-based modelling, making it suitable for real-world IoT security applications (Ishak et al., 2022). Furthermore, exploring lightweight techniques for synchronization, preprocessing, and authentication ensures that the proposed system

can be feasibly deployed on resource-constrained IoT devices without excessive computational or energy overhead. The outcomes of this research are expected to contribute to more secure IoT ecosystems by enhancing device authentication, preventing cloning, and supporting cryptographic key generation in edge computing environments (Korona et al., 2024).

1.3 RESEARCH OBJECTIVES

This research focuses on the design and implementation of a robust FPGA-based PUF authentication system that ensures high reliability, uniqueness, and strong resistance against machine learning (ML) attacks. The specific Objectives of the study are:

i. To design and implement a novel synchronized PUF architecture incorporating an advanced design approach to preserve intrinsic randomness while ensuring synchronized operation in FPGA based edge devices within an IoT network.
ii. To mitigate the proposed PUF architecture's vulnerability to ML modeling attacks.
iii. To evaluate and benchmark the performance of the proposed PUF architectures against other architectures in the open literature.

1.4 RESEARCH HYPOTHESIS AND ASSUMPTIONS

PUFs are primarily used for device authentication and secure key generation. Among them, Arbiter PUF-based security systems are a preferred choice for edge computing devices due to their lightweight design, scalability, and unclonable characteristics. This approach eliminates the need for storing cryptographic keys by leveraging the inherent hardware uniqueness of the device. In addition, the reduced cost of computational resources leads to lower power consumption, making Arbiter PUFs suitable for implementation in both Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs).

Furthermore, the integration of AI-based techniques with PUFs enhances their reliability, scalability, and robustness. Machine learning methods can stabilize noisy

PUF outputs, resist modelling attacks, and enable adaptive authentication. Based on this, the research hypothesis is that optimal calibration and preprocessing of challenge–response pairs (CRPs) from FPGA-based Arbiter PUFs can significantly improve the reliability, uniqueness, and randomness of PUF responses, while also reducing vulnerability to ML-based modelling attacks.

The research work presented in this thesis is based on the following assumptions:

- *Device-Level Uniqueness:* FPGA-based PUF circuits exhibit unique physical characteristics arising from uncontrollable manufacturing process variations, such as gate length, oxide thickness, and dopant concentration. Replicating these atomic-scale variations is not economically feasible.
- *Environmental Stability Through Calibration:* Hashing and whitening based calibration techniques statistically reduce bias, decorrelate response bits, and transform stabilized outputs into more uniform representations, thereby improving the reliability of challenge response pairs (CRPs).
- *Logic Analyzer Accuracy:* PUF responses are digital signals whose stability can be evaluated through repeated measurements. As logic analyzers offer adequate timing accuracy and reliable logic level detection, they are suitable for capturing and analyzing challenge response pairs (CRPs) for noise filtering and reliability assessment.
- *Validity of PUF-ML Attack Model:* It is assumed that adversaries may gain access to CRPs for training machine learning models, but do not know the internal FPGA design details.
- *Statistical Randomness Metrics:* Successful performance on the NIST Statistical Test Suite (STS) indicates that CRP data exhibits high entropy, low bias, and minimal correlation. However, resistance to machine learning attacks must be further validated by assessing the prediction accuracy of trained models on unseen CRPs.

1.5 RESEARCH QUESTION

Conventional security mechanisms that rely on storing cryptographic keys in non-volatile memory are increasingly vulnerable to physical and side-channel attacks.

Moreover, these approaches have proven insufficient for securing IoT devices, as demonstrated by the growing prevalence of malicious hardware-based attacks. In contrast, Physically Unclonable Functions (PUFs) offer a lightweight hardware security primitive with significant potential in applications such as random number generation, key derivation, and device authentication. Against this backdrop, the central research question of this research is:

How can FPGA-based PUF architectures be designed and calibrated to achieve enhanced reliability, uniqueness, and resistance to machine learning–based modeling attacks, while ensuring scalability and suitability for IoT and edge computing environments?

To address this research problem, the following sub-questions are considered:

i. *Uniqueness:* What design strategies can be employed to maximize device-level uniqueness across FPGA-based PUF instances?
ii. *Security:* To what extent can FPGA-based PUF designs resist machine learning–based modeling attacks?
iii. *Benchmarking:* How does the proposed FPGA-based PUF architecture perform in terms of reliability, uniqueness, uniformity, and resistance to machine learning attacks compared to existing PUF designs reported in the literature?

1.6 RESEARCH SCOPE

This research focuses on the design, acquisition, calibration, and evaluation of Physically Unclonable Functions (PUFs) implemented on FPGAs for IoT security applications. The scope is defined as follows:

- PUF Architecture and FPGA Implementation
  - (i) The research work will be limited to Intel Altera Cyclone IVE FPGA-based PUF designs (e.g., Arbiter PUF, Ring Oscillator PUF).
  - (ii) Hardware-based experiments will be conducted to extract Challenge–Response Pairs (CRPs) using a logic analyzer.

- Data Acquisition and Calibration
  (i) CRP collection will be performed under variable environmental conditions (room temperature, supply voltage, and device aging only).
- Performance Evaluation
  (i) PUF metrics like reliability, uniqueness, randomness, and correctness will be computed before and after calibration.
  (ii) Randomness will be validated using the NIST Statistical Test Suite (STS).
- Machine Learning Analysis
  (i) ML models (e.g., logistic regression (LR), Multi-Layer Perceptron (MLP), random forest (RF), and gradient boosting (GB)) will be trained on the CRPs to evaluate modeling resistance.
  (ii) The effect of preprocessing and calibration on ML prediction accuracy will be analyzed.
- Limitations
  (i) The research does not cover ASIC-based PUFs or large-scale commercial deployment.
  (ii) Only selected PUF architectures and ML models will be tested due to resources and time constraints.

1.7 THESIS METHODOLOGY

The research methodology, illustrated in Figure 1.2, begins with a narrative literature review of existing PUF architectures, performance metrics, and vulnerabilities, which leads to the identification of research gaps in FPGA-based PUF designs. Two architectures, Ring Oscillator PUF (RO-PUF) and Arbiter PUF (APUF), were designed and implemented using VHDL on Altera Cyclone IVE FPGA hardware to generate Challenge–Response Pairs (CRPs). The extracted CRPs were evaluated for key metrics such as uniqueness, reliability, and uniformity, and their randomness was validated using the NIST Statistical Test Suite. Security robustness was assessed through machine learning modelling attacks (e.g., Logistic Regression, Support Vector Machine, Random Forest, Gradient Boosting, and Multi-Layer Perceptron) to measure

prediction resistance. Based on the performance and attack outcomes, design modifications and enhancements were introduced to improve entropy, reliability, and resilience against modelling attacks. The enhanced architectures were then benchmarked against existing PUF designs from the literature in terms of performance metrics, randomness quality, and FPGA resource utilization.

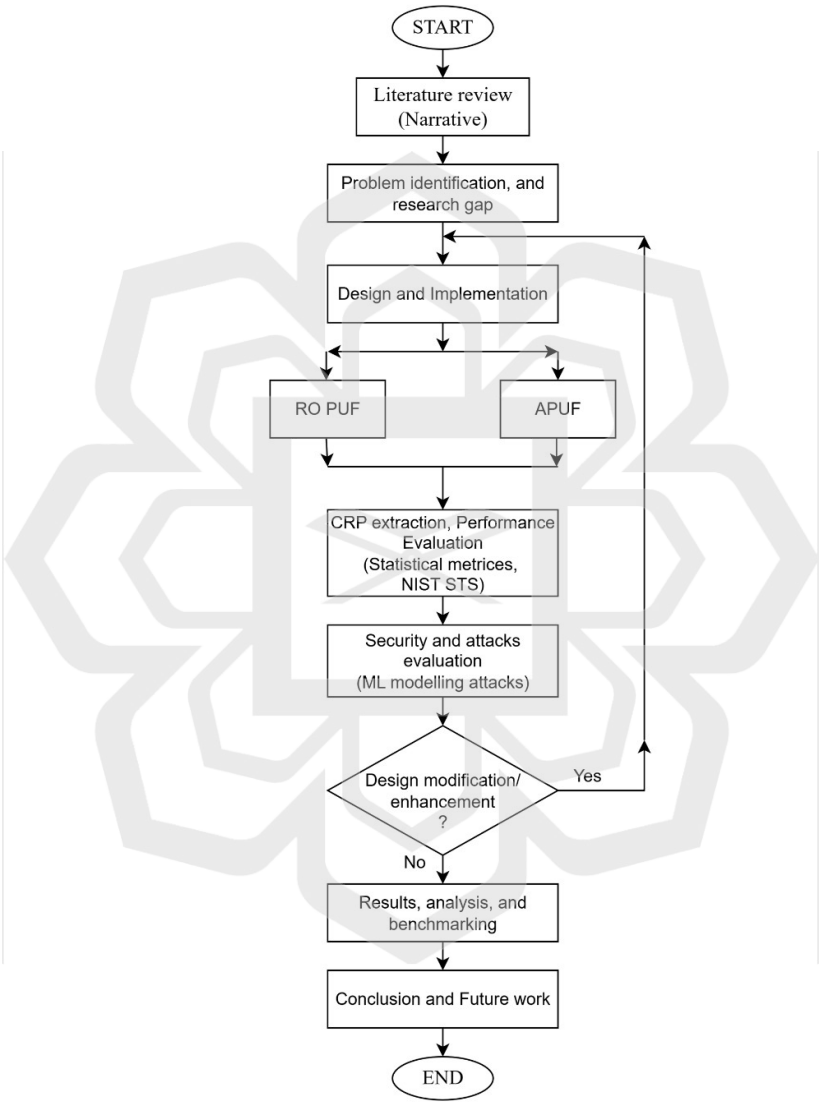


Figure 1.2 System Flowchart

The methodology concludes with a comparative analysis of both baseline and proposed PUF architectures, highlighting improved performance and security, and suggesting directions for future research.

1.8 THESIS ORGANIZATION

This thesis is organized into six chapters, each addressing a specific aspect of IoT security using FPGA based Physically Unclonable Functions. Chapter 1 introduces the IoT and highlights both the opportunities and challenges associated with edge computing, with a particular focus on security concerns. It also presents mitigation strategies such as lightweight encryption, hardware based security primitives, and Physically Unclonable Functions, while introducing the concept of machine learning resistant PUF implementations on FPGA platforms. Chapter 2 provides a comprehensive literature review, beginning with IoT architecture and applications, followed by a detailed analysis of security threats across device, network, and application layers. It explores PUFs as hardware security primitives, including their classifications, and discusses the role of Artificial Intelligence in strengthening IoT security. The chapter also examines the working principles of Arbiter PUFs and Ring Oscillator PUFs, forming the foundation for the proposed work.

Chapter 3 outlines the research methodology, describing the systematic approach used for the design, implementation, and evaluation of FPGA based PUF architectures. It explains the selection of tools such as Intel Quartus, Visual Studio Code, and the NIST Statistical Test Suite, along with the use of the Cyclone IV E FPGA platform and logic analyzer for CRP acquisition. The chapter also introduces preprocessing techniques such as hashing and whitening to enhance reliability and reduce noise. Chapter 4 focuses on the design and implementation of the proposed schemes, detailing the FPGA realization of Arbiter and Ring Oscillator PUFs using VHDL. It describes the complete design flow from challenge input to response generation, including preprocessing and CRP extraction, and presents the machine learning modeling framework used to analyze PUF behavior.

Chapter 5 presents the results and performance evaluation, analyzing the effectiveness of various machine learning algorithms such as Logistic Regression, Support Vector Machines, Random Forests, and Neural Networks in predicting PUF responses. It evaluates the impact of preprocessing and calibration on reducing modeling accuracy and discusses trade offs between security, reliability, and system overhead, demonstrating the superiority of the proposed schemes over conventional

designs. Finally, Chapter 6 concludes the thesis by summarizing the key contributions in PUF design, calibration, and resistance to machine learning attacks. It also identifies limitations and suggests future research directions, including advanced PUF architectures, improved lightweight defense mechanisms, and scalable deployment in large scale IoT environments.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter reviews the existing body of research on the rapid growth of the Internet of Things (IoT), highlighting both its transformative potential and the challenges that accompany its widespread adoption. It examines issues such as scalability, interoperability, limited resources, and latency, with particular emphasis on the pressing security and privacy concerns that arise in large-scale IoT deployments. The survey further explores proposed solutions, focusing on hardware-based primitives like Physically Unclonable Functions (PUFs), which leverage intrinsic circuit variations for device authentication and secure key management. In addition, it discusses the dual role of Machine Learning in this domain, both as a tool used by adversaries to model and attack PUFs, and as a defense mechanism to enhance reliability, detect anomalies, and reinforce IoT security frameworks.

2.2 INTERNET OF THING (IOT)

IoT represents an interconnected digital ecosystem in which physical objects, embedded with sensors, actuators, communication interfaces, and computational intelligence, are uniquely identifiable, network enabled, and capable of autonomous data exchange, monitoring, and remote control (Bhuiyan et al., 2021). The International Telecommunication Union (ITU) has formally defined IoT as, ‘‘A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies (ITU, 2012).’’ Researchers have predicted that millions of new IoT devices will be deployed annually across various application domains. Figure 2.1 presents an estimation of the number of IoT-connected devices by the year 2034, with over 40 billion connected devices worldwide (Vailshery, 2025). Cisco has predicted that there will be approximately 500 billion devices connected to the IoT network by 2030.

Additionally, Telefonica estimates that about 90% of vehicles will be IoT-enabled, and each individual will own an average of 15 connected devices by that time. IoTs are transforming the world into a more innovative, always accessible environment with seamless connectivity. As a result, IoT integrates appliances, services, sensors, actuators, and more to enable efficient communication and data exchange. In addition, it processes real-time data, enhancing system performance and operational efficiency (Zubaydi et al., 2023).

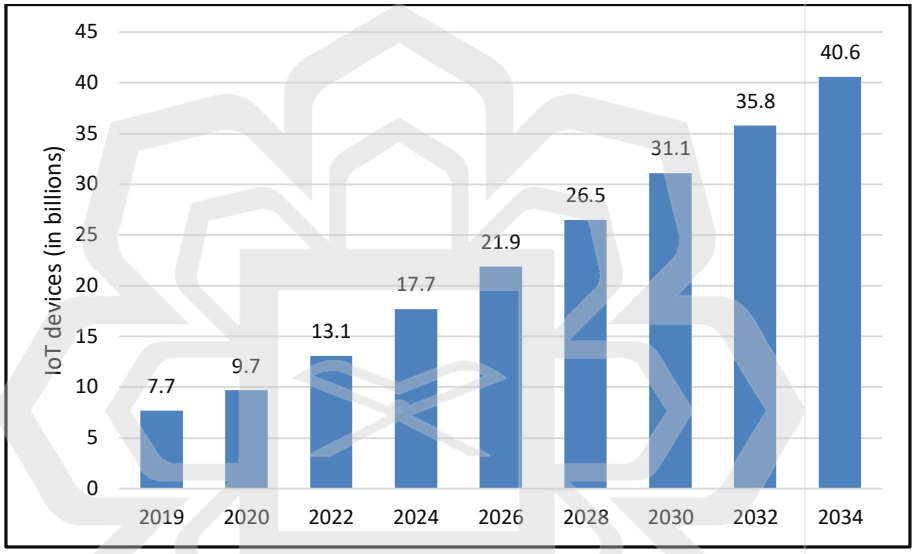


Figure 2.1 Projected IoT devices worldwide.

An IoT ecosystem comprises internet-enabled smart devices equipped with integrated systems such as processors, sensors, and communication hardware. The sensing devices transmit their collected data to an IoT gateway or an edge device, which then carries out local data processing or forwards the data to the cloud. The connectivity, networking, and communication protocols used by these smart devices vary based on the specific IoT application (Laghari et al., 2021).

2.3 ARCHITECTURE OF IOT

The architecture of the IoT refers to a structured framework of interconnected components that allows an IoT system to function effectively and seamlessly. It defines the layers, elements, and their interactions, illustrating the flow of data from the device

layer up to the application layer. The layered, modular design promotes interoperability and scalability by decoupling physical devices from network technologies and application logic, while embedding comprehensive management and security across the architecture. At the heart of the IoT network are the devices, i.e., the "things", which include physical components such as sensors and actuators. Currently, there is no universally accepted standard for IoT platforms and devices. Instead, a variety of standards and technologies are employed across different systems. As a result, deploying an IoT system often requires users or service providers to analyse, configure, and integrate multiple architectures and technologies. This process can be both complex and time-consuming. To address these challenges, several leading standardization bodies in IoT and communication networks have initiated efforts to develop unified technical specifications. Noteworthy among these are the IEEE Standard for an Architectural Framework for the IoT, the ITU-T Y.2060 Recommendation, the International Organization for Standardization / International Electrotechnical Commission (ISO/IEC), the European Telecommunications Standards Institute (ETSI), and the 3rd Generation Partnership Project (3GPP) (Laghari et al., 2021).

The adoption of a unified IoT reference model is important for fast-tracking the development, deployment, and widespread adoption of IoT services and applications. A common reference IoT model forms a standard framework that promotes interoperability, simplifies system integration, and reduces development complexity. Thus, stakeholders like device manufacturers, software developers, service providers, and policymakers stick to consistent communication protocols, data formats, and security practices across various IoT platforms (Heidari & Jamali, 2023). A typical IoT architecture consists of multiple layers and components, each with specific functions. These layers range from physical devices and data acquisition systems to networking elements that transmit IoT data, as well as applications that handle data processing and storage. The simplest form of this architecture is the three-layer model, illustrated in Figure 2.2, which includes the perception layer, the network layer, and the application layer (Jahangeer, 2023; Kenaza et al., 2022).

- The perception layer is also called the device layer. It is responsible for sensing and collecting its environmental data, such as motion, vibration, temperature, and humidity. It then forwards this information to the next layer, which securely delivers it to the data processing system. Sensors,

functioning as data acquisition devices, detect environmental information and translate it into protocol-compliant signals, thereby streamlining data transmission. In essence, the perception layer serves as both the origin and gateway of all data.

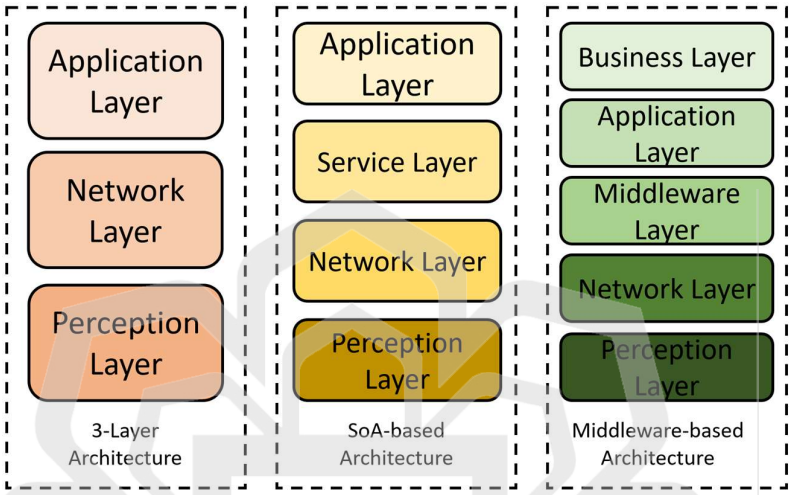


Figure 2.2 Common IoT architectures

- The network layer is also called the transmission layer. It ensures secure delivery of data from the perception layer to the application layer using various communication technologies such as RFID, 5G, Wi-Fi, Zigbee, Bluetooth, and NFC. It links smart devices, network hardware, and servers, facilitating both the transmission and preliminary processing of sensor data. This layer supports a variety of network protocols that define the rules and modes for data transfer, guaranteeing that information is sent safely, reliably, and efficiently.
- The application layer is responsible for providing user-oriented, application-specific services. It defines the various domains where IoT services can be implemented, such as smart homes, smart cities, and innovative healthcare. As a crucial top-layer component of the IoT architecture, it distinguishes and manages the different applications in which IoT systems are deployed. Additionally, it is tasked with delivering tailored services to each application based on the data collected from sensors.

- The middleware layer serves as an intermediary software interface, handling high-capacity data transfers between the sensing and service layers. It consolidates data from the foundational infrastructure before passing it on to higher layers. Additionally, it abstracts the complexity and diversity of the lower-layer technologies, simplifying interaction for both users and developers (Kenaza et al., 2022).
- The business layer supervises the entire IoT ecosystem, including its activities, services, and applications, while converting technical outputs into actionable business insights. By analysing these outputs, it guides decision-making for strategies and roadmaps, ensuring that IoT implementations generate tangible value. This layer employs business intelligence to uncover opportunities and evaluates each layer's performance to continuously optimize services (Pillai et al., 2021).

2.4 APPLICATIONS OF IOT

IoT foster the development of industry-specific and user-centric applications. These applications must ensure that data or messages are received, processed, and acted upon accurately and in real time, especially in critical situations. IoT applications must be designed with a certain amount of built-in intelligence, enabling devices to monitor their surroundings, detect issues, exchange information with each other, and, in some cases, autonomously fix operational issues without human intervention (Lee & Lee, 2015). Regardless of their specific functions, IoT applications aim to authorize users to make informed decisions, seize the best opportunities, deliver intelligent services, and ultimately enhance people's quality of life.

The IoT European Research Cluster (IERC) envisions IoT primary objective as building innovative environments and self-aware devices, such as intelligent transportation systems, products, cities, buildings, rural areas, energy networks, healthcare, and living spaces to address challenges in climate, food, energy, mobility, the digital society, and health (Davies, 2015). IoT applications are developed to meet the specific requirements of numerous domains and environments, each with its own set of requirements and deployment architectures. For example, these include logistics and supply chain management, healthcare, environmental monitoring, smart homes/buildings, and smart agriculture (Ajana El Khaddar, 2021). Exploring IoT

applications deepens the understanding and advancement of IoT technology, guiding the creation of new systems for emerging use cases. At its core, IoT involves collecting everyday data from one object and transmitting it to another. This object-to-object communication significantly broadens the scope of IoT applications, making them diverse, dynamic, and virtually limitless (Hassan et al., 2020). Libelium listed fifty different sensors while exploring the future of IoT and its revolutionary solutions within the verticals of smart cities, sustainability, and agrifood (Libelium, n.d.). 2025). A smart city is defined as one that implements essential services such as administration, education, healthcare, public safety, real estate, transportation, and utilities using advanced information and communication technologies.

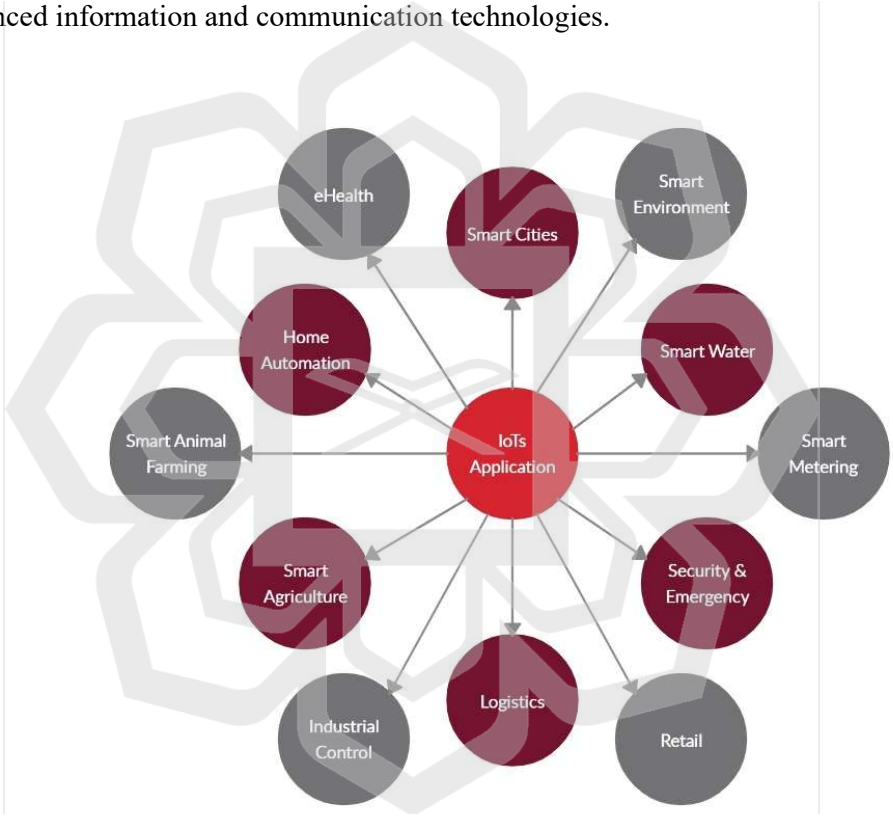


Figure 2.3 IoT application domains

Figure 2.3 presents a comprehensive classification of IoT applications across twelve key domains. The main areas of focus include healthcare, environmental monitoring, smart cities, and the commercial, industrial, and infrastructure sectors. IoT applications offer versatile solutions that enable businesses to streamline, enhance, automate, and manage their operations more optimally. Consequently, these applications support the creation of innovative business models, open new revenue

opportunities, and supply the real-time data essential for developing products and services (Arm, n.d.).

2.5 CHALLENGES OF IOT

The rapid growth and integration of IoT with other technologies have significantly expanded the attack surface. Traditional security mechanisms such as encryption, authentication, and access control are often not feasible for IoT devices due to their limited resources and deployment in unattended or vulnerable environments (Malhotra et al., 2021). Key features and challenges associated with IoT are outlined below:

2.5.1 Heterogeneity

Due to the broad range of application domains covered by the IoT, the ecosystem includes a diverse array of devices, communication protocols, network connectivity methods, and application models. Some of these devices or protocols are legacy infrastructures that rely on proprietary technologies. In contrast, others embrace open standards such as Message Queuing Telemetry Transport (MQTT) and the protocol stack developed by the Internet Engineering Task Force (IETF), which are tailored for constrained IoT devices. This heterogeneity often leads to the formation of vertical silos, which in turn hampers the development of cross-domain, value-added services that leverage these low-resource IoT devices (Van den Abeele et al., 2015; Noaman et al., 2022). Software-Defined Networking (SDN), network virtualization, and the deployment of gateways offer practical solutions for managing the heterogeneity present in large-scale IoT networks.

2.5.2 Scalability

IoT applications should be able to accommodate a growing number of users and computing environments, thus enabling them to manage a high volume of user requests across an expanded physical or virtual space needed for data sensing, aggregation, and processing (Javed et al., 2020). It has been shown that different architectural choices of IoT platforms affect system scalability (Zyrianoff et al., 2018). IoT applications are typically built around three core components: hardware, software, and connectivity. The IoT infrastructure, cloud services, and network connectivity must scale alongside IoT

deployment. A scalable IoT architecture can efficiently accommodate the addition of new sensors, devices, and users while maintaining optimal performance. Key approaches to achieving scalability include:

i. Processing and interpreting a large volume of increasing data through the application of ML techniques.
ii. Adopting decentralized processing techniques like edge and fog computing.
iii. Implementing protocols like IPv6 and 6LoWPAN to enable addressability of a large volume of IoT devices.
iv. Leveraging virtualization and peer-to-peer networking to enhance the flexibility and scalability of the IoT infrastructure.

2.5.3 Energy Consumption

IoT devices are typically compact and powered by on-board energy sources, often relying on various types and sizes of batteries depending on the application requirements (Ma et al., 2020). However, battery-based power solutions can be impractical and costly due to the need for regular recharging or replacement. As a result, there is ongoing research into developing battery-less and perpetual IoT devices using energy harvesting (EH) techniques. A typical IoT device consists of a microcontroller unit (MCU), a radio transceiver, sensors or actuators, a power management unit (PMU), and a supercapacitor. The PMU charges the capacitor using ambient energy sources such as solar, kinetic, thermal, or radio frequency (RF) radiation (Van Leemput et al., 2023). An IoT infrastructure generally comprises three major components: IoT devices, communication protocols, and data storage and processing systems. One of the key strategies to reduce energy consumption is the adoption of lightweight communication protocols tailored to the needs of specific applications. Common examples include Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), and Data Distribution Service (DDS) (Bayılmış et al., 2022).

2.5.4 Privacy and Security

Network security and data protection measures must adhere to the core principles of integrity, authentication, availability, authorization, and confidentiality for users' data (Deep et al., 2022). As more devices connect to shared networks, the likelihood of data breaches, malware infiltration, and digital theft increases (Zaheeruddin & Gupta, 2020). Malware, viruses, and cyberattacks can all compromise data integrity, so IoT architectures, protocols, and implementation strategies must embed security from the outset. Authenticating devices across diverse, interconnected protocols is particularly challenging, especially given the strict constraints of many IoT endpoints—limited energy budgets, small memory footprints, and low processing power must all be accounted for (Azrour et al., 2021). IoT security threats span a broad spectrum, including physical attacks (where an adversary gains proximity to a device), network attacks (manipulating network traffic or infrastructure), software-based attacks (exploiting application vulnerabilities), data attacks, side-channel attacks, cryptanalysis attacks, access-level attacks, and strategy-level attacks (Aqeel et al., 2022). Encryption can also be targeted through side-channel techniques, cryptanalysis, or man-in-the-middle exploits aimed at undermining cryptographic protections (Yang et al., 2017).

Zero Trust Architecture (ZTA) embraces a "never trust, always verify" ethos, treating all users and devices—inside or outside the network perimeter—as untrusted until they pass rigorous checks (Al Tamimi et al., 2024). In this model, blockchain technology underpins user authorization, providing a transparent, decentralized ledger that records every authentication and verification event with immutable timestamps. Physical Unclonable Functions (PUFs), particularly Ring Oscillator PUFs (RO-PUFs), are employed to authenticate IoT devices and safeguard critical assets, such as FPGA bitstream files, from tampering. By combining blockchain's verifiable audit trail with the hardware-rooted security of RO-PUFs, the ZTA implementation strengthens the IoT ecosystem against a wide array of threats, ensuring device integrity and minimizing risk (Kulkarni et al., 2024).

2.5.5 Data Management and Analytics

The widespread adoption of IoT depends on highly efficient big-data analytics, yet managing the vast volumes of information it generates presents significant challenges. Gartner's "three Vs" framework—volume, variety, and velocity—captures the essence of these challenges. Within an IoT ecosystem, data management serves as a middleware layer that gathers, stores, processes, and analyzes streams from billions of connected "things" before they reach consuming applications. Traditional database systems struggle to meet IoT unique demands like geographical dispersion, heterogeneous data flows; support for diverse storage formats; and dynamic routing to final storage or processing nodes (Jeba & Rathi, 2021). Effective big-data analytics in IoT, therefore, requires real-time processing engines and storage solutions built on a range of underlying technologies (Marjani et al., 2017). To unlock the full potential of IoT data, researchers have deployed a variety of algorithms—smart-city prediction models, data-mining techniques, streaming analytics frameworks, and deep-learning networks (Sasaki, 2022).

2.6 IOT SECURITY

Security focuses on preventing unauthorized access to data, while privacy is concerned with safeguarding users' personally identifiable information. It is possible to have security without privacy; however, privacy cannot exist without security. The rapid growth and integration of IoT with various technologies have significantly expanded the attack surface for potential adversaries. Traditional security measures such as encryption, authentication, and access control are often inadequate for IoT systems, which involve a vast number of connected devices. These limitations arise from inherent vulnerabilities, including resource constraints and the deployment of devices in unattended or physically exposed environments (Malhotra et al., 2021). The rapid and widespread adoption of IoT devices, coupled with their growing computational capabilities across diverse application domains, has introduced numerous vulnerabilities within IoT ecosystems. These environments often suffer from a lack of transparency, poor integration of security measures, reliance on vulnerable open-source code, unpatched software flaws, insecure Application Programming Interfaces (APIs), and inadequate testing. To safeguard IoT infrastructures from unauthorized access, data

tampering, or loss, comprehensive cybersecurity measures must be implemented throughout every stage of the IoT lifecycle (Tariq et al., 2023).

A typical IoT application can be divided into four layers: (i) sensing layer; (ii) network layer; (iii) middleware layer; and (iv) application layer. Each of these layers is implemented on diverse technologies, consequently opening numerous issues and security challenges (Baniya et al., 2024). Figure 2.4 illustrates possible attacks on these four layers as well as security concerns associated with the gateways that connect these layers (Mazhar et al., 2023).

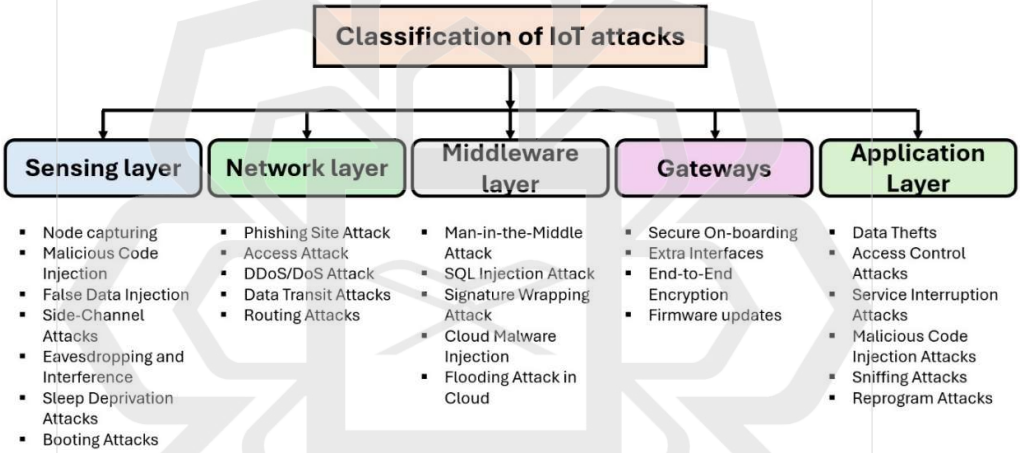


Figure 2.4 Classification of IoT attacks

Physical devices in IoT networks possess six distinct characteristics: spatiotemporal inconsistency, coexistence of multiple identities, high heterogeneity, resource constraints, dynamic behaviour, and social awareness. These features give rise to unique security and privacy challenges that traditional protection mechanisms are often ill-equipped to address. Existing security architectures typically concentrate on threats within the perception, network, and application layers (Rao & Deebak, 2023). Security challenges exist across all layers of IoT architectures, regardless of the specific implementation. Each threat must be addressed with an appropriate security solution tailored to its respective layer. At the application layer, common threats include data theft, malware propagation, and botnet attacks. The network layer is particularly vulnerable due to weaknesses in existing protocols and standards, making it susceptible

to denial-of-service (DoS) attacks, session hijacking, and data corruption. In cloud computing environments, a broad range of security risks arises from issues such as misconfiguration, poor identity management, inadequate data access controls, infrastructure vulnerabilities, and violations of data privacy. At the edge layer, devices with limited computational resources and fixed physical locations are exposed to various threats, including node sabotage, failure or disconnection, offline data harvesting, false message injection, resource exhaustion, Sybil attacks, jamming, and physical tampering (HaddadPajouh et al., 2021).

2.6.1 Security Mechanisms for IoT

It is observed that most IoT security solutions in recent times favor software-centric approaches over conventional tool-centric ones. The security of IoT devices with limited resources requires an optimal approach that considers both technical limitations and security needs. IoT devices with constraints, such as memory, battery charge, and energy consumption, require strategies that enhance the security levels (Rozlomii et al., 2024).

- *Lightweight Cryptographic Algorithms:* A lightweight protocol or cryptographic algorithm, as defined in ISO/IEC 29192, is designed for use in constrained environments such as healthcare devices, sensors, RFID tags, and contactless cards. These algorithms can be implemented in either software or hardware. A lightweight software implementation is characterized by an optimized code size and minimal RAM usage, while energy consumption and chip size are the key considerations in hardware implementations. The four main categories of lightweight cryptographic methods include Lightweight Block Ciphers (LWBC), Lightweight Stream Ciphers (LWSC), Hash Functions, and Elliptic Curve Cryptography (ECC) (Pandey & Bhushan, 2024).
- *Efficient Energy Management Algorithms:* Energy management is a critical aspect of IoT networks, as many edge devices, such as sensors, are battery-powered and have a limited operational lifespan. Accurate power modelling of IoT networks is essential for effective energy management. Data routing protocols, along with the routing process between source and destination nodes, significantly influence overall power consumption. Implementing

algorithms that minimize energy usage without compromising security can extend device runtime and enhance the reliability of security systems (Albreem et al., 2021).

- *Memory Usage Optimization:* Memory management involves the allocation and sharing of memory space among processes, allowing multiple programs to use the exact memory location, though only one at a time. Techniques for optimizing limited memory resources include compact storage of encryption keys and efficient use of memory for security functions (Comeagă & Marin, 2023).
- *Adaptive Security Mechanisms:* Adapting to changing resource constraints is essential to maintain an optimal level of protection under varying operating conditions. Adaptive security continuously monitors threats and evolves in response to shifting cybersecurity risks. AI algorithms analyse user behaviour patterns to enable continuous authentication, creating a dynamic and adaptive authentication process. This self-defending capability enhances the resilience and responsiveness of IoT systems, reducing the likelihood of successful cyberattacks and minimizing the impact of security breaches (Humayun et al., 2024).
- *Improvement of Authentication Algorithms:* Authentication is the process of verifying a device's identity, while authorization involves permitting it to access resources. Only authorized devices are allowed to communicate with other devices, gateways, cloud accounts, and applications. Traditional authentication and key management protocols are not entirely suitable for IoT environments. Public key cryptography, while effective, is often impractical in this context due to its reliance on a Trusted Third Party (TTP) and its high computational demands (Bin Rabiah et al., 2021). Instead, authentication protocols often use cryptographic hash functions and XOR operations to verify device identities and transmitted data securely. Among the most widely used advanced cryptographic primitives are Elliptic Curve Cryptography (ECC) and Physically Unclonable Functions (PUFs) (Cetintav & Sandikkaya, 2025).
- *Secure Communication Protocols:* Communication protocols are sets of rules that ensure reliable data exchange between IoT devices and

applications. These rules define syntax, semantics, synchronization, and error recovery mechanisms. IoT devices use a variety of protocols, such as MQTT, CoAP, HTTP/HTTPS, and proprietary solutions. Protocols tailored to meet specific application requirements include Representational State Transfer (REST), Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), Data Distribution Service (DDS), Extensible Messaging and Presence Protocol (XMPP), and Simple Object Access Protocol (SOAP) (Dragomir et al., 2016).

Table 2.1 Mitigation strategies against IoT Security threats

IoT Layer	Threats	Mitigation Strategy
Perception Layer	Node capture, Fake nodes, Physical damage, Malicious code injection, and Data manipulation.	1. Physical security and tamper resistance 2. Device authentication and identity management 3. Lightweight cryptography for resource-constrained devices 4. Secure Firmware and Software Updates 5. Intrusion detection and anomaly monitoring 6. Redundancy and backup
Network Layer	DoS attacks, Eavesdropping, Routing attacks, Session hijacking, Sybil attacks.	1. Encryption and secure communication protocols 2. Authentication and access control 3. Intrusion Detection and Prevention Systems (IDS/IPS) 4. Secured routing protocols 5. Protection against wireless attacks 6. Session and key management

Application Layer	Data breaches, Malware injection, Botnet attacks, and Unauthorized access.	1. Access control and authentication 2. Secured API design 3. Data protection 4. Malware and botnet prevention 5. Privacy protection 6. Security monitoring and incident response
Cloud Layer	Misconfiguration, Data leakage, Insecure interfaces, Lack of identity control.	1. Identity and Access Management (IAM) 2. Data security 3. API and Interface Protection 4. Infrastructure security 5. Monitoring and incident response 6. Compliance and governance
Edge/Fog Layer	Node failure, False data injection, Sybil attacks, Jamming, Physical tampering.	1. Redundancy and fault tolerance 2. Anomaly and intrusion detection 3. Secure node authentication 4. Data integrity and confidentiality 5. Physical security measures 6. Lightweight Blockchain and Distributed Ledger Technologies 7. Jamming and interference mitigation

Table 2.1 presents mitigation strategies against security threats across different IoT layers (Deep et al., 2022; Conti et al., 2018; Zhang et al., 2018; Mosenia & Jha, 2017). At the perception layer, security can be strengthened through tamper-resistant hardware, secure physical deployment, mutual authentication using unique hardware-based identifiers, and lightweight cryptographic techniques to ensure data confidentiality and integrity. Devices should also support secure firmware updates with digital signature verification and a trusted boot process. Additionally, intrusion detection and anomaly monitoring help identify suspicious activities, while redundant and fault-tolerant architectures ensure uninterrupted services.

At the network layer, security is achieved through end-to-end encryption and lightweight communication protocols that enable mutual authentication and robust access control. Intrusion detection and prevention systems are employed to monitor and block malicious activities. Efficient routing protocols utilize frequency hopping and spread spectrum techniques to counter jamming attacks. At the same time, intense sessions and key management mechanisms protect against threats such as DoS, eavesdropping, Sybil attacks, routing manipulation, and data corruption (Alaba et al., 2017). Application layer threats are mitigated by implementing strong authentication and access control mechanisms, as well as securing APIs through validation. Data protection is achieved using encryption, hashing, and secure key management. Regular updates, along with malware detection and intrusion detection systems, prevent the entry of botnets and malicious payloads into the network. Data minimization, anonymization, and adherence to regulatory compliance achieve privacy through continuous monitoring and incident response. At the cloud layer, identity and access management are enforced through multifactor authentication and least-privilege policies to secure data during transmission and processing. At the same time, APIs are protected with strong authentication, encryption, and input validation. The cloud infrastructure is safeguarded through regular patching, network segmentation, and DDoS protection. Compliance with security standards and routine audits further ensures robust governance and maintains the overall resilience of the cloud environment (Zhang et al., 2019; Kolias et al., 2017). At the edge and fog layers, lightweight anomaly and intrusion detection mechanisms are employed to identify attacks. At the same time, strong cryptographic authentication methods such as PUFs are used to verify device identities. User data should be encrypted and integrity-checked before transmission, and physical security measures must be applied to protect devices from tampering. Furthermore, lightweight blockchain solutions tailored for resource-constrained and physically vulnerable edge devices can be leveraged to enhance trust among nodes (Diro & Chilamkurti, 2018; Makhdoom et al., 2019; Yang et al., 2017).

2.6.2 Hardware-Assisted IoT Security

As illustrated in Figure 2.5, HAS is classified based on hardware security primitives, chip-level, and system-level approaches. PUFs are hardware security primitives that map a challenge input to a response output through inherent physical variations in

hardware. This mapping is unique to each hardware instance and cannot be cloned (Kareem & Dunaev, 2021). PUF architecture is typically classified into two categories: strong PUFs and weak PUFs. Strong PUFs provide a large set of challenge–response pairs (CRPs) and can be directly employed for authentication without requiring additional cryptographic hardware.

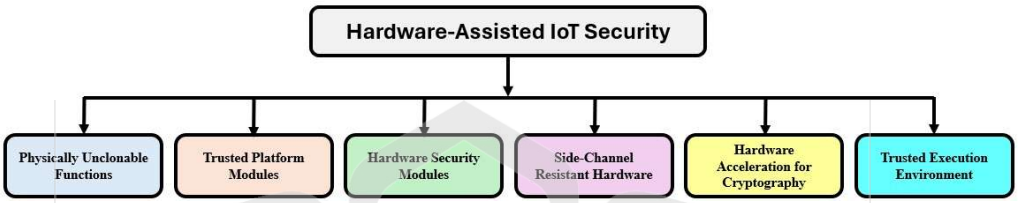


Figure 2.5 Classification of hardware-assisted IoT security

In contrast, weak PUFs generate only a limited number of CRPs, making them more suitable for applications such as cryptographic key generation, seeding pseudo-random number generators (PRNGs), intellectual property (IP) protection, and device identity generation (Anandakumar et al., 2020). PUF circuits can be implemented using various CMOS technologies, most commonly on memory chips, Application-Specific Integrated Circuits (ASICs), or Field-Programmable Gate Arrays (FPGAs).

Trusted Platform Module (TPM): Based on the Trusted Computing Group (TCG) specifications, the Trusted Platform Module (TPM) is a dedicated hardware device designed to provide secure computing. Its architecture ensures that associated data is protected against both software-based and physical attacks. Private keys generated within the TPM are safeguarded using asymmetric cryptographic functions and a hardware-based random number generator, even during usage (Muñoz, 2022). TPM securely stores sensitive information such as user credentials, passwords, fingerprints, certificates, encryption keys, and other critical data behind a hardware barrier, preventing exposure to external threats (Intel, n.d.). Furthermore, it integrates a hardware cryptographic engine that is resistant to side-channel attacks. Due to concerns about sensitive and proprietary design information leakage, TPM processors are typically manufactured in limited quantities (Khan et al., 2024).

Hardware Security Module (HSM): Also referred to as a Tamper-Resistant Security Module (TRSM), an HSM is a dedicated hardware device designed to perform cryptographic functions such as data encryption and decryption, as well as certificate management (Mavrovouniotis & Ganley, 2013). HSMs securely manage digital keys to enable strong data protection and provide additional layers of security through encryption, decryption, authentication, and digital signature services across a wide range of applications. Many HSMs also incorporate physical protection mechanisms, including tamper-proof features that trigger alerts upon interference detection and, in some cases, render the device inoperable to prevent unauthorized access (Amael et al., 2024).

Hardware-Accelerated Cryptography: It is the implementation of dedicated hardware components that perform cryptographic operations more efficiently than software-based solutions. These hardware-based cryptographic coprocessors are faster and consume less energy than software implementations. Hardware acceleration has become a widely adopted methodology to improve the efficiency of computation-intensive tasks, including Machine Learning (ML), Image Processing, and Cryptographic computations by offloading such workloads to specialized hardware. Some standard specialized hardware includes a Graphics Processing Unit (GPU), Field Programmable Gate Array (FPGA), and Application Specific Integrated Circuit (ASIC) (Boeckmann et al., 2022).

Trusted Execution Environment (TEE): A TEE is an isolated environment within a processor that protects sensitive code and data from potentially malicious software, including the kernel. TEEs must meet two fundamental requirements: (i) memory isolation, which ensures that the TEE's address space is separated and protected from the kernel, and (ii) attestation, which provides a mechanism to verify the integrity and authenticity of the code running within the TEE. Implementations such as ARM TrustZone provide a hardware-based secure sandbox that shields sensitive programs from unauthorized access (Yuhala, 2023). In general, TEEs are secure areas of processors or devices that execute code with a higher level of security than the standard operating environment. They are widely used in mobile devices, cloud computing infrastructures, and other embedded hardware platforms (Sommerhalder, 2023).

2.7 EDGE COMPUTING

In 2016, the Edge Computing Consortium was established through a collaboration between leading enterprises and research institutions, including Huawei, Intel, Arm, iSoftStone, the China Academy of Information and Communications Technology, and the Chinese Academy of Sciences (Lu et al., 2023). Edge computing is distinguished by its higher bandwidth, lower latency, and real-time access to network data. The concept was initially introduced by Pang and Tan (Pang & Tan, 2004).

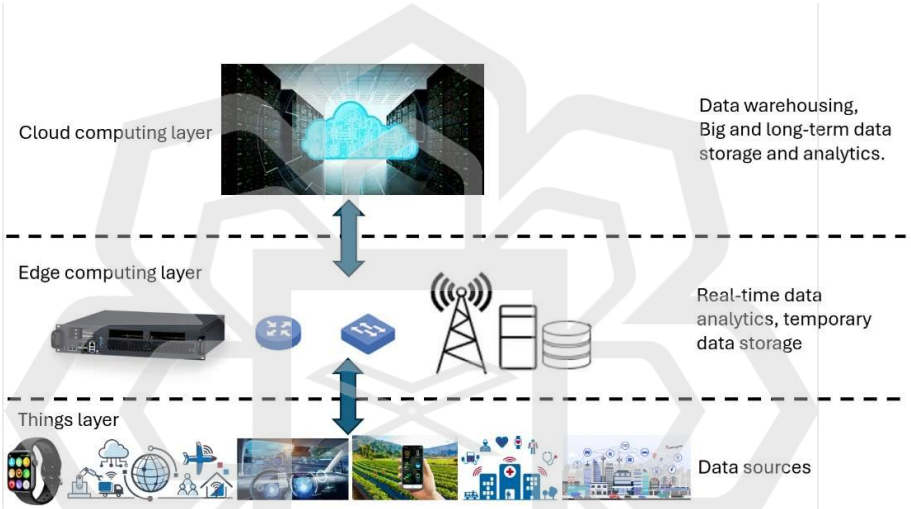


Figure 2.6 Edge Computing architecture

The EC architecture, illustrated in Figure 2.6, is typically organized into three layers: the edge devices layer, the edge nodes/gateways layer, and the cloud layer.

- *Edge Devices layer (Things layer):* This layer consists of IoT sensors, smartphones, cameras, and industrial machines that either generate or collect data. These devices generally have limited processing capabilities and perform only basic operations such as data filtering or preliminary processing. To minimize terminal service delay, only the perception capabilities of these devices are utilized, while computationally intensive tasks are delegated to higher layers.

- *Edge computing layer:* This intermediate layer includes routers, edge servers, and micro–data centers that provide more substantial computing power. They aggregate data, perform localized processing, and enable decision-making closer to the data source. By reducing the volume of data transmitted to the cloud, edge nodes support real-time analysis and intelligent processing more efficiently and securely than relying solely on cloud computing.

- *Cloud computing layer:* The centralized cloud infrastructure comprises high-performance servers and large-scale storage systems. It is responsible for handling complex computations, long-term data storage, and advanced analytics that exceed the capabilities of edge nodes. Cloud computing plays a critical role in applications requiring extensive data analysis, such as predictive maintenance and business decision support (Cao et al., 2020).

A robust network infrastructure leveraging Wi-Fi, 5G, or wired technologies interconnects the three layers to ensure efficient and reliable data transmission. By providing localized computing resources, edge servers enhance scalability and meet the demands of data-intensive applications such as smart cities and autonomous vehicles (Kong et al., 2022). IoT and Edge Computing complement each other in an EC-driven IoT system, where IoT focuses on endpoint sensing, while edge computing focuses on near-field computation. IoT applications benefit from EC-driven IoT systems from three unique aspects (Kong et al., 2022),

- *Real-time response and high quality of services (QoS):* Edge computing reduces network latency compared to cloud computing, as edge servers are geographically closer to IoT devices. Hence, EC-driven systems are well-suited for high-demand, real-time IoT applications. Since most of the data is processed at the edge, the volume of data offloaded to the cloud is significantly reduced, thus resulting in higher Quality of Service (QoS) for real-time applications.
- *Low energy consumption:* Most IoT nodes operate under strict power constraints, and transmitting large volumes of sensing data to remote cloud

servers can lead to significant energy waste. By leveraging edge computing, these nodes only need to communicate with nearby edge servers, which significantly reduces their energy consumption. Consequently, edge computing–driven IoT can extend the operational lifetime of IoT devices and lower maintenance requirements.

- *High scalability:* In EC-driven IoT, edge servers such as base stations offer moderate computing resources in a distributed fashion. This architecture enables excellent scalability, making it well-suited to meet the demands of large-scale IoT applications, such as smart cities and autonomous driving. Figure 2.7 illustrates the classification of Edge computing and its related concepts.

2.7.1 EC Challenges

The heterogeneous ecosystem of IoT comprises a range of hardware and communication protocols. The feasibility of IoT services lies in a unified framework that enables IoT devices and edge nodes to integrate and operate seamlessly. Security and privacy remain the primary concerns, complicated by the heterogeneity and resource constraints (i.e., memory, battery power, etc.) of edge-driven IoT. Despite many benefits, the EC community needs to standardize definitions, architectures, and protocols (Kong et al., 2022). A summary of challenges before EC is discussed in the following paragraphs (Grzesik & Mrozek, 2024).

- *Constrained Devices & Computation Offloading:* Edge devices are typically equipped with limited computational and storage resources, ranging from industrial PCs to microcontrollers. Workloads must be optimally balanced according to device capabilities, which may come at the expense of computation speed or accuracy. Compared to cloud platforms, edge nodes have significantly fewer resources, making them insufficient to handle resource-intensive applications such as virtual/augmented reality (VR/AR), machine learning, and 3D gaming. Computation offloading addresses this limitation by distributing tasks to other nodes or cloud servers, thereby enhancing scalability and reducing bottlenecks (Lin et al., 2022). However, computation offloading involves

a trade-off between the benefits of remote execution and the cost of data transmission. Offloading is feasible only when the local execution time exceeds the total offloading time, which includes both communication time and remote execution time, as illustrated in equation 1.1 (Lin et al., 2019).

$$T_{\text{local}} > T_{\text{offloading}} = T_{\text{comm}} + T_{\text{remote}} \qquad (1.1)$$

T_{comm} indicates the communication time, and T_{local} and T_{remote} refer to the execution time at local and remote, respectively.

- *Energy Consumption:* Energy consumption is crucial in EC, as devices often operate under limited battery capacity or constrained energy budgets. Efficient task scheduling algorithms play an essential role in optimizing energy usage without compromising on the QoS. However, the heterogeneity of cores within heterogeneous multi-processors (HMPs) introduces significant complexity in task allocation and resource management. Also, the dynamic and unpredictable nature of edge workloads necessitates adaptive and responsive scheduling strategies (Liu et al., 2025). Battery-powered and remote devices face stringent energy constraints, particularly when executing machine learning (ML) workloads that rely on GPUs. Limited battery life and restricted power availability significantly constrain the computational resources that can be allocated to AI tasks, thereby affecting both performance and functionality. Consequently, system designers are often required to make trade-offs between model complexity, accuracy, and energy consumption.

- *Device Fleet Management:* Unlike homogeneous cloud data centers, edge deployments take place on diverse edge devices (MCUs, single-board computers, ARM/x86 servers, TPUs/VPUs/GPUs), operating systems (real-time OSes, Linux variants, Android, proprietary RTOS), and application stacks distributed across many geographic locations. That heterogeneity creates three interlocking problems: interoperability,

faults and physical hardware failure, and complex software deployment & lifecycle management.

- *Service Migration:* It ensures continuous service as users move between coverage areas of different edge servers. Since multiple servers may overlap, the system must decide the best server to migrate to. This decision involves a trade-off: migration improves performance by reducing latency and enhancing QoS, but it also introduces costs related to state transfer overhead, service disruption, and additional resource use. Designing an effective migration solution is challenging due to user mobility, heterogeneous server capacities, and application constraints (Kong et al., 2022).

- *Scaling and load balancing:* The management of regional traffic spikes is a critical challenge for development teams. Unlike centralized cloud environments, edge computing involves geographically distributed nodes, which makes scaling inherently more complex. Efficiently distributing workloads across these nodes is essential to ensure that no single node becomes overwhelmed or underutilized. Inadequate load balancing can result in increased latency, system downtime, and resource inefficiencies, all of which can degrade the user experience. Therefore, robust strategies for dynamic scaling and intelligent load distribution are crucial for maintaining performance and reliability in edge environments.

- *Integration with emerging technologies (5G, AI accelerators):* The effectiveness of edge computing is increasingly tied to its ability to integrate with emerging technologies. The rollout of 5G networks provides ultra-low latency and high bandwidth, enabling edge nodes to support real-time applications such as autonomous driving, industrial automation, and AR/VR. Similarly, the adoption of AI accelerators such as GPUs, TPUs, and FPGAs empowers edge devices to execute complex machine learning models directly on site. This reduces reliance on centralized cloud resources, minimizes data transfer overhead, and

enhances data privacy. However, integrating these technologies introduces challenges such as interoperability between heterogeneous hardware, the need for standardized APIs, and efficient workload partitioning between accelerators and general-purpose processors. Achieving seamless integration requires coordinated advancements in hardware, software frameworks, and edge orchestration platforms.

2.7.2 Classification of Edge Computing Security Threats

According to Statista's 2017 report, approximately 159,700 cyberattacks targeted edge networks and were grouped under six distinct groups: side-channel attacks, malware injection attacks, DDoS attacks, man-in-the-middle attacks, authentication and authorization attacks, and corrupt data injection attacks. The percentage share of each class of attacks is shown in Figure 2.7. User privacy and data security are the most essential factors from the service provider's perspective.

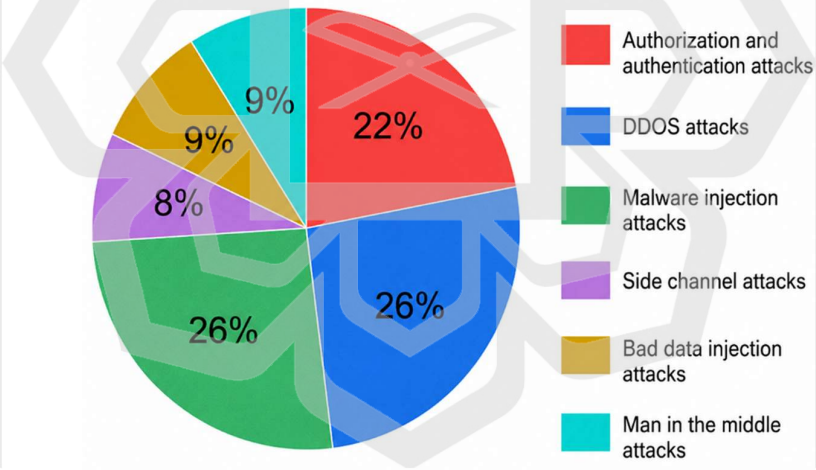


Figure 2.7 Attack percentage on EC

Data acquired from sensing networks can expose substantial volumes of sensitive information, thereby raising significant privacy concerns. For instance, unauthorized access to data generated by smart electricity and water metering systems may enable adversaries to infer tenant occupancy patterns, daily routines, behavioral activities, and resource consumption characteristics. EC processes data generated by IoT devices closer to, or directly at, the edge of the network. In this context, the edge refers to devices at the network core that directly produce sensed data. Processing data

near its sources not only improves Quality of Service (QoS) for delay-sensitive applications but also enhances user privacy and data security through localized handling.

EC further provides advantages such as heterogeneous distributed architecture, large-scale data processing, parallel computation, location awareness, and mobility support. However, traditional cloud-based security and privacy-preserving mechanisms are no longer sufficient to protect the vast amount of data generated in edge environments (Zhang et al., 2018). EC is often regarded as an extension of cloud computing and therefore inherits several security issues from it. At the same time, EC introduces new security and privacy challenges due to its distinctive characteristics, including geographic distribution, heterogeneity, and low-latency requirements. Ensuring secure data analytics in this environment requires the deployment of appropriate security mechanisms. However, the resource constraints of edge devices make traditional cloud-based solutions unsuitable for the edge framework. Consequently, there is a pressing need to develop lightweight and effective security solutions tailored to EC to support reliable and efficient IoT applications (Liu et al., 2019).

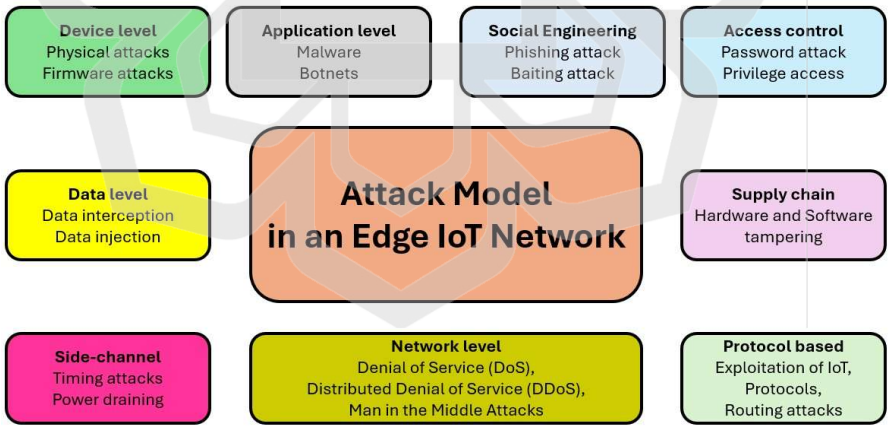


Figure 2.8 Classification of EC security challenges

As shown in Figure 2.8, security and privacy threats in the EC paradigm include malicious hardware and software injection, such as node replication, hardware Trojans, and counterfeit EC nodes. Additionally, side-channel attacks that exploit power, signals, or sensors, as well as authentication or authorization breaches, are also present. At the

communication level, threats such as jamming, Distributed Denial of Service (DDoS), routing manipulation, forgery, replay, and eavesdropping compromise availability and data integrity. Device security is further endangered by physical tampering, while vulnerabilities also arise from unauthorized access and control, integrity attacks on machine learning models, and insufficient logging. In addition, IoT devices are exposed to malware, ransomware, and mobile botnets, with risks intensified by nonstandard frameworks, coding flaws, and the absence of unified security policies. Privacy leakage remains a critical concern, emerging from unauthorized data sharing, location tracking, and simultaneous connections to multiple nodes (Rupanetti & Kaabouch, 2024; Fazeldehkordi & Grønli, 2022).

Table 2.2 Strategies against EC security threats (Kong et al., 2022).

Strategy	Network Layer	Limitations
Cryptographic Schemes	Communication Layer	Power efficiency, computational ability, storage, etc.
Secured data aggregation, deduplication, and analysis	Data layer	Consumes power, renders sensitive data to intruders' network bandwidth
Combined with Blockchain	Architecture layer	A complex system with more computing capability
Intrusion Detection System (IDS)	Communication Layer	Resource consumption

Table 2.2 summarizes various strategies for mitigating security threats in EC along with their associated limitations. Cryptographic schemes, implemented at the communication layer, enhance data confidentiality and authentication but are limited by power consumption, computational requirements, and storage constraints. Secured data aggregation, deduplication, and analysis, applied at the data layer, improve secure data handling and redundancy reduction, although they consume additional power and network bandwidth, potentially exposing sensitive information. Blockchain based

approaches, used at the architecture layer, strengthen trust and tamper resistance but introduce increased system complexity and computational overhead. Meanwhile, Intrusion Detection Systems (IDS) at the communication layer enable real time attack detection, though their effectiveness comes at the cost of higher resource consumption.

2.8 ARTIFICIAL INTELLIGENCE FOR IOT SECURITY

AI enables computers to mimic human intelligence by solving problems and learning from data. Machine Learning (ML), a subset of AI, focuses on creating algorithms that can predict outcomes and make decisions without being explicitly programmed (El Gharbaoui et al., 2024). AI-driven threat detection is transforming IoT security by enabling real-time monitoring, rapid anomaly identification, and predictive analytics. The traditional security methods rely on static rules and signature-based detection, which are inadequate against the dynamic nature of modern cyber threats. AI continuously learns from data patterns and adapts its models to identify both known and emerging attacks. By leveraging machine learning algorithms, AI can detect elusive deviations in network behaviour, flagging potential breaches before they escalate (Imran & Shah, 2023). AI-based systems have revolutionized data analysis, automation, and decision-making across various domains by utilizing large volumes of data to generate insights and predictions. When integrated with IoT, AI enhances automation, increases efficiency, and supports more informed decision-making (Ekolama & Ebregbe, 2023).

AI taxonomy encompasses a wide range of techniques, including Machine Learning (ML), Deep Learning (DL), Natural Language Processing (NLP), Computer Vision (CV), and Robotics. As illustrated in the Venn diagram of Figure 2.9, AI, ML, DL, data science, and data mining are closely related fields. ML, a subset of AI, learns from historical data, while DL, a more specialized branch of ML, processes information through multiple nonlinear transformations. Compared to traditional ML, DL techniques demonstrate superior capability in extracting and processing complex data, though they require greater computational resources. Decentralized deep learning (DDL) approaches, such as federated learning (FL) and swarm learning, enhance data security by enabling processing directly at edge devices.

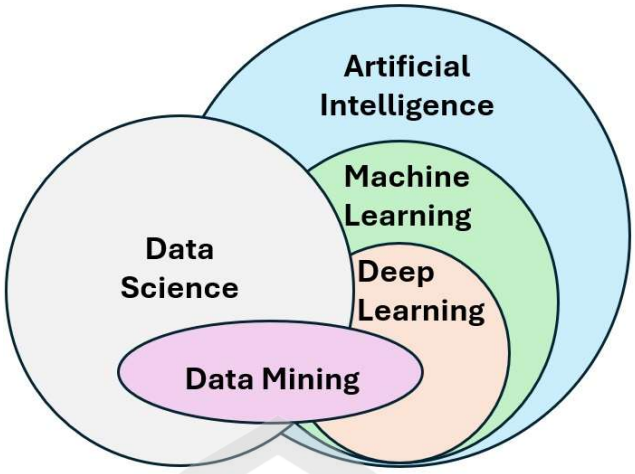


Figure 2.9 AI taxonomy

In data security applications, ML algorithms are often categorized into transaction algorithms, which focus on data exploration and preprocessing, and decision algorithms, which support business decision-making. One of the key advantages of DL over traditional ML is its ability to extract complex, high-level features from raw data automatically. Using hierarchical neural network models, DL can learn effectively from unstructured inputs such as images, audio, text, and video. Meanwhile, data science encompasses the full lifecycle of data processing, including collection, storage, analysis, cleaning, visualization, interpretation, decision making, and value creation. In contrast, data mining focuses on uncovering hidden patterns and generating new knowledge from data (Sheikh et al., 2025).

The integration of AI and IoT combines two transformative technologies that complement each other's strengths. AI techniques such as Machine Learning (ML), Deep Learning (DL), and Natural Language Processing (NLP) enable IoT systems to analyse large volumes of data, extract insights, and make intelligent decisions. These capabilities allow IoT systems to adapt to dynamic environments, detect anomalies, and automate security operations against cyberattacks. By leveraging AI, IoT networks can strengthen cyberattack detection, secure communication, and protect data privacy, resulting in a more resilient ecosystem. Specifically, AI enhances IoT security by providing intelligent, adaptive, and automated protection across connected devices and

networks. It detects anomalies, predicts emerging threats, and identifies zero-day attacks by analyzing both real-time and historical data. AI also automates incident responses, such as isolating compromised devices, and strengthens authentication through behavioural analysis. In addition, it dynamically updates security policies, manages vulnerabilities, and prioritizes critical firmware updates, while monitoring data flows to prevent privacy breaches and unauthorized transmissions (Talpur, 2023). Overall, AI shifts IoT security from a reactive, manual process to a proactive, self-learning defence system. Beyond security, AI subfields extend IoT capabilities: NLP enables language understanding and generation in applications like chatbots, translation, and sentiment analysis; Computer Vision (CV) allows machines to process visual data for tasks such as image classification, object detection, and segmentation; while speech recognition, robotics, expert systems, and reinforcement learning further expand AI applications across industries (Gupta, & Tewari, 2025).

2.8.1 Integrating AI with Cryptographic techniques

AI-driven encryption techniques leverage machine learning (ML), deep learning (DL), and neural networks to detect trends in IoT data, identify anomalies, and generate dynamic encryption keys resilient to brute-force attacks. By employing stochastic and heuristic methods, these techniques continuously enhance encryption strategies, making it increasingly difficult for adversaries to decipher protected information (Ruth et al., 2024). The convergence of AI and blockchain creates powerful synergies by combining decentralized learning with the strengths of both technologies while mitigating their respective limitations (Wena, 2024). Together, blockchain and AI enhance data analysis, pattern recognition, and anomaly detection, enabling real-time IoT network monitoring, predictive maintenance, and efficient resource management (Shinde et al., 2023). Blockchain's tamper-resistant properties further ensure the integrity and transparency of AI models, preventing unauthorized manipulation of algorithms and safeguarding the security of data shared across multiple parties. This directly addresses privacy concerns often associated with AI systems (Kuznetsov et al., 2024).

AI-enhanced encryption also automates key management, reduces human error, and lowers the risk of compromised keys. Unlike traditional cryptographic systems, which rely on static encryption and decryption mechanisms that may become predictable over time, AI-based approaches dynamically evolve encryption strategies,

making decryption attempts exponentially more difficult. Furthermore, deep learning techniques allow AI algorithms to analyze network traffic patterns and detect anomalies that may signal potential security breaches. Despite these advantages, AI-based encryption faces challenges, most notably the computational overhead of cryptographic models, which require significant processing power. This limitation makes them less suitable for low-power devices such as IoT sensors and mobile communication nodes. To address this, researchers are actively developing lightweight AI-based encryption models that strike a balance between security and computational efficiency, enabling even resource-constrained devices to benefit from advanced cryptographic protection (Lingishetty et al., 2025).

PUF-based authentication is desirable for IoT devices because of its low computational overhead and strong resistance to cloning attacks. PUFs generate unique responses derived from inherent manufacturing variations in devices, requiring only minimal hardware operations such as delay-based measurements. For instance, ring oscillator PUFs consume only a few hundred nanowatts of power and need just a few kilobytes of memory for challenge–response storage, making them well-suited for resource-constrained IoT environments. These architectures are often integrated into microcontrollers (MCUs) and FPGAs to provide lightweight authentication, with error correction mechanisms ensuring reliability under environmental variations. However, large-scale deployment still requires standardized interfaces and protocols to enable seamless integration across heterogeneous IoT ecosystems (Dritsas & Trigka, 2025).

A variety of applications have been proposed for PUFs across the fields of hardware security and authentication. Some of the most common uses include device identification, key generation and storage, authentication protocols, and supply-chain security. Table 2.3 shows an overview of the general uses mentioned and examples of PUFs being applied in these areas.

Table 2.3 PUF Application domains

Application domain	Example
Key generation	This process plays a crucial role in securing key generation by replacing vulnerable stored secrets with reproducible, unclonable, and hardware-rooted keys.
Devices identification	Provide a lightweight, low-cost, and tamper-resistant solution for device identification, making them particularly attractive for IoT devices, embedded systems, and supply-chain security applications.
Key storage	PUFs enable secure key storage without physically storing the key. Instead, keys are dynamically generated from hardware fingerprints, making them a promising solution for IoT devices, smart cards, embedded systems, and cryptographic processors.
Chip identification	PUFs provide an intrinsic, unclonable signature for each chip, making them highly effective for chip identification in applications such as counterfeit detection, secure device provisioning, and supply-chain integrity verification.
Hardware Obfuscation	PUFs enhance hardware obfuscation by embedding device-specific, unclonable, and dynamic secrets into IC designs.
Clone prevention	PUFs act as a hardware root-of-trust for clone prevention by binding device identity to uncontrollable physical variations, making it infeasible to replicate or forge the device at the physical level.

2.8.2 Machine Learning (ML)

Machine learning (ML) differs from traditional security solutions such as user authentication, access control, firewalls, and cryptographic systems, which often fall short in addressing emerging IoT cybersecurity challenges. ML is closely related to data mining, computational statistics, analytics, and data science, enabling systems to learn

from historical data (Ahsan et al. 2022). ML algorithms build models using training data to make predictions on new or unseen data. As shown in Figure 2.10, ML techniques are categorized into four groups: supervised, unsupervised, semi-supervised, and reinforcement.

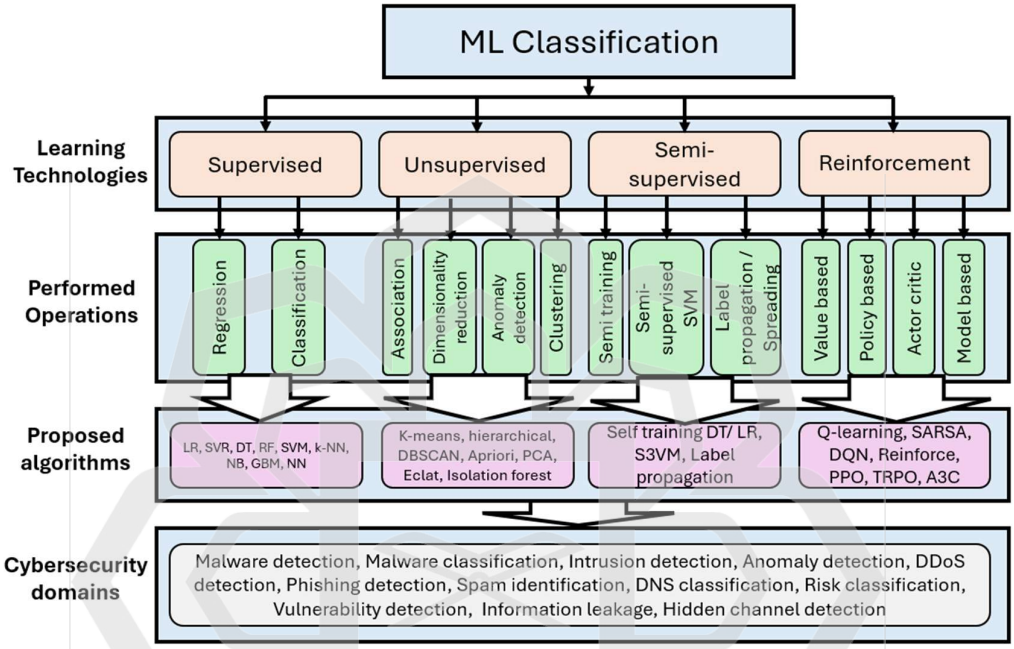


Figure 2.10 Classification of ML algorithms

In supervised learning, the target labels or classes are already known, and these are used to guide the computations, such as in classification and regression tasks. In unsupervised learning, output values are not available, and the focus is on discovering patterns or relationships among samples, such as clustering. Semi-supervised learning lies between the two, where part of the dataset is labeled or human input is required during data acquisition. Finally, reinforcement techniques train ML models to acquire decisions that optimize rewards (Shaukat et al., 2020).

2.8.2.1 Machine Learning in IoT Security

Machine learning techniques provide promising approaches for enhancing the security of IoT systems, including (Zubaydi et al., 2023):

- *Intrusion detection and prevention:* Intrusion detection and prevention systems (IDPS) leverage ML algorithms to analyze network traffic, device logs, and other data sources to identify patterns of known attacks or detect suspicious activities.
- *Anomaly detection:* ML algorithms use anomaly detection to model normal IoT behavior, enabling real-time detection of security breaches such as unauthorized access or malicious activity.
- *Threat intelligence and prediction:* ML analyzes large-scale security datasets to uncover emerging risks, anticipate potential attack vectors, and support IoT security practitioners in proactive threat management.
- *Firmware and software vulnerability analysis:* By training on known weaknesses and coding patterns, ML models can identify security flaws, enabling manufacturers to address them before deployment or expedite the release of security patches.
- *Behavior-based authentication:* ML algorithms model IoT device and user behavior by analyzing usage patterns to create predictive profiles. Any significant deviations from these profiles can trigger additional authentication requirements or alerts for a potential threat.
- *Data privacy and encryption:* ML enhances data privacy and security in IoT systems with approaches such as homomorphic encryption, which enables computation on encrypted data, as well as data anonymization and de-identification.

2.8.2.2 Factors affecting the ML model's performance

The performance of a machine learning (ML) model depends on several factors, including the quality, quantity, and representativeness of the data, as well as the suitability and complexity of the chosen model. Model-related aspects such as effective feature selection, hyperparameter tuning, and regularization play a crucial role in determining whether the model underfits, overfits, or learns effectively. The key factors influencing ML model performance are discussed below.

- *Data quality:* A model's effectiveness largely depends on the quality of the data used for training. Its performance diminishes when the training data is

flawed, containing errors, inconsistencies, duplicates, missing values, or incorrect labels and annotations. Imbalances in the dataset, such as overrepresentation of specific scenarios or insufficient diversity to capture meaningful correlations, can also result in biased or skewed outcomes.

- *Data leakage:* It often arises from preprocessing errors or improper splitting of data into training, validation, and test sets. As a result, data leakage can hinder a model's ability to generalize to unseen data, produce inaccurate or unreliable predictions, and distort performance metrics by inflating or deflating them.
- *Feature selection:* It identifies the most relevant features in a dataset for model training. ML algorithms adjust their weight based on data features, thus playing a crucial role in determining model performance.
- *Model fit:* Overfitting of an ML model occurs when it is overly complex and fits too closely, or even exactly, to the training data, reducing its ability to generalize to new data. In contrast, underfitting arises when a model is too simple to capture the underlying patterns, leading to poor performance on both training and testing data.
- *Model drift:* A decline in a model's performance due to changes in the data or in the relationships between input and output variables is referred to as model drift.
- *Bias:* Bias is prevalent during data processing and model development. Data bias arises when training and fine-tuning datasets are unrepresentative, which negatively impacts model behavior and performance. Algorithmic bias stems from the way data is collected, labeled, and used, as well as from how machine learning models are designed and implemented.

2.9 PHYSICAL UNCLONABLE FUNCTIONS FOR IOT SECURITY

PUF carries out an authentication process for an unknown device in two stages, i.e., enrolment and verification. The PUF module receives the challenge bits from the server, and the corresponding response bits are stored back into the server by the PUF circuit during the authentication phase. During the verification stage, the server sends the previously stored challenge bits to the IoT device, and the PUF circuit embedded into

the device generates response bits. The generated response bits are compared with the CRP look-up table entries to authenticate the IoT devices. Also, the response bits are used to extract the secret key to ensure confidentiality during data exchanges (Al-Meer & Al-Kuwari, 2022). PUFs are classified based on their security capabilities, fabrication methodology, physical characteristics, and delay characteristics. Many researchers have presented a taxonomy of PUF under categories like fabrication process and security, as illustrated in Figure 2.11.

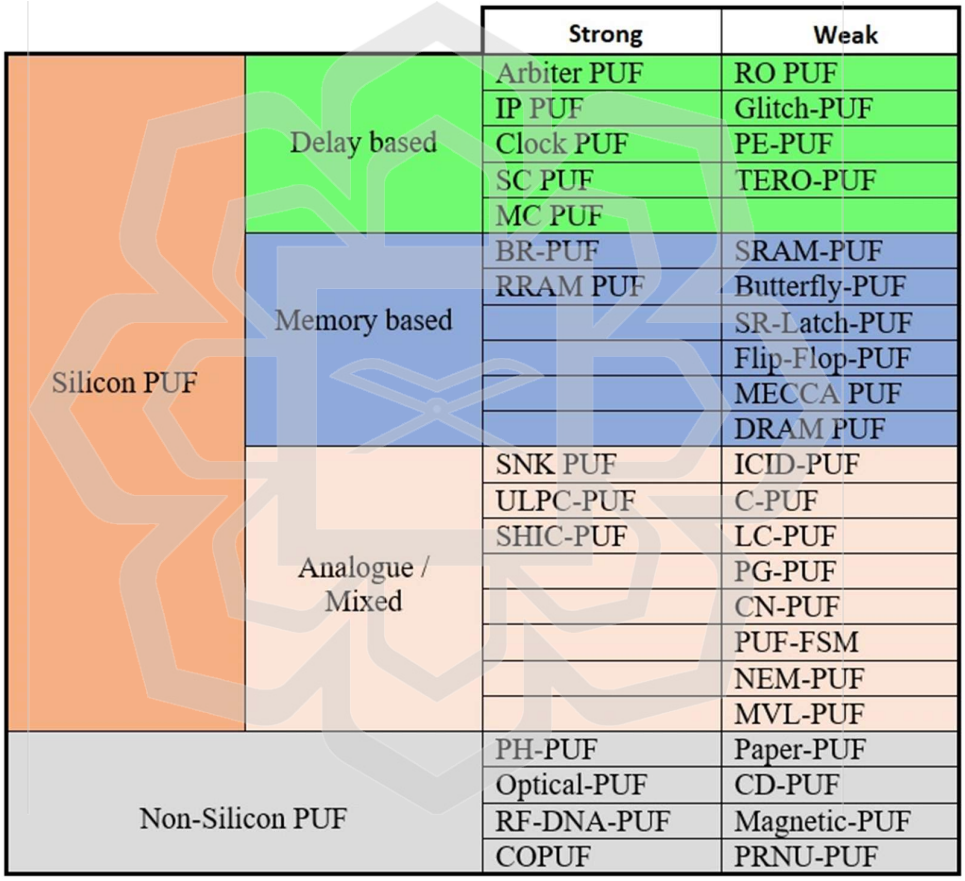


Figure 2.11 Classification of PUFs

PUFs are categorized into two types, strong PUFs (SPUFs) and weak PUFs (WPUFs), depending on the number of CRPs—the number of CRPs in SPUFs scales exponentially and linearly in WPUFs with increasing PUF cells. WPUFs are used in storing secret keys or serve as a seed in a random sequence generator (Liu et al., 2017), while SPUFs can be used for authentication, ID, or key generation (Shah et al., 2021).

Arbiter PUFs fall under SPUFs, whereas SRAM PUF and butterfly PUF are WPUFs (Xu et al., 2020). However, the responses of SPUFs are inherently correlated and highly susceptible to ML attacks, including modelling techniques like Logistic Regression (LR), support vector machines (SVMs), artificial neural networks (ANNs), and ANN-based approximation attacks (Ma et al., 2023)—the variations in the manufacturing process result in silicon and non-silicon PUF types. The fundamental physical properties of silicon PUFs give rise to three types: analog electronic PUFs, memory-based PUFs, and delay-based PUFs (Cao et al., 2023). Non-silicon PUFs create unique characteristics by extracting keys from light beams or lasers, as well as magnetic field strength and radio frequencies, while avoiding the use of electronic signals (Su, 2021; Magyari & Chen, 2023).

An arbiter PUF is a delay-based strong PUF that belongs to the category of silicon PUFs. Figure 2.12 illustrates an N-stage arbiter PUF made up of n pairs of 2-to-1 multiplexers, with each pair in a stage controlled by identical challenge bits. The output, referred to as the "Response", is determined by the differences in path delays. In a standard N-stage arbiter PUF, a rising edge signal travels through one of the 2^N possible paths, guided by the N-bit "Challenge" inputs. An arbiter generates the final response, typically implemented with a D-latch, which decides the output based on the first signal to arrive (Zhuang et al., 2021; Avvaru et al., 2016).

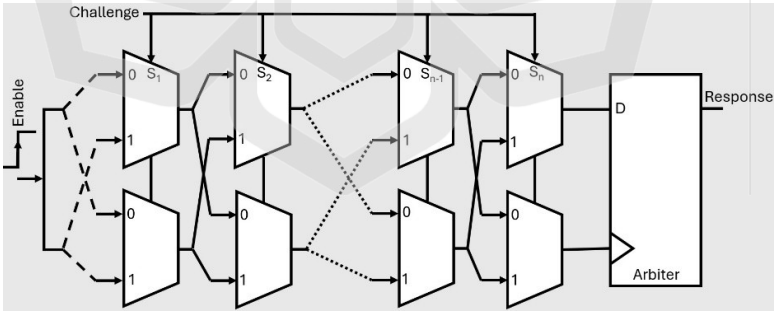


Figure 2.12 Arbiter PUF architecture

Optical PUFs have an edge over other PUF types as they are less noise-sensitive and leverage the complexity of light diffraction, making them stable and difficult to duplicate (Wanga et al., 2024). Light acts as the challenge input and generates a unique random pattern as the response (Chung et al., 2024). Usually, optical structures are not

compatible with solid-state integration. However, a recently proposed CMOS imager PUF uses photodiode responsivity under uniform ambient light and dark current variations to generate unique identifiers for camera authentication (Lu et al., 2018). The basic architecture of an RO PUF, as shown in Figure 2.13, includes an odd number of inverter gates and an AND gate to turn the feedback loop on or off. The conventional RO-PUF consists of several essential components: n ring oscillators (ROs), two n-to-1 multiplexers (MUXs), two counters, and a comparator circuit. The outputs from each RO are routed to the inputs of both MUXs, and the selected outputs from these MUXs serve as clock signals for the counters.

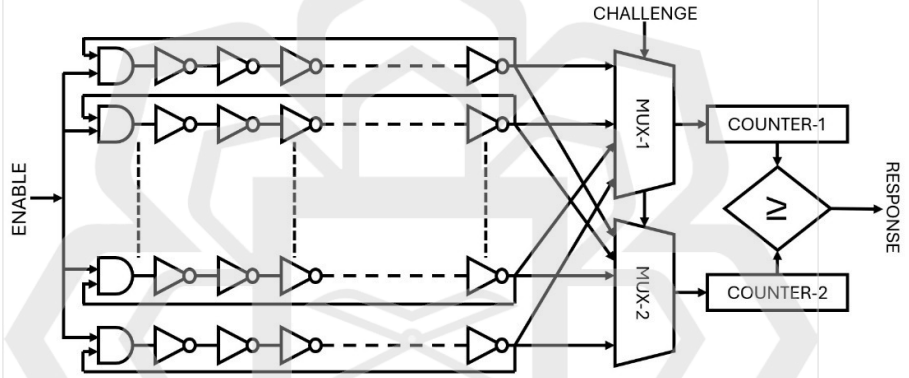


Figure 2.13 RO PUF architecture

Each counter increments based on the oscillation frequency of the specific RO chosen by its respective MUX. Finally, the comparator compares the values stored in the two counters to produce the RO-PUF's response, corresponding to the applied challenge inputs. The frequency of each RO depends on the delay of the inverters, which is influenced by variations in the manufacturing process.

2.9.1 Strong Versus Weak PUFs

The security and performance characteristics of edge devices in a distributed, uncontrolled environment with limited resources vary significantly depending on the choice of PUF types. Choosing between weak and strong PUFs in an EC ecosystem depends on numerous factors, including resource requirements, security against various threat types, authentication capabilities, reliability, and robustness against physical attacks. Environmental factors like temperature and voltage variations are detrimental

to both kinds of PUFs. The efficient and straightforward weak PUFs, e.g., SRAM PUFs, are suitable for key generation in secure boot or communication. In contrast, strong PUFs like APUFs are used in devices that require frequent authentication or cryptographic security.

Table 2.4 Weak versus Strong PUFs

PUF Types	Merits	Demerits
Weak PUFs	Low Resource Consumption: Weak PUFs are simple in implementation, requiring limited hardware resources. For example, SRAM and RO PUFs exploit on-chip memory blocks, making them ideal for resource-limited edge devices.	Environmental variations such as temperature or supply voltage can impact weak PUF performance (Yavuz et al., 2024).
	Low Power Usage: SRAM's minimal static power requirements and quick access times, or energy-efficient oscillator frequencies in RO PUFs, make them ideal for edge devices (Yuyuan et al., 2024).	Physical Attack Vulnerability: Weak PUFs are prone to probing or cloning attacks; if an attacker gains physical access, they may reproduce their response (Rührmair et al., 2016).
	Fast Response Time: The simple architecture and limited set of CRPs of weak CRP suits applications like secure boot due to their quick response, supporting real-time experience in EC.	Error Correction Overhead: Responses processed internally as a secret key require error correction and on-chip storage, adding some overhead (Rührmair et al., 2010).

Strong PUFs	Ample CRP Space: A strong PUF generates a large pool of CRPs, suitable for authentication tasks between edge devices and servers without repeating challenges (Zhang et al., 2020). Hence, no need to store a secret key as the PUF itself acts as the authentication mechanism.	Higher Resource Requirements: Strong PUFs often use complex circuits (e.g., delay lines in APUFs), which require more area and power, thereby stressing limited edge resources. Susceptibility to Modeling Attacks: Large CRP sets make strong PUFs vulnerable to ML attacks that predict their behavior (Kroeger et al, 2022). Power and Complexity Trade-Off: Managing large CRPs consumes more energy and requires sophisticated hardware, unsuitable for low-end edge devices.

Table 2.4 compares various trade-off factors of weak and strong PUFs. Table 2.5 presents a comparison of PUF performance metrics mentioned in the previous section. It is inferred from the table that the Uniqueness and Uniformity performance metrics of Lattice PUF remain closer to ideal values, whereas RC-PUF is the lowest-performing one. The work of Wang et al., establishes functional correctness and acceptable statistical randomness, but its security claims are not rigorously substantiated. The proposed countermeasures such as the self-incrementing counter are not formally analyzed or validated under attack models, making their security impact unclear. The security evaluation of FLAM-PUF by Wu et al. relies on the unrealistic assumption that internal responses and intermediate CRPs are inaccessible, ignoring practical threats such as side-channel and fault injection attacks that can expose hidden states. Its validation is limited to simulations, lacking FPGA or silicon implementation, and therefore does not capture real hardware effects like noise, process variation, and metastability.

Table 2.5 Comparing the performance of PUFs

Article	PUF Type	Stages	Uniqueness [mean|std]	Uniformity [mean|std]	Reliability [mean|std]
(Wang et al., 2020)	Lattice PUF	1000	50.00% (1.58%)	49.98% (1.58%)	1.26% (2.88%)
(Wu et al., 2022)	FLAM-PUF	64/ 128	49.73% / 49.99%	49.81% / 49.85%	95.59% / 96.58%
(Zhu et al., 2023)	Strong response–feedback PUF	32/ 64/ 128	50.17 (1.41) / 50.00 (0.31) / 49.99 (0.21)	49.54 (3.67) / 50.05 (2.79) / 49.93 (1.78)	
(Wang et al., 2024)	DDQ-APUF	64/ 128	47.28% / 47.65%	50% / 50%	99.59% / 99.91%
(Li et al., 2023)	FOM-CDS PUF	17	47.38% (RO mode) / 53.79% (TERO mode) / 50.33% (Full mode)	47.71% (RO mode) / 56.23% (TERO mode) / 53.68% (Full mode)	3.1% CRO-PUF / 9.14% Dual mode / 7.91% FOM-CDS PUF
(Dubrova et al., 2019)	CRC-PUF	128	49.9978%	50.0777%	
(Lee et al., 2019)	RC-PUF	32	27.3% (bit delay = 2 µs) / 30.9% (bit delay = 32 µs)	50.3% (bit delay = 2 µs) / 50.3% (bit delay = 32 µs)	96.2% (bit delay = 2 µs) / 98.5% (bit delay = 32 µs)

The LFSR–APUF design by Zhu et al. relies on the impractical assumption that internal states and intermediate responses remain inaccessible, overlooking realistic attack vectors such as side-channel leakage. The incorporation of an LFSR does not remove its inherent linearity, and the work lacks formal analysis to quantify nonlinearity or entropy enhancement. Additionally, the evaluation is confined to simulations without FPGA or silicon validation, limiting confidence in its real-world security and robustness. The DDQ APUF proposed by Wang et al. introduces a pronounced latency energy trade off, as each challenge requires 13 distinct configuration cycles (Wang et

al., 2024). This repeated reconfiguration increases response generation latency and elevates switching activity within the underlying hardware, thereby impacting overall dynamic power consumption. The research by Dubrova et al. regarding the CRC-PUF architecture exhibits critical gaps in its area-efficiency claims and security framework (Dubrova et al., 2019). The reported gate equivalent (GE) count represents only the PUF core and reconfigurable LFSR, omitting the silicon overhead of the cryptographic PRNG (e.g., a stream cipher) required to generate and update the secret generator polynomial g(x).

The reviewed PUF based IoT security schemes exhibit several key research gaps that limit their practical deployment. Most approaches provide functional validation and statistical randomness testing but lack rigorous security proofs against realistic adversarial models, including machine learning, side channel, and fault injection attacks. Many designs rely on impractical assumptions that internal states, intermediate responses, or configuration data remain inaccessible, which is unrealistic in IoT hardware environments. In addition, a significant number of studies are restricted to simulation based evaluation without FPGA or silicon implementations, preventing accurate assessment of real world effects such as noise, process variation, and metastability. Energy efficiency and hardware overhead are also insufficiently analyzed, as seen in designs like DDQ APUF where latency and switching costs are introduced without quantitative power analysis, and CRC PUF where silicon cost is underestimated by omitting supporting cryptographic components. Furthermore, proposed countermeasures such as LFSR based obfuscation or self incrementing mechanisms are often not formally validated for entropy improvement or attack resistance. Overall, there is a clear gap between theoretical proposals and practical, attack resilient, energy efficient, and hardware validated PUF based IoT security solutions.

2.9.2 PUFs as a Root of Trust

A layered defense model, as shown in Figure 2.14, is preferred for a secure system with outermost layers managing the regular operations of the device and acting as protection barriers for inner layers. RoTs serve as a fundamental source for various secure schemes that enforce access to cryptographic modules and security resources at the hardware

level. The software security built on top of hardware-based RoT provides extra layers of flexibility and protection. These hardware-based RoTs build a trusted execution environment (TEE) for running privileged software, performing cryptographic operations, and offering constant tamper protection. This design approach minimizes the attack surface area and makes inner layers easier to secure because they have fewer, highly controlled tasks. The trust–validation sequence continues moving towards inner layers up to the system core, known as the Root of Trust (RoT) (Fedorkow et al., 2024).

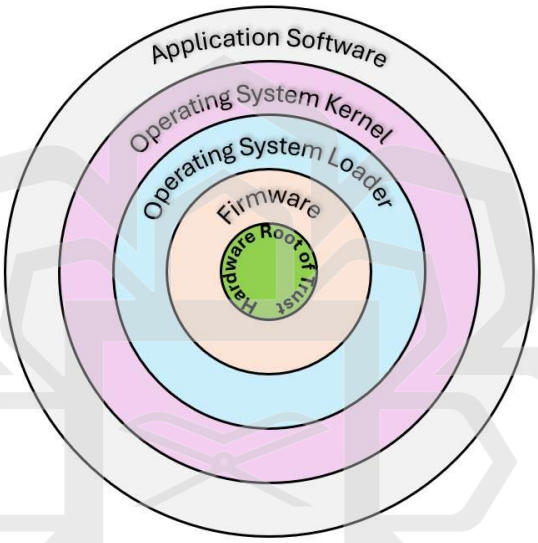


Figure 2.14 Layered defence model.

Edge devices leverage RoTs to establish a protected environment for cryptographic processes required for data encryption and device authentication in backend systems (CYSEC, 2024). RoT applies various code validation mechanisms before executing the code on secured CPUs, providing some protection against physical attacks. Thus, a Chain of Trust is established when each component in this chain trusts the codes it runs as they are validated by the previous link, creating an unbroken line of trust back to the Root of Trust (Knowledge Center, 2024; Zimmer et al., 2016). The hardware RoT secures EC operations by providing the cryptographic keys in the booting process. Hardware-based RoT is typically a small, dedicated chip embedded within an IoT device, leveraging upon intrinsic hardware characteristics (Ehret et al., 2023). PUFs are ideal for hardware-based RoT that hosts cryptographic functions, such as private and public key encryption (Chaintoutis et al., 2019). The unique keys generated from the

edge device's PUF and the secure boot process ensure that only authorized firmware or updates are loaded, preventing Trojan or malware attacks.

Rojas et al. proposed a hardware Root of Trust (RoT) architecture utilizing a Zynq-7000 SoC FPGA (Xilinx Inc., San Jose, CA, USA) and integrating various cryptographic components. These components include PUFs for device authentication, the Advanced Encryption Standard (AES) for data encryption, Secure Hash Algorithms (SHA-2 and SHA-3) for ensuring data integrity, and the Edwards-curve Digital Signature Algorithm (EdDSA) for digital signature verification (Rojas-Muñoz et al., 2023). A hardware RoT is proposed in (Chuang et al., 2021), leveraging Quantum Tunneling PUFs to identify ICs digitally. In contrast to SRAM PUFs, Quantum Tunneling PUFs operate without the need for error correction. The software-based PUF (SW-PUF) combines physical chip variations with delays in software instructions to generate unique IDs within a secure Root of Trust (RoT). This approach supports secure boot and remote attestation, ensuring that only authenticated, tamper-free software is executed (Hamadeh et al., 2019). A secured IoT architecture proposed in (Bathalapalli et al., 2023) combines PUF with a Trusted Platform Module (TPM) and Tangle Distributed Ledger Technology (DLT), which acts as a RoT, establishing a unique digital identity for each device. The proposed architecture implements a Security-by-Design (SbD) approach at the hardware level, strengthening attack resistance and defending device and data integrity. Quantum channels are vulnerable to diverse noise sources, which include environmental interactions and eavesdropping attempts. A key reconciliation protocol is proposed in (Colombier et al., 2017), allowing transmission of a bit stream through insecure and noisy quantum channels. Also, the researchers claim that the proposed protocol can reconcile two PUF responses obtained from the same challenge but at a different time. Also, minor noise levels in the PUF responses are mitigated through the application of a fuzzy extractor, designed to produce stable cryptographic keys from marginally erratic PUF responses (Schrijen et al., 2018).

2.10 Integration of FPGAs-Based PUFs with Edge AI

Artificial Intelligence (AI) assisted data analytics at the edge, allowing for improved interpretation of raw and unstructured data from the physical world. AI at the edge has the potential to automate complex and advanced tasks while simultaneously preventing

user-sensitive data from being transmitted over the network and into data centers. Edge AI models human reasoning, thus enabling machines to sense, comprehend, perform intelligent detection, and communicate results to the cloud for long-term storage or big data processing. It can recognize and combat cyberattacks and other cyber threats by continuously analyzing data, identifying patterns, and tracking attacks. Data privacy and security breaches need to be taken seriously as they may cause business interruptions, revenue losses, and panic among the public (McCarthy et al., 2022). The human brain comprises nearly 100 billion neurons, and over 100 trillion connections are established to form a network that significantly influences the brain's capabilities. The interconnectivity within an FPGA resembles the neural wiring of the human brain, and its programmable logic fabric offers the flexibility of the brain (Ahmad et al., 2022).

The dynamically reconfigurable and customizable hardware architecture of Field Programmable Gate Arrays (FPGAs) has offered a promising solution in accelerating compute-intensive workloads (Manan, 2006). FPGA-based edge network accelerators offload intelligence, data processing, analytics, and communication capabilities from the cloud to where the data originates (Xu et al., 2022). Cloud computing provides the infrastructure needed for securing users' data as well as maintaining their integrity and privacy. However, there is no foolproof technique yet that guarantees data protection, nor is there a processor that can isolate the execution of users' applications from data theft. FPGAs are capable of providing stronger security guarantees because there is no need to involve vulnerable operating systems, drivers, compilers, or any other system software (Al-Asli et al., 2018). The possibility of incorporating general-purpose processors, such as soft cores on FPGAs, makes these reconfigurable devices suitable for IoT applications. They can provide solutions with enhanced security, reduced size, energy consumption, and cost (Martínez Rodríguez et al., 2022). Silicon chip fabricators and designers have integrated FPGA and ARM processor cores for efficient edge AI processing. Also, the benefits of shorter development time make an FPGA-based solution the ideal choice for an intelligent edge device (Mukhtar et al., 2021).

2.11 SUMMARY

This chapter presents a comprehensive literature review on technologies and concepts underpinning FPGA-based PUFs for IoT security. It begins with an overview of the IoT,

its layered architecture, and diverse applications, followed by an analysis of major challenges such as heterogeneity, scalability, energy efficiency, privacy, and data management. The chapter further explores the role of edge computing in enhancing IoT performance while addressing its associated security threats. It also highlights the integration of artificial intelligence and machine learning techniques for anomaly detection, authentication, and cryptographic enhancement. It concludes by examining PUFs as a lightweight, hardware-based root of trust for secure identification and key generation, with a particular emphasis on FPGA-based PUF implementations combined with Edge AI to achieve reliable and attack-resilient IoT security solutions.

CHAPTER THREE

METHODOLOGY

3.1 INTRODUCTION

This chapter presents the methodology adopted for the design, implementation, and evaluation of a Physically Unclonable Function (PUF)-based lightweight solution to secure an IoT device's authentication system. It provides a framework that ensures a systematic approach, which is both technically rigorous and practically viable for securing resource-constrained edge devices. The chapter outlines the overall research design that integrates theoretical modeling, simulation-based validation, and hardware prototyping. The methodology adopted is to address key research objectives, including (i) designing a lightweight and reliable Arbiter PUF architecture suitable, and (ii) evaluating the proposed system under realistic conditions. A layered approach is adopted for the design implementation, which includes, first, mathematical modeling and algorithm design to establish the theoretical foundation of the proposed APUF architecture and authentication scheme. Second, synthesis, timing closure, and simulation of the proposed design using Electronic Design Automation (EDA) software tools Altera Quartus Prime and ModelSim were performed to validate the functionality and further assess performance metrics such as uniqueness, reliability, and uniformity of PUF responses. Third, hardware implementation using Cyclone IVE FPGA-based testbeds and logic analyzer is used to demonstrate the feasibility of the design in real-world scenarios.

One of the most significant security threats to cryptography techniques like PUFs is adversaries' attempts to carry out machine learning–based modeling of PUF behavior. The collected CRP datasets are analyzed using ML classifiers, such as Gradient Boosting, Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), Logistic Regression (LR), and Random Forest. This analysis provides an empirical understanding of the PUF’s resistance to machine learning attacks, as well as insights into the optimal PUF design choices that minimize predictability. The integration of

PUF design and implementation, hardware experimentation, and machine learning–based security evaluation ensures a holistic treatment of the research objectives. The following sections detail the research design, CRP data acquisition and preprocessing, machine learning modeling process, hardware setup, performance evaluation metrics, and validation framework. Both Arbiter PUFs (APUFs) and Ring Oscillator PUFs (RO PUFs) offer lightweight and optimal hardware-based cryptography security solutions for IoT edge devices. Together, these characteristics make both PUF types highly effective and suitable for enhancing IoT security. APUFs fulfill the scalability and speed requirements, while RO PUFs provide robustness and stability.

3.2 RESEARCH DESIGN AND APPROACH

The research implements a mixed-methods approach, integrating theoretical modeling, simulation-based validation, hardware implementation, and analytical evaluation. The figure 3.1 illustrates the overall research design adopted in this study, presenting a structured and sequential workflow for the development, implementation, and evaluation of the proposed synchronized PUF architecture.

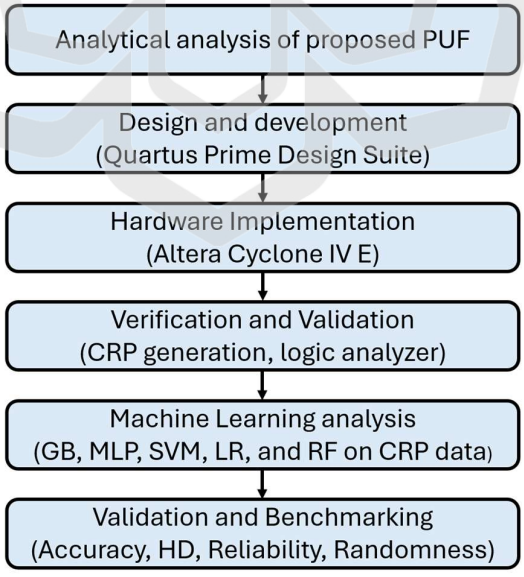


Figure 3.1 Research design and approach flow

This approach is appropriate because the study addresses the conceptualization, design, and implementation of PUF-based authentication mechanisms, along with the security evaluation of their vulnerability to machine learning attacks. Initially, the research is conceptually analyzed, involving the mathematical modeling of Arbiter PUF and Ring Oscillator PUF architectures, as well as the formulation of an authentication protocol. This stage requires a strong foundation and establishes the expected performance parameters, such as uniqueness, reliability, and randomness. The next stage is simulation-based, where the proposed PUF architectures are implemented in environments such as HDL-based tools like Altera Quartus Prime and ModelSim. Simulation enables large-scale challenge–response pair (CRP) generation and early performance testing without incurring the cost and time of hardware realization. Following simulations, research work advances to hardware implementation, deploying FPGA-based testbeds and a logic analyzer with a maximum sampling rate of 500 MHz and an input voltage range from -50 V to +50 V to prototype the PUF designs and authentication protocol. FPGA is a highly suitable platform for PUF implementation because it provides flexibility, reconfigurability, and a hardware-based environment that closely reflects real-world device variations. Since PUFs exploit inherent manufacturing process variations in delay paths, ring oscillators, or memory cells, FPGAs offer an ideal medium to design, test, and evaluate different PUF architectures without the need for costly ASIC fabrication. The FPGA implementation process includes writing VHDL code to describe the circuit, simulating it for correctness, synthesizing it into FPGA logic, performing place-and-route, and generating a bitstream to program the device. Key considerations include timing, resource utilization, and parallelism.

The machine learning analysis is carried out by preparing the dataset through cleaning, feature extraction, feature scaling, and splitting into training and testing sets. Python is used as the primary programming language due to its extensive machine learning libraries, such as NumPy, Pandas, scikit-learn, Matplotlib, and Keras. The models are implemented and executed in Microsoft Visual Studio (VS Code) for efficient coding and debugging. Preprocessing ensures balanced and normalized input data before applying multiple models, including LR, SVM, RF, GB, and MLP. Each model is trained on the training dataset and tested on the unseen data to evaluate performance. Key evaluation metrics such as accuracy, precision, recall, and F1-score

are used to assess the models. This quantitative analysis provides measurable insights into the vulnerability of PUF architectures against modeling attacks. This mixed design, combining theory, simulation, hardware prototyping, and ML-based security analysis, ensures that the proposed methodology is both comprehensive and practical, bridging the gap between academic models and real-world IoT security applications.

3.3 SYSTEM MODELING

The system model provides the mathematical foundations and architectural description of the proposed PUF-based authentication framework. It defines how the PUF architectures are modelled, how CRPs are generated, and how the authentication process operates under realistic conditions. This section also outlines the assumptions and boundary conditions adopted for analysis.

3.3.1 Arbiter PUFs

An APUF is a delay-based PUF circuit that derives its output response bits from the time delay differences between two identical signal propagation paths. A race condition is created by simultaneously launching a standard signal through both paths. The path that completes first determines the output response bit. These path-to-path delay variations arise due to device-specific manufacturing imperfections, yielding unique and effectively unpredictable responses for each chip (Ali et al., 2021). Mathematically, a PUF can be defined as (Herder et al., 2014),

$$f: \mathcal{C} \rightarrow \mathcal{R} \tag{3.1}$$

$$f(c) = r, \quad \text{where } c \in \mathcal{C},\ r \in \mathcal{R}. \tag{3.2}$$

Here, $\mathcal{C}$ denotes the set of input challenges, and $\mathcal{R}$ denotes the set of corresponding responses generated by the PUF circuit.

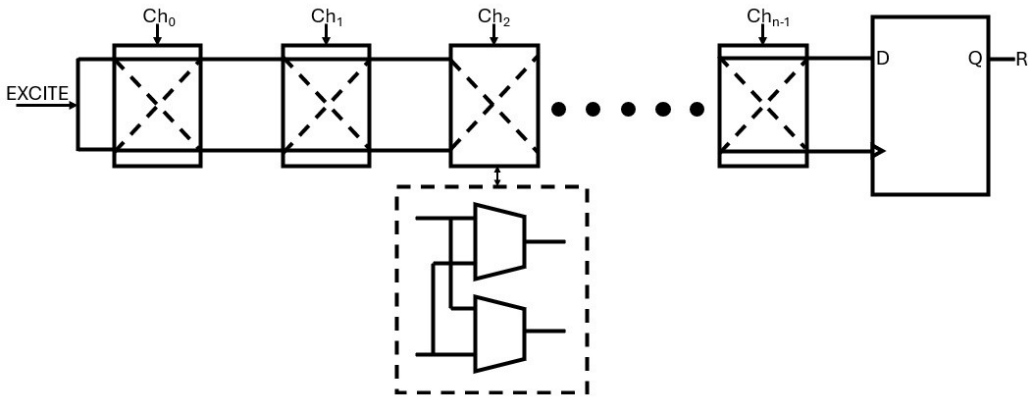


Figure 3.2 APUF design

The Arbiter PUF (APUF) proposed by Gassend et al. has been extensively studied as a strong PUF (Gassend et al., 2002). It consists of two parallel n-stage multiplexer chains that feed into an arbiter stage to generate a 1-bit output of an N-bit PUF, as illustrated in Figure 3.2. Each multiplexer is configured as either a cross- or straight-through connection depending on the input challenge bit. The arbiter, typically implemented as a flip-flop (FF), compares the arrival times of the two signals and outputs the response bit based on which signal arrives first. This timing difference varies across devices due to inherent manufacturing variability. An arbiter at the end evaluates the response bit based on which signal arrives first—either on the top or bottom line. The signals propagate through the two delay paths, competing to reach the end first (Gu et al., 2021). The output is set to '1' if the signal reaching the latch's data input (EXCITE) is faster; otherwise, the output is '0'. The delay differences are considered to follow a Gaussian distribution due to random manufacturing process variations of individual APUF stages that combine additively. This means the total delay difference between the two signals is the sum of the delay differences across all stages. Mathematically, the propagation delay difference at the δ_i at the i^{th} stage is expressed as (Ruhmair et al., 2009):

$$\delta_i = (1 - 2c_i) \cdot w_i \tag{3.3}$$

$c_i \in \{0,1\}$ is the challenge bit at stage i,

w_i is the delay weight associated with process variations.

The overall delay difference for an n-stage APUF is:

$$\delta = \sum_{i=1}^{n+1} \phi_i(c) \cdot w_i \tag{3.4}$$

Where $\phi_i(c)$ is a parity function of the challenge vector defined as,

$$\phi_i(c) = \prod_{j=i}^{n} (-1)^{c_j}, \quad i = 1, \ldots, n, \quad \text{and} \quad \phi_{n+1}(c) = 1. \tag{3.5}$$

The arbiter PUF response bit is then given by, $r = \text{sgn}(\delta)$ with $r \in \{0,1\}$

Algorithm 1: Arbiter PUF

Input: N: number of challenge bits per PUF instance
challenge$[N]$: vector of size N bits
clk: system clock signal
Output: response bit

```
/* Initialization                                          */
path_A[1] ← 0
path_B[1] ← 1
for i ← 1 to N − 1 do
    /* Generate delay paths using multiplexers             */
    if challenge[i] = 1 then
        path_A[i + 1] ← path_B[i]
        path_B[i + 1] ← path_A[i]
    else
        path_A[i + 1] ← path_A[i]
        path_B[i + 1] ← path_B[i]
    end
end
/* Determine the response bit                              */
if posedge of clk then
    /* If path_A wins, response is 0; else 1               */
    response ← path_A[N]
end
return challenge, response
```

As shown in Algorithm 1, Arbiter PUF is triggered by an *n*-bit challenge input and generates a 1-bit response based on path delay differences. There are two parallel paths (A and B) that are initialized and propagated through n stages of multiplexers. Each challenge bit determines whether these paths will cross or proceed straight at every multiplexer stage. After traversing all stages, an arbiter at the end of the multiplexer chain compares the arrival times of the signals. If the path A is faster, then the output response is 0; otherwise, it is 1. The challenge input and corresponding response output

from the CRP uniquely depend on the device's manufacturing variations, making APUFs useful for authentication and hardware security.

3.3.2 Ring Oscillator PUFs

A Ring Oscillator (RO) typically consists of an odd number of inverters connected in a closed loop, whose output oscillates due to the delay difference between the chain of inverters. A NAND gate with an even number of inverters or an AND gate with an odd number of inverters ensures that a RO only oscillates when an enable signal is asserted. Due to manufacturing variability, the ROs oscillate at slightly different frequencies. When a challenge input is applied to an RO-PUF, the corresponding pair of ROs gets triggered, resulting in oscillations, and the resulting output response is computed based on the frequency difference of the pair, as illustrated in Figure 3.3. Consequently, all CRPs of RO-PUF are extracted by measuring the frequency difference of all available pairs of ROs (Gutierrez, 2024).

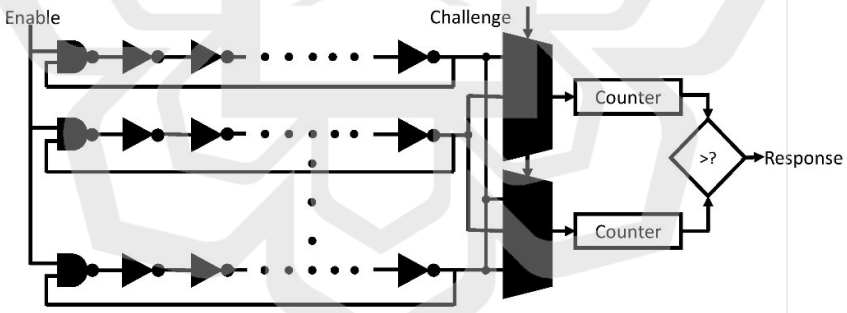


Figure 3.3 RO PUF design

Algorithm 2 demonstrates the working principle of the RO-PUF, which leverages frequency variations between pairs of ring oscillators caused by manufacturing differences. The challenge input selects two oscillators through multiplexer pairs, and their outputs are connected to separate counters. Each counter counts the number of oscillation pulses from its respective oscillator within a fixed time interval. After counting is disabled, the counter values are compared: if the first

oscillator produces more pulses (higher frequency), the response bit is set to 1; otherwise, it is set to 0.

The frequency of the k^{th} oscillator is modeled as:

$$f_k = \frac{1}{2N \cdot (t_{inv} + \delta_k)} \tag{3.6}$$

Where N = number of inverters per ring,

t_{inv} = nominal inverter delay, and

δ_k = variation in delay due to manufacturing randomness.

For a given challenge (i, j), the PUF response is:

$$Response = \begin{cases} 1, if\ f_i > f_j \\ 0,\ \ Otherwise \end{cases} \tag{3.7}$$

Algorithm 2 Ring Oscillator PUF (RO-PUF) Algorithm

Require: Challenge input $C = \{Sel\_A, Sel\_B\}$, Time window T
Ensure: Response bit R
Initialize N Ring Oscillators: $RO[0], RO[1], \ldots, RO[N-1]$
Initialize two N-to-1 MUXs, two Counters: $Counter\_A$, $Counter\_B$, and a Comparator
Select $RO\_A = RO[Sel\_A]$ via MUX A
Select $RO\_B = RO[Sel\_B]$ via MUX B
Connect RO_A output to $Counter\_A$ clock
Connect RO_B output to $Counter\_B$ clock
Enable $Counter\_A$ and $Counter\_B$ for fixed time window T
During T, increment counters on each RO oscillation pulse
After T, disable both counters
if $Counter\_A > Counter\_B$ **then**
    $R \leftarrow 1$
else
    $R \leftarrow 0$
end if
return R

An FPGA based implementation is adopted to capture intrinsic propagation delay variations, routing asymmetries, and metastability effects that are not represented in simulation based models, thereby enabling a realistic evaluation of PUF reliability, uniqueness, and stability under practical conditions. To realize this, Hardware Description Languages such as VHDL and Verilog are used to model and implement the design with precise control over concurrency and timing behavior, ensuring that

critical delay dependent characteristics are preserved from abstraction to physical hardware.

3.3.3 Novelty of the Proposed APUF and RO PUF Design

The proposed design introduces a synchronized PUF architecture implemented on FPGA, which enhances both security and practical deployability. A key novelty lies in the integration of clock synchronized operation for both APUF and RO PUF circuits, ensuring controlled evaluation and consistent response generation under real hardware conditions. The combination of FPGA based implementation and synchronization techniques provides a realistic evaluation environment that captures process variations, routing asymmetry, and noise effects, which are often ignored in simulation only approaches. The proposed approach offers several key advantages:

- *Improved Reliability and Stability:* Clock synchronized operation ensures that responses are sampled at well defined time instances, reducing noise induced variations and improving repeatability. This results in low bit error rates and stable PUF behavior across multiple evaluations.
- *Accurate and Controlled Measurement:* For RO PUF, synchronization enables precise frequency comparison using fixed time windows, improving the accuracy of oscillation based measurements. For APUF, it ensures consistent delay evaluation.
- *Seamless FPGA Integration:* The synchronous design aligns naturally with FPGA based digital systems, simplifying integration with other modules such as processors, memory, and communication interfaces without complex timing management.
- *Energy Efficient Operation:* Clock controlled activation (clock gating) allows selective evaluation of the PUF, reducing unnecessary power consumption which is critical in edge devices.

3.4 HARDWARE DESCRIPTION LANGUAGE

Hardware Description Language (HDL) design is based on the use of textual descriptions to model digital logic circuits and systems. An HDL enables the representation of a circuit at multiple levels of abstraction, in accordance with the

language's syntax and semantics, ranging from basic gate-level descriptions to higher-level behavioral specifications. At the highest abstraction level, the design process begins with the formulation of the system concept, which serves as the initial description and provides the overall design specification. HDLs solve some of the limitations previously posed, as listed below (Riesgo et al., 2002).

- Shorter development phases in the projects.
- Continuous checking and verification of the system's performance and behavior.
- Independent of the target technology and the final implementation details, in the early stages of the project.
- A common language throughout most design phases.
- Serves as a standard interface between different teams involved in the project and between designers and CAD tools.

The most widely used HDLs for digital design are VHDL and Verilog. In recent years, however, SystemVerilog has gained significant recognition in the field of digital design, alongside its strong role in verification. The HDL code is written to conform to one of three styles:

- A structural description represents the circuit in terms of the logic gates employed and the interconnections between them, forming a complete circuit netlist.
- A dataflow description specifies how data moves between inputs, outputs, and internal signals within the circuit.
- A behavioral description defines the functionality of a design in terms of circuit or system behavior through the use of algorithms. This high-level representation employs language constructs that closely resemble those of high-level programming languages.

The proposed APUF architecture was implemented using VHDL 2008. VHDL-2008 is a preferred choice for PUFs modelling and implementation on FPGA due to its strong support for structural design, parameterization, and tool compatibility. The 'generate' constructs supported in VHDL 2008 enable efficient implementation of scalable multi stage PUF structures. Importantly, VHDL-2008 allows precise

description of delay paths and modular components, which is essential for capturing the intrinsic variability required for PUF functionality. Moreover, its widespread support in FPGA toolchains ensures reliable synthesis and timing analysis. The design was described at the register transfer level (RTL) and synthesized on an Altera Cyclone IV E FPGA platform.

A system in RTL description is modeled in terms of clocked storage elements, combinational logic, and the controlled movement of data between registers on discrete clock events. Synthesis tools interpret this RTL representation to infer hardware primitives and generate a technology mapped gate level netlist, consisting of standard cells or FPGA resources along with their interconnections and timing constraints. At a finer granularity, FPGA logic elements are physically realized using transistor level implementations, typically based on CMOS devices, where electrical characteristics such as propagation delay, capacitance, and drive strength determine circuit performance. Some HDLs also support switch level modeling, which represents circuits using transistors as controlled switches and captures their conduction states and connectivity. The selection of a particular HDL and level of abstraction is based on multiple factors. Some of the aspects that must be considered are listed below,

- The availability of suitable electronic design automation (EDA) tools is essential to support the use of the language.
- Availability of simulation models.
- Synthesis capabilities.
- Design re-use.
- Existence of standards for the language.
- The ability to create designs at the required abstraction levels and to employ languages or EDA tools that support these levels is essential.
- Accessing support tools for the language, such as automatic code checking utilities and documentation generation tools, is essential.

ModelSim is used as a verification and simulation tool that supports VHDL, Verilog, SystemVerilog, and mixed-language designs. It can be used as a stand-alone program to perform functional simulations, where simulation inputs are specified by drawing waveforms and simulator commands are selected through ModelSim's

graphical user interface (GUI). Additionally, ModelSim can be integrated into other design flows, such as invoking it from within Intel Quartus Prime software, performing timing simulations, specifying simulation inputs with a testbench, or executing simulator commands via script files.

3.5 CREATING A PYTHON PROJECT

A project in Visual Studio is a structured environment that organizes all files required to build a single application. These files usually include source code, configuration files, and other associated resources. The relationships between these components are defined and maintained within the project environment, while ensuring they are correctly built, executed, and managed as a unified application. In addition, a project manages shared external resources that may be referenced across multiple projects, enabling modular and scalable software development. This structured approach supports application growth and maintainability more effectively than manually organizing files. The Python extension for Visual Studio Code provides comprehensive language support for Python development. It includes features such as syntax highlighting, intelligent code completion, linting for error detection, integrated debugging, code navigation, and automated code formatting. It also supports Python specific workflows such as Jupyter Notebook integration, enabling interactive computing and data analysis within the development environment.

An "environment" in Python is the context in which a Python program runs, including global, virtual, and conda environments. An environment consists of an interpreter, a library (typically the Python Standard Library), and a set of installed packages. These components together determine valid language constructs and syntax, as well as operating-system functionality that can access packages.

- Global environments

  A package installed in a global environment ensures its availablity to all projects using that environment. If the environment is in a protected area of the file system, then installing packages requires administrator privileges.
- Local environments

  Two types of environments can be created for a given workspace: virtual and conda. These environments enable installation of

packages without affecting other environments, and subsequently isolating the workspace's package installations.

Virtual environments: They provide an isolated and project specific Python execution context that helps manage dependencies in a controlled manner. In a shared or continuously evolving development system, installing packages globally can lead to dependency conflicts, version mismatches, and reduced reproducibility across projects. To address this, virtual environments are used to create lightweight, self contained directories that include a dedicated Python interpreter along with their own site packages. When a virtual environment is activated, package installation and execution are restricted to that environment, ensuring that dependencies remain isolated from the global Python installation and from other projects.

As a result, each project can maintain its own consistent set of package versions, improving reliability, reproducibility, and compatibility.

Conda environments: A conda environment can be created using the conda command-line tool or through the integrated conda management available in Visual Studio 2017 version 15.7 and later. The use of a conda environment requires the installation of either Anaconda or Miniconda. Conda environments are essential because they isolate dependencies, prevent package conflicts, and ensure consistency across projects, making development, testing, and deployment more reliable.

3.5.1 Python environment tools

The following table lists the various tools involved with Python environments:

- *Pip :* The Python package manager that installs and updates packages. It's installed with Python 3.9+ by default. It is the reference Python package manager used for installing and updating packages in a virtual environment.

- *venv :* Manage separate package installations for different projects, and is installed with Python 3 by default. It allows management of separate

package installations for different projects. It creates a “virtual” isolated Python installation. Switching projects, can create a new virtual environment which is isolated from other virtual environments.

- *condo* : It's a package and environment management tool associated with Anaconda/ Miniconda, that provides a comprehensive package management capabilities. Conda supports the creation of isolated environments, maintaining their own set of packages and Python versions. This isolation helps prevent dependency conflicts between projects and ensures reproducibility of computational workflows.

Algorithms like SVM and LR depend heavily on numerical computations, while ensemble methods like Random Forest and Gradient Boosting rely on randomness and iterative learning. If these models are executed in different environments with inconsistent dependencies, the comparison becomes biased. Isolated environments guarantee that all models use the same versions of underlying libraries such as NumPy and SciPy, ensuring a level playing field.

3.6 HARDWARE

3.6.1 FPGAs at Various IoT Layers

FPGA as a Service (FaaS) is an emerging computing paradigm that provides flexible and scalable access to FPGA resources through cloud platforms, enabling hardware acceleration for tasks such as neural network inference, cryptographic operations, and large-scale data processing. To fully explore the integration of FPGAs into the IoT ecosystem, it is necessary to analyse the role of FPGA technology within each IoT architecture layer, considering different levels of FPGA as a Service deployment (Kujanov & Perepelitsyn, 2024). There is a direct interaction between the perception/sensing layer and the physical environment. At the same time, the performance, accuracy, and fault-tolerance of these elements remain critical to the IoT system’s reliability. FPGAs improve this layer's performance by implementing custom algorithms and control logic for actuators when off-the-shelf controllers or standard peripheral interfaces like I²C, SPI, and UART are insufficient. Additionally, FPGAs can

be employed for data transformation and preprocessing activities, including noise filtering, distortion reduction, and interference mitigation for sensor signals. They also support real-time image and video processing tasks, such as object detection, and enable data compression techniques to reduce transmission overhead. Furthermore, FPGAs can implement both standard and custom communication protocols at the physical connectivity layer, allowing greater flexibility in heterogeneous IoT environments.

In the case of the connectivity/network layer, FPGAs can be utilized to accelerate network-related functions, such as traffic encryption, decryption, and routing. Also, diverse communication protocols are realised across different levels of the OSI model, including the data link layer (e.g., LPWAN, IEEE 802.15.4), the network layer (e.g., 6LoWPAN, IP), and the transport layer (e.g., TCP, UDP). By offloading these functions to FPGA hardware, IoT systems can achieve reduced latency and improved throughput while maintaining high levels of security and adaptability. The data processing layer transforms raw sensor data into meaningful information for decision-making and system control. Depending on system requirements, computations in this layer can be executed by software or hardware. FPGAs offer significant advantages here by accelerating computationally intensive operations, such as signal processing, machine learning inference, and real-time analytics, which would otherwise overwhelm conventional processors. Finally, the application layer focuses on user interaction and overall system experience. Because of its hardware-centric nature, FPGA technology is not typically integrated directly into this layer. Instead, the benefits of FPGA acceleration in the lower layers (perception, network, and data processing) indirectly enhance application performance by delivering faster responses, lower latency, and more reliable system behaviour.

3.6.2 FPGA EDGE DEVICE INTEGRATION

In the experimental framework, the FPGA with the embedded APUF is modeled as an IoT node. Each challenge applied to the FPGA produces a corresponding response, and the resulting CRPs are used for authentication. The distinct responses across devices enable strong device identification and access control, preventing unauthorized entry into the IoT network. The use of lightweight PUF hardware ensures that authentication is achieved without relying on stored cryptographic keys, thereby reducing vulnerability

to physical attacks, reverse engineering, and side-channel leakage. Furthermore, the IoT nodes are interfaced with the network using lightweight communication protocols such as Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), and Hypertext Transfer Protocol Secure (HTTPS). By leveraging these protocols, PUF-based authentication can be seamlessly integrated into IoT systems, ensuring secure and efficient device-to-device and device-to-cloud communication.

3.6.3 LOGIC ANALYZER

Logic analyzers are a type of hardware probe designed to capture protocol-level information exchanged across one or more digital connections. Unlike oscilloscopes, which measure and display analog signals, logic analyzers work with digital systems where discrete voltage levels represent data and system states. The user has complete control over how a logic analyzer is configured and applied. They can select observation points within the system under test, assign meaningful names to input signals, and specify the conditions for capturing data based on the observed signal values. This allows the logic analyzer to be programmed to monitor a specific subset of system states under defined conditions. Similar to oscilloscopes, logic analyzers are available in various forms, ranging from high-performance stand-alone instruments to more compact USB-based devices.

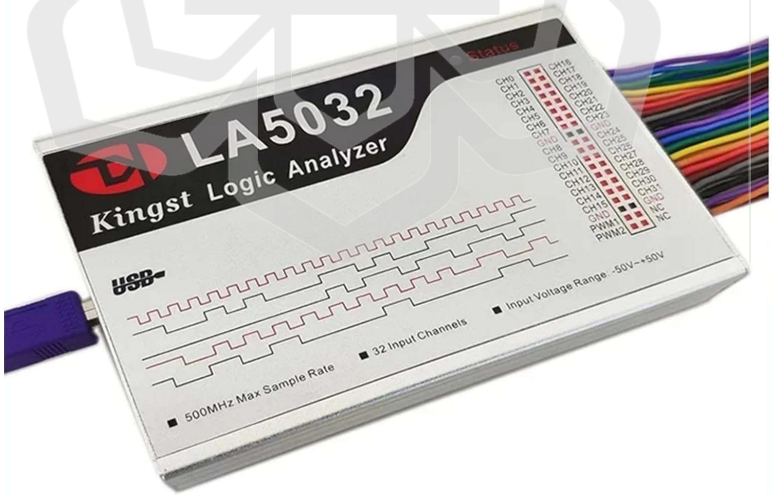


Figure 3.4 32-Channel, 500 MHz logic analyser

The LA5032 is a logic analyzer that offers high performance at a cost, with 32 input channels and a sampling rate of up to 500 MHz. When used with the KingstVIS

PC software, it can capture and sample signals across all 32 channels simultaneously, convert them into digital waveforms, and display them on the computer screen. The captured signals can then be analyzed using multiple standard protocols, providing clear and direct communication data. This makes the LA5032 highly effective for testing and analyzing digital systems such as MCU, ARM, and FPGA platforms, developing and debugging communication programs, and monitoring or recording digital signals over extended periods. When the Kingst Virtual Instruments (VIS) software is launched, the user interface is displayed as shown in Figure 3.13 below.

Figure 3.5 Kingst logic analyser GUI

As illustrated, the software interface is divided into several sections, each providing specific tools and functions for effective signal analysis and visualization.

i. *Toolbar:* Top of the GUI, including standard settings of the current device and the main menu button in the top right corner.

ii. *Threshold voltage*: The top left of the GUI is intended for selecting or customizing the threshold voltage value using the combo box.

iii. *The channel settings bar, located to the left of the GUI, displays the number and name of the* current measurement channels.

iv. *Waveform display windows*: In the middle of the GUI, the topmost is the timeline. The sampled waveform is displayed in the middle, with a scroll bar below it.

v. *Sampling result analysis windows*: On the right side of the GUI, the top half displays frequently used measurement results, and you can add an analyzer decoder to view the results in the bottom half.

3.7 EXPERIMENTAL SETUP

As shown in the proposed model of Figure 3.14, an arbiter PUF is implemented on an Intel Cyclone IV FPGA to generate unique device signatures that are used to encrypt AI or ML models before deployment. This approach ensures that AI models are executed only on authenticated devices, thereby preventing malicious reverse engineering attempts. IoT devices often display recognizable patterns in user interactions and communication with edge or cloud servers. Machine learning models can be trained to analyze these interaction patterns and the characteristics of network traffic. Such ML-based solutions are commonly applied for user authentication, data access control, and detection of DDoS attacks. Supervised ML techniques such as Support Vector Machines, Naïve Bayes, k-nearest neighbors, deep neural networks, and random forests are widely used to identify network intrusions, malware, DDoS attacks, and spoofing attempts.

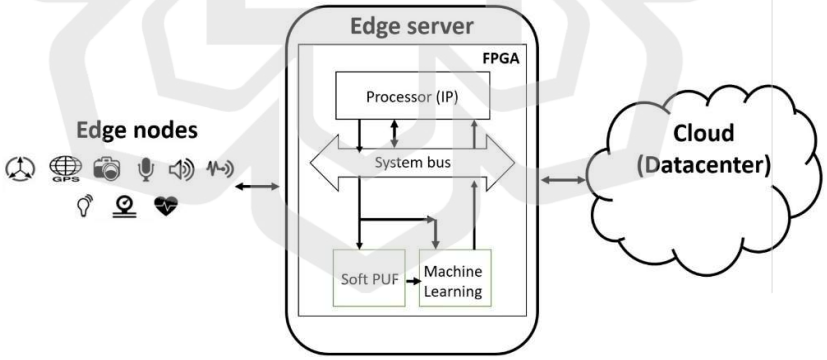


Figure 3.6 Proposed Framework

As illustrated in Figure 3.15, deployment of AI models on an edge device is divided into five stages.

i. *Problem definition:* Initially, it is crucial to define the problem that the proposed AI system will address. Without a precise problem statement, model development may lack direction.

ii. *Data acquisition:* Once the problem statement is agreed upon, an appropriate dataset must be collected. A large and representative dataset is essential to train models effectively and to capture the complexity of real-world scenarios.

iii. *Model training and optimization:* The AI model is trained using the dataset, and the network architecture is tuned to achieve optimal performance while avoiding overfitting. Commonly used frameworks such as Keras, TensorFlow, and PyTorch are written in Python.

iv. *Model translation and resource optimization:* The trained model should be translated into a language compatible with the target device, as most edge platforms do not support Python. In this phase, the model is optimized to fit the limited resources of edge hardware, thus minimizing memory footprint, computation complexity, and energy consumption.

v. *Integration and deployment:* Finally, the optimized AI model is integrated into the edge application by embedding the model within the system software while ensuring seamless integration with sensors, actuators, and communication interfaces.

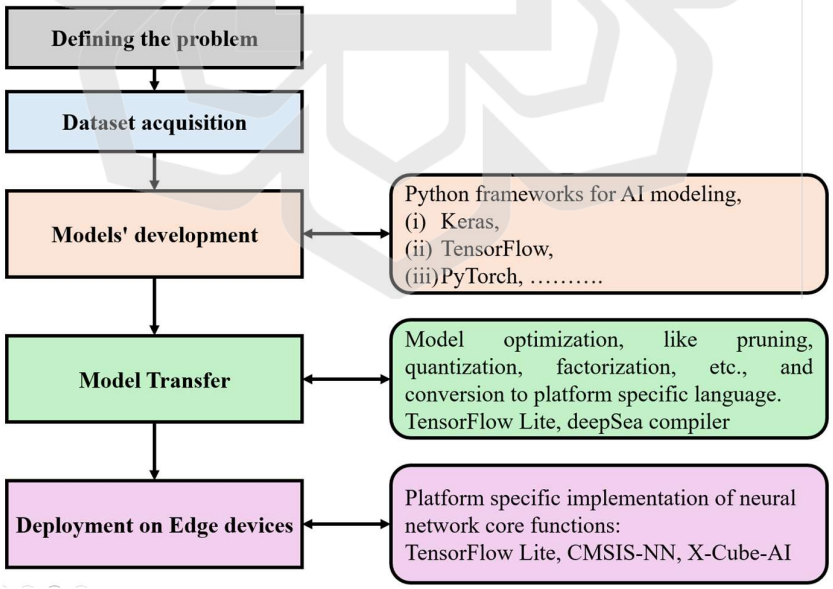


Figure 3.7 AI Implementation on Edge devices

3.8 DATA COLLECTION AND PROCESSING

The performance of PUFs is typically evaluated by acquiring Challenge–Response Pairs (CRPs) in real time. A logic analyzer–based acquisition system provides valuable insight into the digital behavior of the PUF but often requires additional noise filtering and preprocessing. Logic analyzers are widely preferred in PUF experiments because they offer several advantages, including:

i. Multi-channel sampling allows the simultaneous capture of challenge vectors, control signals, and response bits.
ii. High sampling rates ranging from hundreds of megahertz to several gigahertz, enabling the resolution of nanosecond-scale delays in PUF circuits.
iii. Built-in protocol support (e.g., UART, SPI), which facilitates the direct extraction of CRPs when the FPGA transmits responses through serial communication.

3.8.1 Pre-Processing Techniques

PUF responses might suffer from noise, instability, bias, and misalignment, thus making pre-processing essential to prepare the dataset for evaluation and secure applications. Some of the preprocessing tasks carried out include,

i. *Data Alignment and Synchronization:* It is an essential preprocessing step in PUF evaluation to ensure that each challenge is correctly paired with its corresponding response. During high-speed acquisition, timing mismatches, jitter, or transmission delays may cause misalignment, leading to corrupted CRPs. To address this, synchronization signals such as valid or start, embedded markers, or data framing protocols (e.g., UART/SPI) are employed to indicate when a stable response is available.
ii. *Majority Voting / Repetition:* The stability of PUF output is achieved by applying the same challenge multiple times, and the corresponding responses are recorded. The final response bit is then determined by majority voting.

iii. *Glitch Removal / Signal Cleaning:* Techniques are in place to discard pulses shorter than a predefined duration, adjusting logic thresholds on the acquisition system, and applying digital filters to suppress spurious transitions.

iv. *Error Detection:* Data corruption or data loss can be mitigated during the preprocessing stage by embedding parity bits, checksums, or counters in the FPGA output stream.

v. *Normalization and Formatting:* Normalization ensures that CRP data are represented uniformly, such as converting hexadecimal or decimal challenge values into fixed-length binary vectors and mapping them to corresponding standard binary response bits (0/1). Formatting involves organizing the CRPs into specific datasets for CAD tools, such as CSV files, MATLAB matrices, or Python NumPy arrays.

vi. *Dataset Balancing:* Raw responses from PUFs are often biased, meaning the distribution of 0s and 1s is not perfectly balanced. Dataset balancing mitigates this issue by ensuring a more even distribution of response bits. This can be achieved by selectively discarding excess samples of the dominant bit, resampling underrepresented outputs, or applying statistical adjustments to correct the imbalance.

3.7.2 Post-Processing Techniques

The suitability of PUF responses in cryptographic applications, such as secure key generation and authentication, can be further improved by applying hash functions and whitening techniques. Hash functions such as SHA-256 or SHA-3 provide strong cryptographic keys by enhancing randomness, making responses robust against prediction or reverse engineering. In contrast, whitening techniques like Linear Feedback Shift Registers (LFSRs) are lightweight techniques that minimize bias and decorrelate raw response bits, making them suitable for resource-constrained IoT devices. Although whitening improves statistical properties, it is not cryptographically secure and is often combined with hashing. The combination of these approaches has significantly improved the randomness, unpredictability, and resilience of PUF-generated data, enabling its effective use in securing IoT devices.

- *Hashing:* SHA stands for Secure Hashing Algorithm. A SHA reduces the input data into a smaller format, which is difficult to understand through the application of bitwise operations, modular additions, and compression functions. Hashing is similar to encryption, but the key difference is that hashing is one-way. Once the data is hashed, the resulting hash digest cannot be cracked unless a brute force attack is used. The SHA functions are a family of hashing algorithms that have been developed over time under the oversight of the National Institute of Standards and Technology (NIST). Examples of the SHA functions illustrated in Figure 3.16 are SHA-1, SHA-2, and SHA-3. The SHA-1 is gradually being phased out and is not recommended for use in any new designs.

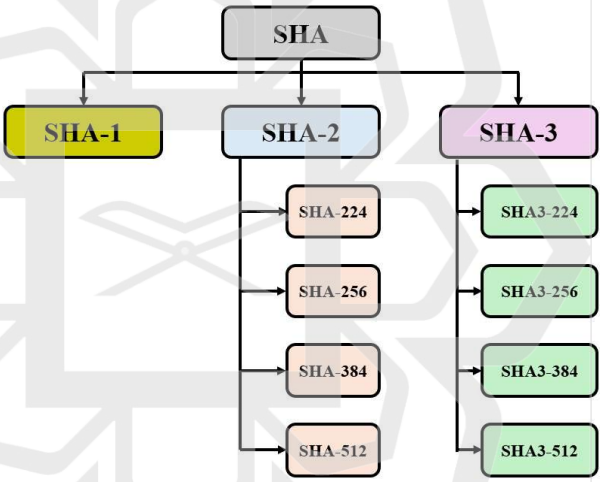


Figure 3.8 Classification of SHA

SHA-256 belongs to the SHA-2 family of algorithms, introduced jointly by the NSA and NIST in 2001. The "256" denotes its fixed hash digest size, meaning it always produces a 256-bit output regardless of the input message length. In contrast, SHA-3 adopts a fundamentally different approach in both algorithm and structural design. While SHA-2 is based on the Merkle–Damgård construction, which processes data in fixed-size blocks using linear operations, SHA-3 relies on the Keccak algorithm and a sponge construction. Unlike SHA-2, which produces outputs of fixed lengths, SHA-3's sponge construction allows for greater flexibility, enabling adjustable hash lengths as required (Baird et al., 2025).

Python provides a built-in library called ‘hashlib’, which offers a unified interface for various SHA algorithms. The module includes constructor methods for each hash type; for instance, the ‘.sha256()’ constructor is used to create a SHA-256 hash. In the context of PUF systems, response data extracted from an FPGA can be converted into a byte array and processed using ‘hashlib’. Applying SHA-256 generates a fixed 256-bit key, while SHA-3 offers enhanced resistance against cryptanalytic attacks and side-channel vulnerabilities. The resulting ‘hash’ ensures that PUF-derived keys are suitable for encryption, authentication, and secure communication protocols.

- *Linear Feedback Shift Register (LFSR) Whitening:* The whitening process transforms raw PUF response bits into a sequence with reduced bias and correlation. Lightweight methods, such as XOR-ing multiple response bits, use Linear Feedback Shift Registers (LFSRs) or simple combinational logic to achieve this mixing. An LFSR is essentially a shift register in which the input bit is generated as a linear function (typically XOR) of selected previous register states, known as taps. When raw PUF responses are passed through an LFSR, the output sequence is mixed in a way that suppresses predictable patterns and reduces bias. This whitening process produces a more balanced distribution of 0s and 1s, thereby enhancing the entropy and overall randomness of the dataset.

3.9 PERFORMANCE METRICS

The efficacy of ML-based PUF security systems in IoT can be examined through performance parameters such as accuracy, reliability, uniqueness, scalability, robustness, latency, and energy consumption. Careful optimization of these parameters is essential to ensure that IoT networks remain secure, resilient, and efficient while operating under the constraints of resource-limited devices. In the following subsections, the principal performance metrics of ML and PUFs are discussed briefly.

3.9.1 PUF Performance Indicators

The performance of PUF-based authentication systems is commonly assessed using intra-class and inter-class metrics. Intra-class variation reflects the reliability of the system, indicating how consistently a device produces identical responses under different environmental conditions. In contrast, inter-class variation measures the uniqueness of the system by quantifying the differences between responses from other devices when presented with the same inputs, thereby ensuring distinguishability (Yu et al., 2012).

Figure 3.9 PUFs Performance parameters

Figure 3.17 illustrates metrics such as uniqueness, reliability, randomness, correctness, etc., which are used to evaluate PUF quality and assess its applicability in IoT security. These metrics are obtained by collecting response bits generated from a set of challenges applied to the PUF. Since each application has distinct requirements, the relative importance of these metrics may vary (Joshi et al., 2017; Ram et al., 2022).

- **Uniqueness**: Uniqueness is a key characteristic of a PUF that reflects its ability to generate distinct responses when subjected to an identical set of challenges across different dies in a lot. It is quantified as the average inter-chip Hamming Distance (HD) of the responses obtained from a group of

chips. For an ideal PUF, the uniqueness value is approximately 50%, indicating that half of the response bits should differ between devices. For instance, in an FPGA-based k-n-bit PUF with responses P_1, P_2, P_3, …, P_k, the average Hamming Distance, as defined in Equation (3.8), serves as the measure of uniqueness (Zhang et al., 2023).

$$u = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(P_i, P_j)}{n} \times 100\% \qquad (3.8)$$

- **Reliability**: The reliability of a PUF, defined as its ability to consistently reproduce the same response under noisy and varying environmental conditions, is a critical performance parameter. Factors such as temperature fluctuations, supply voltage variations, and device aging can introduce deviations in the PUF signatures. The ideal reliability is 100%, and it can be quantified using Equation (3.9).

$$u = \frac{1}{x} \sum_{y=1}^{x} \frac{HD(R_i, R_{i,y})}{n} \times 100\% \qquad (3.9)$$

Where x represents the times of sampling, n is the number of bits of a signature generated by a PUF; $R_{i,y}$ is the y^{th} sampling of R_i.

- **Randomness:** It is a measure of a PUF's ability to produce '0' and '1' in its response bits with equal probability. Ideally, the randomness of a PUF should be 100%. PUF-based authentication protocols rely on inherent physical variations introduced during the semiconductor manufacturing process, which provide static randomness. However, dynamic sources of randomness, such as noise, can corrupt the extracted identification (ID) from the PUF and consequently reduce its reliability (Schaub et al., 2020).

$$Randomness = 1 - |P_r(ID = 0) - P_r(ID = 1)| \qquad (3.10)$$

For 2^M challenges, the probability of obtaining an ID at 0 and 1 can be given as,

$$Randomness = 1 - \left| erf\left(\frac{E(D_R)}{\sigma\sqrt{2 \cdot M}}\right)\right| \quad (3.11)$$

Where D_R is the pdf of $\sum_{i=1}^{M} d_{c_i}^i$

For a variance of $M\sigma^2$, the randomness expression Equation (3.12, 3.13) is given by (Jouini, Danger, & Bossuet, 2011)

$$P_R(ID = 0) = 1 - P_R(ID = 1), \text{ and} \quad (3.12)$$

$$= P_r\left(\sum_{i=1}^{M} d_{c_i}^i < 0\right) \quad (3.13)$$

- **Correctness:** It is defined as the probability that the response R generated by the PUF for a given challenge C is always the same across multiple evaluations under varying conditions such as temperature, voltage fluctuations, aging, or noise.

$$C = 1 - BER \quad (3.14)$$

Where, Bit Error Rate (BER is the fraction of bits in the response that flip compared to a reference response when the same challenge is reapplied.

$$BER = \frac{1}{n}\sum_{j=1}^{n} HD(R_{ideal}, R_i) \quad (3.15)$$

R_{ideal}, is the golden response (reference).

R_i, is the response observed under test condition *i*, and

HD is the hamming distance.

Ideal value of Correctness ≥ 95%.

- **Uniformity:** The proportion of '0's and '1's in the response bits of a PUF is defined by uniformity. For truly random PUF responses, this proportion must be 50%. For the i^{th} PUF instance, uniformity is defined as

$$(Uniformity)_i = \frac{1}{n}\sum_{l=1}^{n} r_{i,l} \times 100\% \quad (3.16)$$

Where n is the total number of response bits, and

$r_{i,l}$ is the value of the l^{th} bit (0 or 1) in the response of the i^{th} PUF.

- **Bit Aliasing:** Bit-aliasing measures how evenly each response bit of a PUF outputs a 0 or 1 across a set of devices when subjected to the same challenge. Bit-aliasing of the I^{th} bit in the PUF identifier is estimated as the percentage Hamming Weight (HW) of the I^h bit of the identifier across k devices.

$$(\text{Bit-aliasing})_l = \frac{1}{k}\sum_{i=1}^{k} r_{i,l} \times 100\% \quad (3.17)$$

Where, k = number of PUF instances (chips/devices),

$r_{i,l}$ = response bit at position l from the i^{th} device,

l = bit position index.

- **Steadiness:** The steadiness measures the degree of bias of a response bit towards '0' or '1' over T samples. It is defined as,

$$S_n = 1 + \frac{1}{K \cdot L}\sum_{k=1}^{K}\sum_{l=1}^{L} log_2 \max\left(p_{n,k,l}, 1 - p_{n,k,l}\right) \quad (3.18)$$

Where, $p_{n,k,l} = \frac{1}{T}\sum_{t=1}^{T} r_{n,k,t,l}$

This parameter is somewhat like the correctness parameter. A lower value of steadiness will produce a lower correctness.

- **Security:** PUF Security is the ability of the PUF to withstand prediction, cloning, and modelling attacks. It is evaluated using entropy measures, ML attack resistance, and unpredictability of CRPs.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p\,(x_i) \quad (3.19)$$

Shannon Entropy measures the uncertainty in the PUF responses. The higher entropy value leads to better security.

- **Diffuseness:** Diffuseness measures how well the PUF responses spread out across the output space when different challenges are applied to the same PUF instance. Diffuseness is usually defined as the Hamming distance between two response vectors of the same PUF instance under different challenges, averaged across many pairs of challenges.

$$\text{Diffuseness} = \frac{1}{M}\sum_{i=1}^{M}\frac{HD\left(R(C_i), R(C_j)\right)}{n} \times 100\% \qquad (3.20)$$

Where, $R(C_i)$ = response vector of the PUF for challenge C_i,
HD = Hamming distance function,
n = response length (number of bits),
M = number of challenge pairs tested.

3.9.2 ML Model Performance Metrics

A machine learning (ML) model is validated by assessing its prediction performance. Performance metrics shown in Figure 3.18 provide a way to quantify how well the model works.

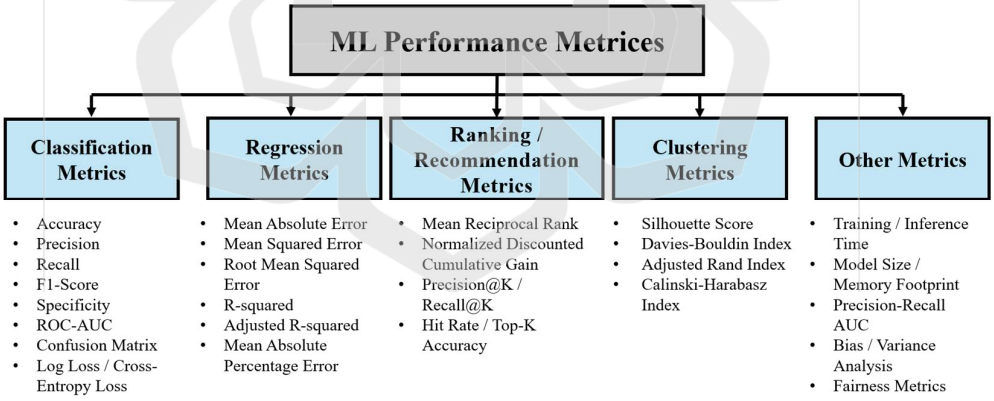


Figure 3.10 Classification of ML Model evaluation metrices

Evaluating these metrics is essential for optimizing an ML model before deployment and for improving it after release into production (V7 Labs, n.d.).

3.9.2.1 Classification Metrics

In classification modelling, various performance metrices can be predicted from a matrix called the confusion matrix illustrated in Figure 3.19. For binary classification, the confusion matrix divides the test samples into four categories, depending on their true and predicted labels.

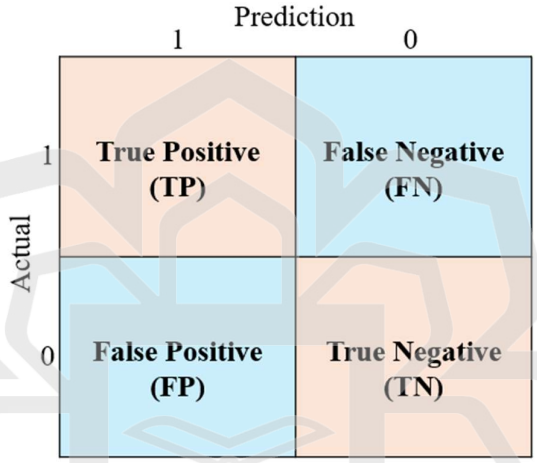


Figure 3.11 Confusion matrix

True Positives (TP): Samples for which the true and predicted labels are both 1.
True Negatives (TN): Samples for which the true and predicted labels are both 0.
False Positives (FP): Samples for which the true label is 0 and the predicted label is 1.
False Negatives (FN): Samples for which the true label is 1 and the predicted label is 0.

- *Accuracy:* It's a measure of the overall correctness of a model's predictions by calculating the ratio of correct predictions to the total number of predictions.

$$\text{Accuracy} = \frac{\sum \text{TruePositive} + \sum \text{TrueNegative}}{\text{SampleSize}} \tag{3.21}$$

- *Precision:* Precision measures the ratio of true positive predictions to the total predicted positives. It reflects the model's effectiveness in reducing false positives and ensuring the accuracy of its positive predictions.

$$\text{Precision} = \frac{\sum TruePositive}{\sum TruePositive + \sum FalsePositive} \quad (3.22)$$

- *Recall, also known as sensitivity or the true positive rate, evaluates a model's ability to detect positive instances from all actual positive cases correctly.*

$$Recall = \frac{\sum TruePositive}{\sum TruePositive + \sum FalseNegative} \quad (3.23)$$

- *F1 Score:* The F1 Score merges precision and recall into a single metric, offering a balanced evaluation of both. It represents the harmonic mean of precision and recall and is particularly valuable when managing the trade-off between false positives and false negatives.

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}} \quad (3.24)$$

- *Specificity:* Specificity measures a model's ability to identify negative cases among all actual negatives correctly. It represents the proportion of true negative predictions relative to the total number of actual negative instances.

$$\text{Specificity} = \frac{\sum TrueNegative}{\sum TrueNegative + \sum FalsePositive} \quad (3.25)$$

- *Area Under the ROC Curve (AUC-ROC):* The ROC curve illustrates a model's performance across different classification thresholds by plotting recall (sensitivity) on the y-axis against specificity on the x-axis. The Area Under the ROC Curve (AUC-ROC) condenses this information into a single value that reflects the model's overall discriminative ability. A higher AUC-ROC value indicates stronger predictive performance, with an AUC of 1 representing a perfect classifier.

Regression metrics are used to evaluate the performance of algorithms that predict continuous numerical values. Some of the critical regression metrics are discussed below.

3.9.2.2 Mean Absolute Error (MAE)

It measures the average magnitude of errors between predicted and actual values without considering their direction. MAE is especially useful in applications that aim to minimize the average error and is less sensitive to outliers than other metrics like Mean Squared Error (MSE).

A dataset with 'n' observations, where 'y' is the actual value and '$\hat{y}_i$' is the predicted value for the i^{th} data point in the dataset, the Mean Absolute Error (MAE) can be calculated using equation (3.25),

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.26}$$

3.9.2.3 Mean Squared Error (MSE)

It measures the average of the squared differences between predicted and actual values, thereby placing greater emphasis on larger errors. MSE is typically preferred when the objective is to penalize large deviations more heavily or when the error distribution is assumed to follow a Gaussian pattern.

3.9.2.4 Root Mean Squared Error (RMSE)

It has the same unit as the target variable, making it more interpretable and easier to relate to the problem context than MSE.

3.10 NIST RANDOMNESS TEST

Randomness is a fundamental characteristic in the field of cryptography. The generation of random numbers is a challenging task, and equally critical is the evaluation of the quality of the data generated. Statistical randomness tests typically produce a p-value, which represents the probability that a perfect random number generator would produce fewer random sequences than the sequence under evaluation. Most empirical randomness tests, including those from the National Institute of Standards and Technology (NIST) Statistical Test Suite (STS), are based on statistical hypothesis testing.

The NIST STS suite comprises 15 distinct empirical tests specifically designed to analyze binary sequences. These tests assess the randomness of data based on individual bit statistics or the statistical properties of blocks of bits, while the entire bitstream is evaluated for global randomness. To detect local non-randomness, the bitstream is divided into several large segments, and specific characteristics are computed for each segment before being combined into the final test statistics. Each NIST STS test is defined by a particular test statistic, which falls into one of the following three categories:

- *bits*: Various characteristics of bits are analyzed, like the proportion of bits, frequency of bit change, and cumulative sums.
- *m-bit blocks:* Analyze distribution of m-bit blocks ($m < 30$) within the sequence or its parts,
- *M-bit parts:* Analyses complex properties of M-bit ($M > 1000$) parts of the sequence, like rank of the sequence viewed as a matrix, spectrum of the sequence, or linear complexity of the bitstream.

Most randomness tests are parameterized by ‘n’, which represents the bit length of the binary sequence under evaluation. A second parameter, denoted as ‘m’ or ‘M’, is also used depending on the specific test. Table 3.1 summarizes the number of sub-tests included in the NIST STS suite. Notably, the non-overlapping template matching test has a variable number of sub-tests determined by the value of ‘m’ (Marton, & Suciu,

2015). A brief description of NIST randomness tests is presented in the following paragraphs (National Institute of Standards and Technology, n.d.; Bassham et al., 2010).

Table 3.1 Recommended Bitstream Parameters

#	Name of the Test	n	M or m	Sub-Test #
1	Frequency	$n \geq 100$		1
2	Frequency within a block	$n \geq 100$	$20 \leq M \leq n/100$	1
3	Runs	$n \geq 100$		1
4	Longest run of ones	$n \geq 128$		1
5	Rank	$n > 38912$		1
6	Spectral	$n \geq 1000$		1
7	Non-overlapping Template Matching	$n \geq 8m - 8$	$2 \leq m \leq 21$	148*
8	Overlapping Template Matching	$n \geq 10^6$		1
9	Maurer's Universal	$n > 387840$		1
10	Linear Complexity	$n \geq 10^6$	$500 \leq M \leq 5000$	1
11	Serial		$2 < m < [\log_2 n] - 2$	2
12	Approximate Entropy		$m < [\log_2 n] - 5$	1
13	Cumulative Sums	$n \geq 100$		2
14	Random Excursions	$n \geq 10^6$		8
15	Random Excursions variant	$n \geq 10^6$		18

- *Frequency (Monobits) test:* This test verifies whether the number of ones and zeros in a sequence is approximately equal, as expected in a truly random sequence.
- *Test for frequency within a block:* The test evaluates the proportion of zeros and ones within M-bit blocks. It verifies whether the frequency of ones in each M-bit block is approximately equal to M/2.

- *Runs test:* This test measures the occurrence of consecutive runs of zeros and ones throughout the entire sequence. A run of length ‘k’ is defined as a substring of exactly ‘k’ identical bits, bounded on both sides by a bit of the opposite value. The test evaluates whether the frequency of transitions between such runs indicates oscillations that are either too rapid or too slow for a truly random sequence.
- *Test for the longest run of ones in a block:* This test examines whether the length of the longest run of ones within ‘M-bit’ blocks of the sequence is consistent with the expected distribution for a truly random sequence. Any deviation in the observed length of the longest run of ones also implies a corresponding irregularity in the longest run of zeros.
- *Random binary matrix rank test:* This test evaluates the rank of disjoint sub-matrices derived from the sequence. Its purpose is to detect linear dependencies among fixed-length substrings of the original sequence, which would indicate deviations from true randomness. For instance, if the rank of many sub-matrices is lower than expected, it suggests redundancy or correlation among the bits, meaning the sequence lacks the independence characteristic of a truly random source.
- *Discrete Fourier Transform (Spectral) test:* This test focuses on the peak heights in the discrete Fast Fourier Transform (DFT) of the sequence. Its purpose is to detect periodic features, such as repetitive patterns occurring in proximity, which would indicate deviations from true randomness.
- *Non-Overlapping (Aperiodic) template matching test:* The purpose of this test is to reject sequences that contain an excessive number of occurrences of a given aperiodic (non-periodic) pattern. In this test, as well as in the Overlapping Template Matching Test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window shifts forward by one bit. When the pattern is detected, however, the window is reset to the position immediately following the identified pattern, and the search continues.
- *Overlapping (Periodic) template matching test:* This test focuses on counting the occurrences of predefined target substrings within the sequence. Its purpose is to reject sequences that deviate from the expected

number of runs of ones of a specified length. A deviation in the runs of ones of a given length inherently implies a corresponding deviation in the runs of zeros. In this test, when the target pattern is found, the m-bit window slides forward by only one bit, and the search resumes, allowing overlapping occurrences to be detected.

- *Maurer's universal statistical test:* This test measures the number of bits between matching patterns in the sequence. Its purpose is to determine whether the sequence can be significantly compressed without losing information. If the sequence is overly compressible, it indicates redundancy and, therefore, a lack of true randomness.
- *Linear complexity test:* The length of the generating feedback register required to reproduce the sequence is estimated in this test. Truly random sequences typically require more extended feedback registers, whereas sequences that can be generated with short registers indicate predictability and non-randomness.
- *Serial test:* This test examines the frequency of all overlapping m-bit patterns throughout the entire sequence. Its purpose is to determine whether the occurrences of the 2^m possible m-bit patterns are approximately equal, as expected in a truly random sequence. The patterns are allowed to overlap during the analysis.
- *Approximate Entropy test:* This test evaluates the frequency of overlapping 'm-bit' patterns in a sequence. It compares the observed frequencies of overlapping blocks of lengths 'm' and 'm+1' with the expected frequencies for a truly random sequence, providing a measure of the sequence's regularity or predictability. If overlapping m-bit sequences appear with unexpected frequencies, it indicates that the PUF outputs are partially predictable, reducing their effectiveness for secure key generation.
- *Cumulative Sum (Cusum) test:* This test examines the maximal deviation from zero of the random walks formed by the cumulative sum of the adjusted sequence values, where each bit is mapped to -1 or +1. Its purpose is to determine whether the cumulative sum of partial sequences in the tested bitstream is unusually large or small compared to what is expected for a truly random sequence. In effect, the cumulative sum represents a

random walk: for a random sequence, the walk should stay near zero, whereas non-random sequences produce excursions that are too large, indicating deviation from randomness.

- *Random Excursions test:* The focus of this test is the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is found if partial sums of the (0,1) sequence are adjusted to (-1, +1). A random excursion of a random walk consists of a sequence of n steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number of visits to a state within a random walk exceeds what one would expect for a random sequence.
- *Random Excursions Variant test:* This test analyzes the number of times a particular state is visited in the cumulative sum random walk of the sequence. Its purpose is to identify deviations from the expected frequency of visits to different states, helping to detect non-random patterns in the sequence.

3.11 CALIBRATION METHODS

Calibration plays a critical role in ensuring the accuracy, reliability, and reproducibility of CRP acquisition when implementing PUFs on FPGAs. Since PUF responses are highly sensitive to environmental conditions like temperature, voltage fluctuations, aging, as well as measurement artifacts such as timing misalignment, noise, and signal distortions, calibration methods are employed to minimize errors before feeding the data into ML-based security models.

- *Timing Calibration:* Logic analyzers capture digital signals at high sampling rates varying between hundreds of MHz and GHz. However, misalignment between challenge signals, response outputs, and control clocks can lead to corrupted CRPs.

  Calibration method:

  i. Use a known reference signal, such as an FPGA internal clock or test pattern generator, to align acquisition channels.

ii. Apply trigger-based synchronization in the logic analyzer to ensure consistent alignment of challenge vectors with corresponding responses.

- *Voltage and Signal Level Calibration:* FPGA outputs may degrade due to voltage drop, temperature changes, or I/O mismatches. This can cause logic analyzers to misinterpret "0" and "1" levels.

  Calibration method:

  i. Adjust threshold voltage levels on the logic analyzer to match FPGA I/O standards (e.g., LVTTL, LVCMOS).
  ii. Periodically recalibrate using known test vectors to verify that digital transitions are correctly captured.

- *Environmental Calibration:* PUF responses are known to vary with temperature, supply voltage, and device aging.

  Calibration method:

  i. Use environmental profiling, where CRPs are collected across controlled temperature/ voltage ranges, and corrections are applied.
  ii. ML preprocessing may include normalization or majority voting to compensate for environmental drift.

- *Noise Filtering and Signal Cleaning:* High-frequency noise or glitches can distort CRP acquisition.

  Calibration method:

  i. Apply digital filtering techniques (e.g., glitch removal, debouncing) during preprocessing.
  ii. Repeated measurements followed by majority voting ensure that transient noise does not bias the dataset.

- *Data Alignment and Synchronization:* Multi-channel acquisition can result in skew between channels.

  Calibration method:

  i. Perform multi-channel skew calibration by applying the same known signal to all acquisition channels and adjusting offsets.

ii. Use post-processing alignment algorithms to re-synchronize challenge-response mapping before ML training.

- *Statistical Calibration for ML Training:* Before feeding CRPs into an ML model, statistical calibration ensures data quality.

Calibration method:

i. Compute intra-class and inter-class metrics to check the reliability and uniqueness of CRPs.

ii. Apply whitening (e.g., LFSR or hash-based methods) to eliminate bias in raw PUF data.

iii. Normalize datasets so that ML models are not biased by imbalanced or skewed responses.

3.12 RESEARCH ETHICS

This research was conducted following responsible research practices. Machine learning based attacks on PUF challenge response pairs were implemented solely for academic evaluation of security robustness. The findings are intended to support the design of more resilient hardware security mechanisms and not to facilitate unauthorized exploitation or misuse of security vulnerabilities.

3.13 SUMMARY

This chapter outlines the methodology employed for designing, implementing, and evaluating the proposed PUF architectures for IoT security. The chapter describes the tools utilized, including Quartus Prime for FPGA synthesis, HDL for hardware design, and Visual Studio Code for Python-based data analysis. It also details the hardware setup involving Intel Cyclone FPGA devices, their application across IoT layers, and the use of a logic analyzer for signal observation. The experimental setup, data collection, and processing methods are explained, with a focus on pre- and post-processing techniques to ensure data reliability. Performance evaluation focuses on PUF indicators such as uniqueness, reliability, and randomness, as well as ML model metrics like accuracy. The chapter concludes with the application of NIST randomness tests and calibration methods to ensure robustness, stability, and secure operation of the proposed FPGA-based PUF systems. The separation between Chapter 3 (Methodology) and

Chapter 4 (Design of Proposed Schemes) is intentional to maintain a clear and logical structure in the thesis.

CHAPTER FOUR

DESIGN OF PROPOSED SCHEMES

4.1 INTRODUCTION

This chapter presents the design of the proposed FPGA-based PUF authentication framework. The system architecture is first described, emphasizing the integration of Arbiter and Ring Oscillator (RO) PUF designs to achieve robust device authentication. The subsequent sections elaborate on the FPGA implementation details, including hardware resource utilization, clock synchronization mechanisms, and configuration procedures. The testbench and simulation environment are then introduced to validate the functional correctness of the proposed schemes before hardware realization. Finally, the chapter discusses security enhancements against machine learning attacks, with particular focus on countermeasures such as Challenge–Response Pair (CRP) filtering and randomness validation through standard statistical test suites.

4.2 PROPOSED RESEARCH WORK

The selection of a delay based PUF for implementation on the Intel Cyclone IV E FPGA is motivated by its compatibility with the FPGA's digital fabric, where intrinsic routing delays can be exploited to generate unique responses. Both Arbiter PUF (APUF) and Ring Oscillator (RO) PUF architectures are implemented, where the APUF uses symmetric multiplexer based delay paths controlled by a challenge input and a D flip flop as an arbiter, while the RO PUF relies on frequency differences between identically designed ring oscillators. During implementation, synthesis constraints are applied to preserve delay variations, and manual placement of logic is avoided to maintain natural routing randomness and enhance entropy. CRPs are extracted by applying input challenges and capturing output responses using a logic analyzer interface, enabling efficient CRP dataset collection for both architectures. The collected data is then evaluated in terms of uniqueness, reliability, and uniformity.

4.2.1 Selection of Delay-Based PUFs

The selection of delay-based PUFs over other PUF architectures is preferred in the work due to their digital nature, scalability, and suitability for FPGA-based implementations. Compared to memory-based (e.g., SRAM PUFs) or optical PUFs, delay-based architectures are easily realizable using standard digital logic elements, such as multiplexers, flip-flops, and inverters, without the need for specialized fabrication or analog components (Plusquellic et al., 2022). Secondly, delay-based PUFs support a large CRP space, unlike memory-based PUFs that produce a limited number of fixed responses. This large CRP capability enhances security, reconfigurability, and resistance to modeling and replay attacks, which are essential for robust device authentication (Plusquellic et al., 2024). Furthermore, delay-based PUFs are characterized by low hardware overhead and power consumption, making them ideal for resource-constrained IoT and embedded environments (Zhou et al., 2019). Therefore, delay-based PUFs are chosen in this research for their balance of simplicity, scalability, and security effectiveness. Their ability to be efficiently implemented on FPGA platforms and to produce statistically random, unique, and stable outputs makes them well-suited for the proposed hardware-assisted security architecture for IoT applications.

4.3 IMPLEMENTATION PLAN

The vulnerability of APUF implemented on an FPGA to machine language modeling attack is assessed in the proposed research work. If an ML model accurately predicts the PUF's response, it is considered broken from a security perspective. Figure 4.1 illustrates the implementation of the proposed PUF-based authentication system on an FPGA device, followed by ML modeling. During the initial setup, a dataset comprising challenges, their corresponding PUF responses, and timestamps is extracted from the FPGA I/O interface to train the desired ML model. Python 3.12.3 on the Visual Studio framework, along with native libraries, is used to develop an ML model for tasks such as authentication, key generation, and security. Various stages in research work are discussed briefly in the following subsections.

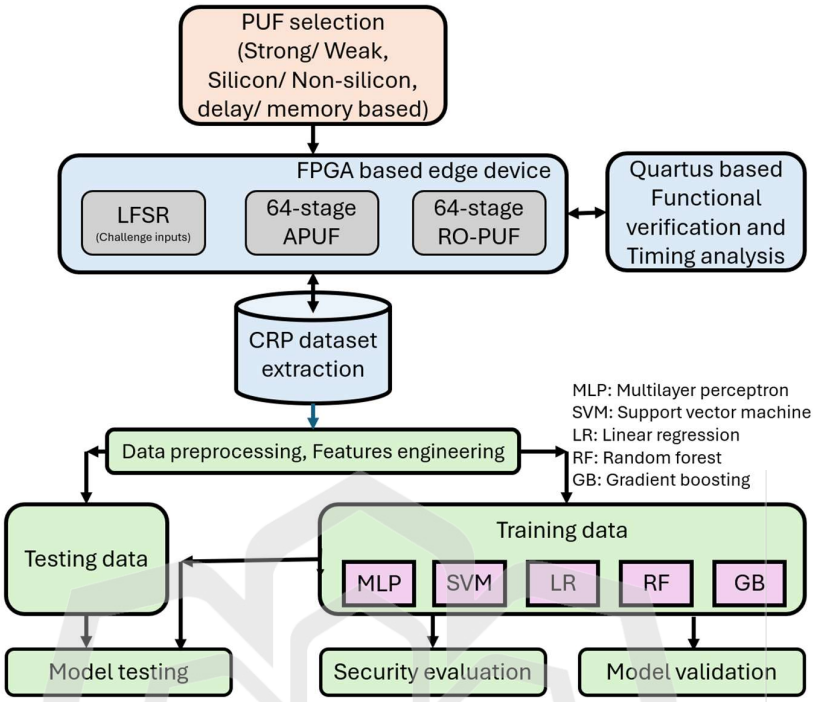


Figure 4.1 Design of PUFs' authentication architecture

- PUF Selection & Characterization
  i. *Selection of specific PUF:* Arbiter PUF is a delay-based PUF based on the path differences of a signal traveling through two symmetrical layouts. The arbiter PUF can be applied to devices like ASICs and FPGAs. Arbiter PUFs can be used in authentication, anti-counterfeiting, and lightweight cryptographic primitives.
  ii. *Type:* A 64-stage Arbiter PUF (APUF).
  iii. *Hardware platform:* Implemented on an Intel Altera Cyclone IVE FPGA-based edge device.
- Challenge Generation & Data Collection
  i. Generating the 64-bit inputs ("challenge") for the PUF and recording its corresponding outputs ("response") to create a dataset.
  ii. LFSR (Linear Feedback Shift Register): It is used to generate a long, pseudo-random sequence of challenge input bits. It feeds random challenges (C_0 to C_{63}) into the 64-stage APUF.

iii. CRP Dataset Extraction: For each challenge input, the corresponding response bit (0 or 1) is captured using a logic analyzer. The collected CRPs from the raw dataset are stored in a CSV or TXT file.

- Data Preprocessing & Feature Engineering
  i. Preparing raw CRP data to make it suitable and adequate for training ML models is one of the most critical steps in the pipeline. This process is called data preprocessing or data preparation.
  ii. CRPs are usually high-dimensional, noisy, and device-dependent; thus, a systematic pipeline is needed to make them clean, consistent, and ML-ready.
- Model Training (The Attack)
  i. A portion of the collected CRP data, also called "training data", is used to train various ML algorithms in the hidden relationship between the challenges and the PUF's responses.
  ii. Five standard ML models used in the research are MLP (Multilayer Perceptron), SVM (Support Vector Machine), LR (Linear Regression/Logistic Regression), RF (Random Forest), and GB (Gradient Boosting).
- Model Testing & Evaluation
  i. The performance of the trained models is evaluated on a different set of data ("testing data") to verify the prediction accuracy of unknown responses.
  ii. A higher prediction accuracy of the ML model on PUF CRP data indicates vulnerability, making the model insecure. Lower accuracy means that the PUF is hard to model, ensuring stronger security.
- Security Evaluation & Functional Verification
  i. The final step is to draw conclusions about the PUF's security and verify the physical implementation.
  ii. Quartus-based Functional Verification and Timing Analysis: Quartus Prime is a primary software tool for FPGA design (for Intel FPGAs).
    a) Functional Verification: Ensures that the VHDL PUF correctly implements the intended logic, generates the expected responses

for predefined test vectors, and maintains stable behavior under repeated sampling.

b) Timing Analysis involves examining combinational path delays, setup and hold times, and metastability behavior that influence PUF outputs. By back-annotating these delays into the simulation, real silicon-like behavior can be reproduced. This process is essential for validating ML model results and for understanding the physical characteristics that may have contributed to the success of an attack.

4.4 DESIGN OBJECTIVES AND REQUIREMENTS

The design of the proposed APUF and RO PUF circuits is driven by objectives and requirements that ensure both security robustness and hardware efficiency on FPGA platforms. The proposed security system should ensure uniqueness and reliability under environmental variations and remain unclonable.

4.4.1 Design Guidelines

The design must ensure perfect nominal symmetry, as any structural asymmetry introduced during the design phase may overshadow the subtle manufacturing process variations that are essential for PUF operation.

- Arbiter PUF: Dual signal paths should be perfectly symmetric, with an equal number of switches, LUTs, and identical routing resources. Additionally, manual placement and routing constraints are often essential to balance delays, minimize systematic skew, and maintain the PUF's unpredictability and robustness.
- RO-PUF: Hard macros or placement constraints ensure identical implementation of the Ring Oscillator and counter, minimizing variation. Clock and enable signals are routed through global resources to provide low skew and stable operation.

4.4.2 Post-Processing

Raw PUF responses are inherently noisy and unsuitable for direct use as cryptographic keys. Therefore, preprocessing and stabilization mechanisms are required before subsequent data processing or cryptographic operations can be performed.

- *Whitening (Entropy Extraction / Fuzzy Extraction):* The goal of whitening is to produce a stable and uniform (unbiased) output from the noisy and biased raw PUF response. LFSR whitening is a lightweight post-processing technique in which a pseudo-random sequence generated by a Linear Feedback Shift Register is XORed with PUF output bits. This method is highly hardware-efficient, requiring only a small number of flip-flops and XOR gates, making it suitable for FPGAs and resource-constrained edge devices. However, to achieve cryptographic-grade randomness, LFSR whitening is often combined with stronger techniques such as error correction or hashing.
- *Hashing (Randomness Extraction & Final Key Derivation):* The whitened data is generally stable and has high entropy, but it may still be too long or exhibit uneven entropy distribution across its bits. To address this, hash functions such as SHA-128, SHA-256, or SHA-512 are commonly applied. These functions compress and transform noisy, biased, or low-entropy PUF responses into fixed-length, high-entropy digests, ensuring strong randomness, uniformity, and unpredictability suitable for cryptographic applications.

4.4.3 Hardware and Implementation Requirements

The flowchart in Figure 4.2 provides a step-by-step illustration of the operation of an APUF. Its core principle is to construct two nominally identical delay paths, whose configuration is determined by the input challenge, and then measure which path produces a faster signal. The arbiter compares the two signals, resulting in a single-bit response that is unique to the specific chip and depends on the applied challenge.

- Entity Declaration: The VHDL modeling process begins with the declaration of the input and output ports of the PUF circuit.

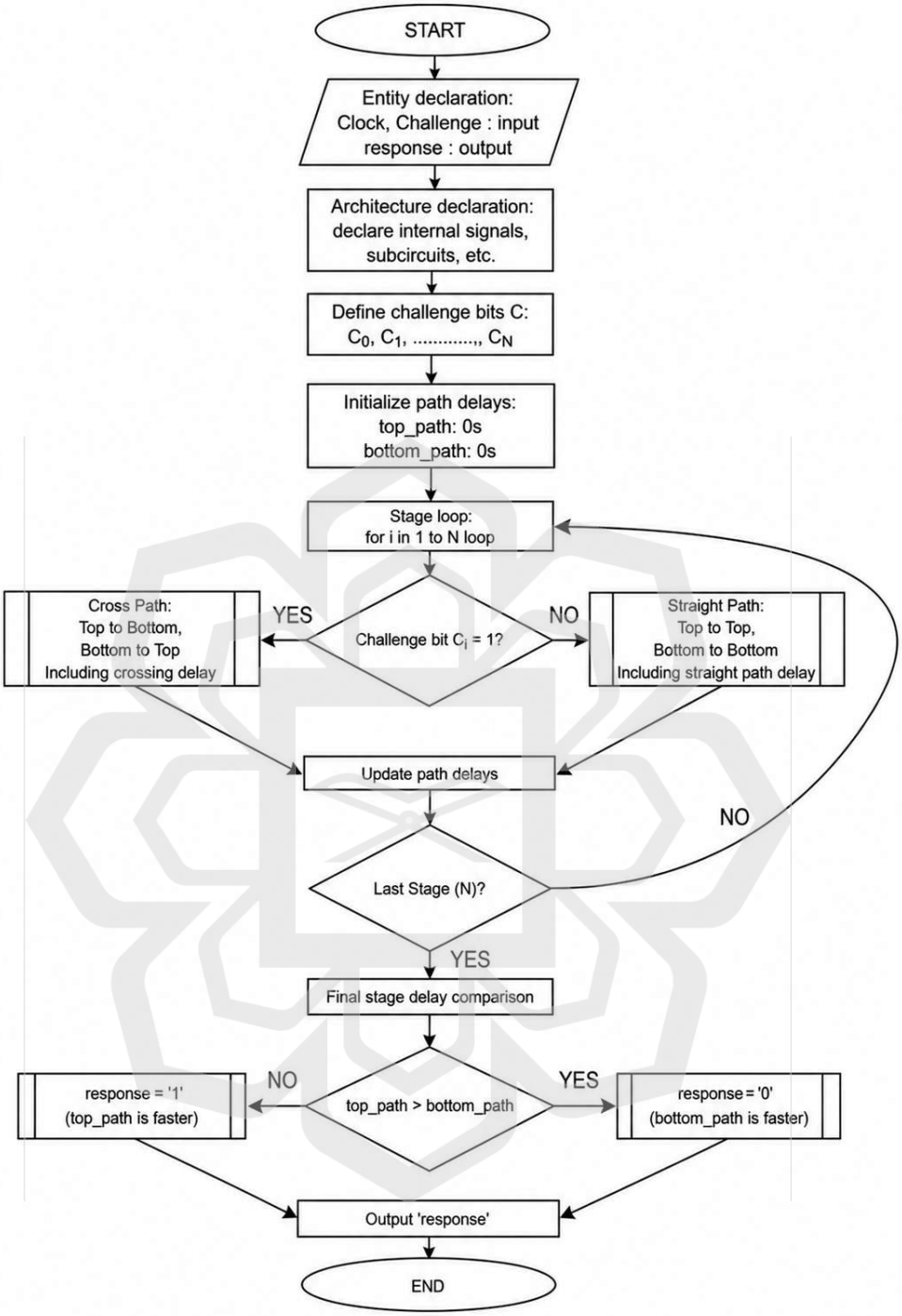


Figure 4.2 VHDL model of APUF circuit

i. Clock: It's an input signal used to synchronize the operations and sample the final output.

ii. Challenge: A multi-bit input used to configure the routings of delay paths through a chain of multiplexers.

iii. Response: It is a single-bit output (0 or 1) from a flip-flop.

- Architecture Declaration: It declares internal signals (like top_path and bottom_path) and sub-components (e.g., multiplexers (MUXes) that form each stage, delay elements, and the final arbiter circuit).
- Initialize Path Delays: Define top_path_delay and bottom_path_delay to zero. These two variables or signals accumulate the total propagation delay for each of the two parallel paths as they go through each configurable stage.
- Configurable Delay Paths: It's the main functional loop that processes N stages corresponding to N challenge bits. For each stage 'i' from 1 to N, the challenge bit C_i is checked to determine how the signal paths are configured for each stage.
    i. If Challenge Bit C_i = 1:
        - Action: The signal propagation paths are configured to cross. The signal propagating on the top path is exchanged with the bottom path for the next stage.
        - The delay intrinsic to the crossing wires and switching elements (MUXes) is added to the respective path's total delay.
    ii. If Challenge Bit C_i = 0:
        - Action: The paths are configured to go straight. The top path continues the top, and the bottom path continues the bottom.
        - The delay of the straight wires and switches is added to the respective path's total delay.
- Update Path Delays: The measured delay for the current stage's path, whether straight or crossed, is added to the running total for both the top and bottom paths.
- Final Arbitration and Response Generation: Once the loop processes all N stages, the final step is to determine the winner of the race.
    i. The accumulated delays, top_path_delay and bottom_path_delay, are compared by a circuit called an Arbiter. Generally, an arbiter is a latch or set of latches configured to detect which of two arriving signals is first.
    ii. If top_path_delay < bottom_path_delay, then arbiter outputs response to '1'.

iii. If bottom_path_delay < top_path_delay, then arbiter outputs response to '0'.

iv. The output 'response' is the unique data from a particular PUF device corresponding to a given challenge input.

4.5 FUNCTIONAL VERIFICATION

Functional verification of an FPGA-based APUF circuit is a critical step to ensure that the design correctly implements the intended logic and produces reliable outputs under various conditions. Since APUFs are used in hardware security applications such as device authentication and key generation, verification must check both functional correctness and robustness. Functional verification usually involves simulation, emulation, and hardware testing. A Pseudo-Random Binary Sequence (PRBS) generator, utilizing an LFSR, is used to drive the PUF challenge input in both simulation and FPGA hardware. PRBS generator in simulation environment provides a wide variety of deterministic challenge inputs to verify PUF functionality, and estimate key metrics such as uniformity, reliability, and uniqueness. On an FPGA development board, the same generator supplies stable challenge vectors to the PUF through a synchronized controller, with seed-loading and enable signals ensuring reproducibility and flexibility. The PRBS output is mapped as a challenge word, applied to the PUF, and the corresponding response outputs are collected via a logic analyzer for statistical analysis. This approach ensures systematic, scalable, and hardware-efficient testing of the PUF design.

4.5.1 Register Transfer Level (RTL)

The purpose of RTL simulation is to verify the functional correctness of the Arbiter PUF design at the register transfer level before synthesis and hardware implementation. It ensures that the multiplexer chain, challenge input configuration, and arbiter decision logic work together to produce valid one-bit responses for different challenge vectors. A testbench is created to apply a series of challenge inputs to the PUF circuit and monitor the output responses. The testbench typically includes:

- Challenge stimulus generates random or sequential challenge vectors to test a vast input space.
- Stimulus Process: Applies challenge vectors to the multiplexer chain of the PUF.
- Clock and Reset Signals: Provide timing references and initial conditions to the circuit.
- Response Monitor: Observes the arbiter output and records the one-bit responses generated.

To handle large-scale simulation efficiently, File I/O operations are incorporated into the testbench.

- Input File: Stores a list of challenge vectors, which are sequentially read and applied to the PUF circuit.
- Output File: Stores the corresponding PUF responses for each challenge. These responses are later analyzed for important security metrics such as uniqueness, reliability, and uniformity.
- The testbench flowchart begins by initializing signals and variables for the Device Under Test (DUT), followed by setting up File I/O, where a challenge input file is opened for reading and a response output file is opened for writing. The testbench then reads each challenge from the file, applies it to the PUF input, waits for the propagation delay, and records the PUF response. The resulting Challenge–Response Pairs (CRPs) are written into the output file and displayed. Once all inputs are processed, the files are closed, and a summary report is generated, including performance metrics such as response uniqueness and distribution. Finally, the simulation time is closed, marking the end of the testbench.

This flowchart of Figure 4.3 details the operation of a Testbench (TB) for an APUF. A testbench is a specific HDL program used to verify the functionality and performance of the APUF circuit or any digital circuit. The key objective of a testbench is to automatically verify the functionality of the implemented APUF design, extract a large dataset of CRPs, and analyze the basic performance metrics of the PUF. This is a critical step in both designing PUF and conducting security research like ML modeling. The stepwise explanation of the testbench development is discussed below.

- *Startup & Initialization:* All signals that will be connected to the DUT, i.e., the APUF module, are declared and assigned initial values. It includes:
  i. Setting up the clock signal to a known state (e.g., '0').
  ii. Initializing the 64-bit challenge input vector to all zeros or a known value.
  iii. Preparing the response signal to be read.
- *I/O file setup:* The testbench opens a pre-generated text file "challenge_input.txt" containing a random challenge input vector (e.g., a PRBS-generated sequence of thousands of 64-bit binary numbers). Thus, the file acts as the stimulus for the test.

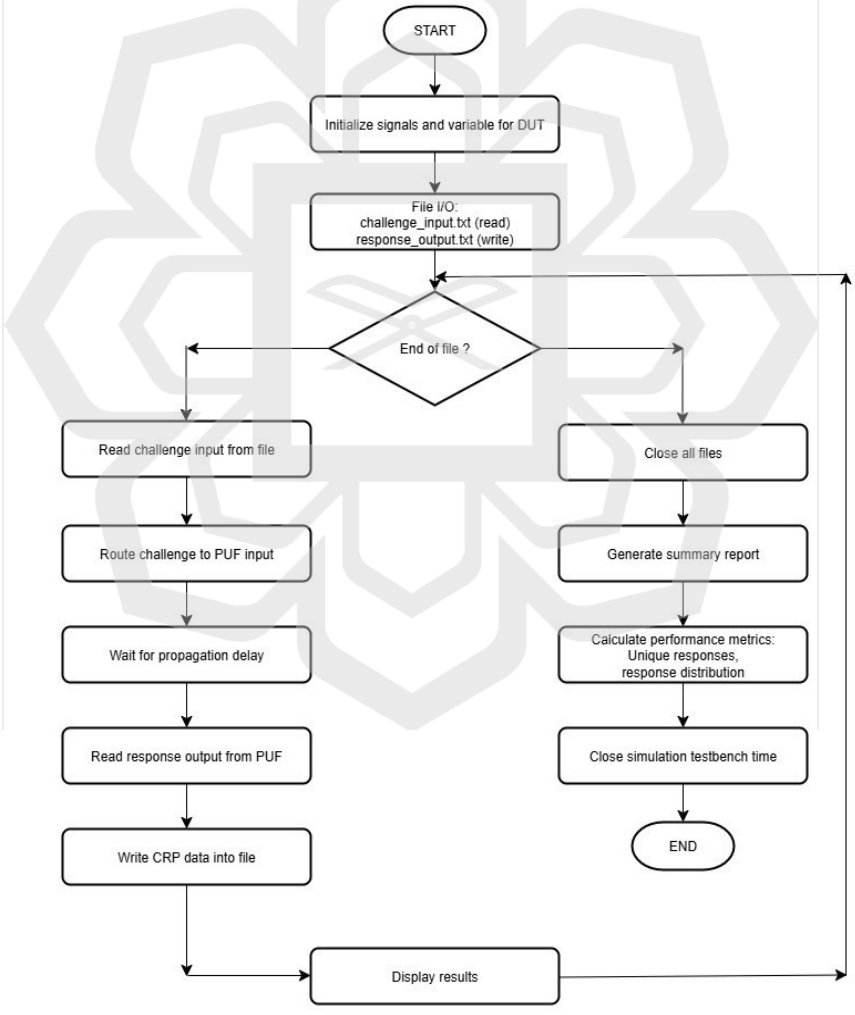


Figure 4.3 Verification and validation of DUT

The testbench opens a new file “response_output.txt” where it writes the simulated results. Each line in this file will typically contain a challenge and its corresponding response, forming a CRP dataset.

- *Simulation Loop:* The loop executes once for each challenge in the challenge_input.txt file, continuing until all challenges have been read and processed. The End-of-File (EOF) condition is used to terminate the loop. Hence,
  i. The testbench reads the next challenge input vector from the text file and stores it in an internal variable, and
  ii. It applies this challenge input vector to the input ports of the DUT (the APUF module).
- The simulation is paused for a fixed duration that exceeds the worst-case propagation delay of the APUF’s combinational logic, ensuring the response signal is fully settled before sampling. After the signal stabilizes, the testbench captures the value of the DUT’s response output ('0' or '1') and records it.
- *Write CRP Data into File:* The testbench writes the current challenge and the newly read response as a single line into the response_output.txt file.
- *Loop Completion and final tasks:* As the challenge inputs from the challenge_input.txt file get processed, the testbench exits the loop and closes the input and output text files.
- *End Simulation:* The testbench ends the simulation, and the simulator software (Intel Quartus Simulator) terminates the process.
- *Estimating the APUF Performance Metrics:* Python scripts executed in VS Code process the response_output.txt file to evaluate key characteristics of the PUF:
  i. Uniqueness: Evaluated by comparing the responses of different PUF instances, whether on the same or on separate FPGA devices, when subjected to the same set of challenges.
  ii. Randomness: The proportion of 0s and 1s in the responses is analyzed. An ideal PUF produces a 50/50 split, indicating balanced and unbiased behavior.

iii. Reliability: The same challenge inputs are applied repeatedly under controlled environmental variations, and the consistency of the resulting responses is measured.

4.5.2 Post-Synthesis Verification of FPGA-based APUF

Following the synthesis stage, the APUF circuit is verified in its synthesized form to confirm that the synthesis process didn't alter or degrade its intended functionality. While RTL simulation validates the higher abstraction level behavioural model of the design, post-synthesis verification aims to validate the gate-level netlist generated by the FPGA synthesis tool. Post-Synthesis process applies various transformations, including logic optimizations, resource mapping, and routing assignments. In the case of an APUF, the synthesis process directly impacts its delay-based operation, which is sensitive to path differences. Resource Mapping of multiplexers, arbiters, and interconnects onto FPGA primitives can change how signals propagate.

Additionally, routing constraints might introduce additional delays or asymmetries in the proposed balanced paths. The uniqueness, reliability, and randomness of APUF circuits rely heavily on the delay variations introduced during manufacturing stages, and even minor synthesis-induced modifications can alter their security characteristics. Therefore, post-synthesis verification ensures that the gate-level implementation still preserves the core functional and statistical characteristics of the APUF.

This post-synthesis verification typically involves running gate-level simulations with back-annotated delays (SDF) or using the Quartus timing simulation environment. By comparing the synthesized behaviour against the RTL reference model, the logical correctness of APUF circuits is verified, and its delay-based challenge–response mechanism continues to function as expected without compromising PUF performance metrics. Also, Post-Place-and-Route (Timing) verification after the placement and routing step captures the actual timing behaviour of the implemented design.

4.6 REAL-TIME CRP DATA EXTRACTION

The CRP extraction process is implemented on an Altera Cyclone IV E FPGA for both Arbiter PUF and Ring Oscillator PUF architectures using an integrated measurement setup. Data Extraction involves capturing the output responses of a PUF immediately as challenges are applied to an FPGA-based APUF. Input Challenges generated by a pseudo random binary sequence generator (PRBS) generator are applied to the PUF in real time, and the corresponding responses are recorded as they occur. A Finite State Machine (FSM) sequences the challenge application, asserts a strobe signal to latch the PUF response, and streams the data to a host interface. A logic analyzer plays a critical role in this process by providing high-resolution, accurate monitoring of both the challenge inputs and response outputs. For the Arbiter PUF, PRBS produce challenges, which are directly applied to the multiplexer stages defining the delay paths. After a defined stabilization interval, the arbiter generates a single bit response based on the delay difference between the two paths. The challenge and corresponding response are observed and captured through a logic analyzer using a clock synchronized trigger. This procedure is repeated continuously, generating a dataset exceeding 90,000 CRPs, with each challenge optionally evaluated multiple times to assess stability. For the Ring Oscillator PUF, oscillator pairs are selected using internally generated control inputs, and their frequencies are compared using on chip counters within a fixed time window. The resulting comparison bit forms the response, which is similarly monitored via the logic analyzer. All captured challenge response pairs are transferred to a host system for storage and further analysis. Reliability is evaluated using repeated measurements and Hamming Distance, while randomness is assessed through bias and entropy metrics, ensuring a statistically meaningful characterization under consistent operating conditions.

The stabilization interval is selected based on the metastability resolution characteristics of bistable elements, modeled as an exponential decay $P(t) = e^{-t/\tau}$. Assuming a conservative FPGA time constant of $\tau \approx 50$ ps, a delay of 100 ns – 200 ns yields $\frac{t}{\tau} > 2000$, resulting in a negligible probability of unresolved metastability. This ensures that the APUF output is fully settled before sampling. Additionally, this interval corresponds to 10 - 20 samples at the 100 Mbps logic analyzer rate, guaranteeing reliable capture of a stable response. For the Ring Oscillator PUF, longer evaluation

windows in the microsecond range are used to enable accurate frequency comparison rather than metastability mitigation. The experimental validation procedure ensures correctness and repeatability of extracted challenge response pairs through repeated evaluations of each challenge on the FPGA platform. For each challenge, multiple responses are collected and compared using Hamming Distance to quantify intra challenge consistency, thereby evaluating response stability under identical operating conditions. The logic analyzer is triggered synchronously with the FPGA system clock, ensuring deterministic capture of stable response values without transition ambiguity.

4.6.1 Hardware-in-the-Loop (HIL) verification

This is the final and most practical stage of validating an FPGA-based APUF design, as it directly tests the circuit's functionality on the physical FPGA device under real operating conditions. Unlike simulation-based verification, HIL incorporates the actual hardware platform in the verification process and uses measurement tools such as a logic analyzer to observe real-time behaviour. This step is essential because APUFs rely heavily on subtle delay variations that are only accurately manifested in silicon rather than simulations. The main goal of HIL verification is to confirm that the implemented APUF circuit on the FPGA:

- Generates consistent challenge–response pairs (CRPs) under real conditions.
- Preserves the statistical properties (uniqueness, reliability, uniformity) observed in simulation.
- The system remains robust against environmental variations such as voltage and temperature fluctuations.
- The system interfaces correctly with external monitoring equipment for response collection and analysis.

The flowchart shown in Figure 4.4 describes the physical testing and validation stage on actual hardware. The process begins by configuring an FPGA with the APUF design and connecting a Logic Analyzer's probes to the board's input/output pins to monitor the “challenge” bus and “response” bit. The analyzer gets triggered by an event occurring on “challenge” or a clock edge, initiating the capture of real digital waveforms. The testing process initiates by driving a “challenge” to the running FPGA,

waiting for the real-world propagation delay to elapse, and capturing the stabilized response output. This process is repeated for multiple challenges to ensure the physical device's behaviour is reliable, correct, and robust under real-world conditions.

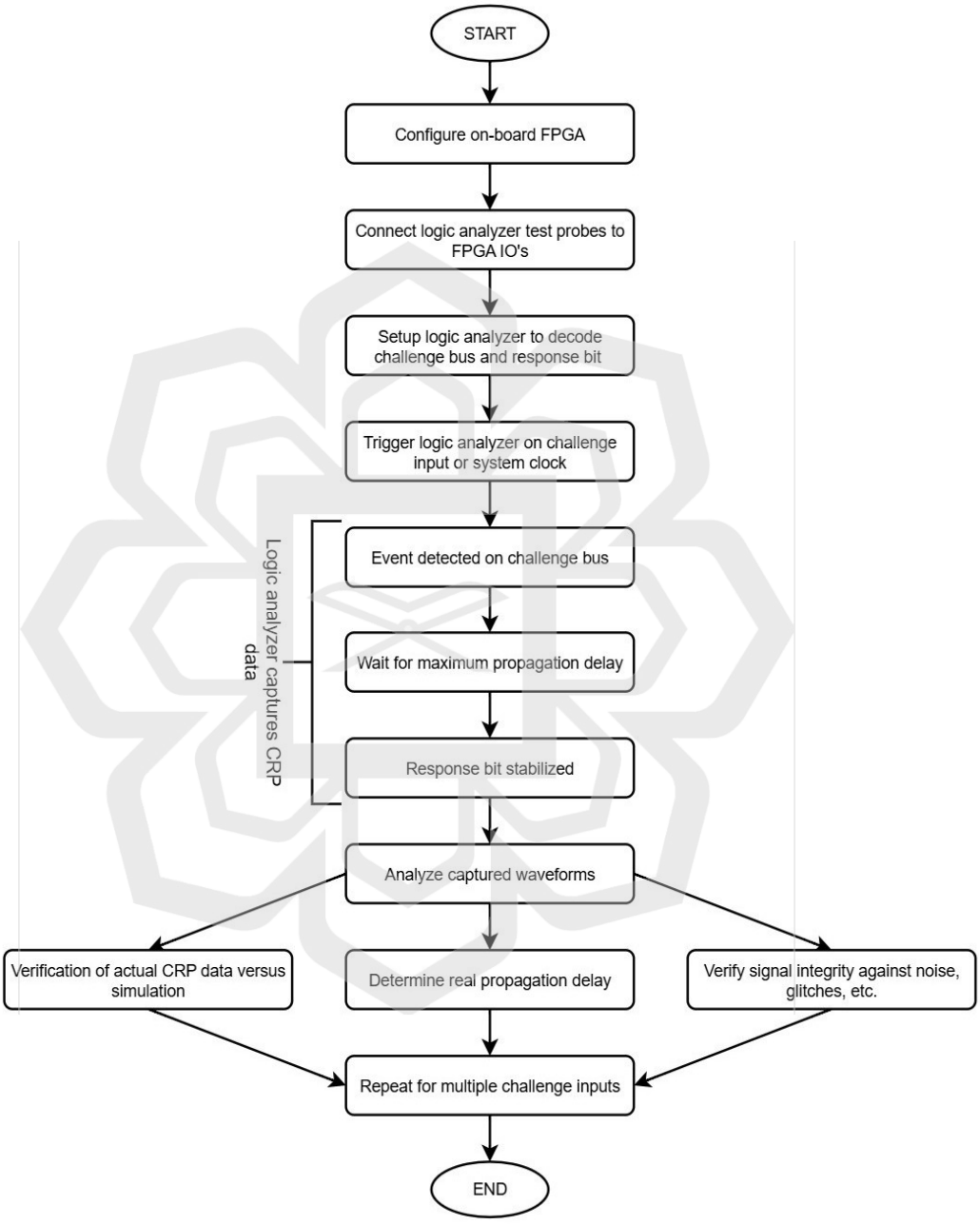


Figure 4.4 Physical PUF Validation with Logic Analyzer

The HIL verification process can be summarized in the following steps:

- *FPGA Configuration:* To configure an Intel FPGA, the hardware design project in Intel Quartus Prime is first compiled to generate an SRAM Object File (.sof). This .sof file is then loaded into the Intel Quartus Prime Programmer, which is used to download the bitstream and program the FPGA device.
- *Challenge Application:* A set of challenge vectors is generated directly on the FPGA using a PRBS generator. These challenges are then applied to the multiplexer chain of the APUF for evaluation.
- *Response Observation:* The arbiter outputs are connected to the logic analyzer probes, allowing real-time observation of the PUF responses. The logic analyzer captures the binary outputs (‘0’ or ‘1’) corresponding to each challenge input.
- *Data Logging:* The captured CRPs are transferred to the host computer for post-processing. Large-scale CRP sets are analyzed for PUF metrics.

4.6.2 Role of the Logic Analyzer

The logic analyzer serves as a crucial verification tool in this setup:

- *Real-Time Monitoring:* The arbiter ‘response’ output is captured with high timing resolution.
- *Signal Integrity Checking:* It detects glitches, metastability, or unstable responses caused by noise or routing imbalances.
- *Timing Measurement:* An estimate of propagation delays across the multiplexer chain and arbiter circuit.
- *Validation of Correctness:* Ensures that each applied challenge produces a single stable output bit without undefined states.

The use of a logic analyzer is significant because oscilloscopes often lack the necessary channel count and data storage for large-scale CRP acquisition. In contrast, logic analyzers can monitor multiple digital signals efficiently.

4.7 CALIBRATION AND ERROR REDUCTION TECHNIQUES

PUF responses are naturally affected by environmental conditions such as temperature changes, supply voltage fluctuations, and long-term device aging. These factors can introduce instability in CRPs, which lowers the overall reliability of the PUF. To address this issue and ensure stable, repeatable outputs, the proposed schemes incorporate calibration and error-reduction techniques. These techniques should be able to compensate for environmental variations, minimize noise effects, and improve the consistency of responses across different operating conditions.

4.7.1 Sources of Instability in PUF Responses

- *Temperature variations:* High operating temperatures change the switching speed of transistors, which in turn affects the delay paths in Arbiter PUFs and the oscillation frequencies in RO PUFs.
- *Voltage Variations:* Fluctuations in supply voltage affect signal propagation delays, which can lead to higher error rates in CRPs.
- *Device Aging:* Long-term wear-out effects, such as Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI), gradually degrade the electrical characteristics of the device.
- *Random Noise:* Thermal and electrical noise can introduce uncertainty in borderline cases, leading to unpredictable flipping of PUF responses.

4.7.2 Calibration Techniques

Calibration is used to counteract environmental and device-induced variations, ensuring that the final PUF response remains stable and reliable.

- Challenge Selection Filtering
  i. Challenges that produce inconsistent responses across repeated measurements are identified and discarded.
  ii. During the enrollment phase, challenges are pre-screened so that only reliable CRPs are retained for authentication.
- Voltage and Temperature Compensation
  i. On-chip sensors continuously monitor supply voltage and temperature.

ii. Compensation factors are applied dynamically to adjust timing thresholds in Arbiter PUFs or counter windows in RO PUFs, maintaining stable responses under varying conditions.

- Normalized Frequency Comparison (for RO PUFs)
  i. Rather than comparing raw oscillator frequencies, frequency ratios are used, reducing sensitivity to absolute changes in environmental conditions.
- Delay Path Balancing (for Arbiter PUFs)
  i. Strategic placement and routing constraints during FPGA implementation minimize systematic bias in delay paths.
  ii. A symmetric layout ensures that PUF behavior is governed primarily by random manufacturing variations rather than deterministic placement differences.

4.8 SUMMARY

This chapter presents the design and development process of the proposed PUF architectures aimed at enhancing security in IoT systems. The design objectives and requirements section defines the key goals, such as low power consumption, high reliability, uniqueness, and robustness against environmental variations. Clear design guidelines, post-processing strategies, and hardware implementation requirements support it. The functional verification process is described at both the Register Transfer Level (RTL) and post-synthesis stages to ensure the correctness and performance of the FPGA-based APUF implementation. The chapter also explains real-time CRP data extraction, facilitated by Hardware-in-the-Loop (HIL) verification and the use of a logic analyzer for accurate signal monitoring and data validation. Finally, it discusses calibration and error reduction techniques, identifies the primary sources of instability in PUF responses, and details the methods used to achieve stable, reliable, and repeatable outputs. Overall, the chapter systematically details the design, verification, and refinement stages of the proposed PUF architectures to ensure their practical viability and resilience in real-world IoT environments.

CHAPTER FIVE

RESULTS AND PERFORMANCE EVALUATION

5.1 INTRODUCTION

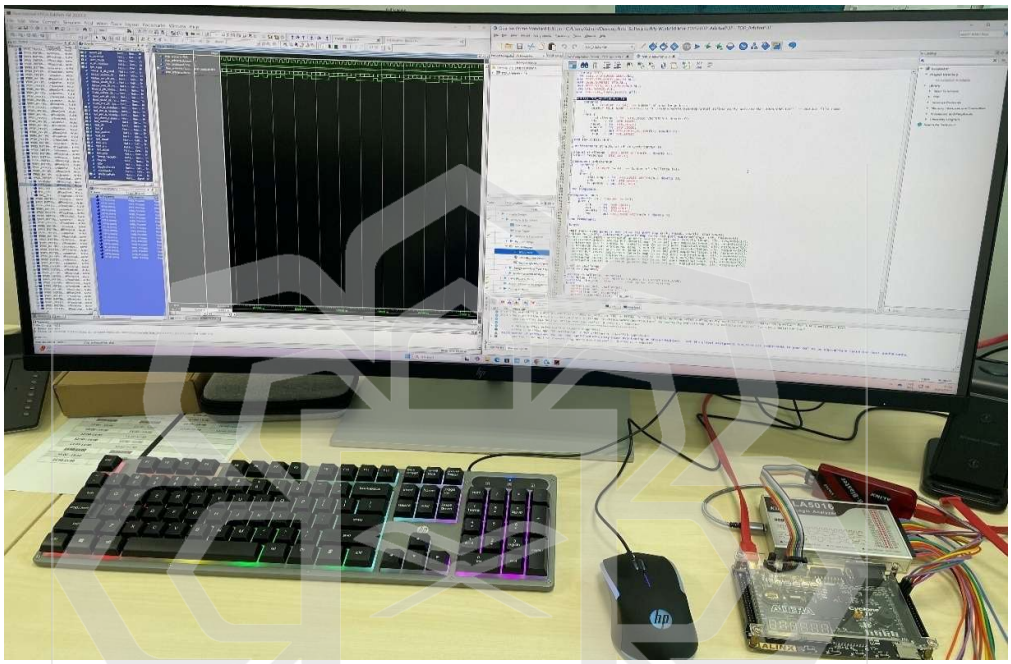
This chapter presents the experimental validation of the proposed FPGA-based PUF authentication system. The performance of the PUF is evaluated using key metrics such as uniqueness, reliability, randomness, and uniformity, further verified through NIST statistical tests. The chapter also examines the robustness of the design against machine learning attacks by comparing prediction accuracies obtained from raw and pre-processed CRPs. The results are benchmarked against existing approaches, emphasizing the trade-offs between complexity, reliability, and security. Overall, the findings demonstrate the effectiveness of the proposed scheme as a lightweight and resilient solution for IoT applications.

5.2 RESULTS OF IMPLEMENTED PUFS ON FPGA

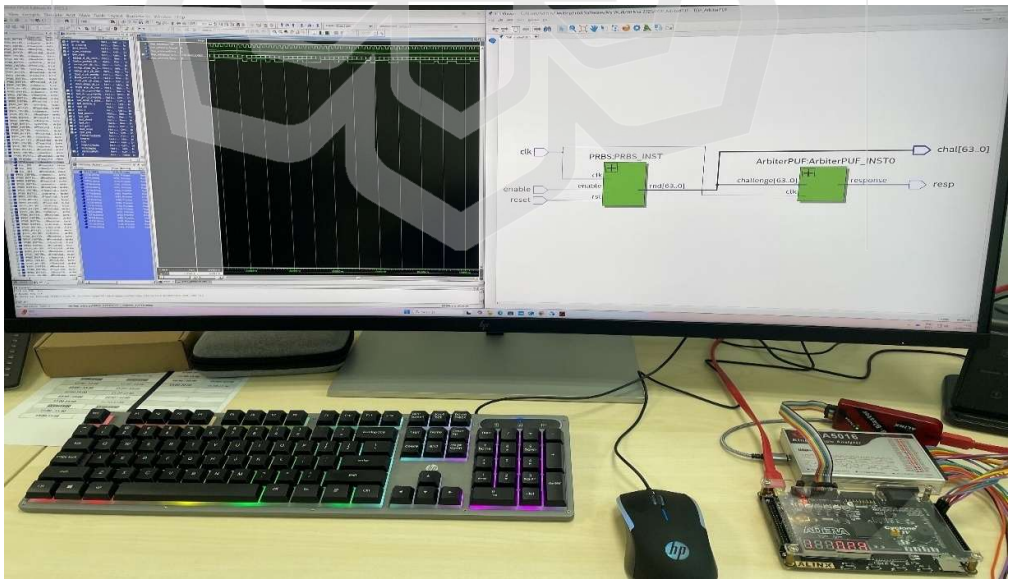
A 64-bit APUF is implemented on the ALINX AX4010 FPGA, where a Pseudo-Random Binary Sequence (PRBS) generator supplies the challenge inputs, and the resulting CRPs are captured using a Kingst logic analyzer. The FPGA used in this study is an Altera Cyclone IV EP4CE10F17C8 device, developed by Intel (Santa Clara, CA, USA). This device is fabricated using 60 nm CMOS technology, operates at a core voltage of 1.2 V, and comes in a 256-pin Fine-Pitch Ball Grid Array (FBGA) package. The choice of Cyclone IV FPGA is motivated by its low cost, energy efficiency, and suitability for lightweight IoT applications where power and area are critical constraints. The experimental setup for validating the APUF design on an FPGA is shown in Figure 5.1. The setup operates on combined hardware and software components for CRP generation and analysis of PUF performance.

The FPGA is powered and programmed via a USB–Blaster interface, along with an external logic analyzer probe to capture the responses. The logic analyzer enables real-time monitoring of the signals, ensuring accurate CRP acquisition for further

analysis. The design is developed and simulated using Intel Quartus Prime Standard Edition (v18.1), with waveform observation and debugging performed in the ModelSim-Altera simulation environment. The PRBS generator supplies 64-bit challenge vectors to the APUF module. For each challenge input, the arbiter evaluates the relative delay paths and outputs a corresponding single-bit response.



(a)



(b)

Figure 5.1 Experimental setup (a) RTl Code (b) Netlist

The captured CRP responses are recorded for subsequent evaluation of key metrics such as uniqueness, reliability, uniformity, and randomness. The collected CRPs are further processed for statistical analysis and robustness assessment, which includes evaluation against machine learning-based prediction attacks. The overall setup integrates FPGA hardware, ModelSim waveform simulation, the Quartus design environment, and external measurement tools, together forming a comprehensive validation framework to verify the feasibility of FPGA-based APUF implementation in IoT authentication systems.

5.2.1 RTL Simulation

The RTL simulation results of both the RO-PUF and APUF designs, as illustrated in Figures 5.2 and 5.3, confirm that each architecture functions correctly at the logic level. In the RO-PUF waveform, the clock, enable, reset, challenge, and response signals operate as intended. When the enable signal activates the measurement window, the oscillators generate pulses that are accurately counted by the counters. Once counting stops, the comparator compares the two counts and outputs a stable response bit. This demonstrates that the challenge selection logic, counter increments, and comparator operations are functioning correctly. Similarly, the APUF waveform shows proper operation of the two propagation paths, path_A and path_B, as signals race through the delay network. The arbiter correctly resolves which path's signal arrives first, producing a deterministic response bit. Functionally, these RTL simulations verify that both PUFs are correctly implemented in terms of digital logic. They confirm that the PUFs accept challenges, perform internal operations correctly, and generate stable responses for each evaluation cycle. However, RTL simulations are purely digital and do not capture the physical properties that make PUFs unique and unpredictable in hardware. Real-world randomness in PUFs arises from manufacturing process variations, timing mismatches, and analog effects that are not modeled at the RTL level. Therefore, while functional verification ensures that the design logic works as intended, it does not validate the PUF's physical randomness, uniqueness, or reliability.

To achieve full verification, further post-layout and variation-aware simulations are necessary. Overall, the RTL results demonstrate that both the RO-PUF and APUF designs are functionally correct and logically verified. Still, comprehensive physical-

level simulations and hardware testing are required to confirm their robustness, unpredictability, and suitability for secure applications.

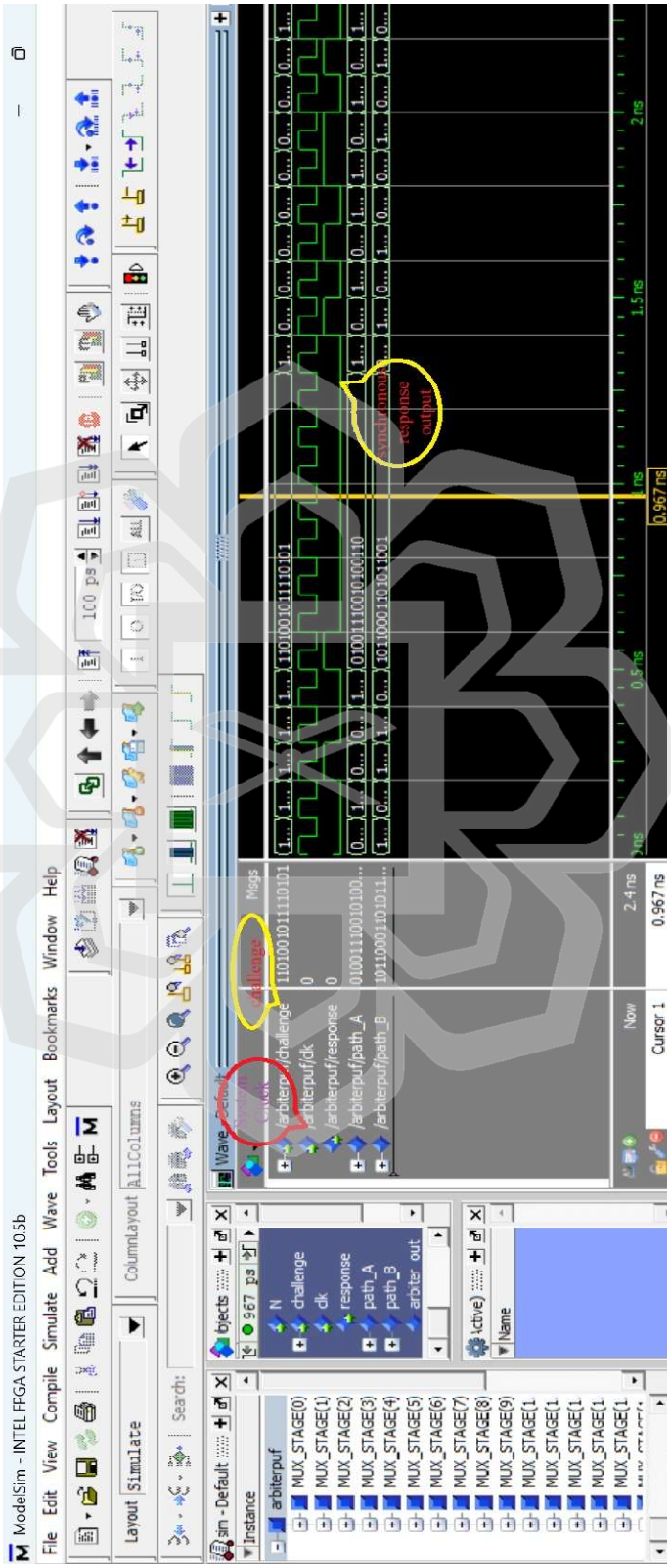


Figure 5.2 ModelSim simulation of APUF

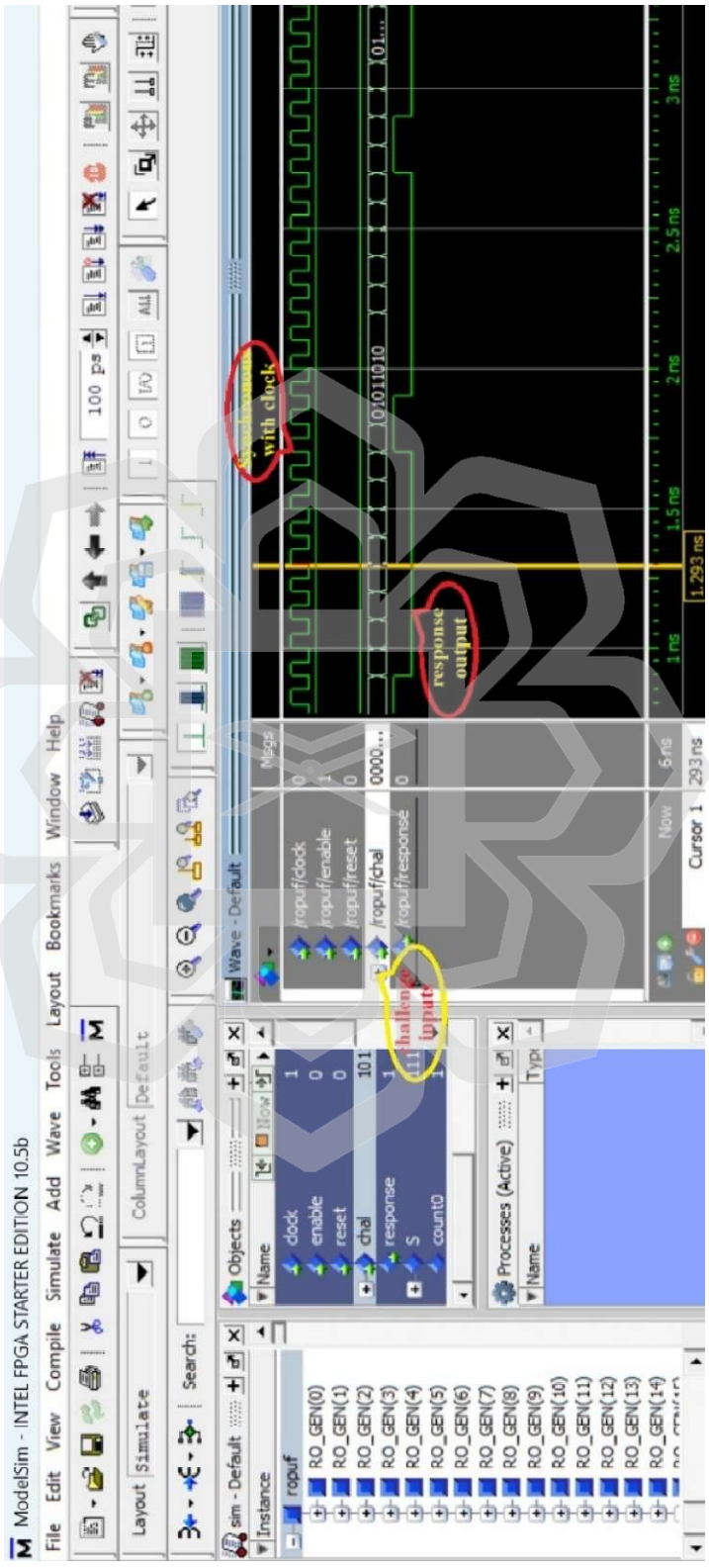


Figure 5.3 ModelSim Simulation of RO-PUF

5.2.2 Implementation Constraints

A 64-bit APUF and a 64-stage RO-PUF were implemented on an Altera Cyclone IV E FPGA to evaluate and compare their suitability for hardware security in IoT applications. In the APUF architecture, each 64-bit challenge vector produces a single-bit response, forming a CRP. The FPGA was configured as an IoT node with an embedded PUF-based authentication engine that uses these CRPs as device-specific fingerprints. Implementing PUFs on the Cyclone IV requires balancing the need to expose device-level physical variability (for uniqueness) with the need for stable, repeatable responses (for reliability). FPGA CAD optimizations can reduce or mask physical variability by balancing routes and packing logic. Therefore, critical elements, including APUF delay paths and RO stages, must be floorplanned and placed adjacent to preserve intended asymmetries or identical oscillator structures. The APUF depends on metastability and precise sampling of the arbiter output; therefore, the design must include metastability-tolerant arbiters and carefully controlled sampling clocks. The RO-PUF, by contrast, requires high-resolution frequency measurement and careful isolation from routing coupling and supply noise to minimize jitter and frequency skew.

While static leakage is dominated by the device, dynamic switching and I/O activity significantly affect power consumption and thermal stability; these effects must be managed through decoupling, careful I/O buffering, and power-aware placement. Finally, CRP exposure must be limited and protected through measures such as rate limiting, controlled-PUF architectures, XORing, and secure helper-data management to reduce vulnerability to ML and side-channel attacks.

5.3 ROUTING STRATEGY

A fundamental design choice in this work is to rely on the Quartus toolchain's default automated placement and routing. While other PUF implementations often use strict manual constraints or complicated macros, this work intentionally allows the tool to distribute the APUF components across the FPGA freely. This approach is critical because it preserves the natural, microscopic variations inherent in the silicon fabric. These inherent variations are the essential source of randomness that makes a PUF unclonable. By avoiding artificial constraints, we ensure that the CRPs generated are a

direct reflection of this true physical randomness, making them highly unpredictable and unique. This strategy also significantly enhances security against machine learning–based modelling attacks, as it prevents the introduction of accidental patterns, symmetry, or bias that can arise from manual layout.

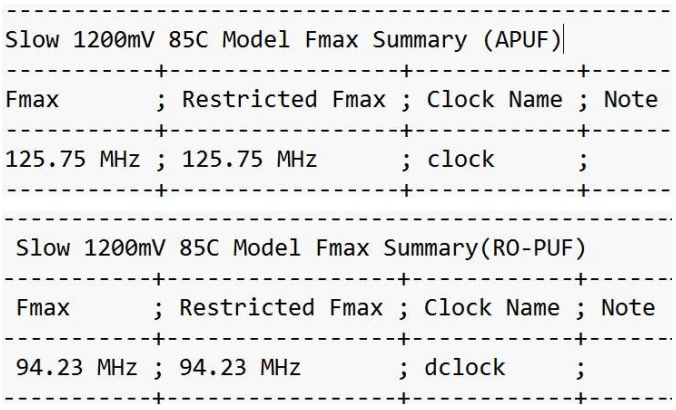
Routing Usage Summary (APUF)	
Routing Resource Type	Usage
Block interconnects	103 / 32,401 (< 1 %)
C16 interconnects	3 / 1,326 (< 1 %)
C4 interconnects	87 / 21,816 (< 1 %)
Direct links	15 / 32,401 (< 1 %)
Global clocks	2 / 10 (20 %)
Local interconnects	64 / 10,320 (< 1 %)
R24 interconnects	7 / 1,289 (< 1 %)
R4 interconnects	97 / 28,186 (< 1 %)

Routing Usage Summary (RO-PUF)	
Routing Resource Type	Usage
Block interconnects	615 / 32,401 (2 %)
C16 interconnects	2 / 1,326 (< 1 %)
C4 interconnects	227 / 21,816 (1 %)
Direct links	125 / 32,401 (< 1 %)
Global clocks	2 / 10 (20 %)
Local interconnects	925 / 10,320 (9 %)
R24 interconnects	6 / 1,289 (< 1 %)
R4 interconnects	284 / 28,186 (1 %)

Figure 5.4 Routing summary results

The routing resource analysis shown in Figure 5.4 illustrates a summary of the implementation of APUF and RO-PUF on the target FPGA device. The RO-PUF design utilizes higher routing resources and interconnect types, particularly in block, local, and short-range (R4) interconnects. This validates a denser and more complex routing network, consistent with its oscillator-based architecture. In contrast, the APUF exhibits lightweight routing usage due to its simpler, linear delay path structure, resulting in lower area utilization, reduced routing delay, and better scalability. Both designs use the same number of global clocks, showing similar clocking requirements. From a design perspective, RO-PUF's dense routing enhances entropy and randomness, making it suitable for security-critical applications. In contrast, APUF's simpler routing ensures speed, power efficiency, and ease of implementation. Overall, RO-PUF offers higher randomness at the cost of increased routing complexity and power consumption, while APUF remains more resource-efficient and scalable.

The maximum frequency (Fmax) analysis shows that the APUF achieves a higher operating frequency of 125.75 MHz, while the RO-PUF operates at 94.23 MHz, as reported in Figure 5.5. This indicates that APUF can function at a faster clock rate, offering better timing performance and quicker response compared to RO-PUF, which trades off speed for potentially lower power consumption and design stability.



Slow 1200mV 85C Model Fmax Summary (APUF)

Fmax	Restricted Fmax	Clock Name	Note
125.75 MHz	125.75 MHz	clock	

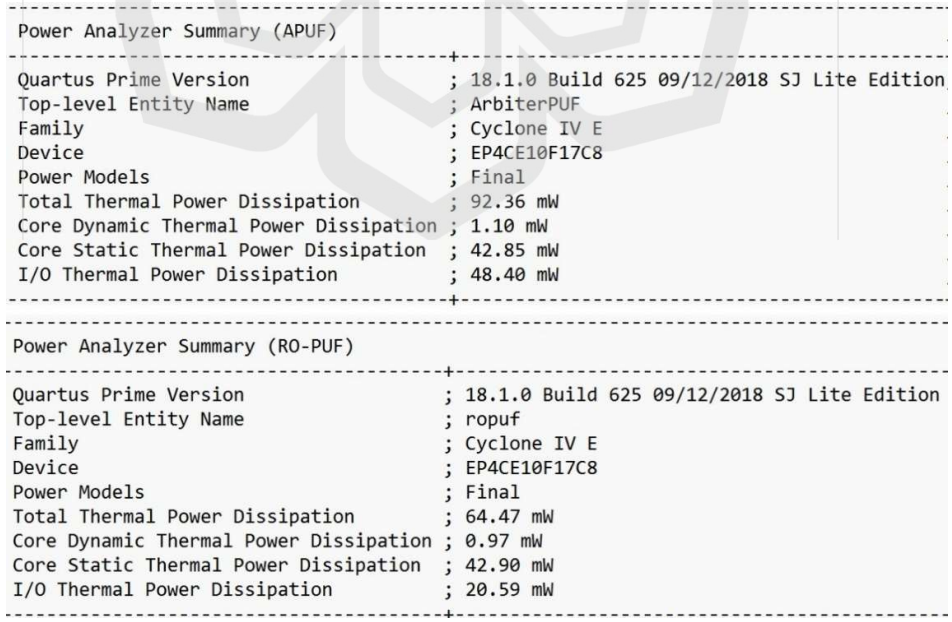
Slow 1200mV 85C Model Fmax Summary(RO-PUF)

Fmax	Restricted Fmax	Clock Name	Note
94.23 MHz	94.23 MHz	dclock	

Figure 5.5 Maximum achievable clocking frequency

5.4 ESTIMATED POWER CONSUMPTION ANALYSIS

The power analysis comparison between the APUF and RO-PUF implemented on the Cyclone IV E FPGA highlights notable differences in their power behavior. As illustrated in Figure 5.6, the total thermal power dissipation of APUF is 92.36 mW, which is significantly higher than RO-PUF's 64.47 mW, indicating that APUF consumes more power overall.



Power Analyzer Summary (APUF)

Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Top-level Entity Name	ArbiterPUF
Family	Cyclone IV E
Device	EP4CE10F17C8
Power Models	Final
Total Thermal Power Dissipation	92.36 mW
Core Dynamic Thermal Power Dissipation	1.10 mW
Core Static Thermal Power Dissipation	42.85 mW
I/O Thermal Power Dissipation	48.40 mW

Power Analyzer Summary (RO-PUF)

Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Top-level Entity Name	ropuf
Family	Cyclone IV E
Device	EP4CE10F17C8
Power Models	Final
Total Thermal Power Dissipation	64.47 mW
Core Dynamic Thermal Power Dissipation	0.97 mW
Core Static Thermal Power Dissipation	42.90 mW
I/O Thermal Power Dissipation	20.59 mW

Figure 5.6 PUF estimated static and dynamic power consumption analysis

This increased power usage is primarily due to the higher I/O thermal power dissipation of 48.40 mW in APUF compared to 20.59 mW in RO-PUF, suggesting that APUF's design involves more intensive I/O activity or higher signal transitions. Both architectures exhibit nearly identical core static power values ranging between 42.8–42.9 mW, implying that their static leakage characteristics are similar and mainly dependent on the FPGA fabric rather than the design architecture. However, APUF has a slightly higher core dynamic power of 1.10 mW compared to 0.97 mW in RO-PUF, indicating marginally greater internal switching activity. In summary, APUF demonstrates higher power consumption due to its greater I/O and dynamic power usage, which aligns with its faster operational performance. Conversely, RO-PUF is more power-efficient, making it more suitable for low-power or resource-constrained IoT applications where energy conservation is critical.

5.5 CRP DATA ANALYSIS RESULTS

The analysis of CRP data is crucial for evaluating the performance and security of PUFs. A dataset comprising more than 900,000 Challenge Response Pairs (CRPs) was collected through hardware based acquisition using a logic analyzer, enabling comprehensive characterization of the implemented PUF. Challenge generation was realized using an LFSR based Pseudo Random Binary Sequence (PRBS) generator implemented on the Cyclone IV FPGA, where generated challenges were directly applied to the on chip PUF instances. The logic analyzer sampling rate was appropriately configured to reliably capture CRPs from the FPGA operating at a system clock frequency of 50 MHz, thereby ensuring temporal synchronization and data integrity during acquisition. To evaluate repeatability and reliability, identical challenge patterns were repeatedly applied under identical operating conditions, and response consistency across repeated measurements was systematically analyzed to assess the stability, reproducibility, and reliability of the PUF responses. For machine learning based security evaluation, the collected CRP dataset was partitioned using an 80:20 ratio for training and testing, respectively. The training subset was utilized to develop and optimize the predictive capability of multiple machine learning models, including Logistic Regression (LR), Random Forest (RF), Gradient Boosting (GB), and Multi Layer Perceptron (MLP), while the testing subset was employed to evaluate model generalization performance and assess the susceptibility of the PUF architecture to

modeling attacks. Through CRP analysis, key characteristics such as uniqueness, reliability, uniformity, and randomness can be quantified, which collectively define the strength of a PUF circuit. Moreover, statistical and machine learning-based evaluations of CRPs help assess the robustness of PUFs against adversarial attacks and provide valuable insights for improving their architecture, implementation, and post-processing techniques. Overall, CRP data analysis is essential for validating the feasibility of PUFs as lightweight, hardware-based security primitives in IoT and other embedded systems. The subsequent sections present and discuss the results of various CRP data analyses.

5.5.1 Analysis of APUF CRP

The matrix in Figure 5.7 illustrates the pairwise Pearson correlation coefficients between the 15 challenge bit channels (ch0…ch14) and the final response (R). The diagonal entries represent each variable's correlation with itself, typically having a value of 1.00. In contrast, the off-diagonal entries quantify the linear dependence between pairs of bits or between a bit and the response. This correlation matrix serves as a measure of the linear relationships between challenge bits and the PUF response. The confusion matrix analysis was limited to 15 channels of the 64 bit PUF circuit due to hardware constraints associated with the logic analyzer used for experimental validation. Specifically, the available logic analyzer supports a maximum of 16 input channels, of which one channel was reserved for clock or control signal synchronization, leaving 15 channels available for simultaneous PUF response acquisition. Despite this limitation, the selected 15 channels provide a representative subset for evaluating inter channel behavior, response distinguishability, and classification performance. Since the PUF architecture exhibits consistent operational characteristics across all response bits, the obtained confusion matrix remains sufficient to demonstrate the effectiveness and reliability of the proposed design without compromising the validity of the overall analysis. Values near 0.00 (dark blue) indicate little or no linear correlation, which is desirable for ensuring unpredictability and independence. In contrast, values approaching +1.0 or −1.0 represent strong positive or negative linear dependence, which reduces unpredictability. Small positive values in the range of 0.01 to 0.05 reflect negligible linear relationships. However, the notable correlation of 0.22 observed between ch7 and the response indicates a moderate dependence, suggesting potential bias or information leakage. Such correlations, even

when modest, warrant further investigation as they can be exploited by attackers or reduce the randomness of responses, particularly when large CRP datasets are available.

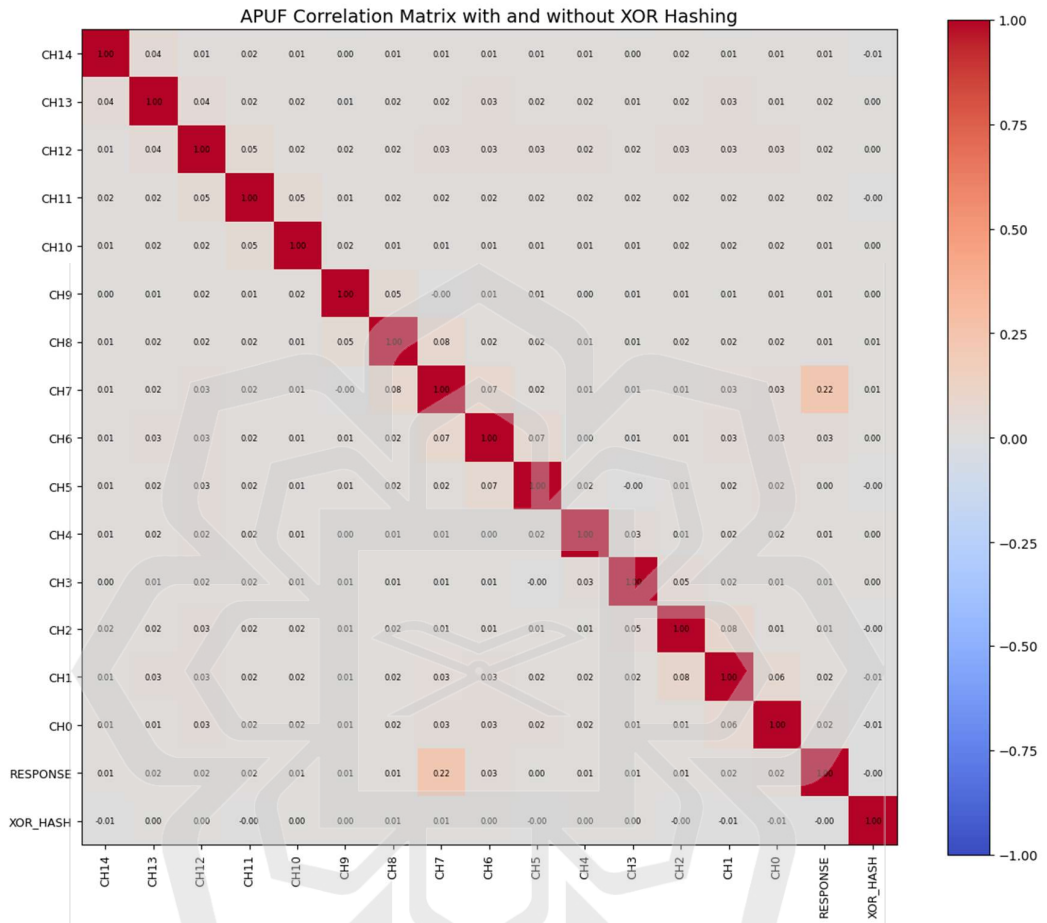


Figure 5.7 APUF Correlation matrix

The correlation matrix reveals several favourable and one concerning property of the PUF. A moderate correlation observed for bit 7 (≈0.22) indicates that this stage exerts a more substantial systematic influence on the final arbiter outcome than the others. Although 0.22 is still a moderate value but indicates the presence of a measurable bias or structural sensitivity at the corresponding stage, which may stem from systematic delay asymmetries, imbalance in the CRP dataset, or implementation related artifacts inherent to the APUF architecture. The correlation coefficients associated with the XOR-hashed output, both with respect to individual challenge bits and the raw response, are effectively zero. This indicates that the XOR post-processing stage has suppressed linear dependencies between the input challenges and the output response.

Notably, the previously observed correlation between CH7 and the raw response is no longer discernible after hashing. Consequently, the XOR-hashed output exhibits characteristics consistent with a decorrelated, statistically balanced bitstream, thereby enhancing its suitability for security-oriented applications. The potential implications of the APUF circuit design from a security and randomness perspective can be summarized as:

- The implemented APUF circuit has generated responses that are largely uncorrelated with the corresponding challenge bits, indicating unpredictability and independence.
- The correlation value of 0.22 signifies that while the PUF generally exhibits low interdependence, there exists a measurable relationship between one challenge bit and the response, which can slightly weaken the PUF's resilience to modeling and prediction attacks. Maintaining all correlation values close to zero is essential for ensuring strong unpredictability, uniqueness, and resistance to security threats (Chatterjee, Hazra, and Mukhopadhyay, 2018).
- Because Pearson correlation measures *linear* relationships only, low correlations do not rule out nonlinear dependencies.
- With XOR hashing, the correlations are significantly suppressed across all challenge bits, with coefficients approaching zero including CH7. This demonstrates that XOR hashing effectively decorrelates the input challenge space from the output response, masking linear relationships and distributing the influence of individual stages more uniformly.

5.5.2 APUF intra-hamming distance (HD)

The histogram of intra-PUF Hamming distances (HD) in Fig. 5.8, illustrates the stability of the APUF when the same challenge is repeatedly applied under varying conditions such as temperature, voltage, and noise. Figure 5.8 shows the distribution of intra-PUF HD, where the x-axis represents the percentage of differing bits between repeated responses to the same challenge, and the y-axis indicates the number of challenge groups observed at each level. An HD of 0% corresponds to perfectly stable and reproducible responses, while higher HD values signify increasing instability in the

PUF outputs. Taller bars on the plot indicate that more challenge groups exhibit that level of stability or instability.

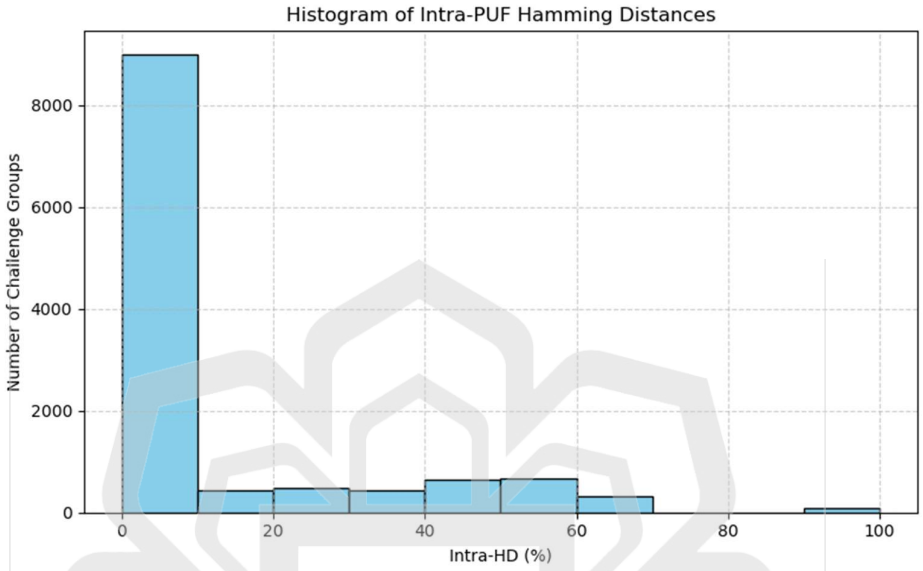


Figure 5.8 APUF Intra-HD

The results demonstrate that most challenge groups fall within a very low intra-HD range (0–5%), confirming high stability and reliable PUF responses. A smaller portion of challenge groups exhibit moderate intra-HD values (10–60%), reflecting instability and noticeable variation in repeated outputs, which are generally unsuitable for secure applications. Although a 100% HD indicates completely unreliable responses, this case occurs only rarely. Overall, the histogram confirms that the proposed APUF design provides good reliability, as most of the challenge–response pairs show very low intra-HD, ensuring consistent and stable outputs. However, the presence of a small fraction of unstable CRPs highlights the influence of environmental variations or inherent circuit noise. These cases can be managed by discarding volatile responses or employing error-correcting codes to maintain robustness. In practice, stable CRPs with near 0% HD are best suited for secure key generation, while less stable ones can be filtered or corrected to preserve overall system security and dependability. Thus, it can be concluded that the implemented APUF exhibits strong stability and reliability, with most responses reproducible and only limited cases requiring post-processing measures.

5.5.3 Entropy, Randomness & Reliability Results of APUF CRP

The APUF CRP analysis results shown in Figure 5.9 demonstrate excellent randomness and entropy but low reliability. The randomness stays close to 0.5, indicating balanced output bits, while entropy remains near 1.0, confirming high unpredictability. However, reliability fluctuates around 0.5, with a drop at higher CRPs, revealing instability in repeated responses due to noise and delay variations. Overall, the APUF provides strong uniqueness and randomness but needs improvement in reliability for consistent performance.

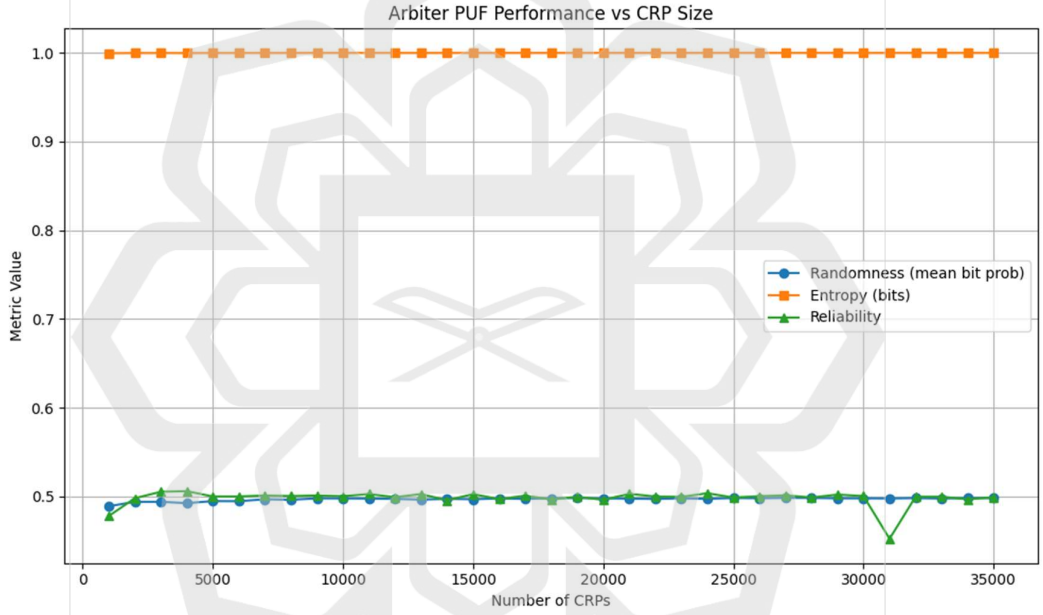


Figure 5.9 Analysis of APUF CRP

5.5.4 ANALYSIS OF RO-PUF CRP

The quality of CRP data from RO-PUFs is commonly evaluated using statistical measures such as bitwise probability (P) and entropy (H), which serve as indicators of the randomness, reliability, and security of the generated responses. Bitwise probability (P) measures the likelihood of obtaining a '1' or '0' in each response bit, with the ideal value of $P \approx 0.5$ ensuring balanced and unbiased outputs. Entropy (H) quantifies the uncertainty or randomness of the responses, reaching its maximum value of 1 bit when

P(0) = P(1) = 0.5, indicating highly unpredictable behaviour. Conversely, an entropy value of 0 signifies fully deterministic outputs.

The bar chart in Figure 5.10 shows the variation of bitwise probability (P) and entropy (H) with different CRP sizes. The results from the RO PUF CRP data indicate that the bitwise probability (P) ranges between 0.40 and 0.51, with the ideal value being 0.5 for perfectly balanced randomness. At 25,000 CRPs, P is 0.5123, which is very close to the perfect case. However, as the number of CRPs increases, P gradually declines to around 0.40–0.43, suggesting a slight bias in the responses at higher CRP counts, where one bit value occurs more frequently than the other.

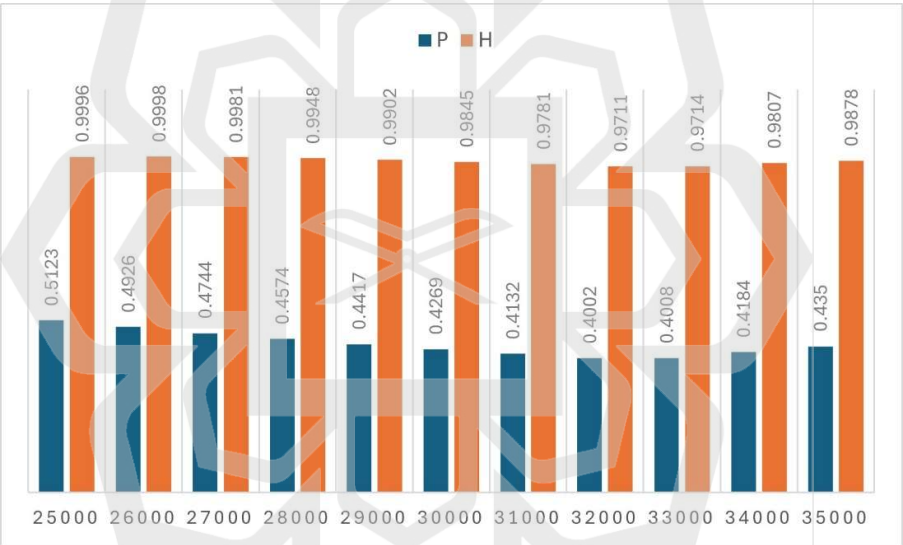


Figure 5.10 Bitwise probability and entropy of RO CRP

In contrast, entropy (H) remains consistently high, ranging from 0.97 to 0.999, which is very close to the maximum value of 1. This shows that the PUF responses maintain substantial unpredictability and randomness across different CRP sizes. Although the bitwise probability deviates slightly from the ideal 0.5, the entropy values confirm that the overall randomness of the response distribution is largely preserved. From this analysis, the following inferences can be drawn:

- The implemented RO PUF exhibits good randomness quality, with bitwise probability close to the ideal 0.5, especially at lower CRPs.

- At higher CRP counts, there is a slight bias in bit distribution, but entropy values indicate that unpredictability remains excellent.
- Hence, it is verified that the RO PUF is suitable for cryptographic applications, though post-processing techniques, which will further improve the balance.

The correlation matrix of the RO PUF CRP data in Figure 5.11 illustrates the degree of dependency between different output bits generated by the RO PUF. Each cell in the matrix represents a correlation coefficient between two response bits, ranging from –1 to +1. The diagonal elements show perfect correlation (value of 1), indicating that each bit is fully correlated with itself. The colors transition from red along the diagonal to blue in the off-diagonal regions, reflecting varying levels of inter-bit relationships.

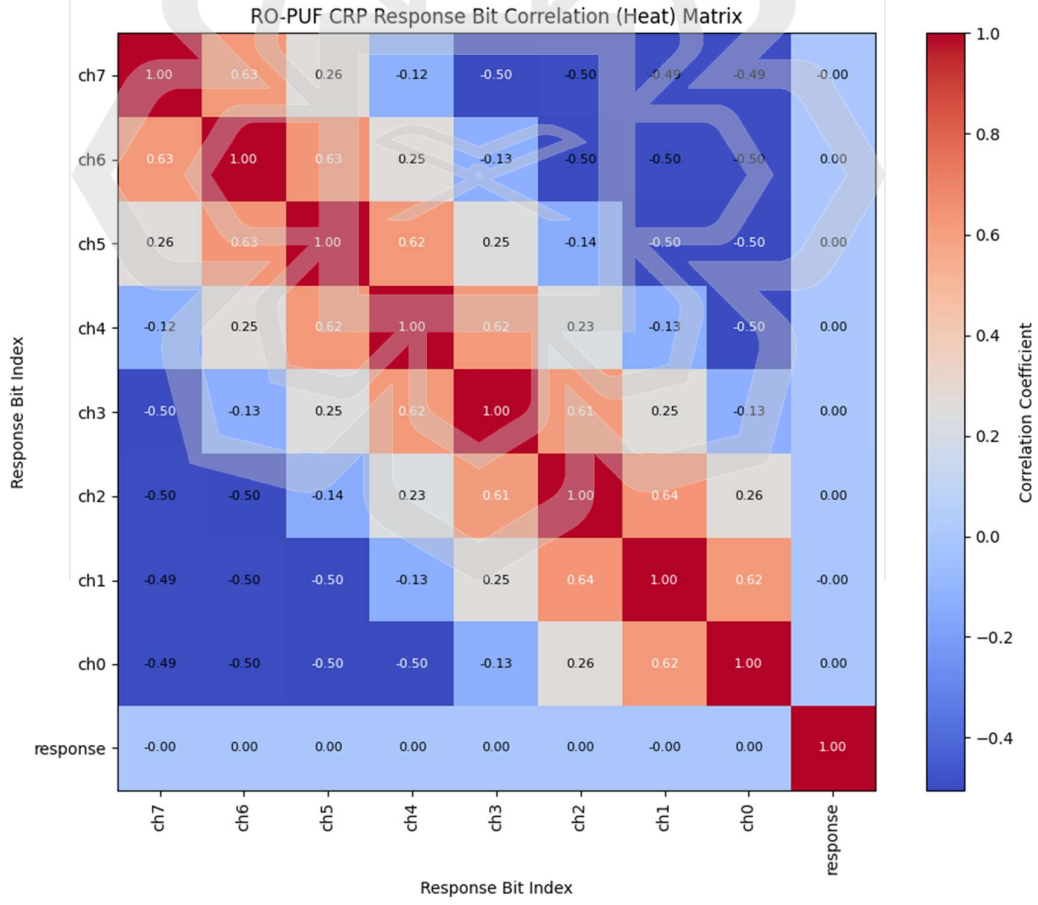


Figure 5.11 RO PUF Correlation Matrix

From the heatmap, it can be observed that adjacent bits exhibit moderate correlation, while distant bits tend to be weakly correlated or nearly independent. This pattern suggests that neighboring ring oscillators may share certain environmental or structural influences, such as local routing, power distribution, or temperature variations, which introduce partial dependency among nearby bits. However, as the distance between bits increases, the influence diminishes, and their correlation approaches zero. The overall low inter-bit correlation indicates that the RO PUF produces outputs that are largely independent and random, fulfilling one of the key requirements of a strong PUF design. The slight correlation observed between adjacent bits is expected in practical implementations due to physical proximity effects. To further enhance independence, post-processing techniques such as XOR-based bit mixing or improved oscillator placement strategies can be employed. In summary, the correlation matrix demonstrates that the RO PUF achieves good randomness and uniqueness characteristics, making it suitable for secure identification and authentication in hardware security applications.

5.5.5 COMPARING APUF VERSUS RO-PUF

The comparative analysis of the APUF and RO-PUF designs, based on their CRP evaluations, reveals distinctive strengths in terms of randomness, reliability, and uniqueness, as presented in Table 5.1. The RO-PUF demonstrates highly balanced output characteristics, with a uniformity of 49.28% and a Shannon entropy of 0.9999 bits, both of which are extremely close to ideal values. This indicates that the RO-PUF generates nearly unbiased and unpredictable responses, an essential property for ensuring strong cryptographic security. In contrast, the APUF shows a slightly biased response distribution, with a uniformity of 56.86% and an entropy of 0.9864 bits, suggesting a minor deviation from perfect randomness. In terms of reliability, both PUF types achieve excellent stability across repeated measurements, with the APUF slightly outperforming the RO-PUF (98.01% vs. 97.99%). This suggests that APUF responses remain more consistent even under environmental variations such as temperature or voltage changes. Both designs also achieve ideal uniqueness (49.99%), meaning that each PUF instance produces distinct responses, ensuring strong chip-to-chip differentiation, an important aspect for secure device authentication and identification. Practically, these results imply that the RO-PUF is more suitable for applications requiring high entropy and unpredictable key generation, such as secure encryption key

derivation or random number generation.The APUF, with its marginally higher reliability, may be preferable in scenarios demanding stable and repeatable responses, such as device fingerprinting or secure access control under varying operating conditions.

Table 5.1 PUF metrices of APUF and RO-PUF

Metric Type	Parameter	APUF	RO-PUF	Ideal Value
Total CRPs Loaded		**999,405**	**985,622**	–
Randomness Metrics	Uniformity (fraction of 1s)	49.28%	56.86%	~50%
	Mean Bit Value	0.493	0.569	–
	Shannon Entropy	0.9999 bits	0.9864 bits	~1.0
Reliability Metrics	Intra-chip Hamming Distance	20,039 bits out of 999,405	19,630 bits out of 985,622	–
	Reliability	97.99%	98.01%	≥ 95%
Uniqueness Metrics	Average Inter-chip Hamming Distance	49.99%	49.99%	~50%

Overall, both PUF architectures demonstrate excellent performance, and the choice between them would depend on whether the target system prioritizes maximum randomness (favoring RO-PUF) or operational stability (favoring APUF).

5.6 ML MODELLING ATTACKS RESULTS

Machine learning (ML) modelling of PUF CRPs provides a powerful approach for anomaly detection, which is essential for ensuring hardware integrity. Accurate ML models of the PUF function (e.g., Logistic Regression) establish a statistical baseline of authentic device behaviour. During the enrolment stage, a trusted set of CRPs from the

genuine PUF instance is used to train a model that captures the unique fingerprint of the device. This baseline enables continuous monitoring, where new CRPs are compared against the model's predicted responses. Any significant deviation from the expected behaviour is flagged as an anomaly. Such deviations may result from counterfeit or cloned devices, which cannot replicate the original physical randomness, or from hardware Trojans that deliberately alter responses. They may also arise naturally due to device aging and environmental stress, which cause gradual drifts in response stability.

By defining a tolerance threshold that distinguishes normal noise from abnormal variations, the ML-based anomaly detection framework strengthens device authenticity, integrity, and long-term reliability. Both supervised and unsupervised models can be employed to address different types of anomalies, while evaluation tools such as the confusion matrix and related metrics (accuracy, precision, recall, and F1-score) quantify effectiveness. High recall is crucial for ensuring that all anomalies are detected, thereby maintaining the robustness and reliability of the APUF system under real-world adversarial and environmental challenges. APUFs are vulnerable to LR based attacks because of their additive linear delay model, whereas RO PUFs exihibit nonlinear frequency variations, making RF, GB, and MLP useful for evaluating deeper hidden patterns. Thus, multiple ML models provides a comprehensive security evaluation as, LR models can test basic predictability, RF, GB based ensemble methods test structured nonlinear dependencies, and MLP test advanced learning capability.

Therefore, evaluating PUFs against LR, RF, GB, and MLP attacks helps to measure the robustness, unpredictability, and practical security of the PUF design against increasingly sophisticated attackers.

5.6.1 APUF analysis using a confusion matrix

A confusion matrix is a performance evaluation tool in ML that summarizes a classification model's outcomes using four categories: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). It provides insight into the accuracy and effectiveness of the model by showing how well predictions align with actual results. From the confusion matrices shown in Figure 5.12 and derived metrics LR and GB in Table 5.2, perform poorly, achieving only about 61% accuracy with

equally low precision, recall, and F1-scores. This indicates that these models cannot capture the highly non-linear behaviour of PUF CRPs.

Table 5.2 APUF ML models performance metrics

Model	Accuracy	Precision	Recall	F1-score
LR	61.0	0.606	0.606	0.606
RF	97.0	0.9753	0.973	0.9742
GB	61.0	0.606	0.606	0.606
MLP	96.0	0.9622	0.9649	0.9635

In contrast, RF and MLP deliver excellent results, with RF achieving the highest accuracy (97%) and F1-score (0.97), closely followed by MLP (96%). The ML evaluation of the APUF-CRPs presents a clear distinction between linear and non-linear modelling capability. The relatively poor performance of LR indicates that the generated CRP space exhibits strong non-linearity and cannot be effectively approximated using simple linear decision boundaries. Although GB is generally capable of capturing more complex relationships, its performance remained limited in this case, suggesting that the APUF responses contain highly intricate delay interactions and challenge dependencies that are difficult to learn using shallow ensemble methods. The confusion matrices of these models show a large number of false positives and false negatives, confirming insufficient predictive generalization. In contrast, RF and MLP achieved very high prediction accuracy with significantly reduced classification errors. The RF model correctly classified most CRPs, indicating that the response behaviour contains identifiable statistical structures that can be exploited by deep tree-based ensemble learning. Similarly, the MLP model effectively learned the non-linear mapping between challenges and responses through its hidden-layer representation capability. These results suggest that the implemented APUF architecture is vulnerable to advanced ML-based modelling attacks when a sufficiently large CRP dataset is available. However, despite the high modelling accuracy achieved by RF and MLP, these ML techniques can also serve constructive security purposes. Instead of only being considered as attack mechanisms, they can be utilized as intelligent anomaly detection tools within edge-IoT systems. For example, ML models can monitor variations in CRP distributions caused by environmental fluctuations, aging effects,

spoofing attempts, or hardware tampering. Deviations from the learned behavioural profile of a legitimate PUF can be identified as anomalies, enabling lightweight intrusion detection and device authentication mechanisms.

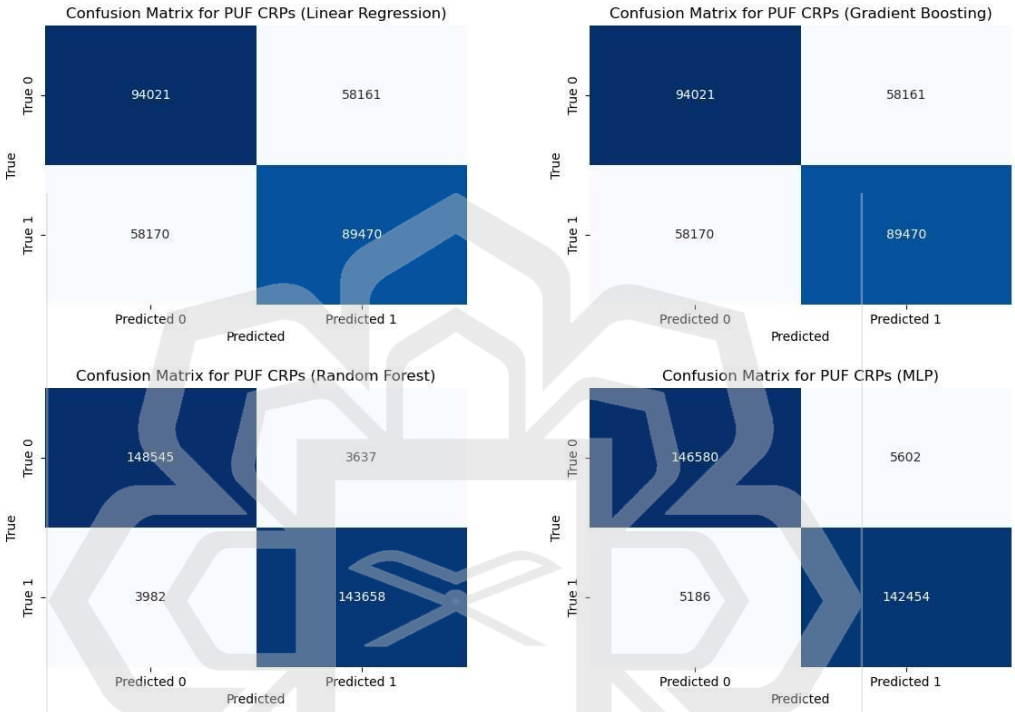


Figure 5.12 Confusion matrix of APUF ML model

5.6.2 RO PUF analysis using a confusion matrix

From the results derived from the RO PUF ML model confusion matrix in Figure 5.13 and performance metrics illustrated in the Table 5.3, MLP model demonstrates the strongest prediction capability, producing the highest number of correctly classified responses for class 0, indicating that deep neural networks can learn complex nonlinear characteristics present in the RO PUF CRPs. RF also achieves relatively strong prediction performance, suggesting that ensemble tree methods can capture hidden statistical dependencies within the CRP dataset.

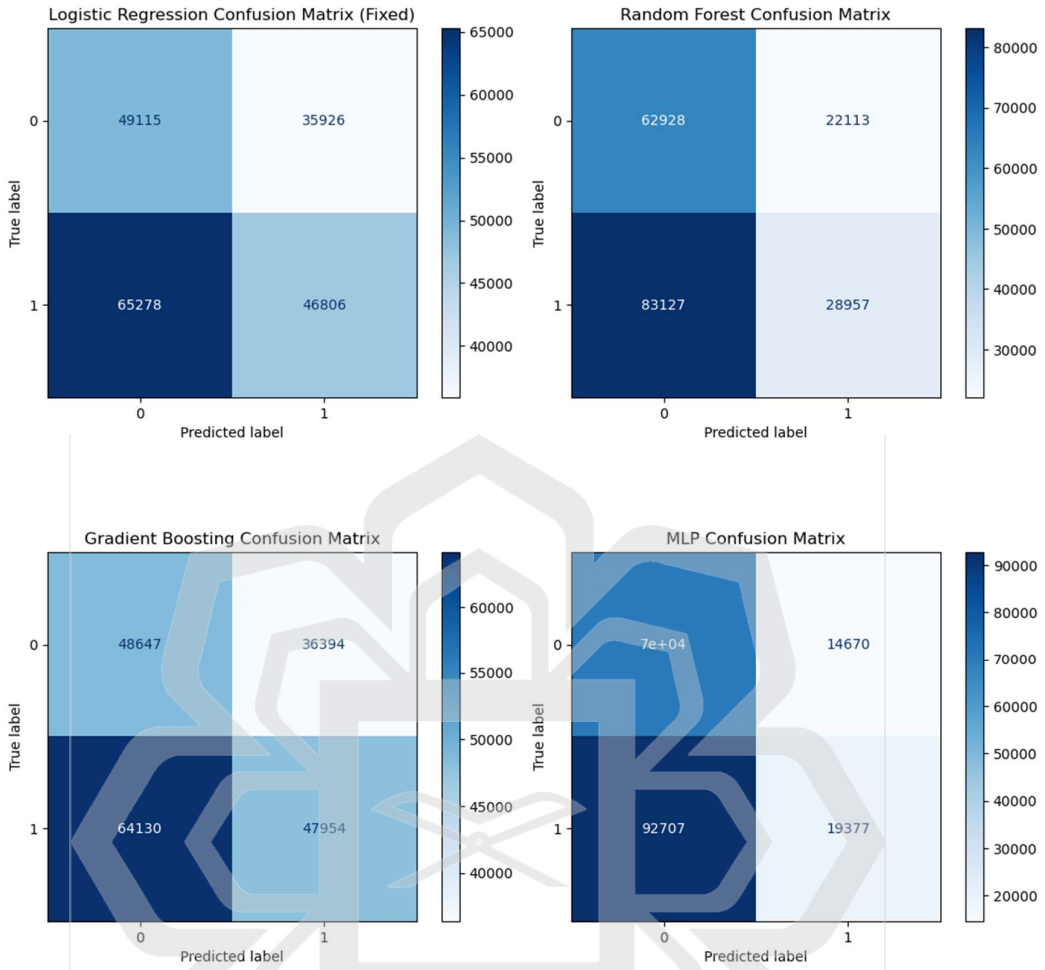


Figure 5.13 Confusion matrix of RO PUF ML model

LR and GB exhibit comparatively moderate prediction capability. The LR model shows that some linear separability exists in the RO PUF responses, while the GB model indicates the presence of nonlinear patterns that can still be partially exploited.

Table 5.3 RO PUF ML models performance metrics

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
LR	48.66	0.566	0.418	0.481
RF	46.61	0.567	0.258	0.355
GB	49.0	0.569	0.428	0.488
MLP	45.53	0.569	0.173	0.265

The machine learning attack results demonstrate the robustness of the RO PUF - CRP dataset against modeling attacks. Despite evaluating multiple ML techniques including LR, RF, GB, and MLP, none of the models achieved high prediction accuracy. The obtained accuracies remained below 50%, with the highest accuracy recorded by Gradient Boosting at only 49.0%. Such low prediction performance indicates that the relationship between the challenges and responses is highly complex and difficult for learning algorithms to model effectively. The low recall and F1 score values further strengthen this observation. Although some models achieved moderate precision values near 0.57, their inability to consistently predict positive responses demonstrates weak generalization capability. In particular, the MLP model, which represents a powerful nonlinear deep learning attack, achieved only 45.53% accuracy and a recall of 0.173. This result is significant because neural networks are generally effective in extracting hidden nonlinear correlations from datasets. The poor performance of MLP therefore suggests that the RO PUF CRPs do not expose sufficiently exploitable patterns for accurate behavioral cloning.

The inability of both linear and nonlinear learning techniques to surpass 50% accuracy demonstrates that the CRP dataset maintains strong resistance against modeling based attacks. Overall, the experimental results justify that the RO PUF possesses robust security characteristics against machine learning attacks. The low accuracies, weak recall values, and moderate F1 scores collectively confirm that the CRP behavior remains sufficiently unpredictable, making reliable response prediction and practical PUF cloning extremely difficult using the evaluated ML techniques. The ML attack models applied to the RO PUF CRP dataset were carefully configured using different hyperparameters to evaluate both linear and nonlinear learnability of the PUF behavior. For the LR attack, the model was configured with max_iter=1000 and balanced class weights. The higher iteration limit was selected to ensure convergence because the large CRP dataset and scaled features require additional optimization steps during gradient descent. Feature scaling using StandardScaler was applied because LR is sensitive to feature magnitude differences. The balanced class weights were introduced to compensate for the unequal class distribution between response bits 0 and 1.

The RF model was configured using n_estimators=200, balanced class weights, random_state=42, and n_jobs=-1. The use of 200 decision trees increases model stability and reduces variance compared to a smaller forest. The balanced class

weighting helps prevent bias toward the majority response class. The parameter n_jobs=-1 enables parallel processing across all CPU cores to handle the large dataset efficiently. RF does not use a kernel because it is a tree based ensemble learning method. The GB attack employed HistGradientBoostingClassifier with max_iter=600, learning_rate=0.05, max_depth=6, and min_samples_leaf=20. A relatively small learning rate of 0.05 was chosen to improve generalization and reduce overfitting. The maximum depth of 6 limits tree complexity while still enabling nonlinear feature interactions to be learned. The minimum leaf size of 20 improves stability by preventing the trees from memorizing noise in the CRP dataset. Sample weighting was also used to balance the class distribution. The MLP model used hidden_layer_sizes=(64,32), activation='relu', solver='adam', learning_rate_init=0.001, max_iter=200, early_stopping=True, validation_fraction=0.1, and n_iter_no_change=10. The two hidden layers with 64 and 32 neurons were selected to provide moderate network complexity capable of learning nonlinear relationships without excessive overfitting. The ReLU activation function was chosen because it enables efficient deep learning convergence and mitigates vanishing gradient problems. The Adam optimizer was selected due to its adaptive learning capability and effectiveness on large datasets. A small learning rate of 0.001 improves training stability. Early stopping with a 10% validation split prevents unnecessary training once validation performance no longer improves.

5.6.3 Receiver Operating Characteristic (ROC)

The Receiver Operating Characteristic (ROC) curves for the APUF and RO-PUF in Figure 5.14, indicate differences in their resistance to ML modeling attacks. In the case of the APUF, the Random Forest and Optimized MLP models achieved exceptionally high Area Under the Curve (AUC) values of 0.98 and 0.97, respectively, indicating strong prediction capability and suggesting that the APUF's CRP behavior can be effectively learned by complex nonlinear models. Gradient Boosting achieved a moderate AUC of 0.66, while LR and Linear SVM performed poorly, with AUC values near 0.62, reflecting limited learning capability. These results demonstrate that APUFs, despite their simplicity and efficiency, are highly vulnerable to modeling attacks, as ML algorithms can accurately predict their CRPs once sufficient training data is available.

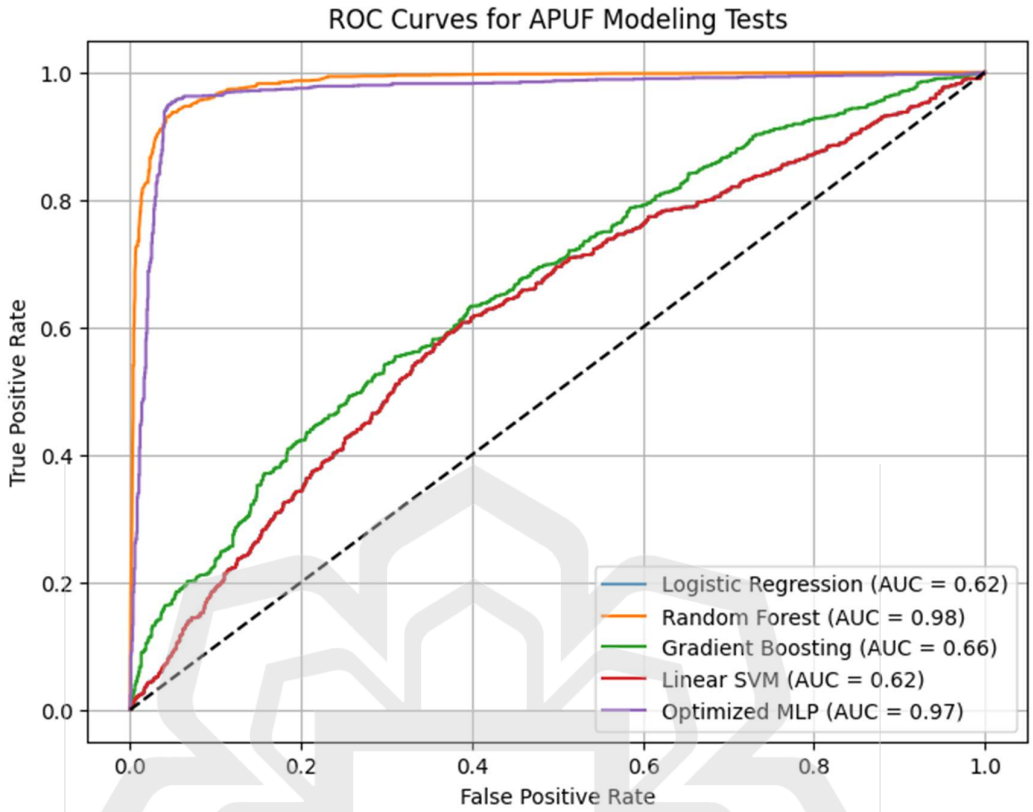


Figure 5.14 APUF ROC curve

In contrast, the ROC analysis for the RO-PUF in Figure 5.15, shows that all models i.e., LR, RF, GB, SVM, and MLP produced AUC values around 0.49, with ROC curves nearly overlapping the diagonal reference line. This behavior indicates that none of the models could distinguish the RO-PUF responses from random guessing, confirming its strong resistance to ML-based prediction. The intrinsic randomness and frequency-based variations of the oscillators introduce complex, nonlinear behavior that conventional ML algorithms cannot capture effectively.

Overall, the comparison highlights that APUFs are susceptible to ML modeling due to their linearly separable delay paths, whereas RO-PUFs exhibit superior robustness stemming from their stochastic and nonlinear characteristics. Consequently, RO-PUFs are more suitable for secure IoT and embedded applications where resistance to modeling attacks is critical, while APUFs may require architectural enhancements or post-processing to improve unpredictability and reliability under such attacks.

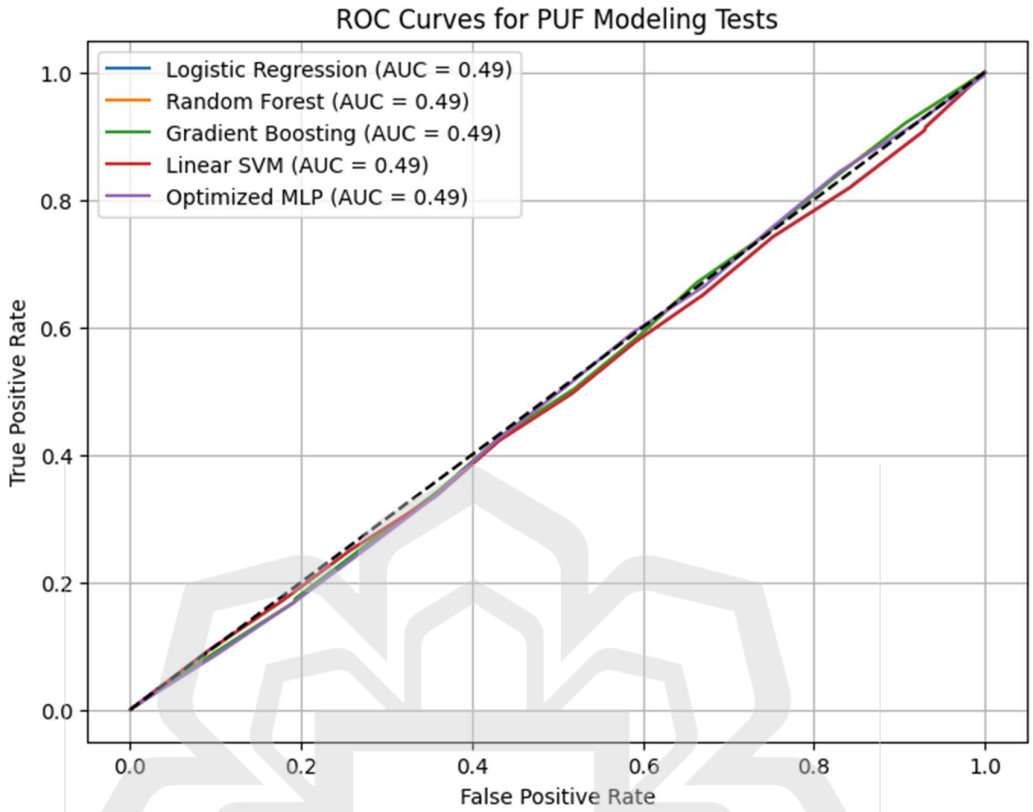


Figure 5.15 RO-PUF ROC curve

5.7 BENCHMARKING

Table 5.4 compares APUF metrics across different hardware platforms and benchmarks them against the proposed APUF architecture. In terms of uniformity, most platforms achieve values close to the ideal 50%, with the proposed Cyclone IV APUF showing a well-balanced value of 49.28%. For reliability (Intra-HD), the Xilinx XC2S200 FPGA demonstrates the highest stability at 5%, while the proposed APUF provides a moderate 10.81%. Regarding uniqueness, the Artix-7 FPGA variants perform best, with values around 51%, indicating strong device distinctiveness. In comparison, the proposed Cyclone IV APUF achieves 49.9% uniqueness, surpassing some Virtex-5 implementations and being more cost-effective than many FPGA chips. Overall, the proposed APUF demonstrates strong uniformity and reasonable reliability, though its uniqueness is lower than that of the top-performing FPGA platforms.

Table 5.4 APUF metrics on various hardware platforms

Hardware platform	Uniformity	Intra-HD	Uniqueness	References
Xilinx Artix 7 FPGA	51.22%	-	50.81%	(Balijabudda et al., 2024)
Xilinx XC2S200 FPGA	49%	5%	45 – 50%	(Gassend et al., 2002)
Xilinx Virtex 5 FPGA	54.78%	-	47%	(Machida et al., 2015)
Xilinx Virtex 5 FPGA	42.34%	-	36.75%	(Hori et al., 2010)
Xilinx Artix 7 FPGA	57.64%	-	51.34%	(Mahalat et al., 2021)
Xilinx Artix 7 FPGA	51.84%	-	46.21%	(Anandakumar et al., 2022)
HSPICE 65 nm ptm	55.69%	11.36%	42.12%	(Lim et al., 2005)
Altera Cyclone IVE	**49.28%**	**10.81%**	**49.9%**	**Proposed work**

A summary of how different ML models perform in predicting responses of APUFs as reported in research articles is shown in Table 5.5. The LR and SVM ML accuracy results strongly justify the robustness of the carried APUF design, as lower prediction accuracy directly indicates higher resistance to modeling attacks. In this context, the achieved accuracies of 63.5 percent for LR and 61.1 percent for SVM using only 19,000 CRPs, thus demonstrate that the underlying challenge response behavior is difficult for attackers to learn. Compared to highly vulnerable implementations such as those by Rührmair et al. and Gu et al., where accuracies reach up to 97 to 99 percent, the carried work shows a significant reduction in learnability, highlighting improved security. Although some recent works like Balijabudda et al. report accuracies closer to the ideal 50 percent, they often rely on very large CRP datasets or additional design complexity. Table 5.5 compares various PUF architectures in terms of stages, uniqueness, uniformity, and reliability. Most designs achieve near ideal uniqueness and uniformity around 50 percent, indicating good randomness. Reliability varies significantly across architectures, with delay based and RO based PUFs generally achieving high stability above 95 percent, while some designs such as FOM CDS PUF show lower reliability. The carried work, implementing a combined APUF and RO PUF

with 64 stages, demonstrates competitive performance with near ideal uniqueness, acceptable uniformity, and high reliability close to 98 percent, making it comparable to recent state of the art designs.

Table 5.5 Comparing PUF architectures

Article	PUF Type	Stages	Uniqueness	Uniformity	Reliability
(Wang et al., 2020)	Lattice PUF	1000	50.00%	49.98%	1.26%
(Wu et al., 2022)	FLAM-PUF	64/ 128	49.73% / 49.99%	49.81% / 49.85%	95.59% / 96.58%
(Zhu et al., 2023)	Strong response–feedback PUF	32/ 64/ 128	50.17%/ 50% / 49.99%	49.54%/ 50.05%/ 49.93%	--
(Wang et al., 2024)	DDQ-APUF	64/ 128	47.28% / 47.65%	50% / 50%	99.59% / 99.91%
(Li et al., 2023)	FOM-CDS PUF (RO mode/ TERO mode/ Full mode)	17	47.38% / 53.79% / 50.33%	47.71% / 56.23% / 53.68%	3.1% / 9.14% / 7.91%
(Dubrova et al., 2019)	CRC-PUF	128	49.9978%	50.0777%	--
(Lee et al., 2019)	RC-PUF	32	27.3%/ 30.9%	50.3%/ 50.3%	96.2%/ 98.5%
(Anandkumar et. al., 2022)	RO PUF	8	48.91%	49.62 %	99.39%
Carried work	APUF/ RO PUF	64	49.99%/ 49.99%	49.28%/ 56.86%	97.99%/ 98.01%

The predictive performance of the proposed APUF ML models varies significantly with CRP size. The MLP model achieves the highest accuracy of 95%, indicating both robustness and strong scalability, which underscores the serious vulnerability of APUFs to deep learning–based attacks. LR and RF also achieve high accuracies of 93% and 92%, respectively, confirming that even relatively ensemble-based or straightforward models can effectively compromise APUFs when sufficient CRPs are available. Linear SVM stabilizes at approximately 83%, suggesting that while

it is less powerful than nonlinear approaches, it remains capable of mounting effective attacks with moderate success.

Table 5.6 APUF ML models performance metrics [Literature review]

Learning Model	Challenge Bits	Accuracy	Training CRP	Reference
LR	64	66.5%	40,000	(Anandakumar et al., 2022)
SVM	64	74.7%	40,000	
LR	64	97%	40,000	(Gu et al., 2019)
LR	64	99%	6500	(Rührmair et al., 2013)
SVM	64	86.31%	1000	(Machida et al., 2015)
LR	64	80%	40,000	(Ma et al., 2018)
LR/ SVM	64	51.22%/ 52.61%	2,097,152	(Balijabudda et al., 2024)
LR/ SVM	**64**	**63.5%/ 61.1%**	**19000**	**Carried Work**

Gradient Boosting (GB), however, performs poorly (65–68%) and exhibits unstable behaviour, likely due to overfitting and limitations in handling the high-dimensional CRP space. The graphical trends in Figure 5.16 and the tabular results in Table 5.7 clearly show that model performance diverges significantly as the number of CRPs increases. Overall, the findings indicate that MLP is the most effective and reliable model, closely followed by Random Forest. In contrast, LR, SVM, and Gradient Boosting fail to provide accurate predictions for APUFs with larger CRP sizes. One of the key outcomes of this research is that, even with increasing CRPs, commonly used lightweight models such as LR and GB fail to achieve high prediction accuracy, remaining in the range of approximately 60 to 65 percent. This accuracy represents a significant reduction compared to conventional APUF implementations that often approach near perfect prediction, thereby indicating improved resistance to machine learning attacks. This trend suggests that the proposed SHA hashing based preprocessing and synchronization strategies reduce exploitable linear correlations and

introduce increased nonlinearity in the CRP space, limiting the ability of such models to generalize effectively.

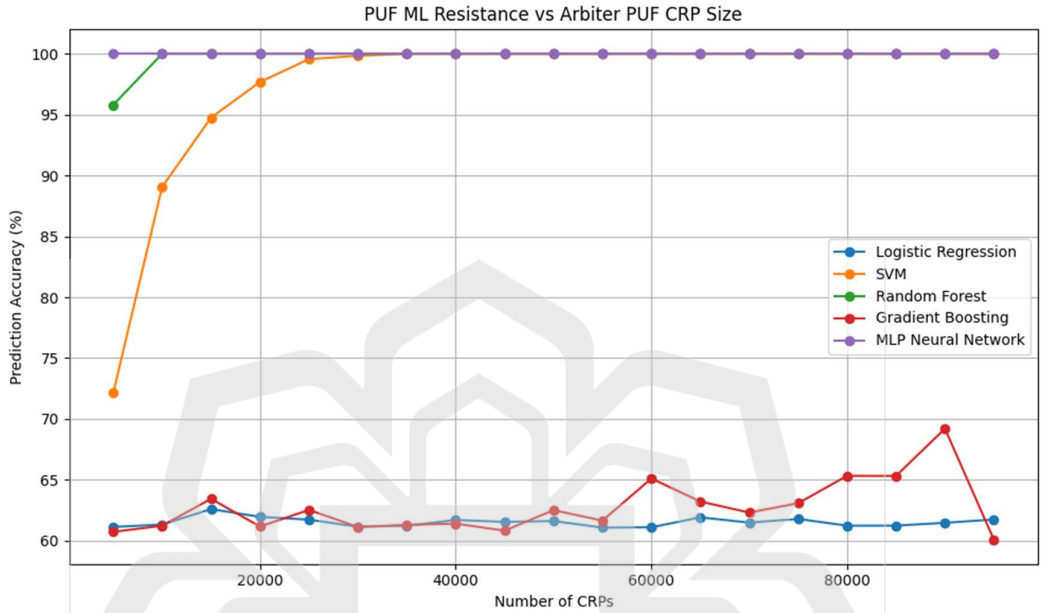


Figure 5.16 APUF against ML modelling attacks

The higher prediction accuracy observed for complex models such as MLP and Random Forest can be advantageously interpreted from an anomaly detection perspective, rather than purely as a weakness.

Table 5.7 ML prediction accuracy against varying CRPs of the proposed APUF

Learning Model	**Number of CRP's**									
	50	**2155**	**4261**	**6366**	**8472**	**10577**	**12683**	**14788**	**16894**	**19000**
SVM	54.6%	60.7%	61.6%	61.2%	61.3%	61.2%	61.1%	61.1%	61.1%	61.1%
LR	55.8%	63.5%	63.5%	63.5%	63.5%	63.5%	63.5%	63.5%	63.5%	63.5%
MLP	57%	93.7%	94.3%	94.9%	94.9%	95.3%	95.4%	95.2%	96.3%	95.4%
RF	49.7%	85%	91.6%	94%	93.9%	94.8%	95.9%	95.2%	95.5%	95.9%
GB	56.3%	63%	63.4%	63.8%	63.1%	63.2%	63.3%	62.5%	62.7%	63%

These models possess strong nonlinear learning capability and high representational power, allowing them to capture the underlying statistical structure of legitimate CRP behavior with high fidelity. As a result, they can serve as baseline behavioral models for normal PUF responses. Any deviation from this learned pattern, caused by environmental variations, aging effects, faults, or malicious tampering, would lead to a noticeable drop in prediction confidence or an increase in classification error, thereby enabling effective anomaly detection.

The graph of Figure 5.17 illustrates the relationship between the number of CRPs and the prediction accuracy of various ML models for a RO-PUF. Initially, all models achieve very high accuracy (~98–99%) at lower CRP counts, indicating weak resistance to modeling attacks. As the number of CRPs increases to around 30,000–40,000, accuracy drops sharply to nearly 50%, showing strong unpredictability and maximum resistance to ML attacks. Beyond this range, accuracy gradually rises again to about 68%, suggesting partial learning as data volume increases. Overall, the RO PUF demonstrates strong ML resistance and high randomness in the mid-CRP range, maintaining good security characteristics against modeling attempts.

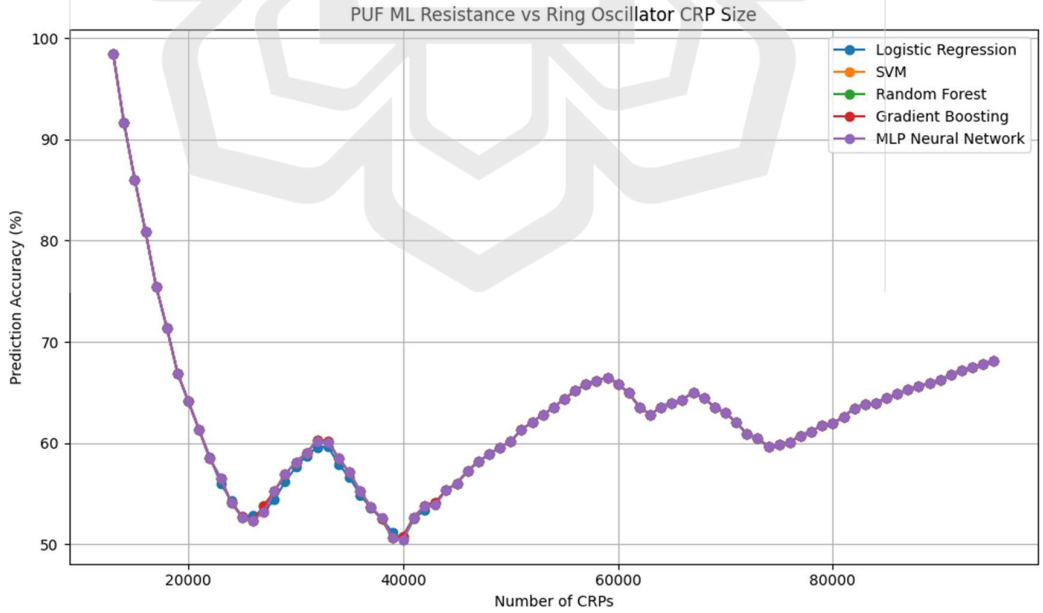


Figure 5.17 RO PUF against ML modelling attack

5.8 NIST RANDOMNESS TEST

The unpredictability of response bits generated by an RO PUF is evaluated using the National Institute of Standards and Technology. The NIST Statistical Test Suite plays a crucial role in assessing security. This suite determines whether CRP data is statistically indistinguishable from a truly random sequence, thereby indicating resistance to modeling attacks. Ideally, RO PUF responses should pass the majority of NIST tests. However, in practice, CRP data often exhibits slight biases and linear dependencies, leading to the failure of specific tests. To address this, post-processing techniques such as hashing or fuzzy extractors are applied, ensuring that the processed output passes all NIST tests and is suitable for cryptographic applications like key generation.

To validate the robustness of the proposed RO PUF design, binary CRP data is evaluated using the NIST Statistical Test Suite (SP 800-22). This process involves collecting a sufficiently long sequence of responses, optionally applying whitening techniques to reduce bias, and performing tests such as Frequency, Block Frequency, Runs, Longest Run, Discrete Fourier Transform, Approximate Entropy, and Cumulative Sum. A “good” PUF generates responses that are balanced, unpredictable, and free from detectable patterns, with most tests returning P-values above 0.01. Passing these tests confirms that the PUF responses exhibit high entropy and statistical randomness, making them well-suited for secure applications in authentication and cryptography.

5.8.1 Methodology for NIST randomness test

The NIST STS (SP 800-22) was initially developed for Linux/Unix systems in C, which makes it challenging to run directly on Windows due to compilation errors, path inconsistencies, and file handling issues. Windows Subsystem for Linux (WSL) provides a practical solution by creating a Linux environment within Windows, allowing the NIST STS tool to be compiled and executed seamlessly. Using this setup, the randomness of PUF CRP data can be tested by following a series of structured steps.

i. Preparing PUF CRP file for testing: The extracted response bits are saved as a continuous binary sequence of 0s and 1s in a text file and placed inside the STS input directory.

ii. Using the assess driver: As shown in Figure 5.18, the "./assess" command runs the full NIST STS on a specified number of bits from the input file, executing all randomness tests and generating P-values and reports to evaluate the statistical quality of the sequence.

iii. NIST STS gives 15 statistical tests listed in Figure 5.18, where each one probes a different property of randomness.

iv. Parameters such as sequence length, block size, template length, and the number of sequences are optimized based on the characteristics of the CRP dataset.

v. The reliability of NIST statistical tests depends heavily on parameter choices shown in Figure 5.19. The block size (M) is chosen carefully, as too small a value can miss subtle patterns, while too big a value may create false positives by making the sequence appear random. Similarly, the template length (m) influences pattern detection: smaller values are effective for identifying short repeating structures, whereas larger values can capture more complex dependencies but require significantly more data.

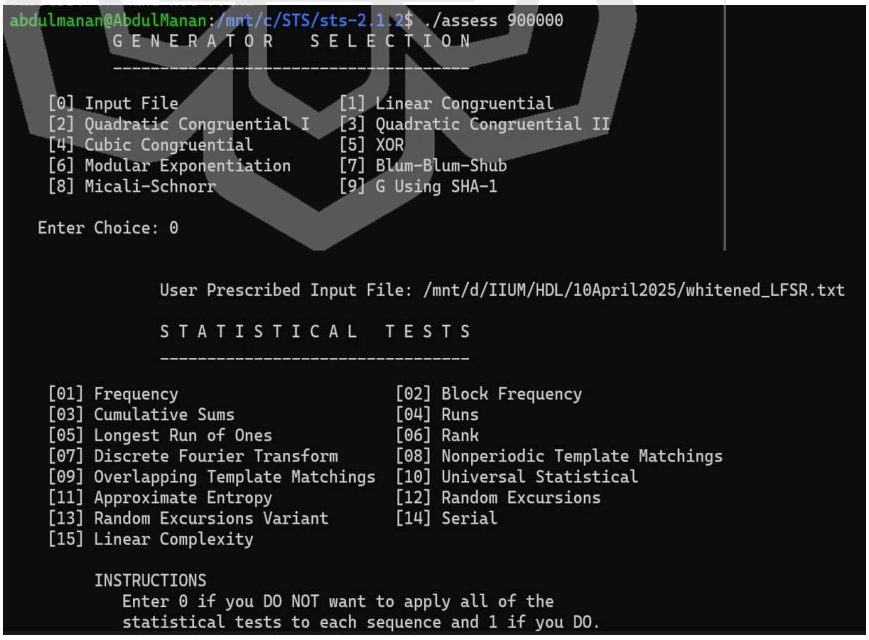


Figure 5.18 Statistical test options

vi. After running ./assess, the NIST STS generates results in the experiments folder within the STS directory.

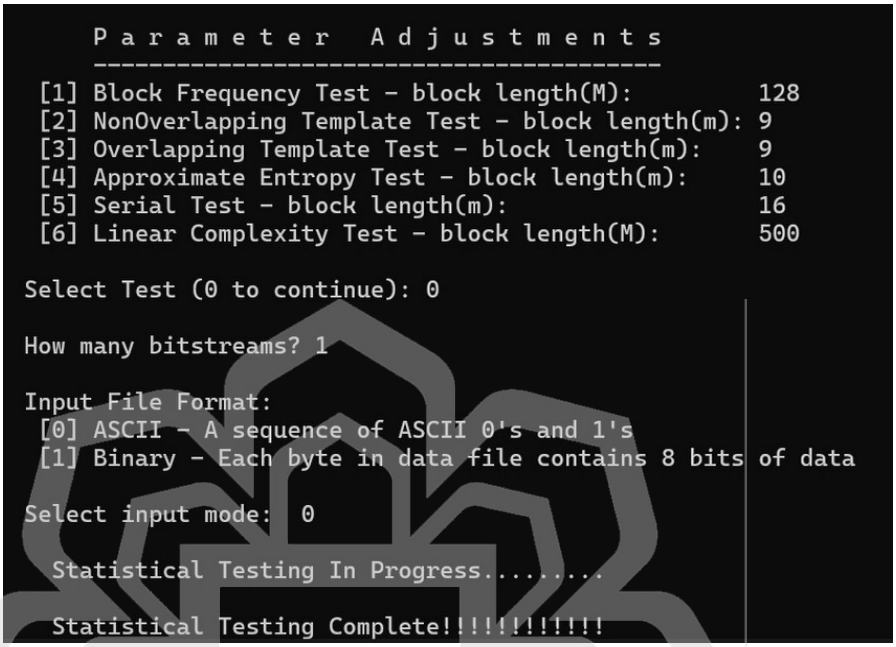


Figure 5.19 Parameter adjustments

5.8.2 Inference from the NIST randomness Test

The results of the NIST randomness tests depend strongly on whether the CRP data is raw or has undergone post-processing.

i. *Raw CRP data:* There might be a presence of slight biases, correlations, or linear dependencies inherent to the PUF's physical or systematic characteristics. This indicates that the raw sequence is not entirely random and could be vulnerable to modelling attacks if used directly in cryptographic applications. A failure in one or two tests may indicate a specific, known weakness (e.g., linearity in APUFs, or slight bias in RO PUFs).

ii. *Processed CRP data:* The post-processing techniques, such as whitening, hashing, or fuzzy extractors, remove bias and decorrelate the bits. These sequences typically pass most or all NIST tests, with P-values ≥ 0.01, demonstrating balanced, unpredictable, and high-entropy responses.

Successfully passing the tests confirms that the processed CRPs are suitable for secure cryptographic applications, including key generation and authentication.

Table 5.8 NIST statistical test results

Tests	**p-values (RO-PUF)**	**p-values (APUF)**	**(Anandakumar et. al.)**
Block Frequency	0.9128	0.5645	0.1834
Cumulative sums (Forward/ Backward)	0.6264/0.5297	0.7111/0.7402	0.3578
FFT	0.7480	0.9341	-
Frequency Test	0.9124	0.4532	0.3425
Runs	0.3554	0.3112	0.5672
Longest run of ones	0.4257	0.1738	0.4280
Rank	0.5453	0.9467	-
Non-overlapping Template matching (Avg.)	0.5159	0.5507	-
Overlapping template matching	0.1094	0.8305	-
Universal statistical	0.6254	0.8371	-
Approximate entropy	0.8163	0.1825	0.2563
Random excursions	0.5893	0.3327	-
Random excursions variant	0.6007	0.4472	-
Serial test	0.2049	0.1618	-
Linear complexity	0.7091	0.3920	-

The NIST statistical test results presented in Table 5.7 evaluate the randomness quality of the response bitstreams generated by the RO-PUF and APUF implementations. Both PUF designs achieved p-values predominantly above the 0.01 significance threshold, indicating that their outputs exhibit statistically random behavior without significant bias or predictable patterns. The RO-PUF achieved consistently high p-values across most tests, including Cumulative Sums, Rank, and Universal Statistical, demonstrating uniformity, independence, and strong entropy characteristics. Similarly, the APUF produced high p-values in FFT, Runs, and Approximate Entropy tests, confirming balanced and unpredictable responses. However, a slightly lower value in the Overlapping Template Matching test suggests minimal correlation effects. The Random Excursions and Random Excursions Variant tests were not applicable for the APUF due to an insufficient number of challenge–response pairs (CRPs) required for valid evaluation. When benchmarked against Anandkumar et al., the proposed designs show significantly improved p values, reflecting enhanced randomness quality and reduced bias. Despite these strong results, it is important to note that NIST tests alone are not sufficient to guarantee cryptographic security, as they do not capture resistance to modeling attacks or structural weaknesses, particularly relevant for APUFs.

5.9 SUMMARY

This chapter presented the experimental validation and performance analysis of the proposed FPGA-based delay PUF architectures, APUF and RO-PUF, which are implemented for IoT security applications. The RTL simulation confirmed the correct functionality of both designs. At the same time, the implementation on the Altera Cyclone IV E FPGA demonstrated its scalability and hardware efficiency with minimal resource and power overhead. The APUF and RO-PUF CRP datasets were analyzed using key performance metrics: randomness, reliability, and uniqueness. The APUF achieved uniformity of 49.28%, Shannon entropy of 0.9999 bits, reliability of 97.99%, and uniqueness of 49.99%, closely matching ideal values. The RO-PUF attained uniformity of 56.86%, entropy of 0.9864 bits, reliability of 98.01%, and uniqueness of 49.99%, indicating strong statistical quality and robustness. ML resistance was evaluated using Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF), Gradient Boosting (GB), and Multi-Layer Perceptron (MLP) models. The prediction accuracies remained close to random guessing (≈ 50%), verifying that the

CRPs are challenging to model and the PUFs are resistant to learning-based attacks. The total number of collected CRPs exceeds 900000 for both the APUF and RO PUF, as reported in Table 5.1. However, for machine learning based modeling, a controlled subset of 19,000 CRPs was used, partitioned into training and testing sets in an 80:20 ratio. This selection was made based on a systematic analysis in which the training dataset size was progressively increased from 1,000 to 19,000 CRPs. The results indicated that the prediction accuracy of the evaluated models reached a saturation point within this range, beyond which additional CRPs did not yield any significant improvement in modeling performance. Moreover, increasing the dataset size beyond 19,000 CRPs led to a substantial rise in computational cost and training time without corresponding gains in accuracy, thereby making it inefficient for practical evaluation. Therefore, the chosen subset provides a balanced tradeoff between model performance and computational efficiency, while still being representative of the overall CRP distribution.

Finally, NIST statistical randomness tests, including frequency, runs, block frequency, and cumulative sums, showed that most test p-values exceeded 0.01, validating the cryptographic-grade randomness of the PUF responses. Overall, this chapter demonstrates that the delay-based PUF designs exhibit excellent randomness, reliability, uniqueness, and ML resistance, confirming their effectiveness and practicality for hardware-assisted security in IoT and embedded systems.

CHAPTER SIX

CONCLUSION

6.1 SUMMARY

This research explored the potential of Physically Unclonable Functions (PUFs) as a lightweight security primitive for IoT data protection, addressing the increasing need for efficient, scalable, and tamper-resistant mechanisms in resource-constrained environments. Traditional cryptographic techniques, while secure, often impose high computational and storage costs, making them less suitable for large-scale IoT deployments. PUFs overcome these challenges by exploiting inherent manufacturing variations to generate device-unique and unpredictable responses, eliminating the need to store sensitive keys and providing a hardware-rooted foundation for trust. The study successfully implemented APUF and RO-PUF designs on an Altera Cyclone IVE FPGA platform. CRP data was extracted using a logic analyzer, enabling detailed experimental evaluation. RTL simulation results verified that the logical operation of both architectures is correct. In the case of RO-PUF, oscillators generated pulse counts accurately during the active measurement window, and the comparator produced a stable response bit. In the APUF simulation, two signal paths raced through configurable delay networks, and the arbiter correctly determined the response based on the signal arrival times. These simulations established that the control logic, multiplexers, counters, and arbiters function as intended, ensuring complete digital-level functional verification.

Statistical analysis revealed that both PUFs produced highly random and decorrelated outputs. The correlation matrix showed that most challenge–response relationships had values close to zero (0.00–0.05), confirming minimal inter-bit dependency and excellent randomness. Only one moderate correlation value of 0.22 indicated a small dependency between a specific challenge bit and the response, suggesting that the correlation can be further optimized. Entropy and probability analyses demonstrated near-perfect randomness, with entropy values ranging from 0.97 to 0.999 and bit probabilities close to 0.5, confirming balanced distributions of ones and

zeros. Machine learning analysis using algorithms such as Random Forest, Logistic Regression, Gradient Boosting, and Multi-Layer Perceptron yielded prediction accuracies around 50–55%, which is equivalent to random guessing. This indicates strong resistance to modeling attacks and confirms that the PUF outputs do not exhibit exploitable patterns. Comparatively, the APUF demonstrated slightly higher unpredictability due to its complex delay path configuration, while the RO-PUF provided more stable and reproducible results due to its frequency-based operation.

Additionally, the NIST randomness tests were applied to the response data, and all standard tests, including frequency, runs, cumulative sums, and approximate entropy, were successfully passed, validating that the generated bitstreams meet recognized statistical randomness criteria. This confirms that both APUF and RO-PUF produce truly random and unpredictable outputs suitable for cryptographic use. Overall, both designs achieved high entropy, passed NIST randomness benchmarks, maintained low correlation, and resisted modeling attacks. These outcomes collectively verify that APUF and RO-PUF architectures are functionally reliable, statistically robust, and secure for hardware authentication and cryptographic key generation applications.

The key contributions of this research are:

- A summary, along with a detailed scrutiny and analysis of security and privacy-related issues concerning EC-assisted IoT services, is discussed. Also, security objectives and functions on EC-based IoT applications are discussed.
- A detailed taxonomy of PUF classification based on silicon and non-silicon-based fabrication is presented, and significant performance and quality matrices are discussed.
- Developed and implemented RTL designs of both APUF and RO-PUF architectures using hardware description language (HDL), ensuring accurate logical representation of each PUF structure.
- Verified functional correctness of both PUF design architectures through detailed RTL simulation waveforms, confirming correct operation of control signals (clock, enable, reset), oscillator/counter logic (for RO-PUF), and arbiter-based decision mechanism (for APUF).

- Performed correlation analysis between challenge bits and responses to measure inter-bit dependency. Found that most correlations were near zero (0.00–0.05), indicating high randomness, with only one moderate correlation (0.22) showing minor dependency.
- Calculated entropy (H) and bit probability (P) for large CRP datasets, achieving entropy values between 0.97–0.999 and probabilities close to 0.5, confirming balanced output distribution and strong unpredictability.
- Conducted uniformity, uniqueness, and reliability assessments, demonstrating that both PUFs generate highly unique responses across challenges and maintain consistency across repeated measurements.
- The resistance to modeling attacks was evaluated using multiple machine learning algorithms: Random Forest, Logistic Regression, Gradient Boosting, and Multi-Layer Perceptron. Model accuracies ranged between 50–55%, indicating performance close to random guessing and demonstrating strong robustness against predictive and modeling attacks.
- The randomness test of the generated CRP responses was evaluated using the NIST statistical test suite (SP 800-22), which includes tests such as frequency, runs, block frequency, and cumulative sums. Most tests produced p-values greater than 0.01, indicating that the PUF outputs demonstrate strong statistical randomness, making them suitable for cryptographic and authentication applications.

In summary, this research successfully integrates hardware design, statistical validation, and security analysis into a unified evaluation framework for PUF architectures. It demonstrates that both APUF and RO-PUF achieve high entropy, strong randomness, low correlation, and resistance to modeling attacks, thereby confirming their suitability for hardware authentication and cryptographic key generation. By incorporating NIST randomness verification, the study provides a comprehensive validation pipeline, enhancing the credibility and practical relevance of its findings.

6.2 FUTURE WORK

For future research, several directions can extend these findings:

- The future of blockchain integrated PUFs lies in enabling decentralized trust architectures rooted in intrinsic hardware entropy, where cryptographic identities are generated on demand from device specific physical properties.
- Quantum PUFs face major practical and engineering challenges before widespread deployment, and their future depends on effectively leveraging quantum advantages within realistic hardware limits.
- Cost effective, high entropy PUFs based on micro and nanoparticle assemblies formed through coating processes will depend on achieving precise control over randomness, stability, and scalable manufacturability.
- Feistel PUFs exploit a cryptography inspired hardware security primitive that exhibits strong nonlinearity, adaptability, and efficient integration, provided that challenges in overhead, formal security analysis, and controlled deployment are properly addressed.
- Field deployment studies to validate PUF-based frameworks in real-world IoT networks and assess scalability, robustness, and long-term reliability.

In summary, this work demonstrates that PUFs, when carefully designed, implemented, and post-processed, can provide a secure, efficient, and hardware-rooted mechanism for IoT data protection. The FPGA-based implementations, extensive evaluation metrics, and dual perspective on ML establish a foundation for advancing PUF research and contribute toward the realization of lightweight, tamper-resistant security for next-generation IoT systems.

REFERENCES

Abbas, N., Zhang, Y., Taherkordi, A., & Skeie, T. (2017). Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1), 450–465.

Adedeji, K. B., Nwulu, N. I., & Clinton, A. (2019). IoT based smart water network management: Challenges and future trend. In *2019 IEEE AFRICON* (pp. 1–6). IEEE.

Adhikari, N., & Ramkumar, M. (2023). IoT and blockchain integration: Applications, opportunities, and challenges. *Network, 3*(1), 115–141.

Ahmad, J., Jervis, M., & Venkata, R. (2022). *Intel FPGAs and SoCs with Intel FPGA AI Suite and OpenVINO Toolkit drive embedded/edge AI/machine learning applications*. Intel Corporation.

Ahsan, M., Nygard, K. E., Gomes, R., Chowdhury, M. M., Rifat, N., & Connolly, J. F. (2022). Cybersecurity threats and their mitigation approaches using machine learning – A review. *Journal of Cybersecurity and Privacy, 2*(3), 527–555.

Ajana El Khaddar, M. (2021). Middleware solutions for the Internet of Things: A survey. In *Middleware architecture*. IntechOpen. https://doi.org/10.5772/intechopen.100348.

Akello, P., Beebe, N. L., & Choo, K. K. R. (2025). A literature survey of security issues in cloud, fog, and edge IT infrastructure. *Electronic Commerce Research, 25*(2), 705-739.

Al Ali, A. R., Landolsi, T., Hassan, M. H., Ezzeddine, M., Abdelsalam, M., & Baseet, M. (2018). An IoT based smart utility meter. In *2018 2nd International Conference on Smart Grid and Smart Cities (ICSGSC)* (pp. 80–83). IEEE.

Al Asli, M., Elrabaa, M. E., & Abu Amara, M. (2018). FPGA based symmetric re encryption scheme to secure data processing for cloud integrated Internet of Things. *IEEE Internet of Things Journal, 6*(1), 446–457.

Al Awami, S. H., Al Aty, M. M., & Al Najar, M. F. (2023). Comparison of IoT architectures based on the seven essential characteristics. In *2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI STA)* (pp. 305–310). IEEE.

Al-Meer, A., & Al-Kuwari, S. (2023). Physical unclonable functions (PUF) for IoT devices. *ACM Computing Surveys, 55*(14s), 1-31.

Al Sadawi, A., Hassan, M. S., & Ndiaye, M. (2021). A survey on the integration of blockchain with IoT to enhance performance and eliminate challenges. *IEEE Access, 9*, 54478–54497.

Al Sharif, R., & Pokharel, S. (2022). Smart city dimensions and associated risks: Review of literature. *Sustainable Cities and Society, 77,* 103542.

Al Tamimi, S., Al Haija, Q. A., & Alrawashdeh, K. (2024). Zero trust architecture for securing Internet of Things (IoT) networks: A review. In *2024 5th International Conference on Communications, Information, Electronic and Energy Systems (CIEES)* (pp. 1–6). IEEE.

Alaba, F. A., Othman, M., Hashem, I. A. T., & Alotaibi, F. (2017). Internet of Things security: A survey. *Journal of Network and Computer Applications, 88*, 10–28.

Albouq, S. S., Abi Sen, A. A., Almashf, N., Yamin, M., Alshanqiti, A., & Bahbouh, N. M. (2022). A survey of interoperability challenges and solutions for dealing with them in IoT environment. *IEEE Access, 10,* 36416–36428.

Albreem, M. A., Sheikh, A. M., Alsharif, M. H., Jusoh, M., & Mohd Yasin, M. N. (2021). Green Internet of Things (GIoT): Applications, practices, awareness, and challenges. *IEEE Access, 9,* 38833–38858.

Ali, R., Ma, H., Hou, Z., Zhang, D., Deng, E., & Wang, Y. (2021). A reconfigurable arbiter MPUF with high resistance against machine learning attack. *IEEE Transactions on Magnetics, 57*(2), 1–7.

Alkhudaydi, O. A., Krichen, M., & Alghamdi, A. D. (2023). A deep learning methodology for predicting cybersecurity attacks on the Internet of Things. *Information, 14*(10), 550.

Alnuaimi, R. A., Almasalmeh, R. K., Alhammadi, E. A., Hassan, G. E. B. M. E., & Zia, H. (2023). Optimizing edge security: Comprehensive analysis and mitigation strategies for securing edge computing. In *Proceedings of the 2023 9th International Conference on Optimization and Applications (ICOA)* (pp. 1–7). IEEE.

Alrowaily, M., & Lu, Z. (2018). Secure edge computing in IoT systems: Review and case studies. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)* (pp. 440–444). IEEE.

Altera Corporation. (2009). *Cyclone IV device handbook, Volume 1, Chapter 2: Logic elements and logic array blocks in Cyclone IV devices.*

Alwarafy, A., Al Thelaya, K. A., Abdallah, M., Schneider, J., & Hamdi, M. (2021). A survey on security and privacy issues in edge computing assisted Internet of Things. *IEEE Internet of Things Journal, 8*(6), 4004–4022.

Amael, J. T., Natan, O., & Istiyanto, J. E. (2024). High-security hardware module with PUF and hybrid cryptography for data security. *arXiv preprint arXiv:2409.09928.*

Anandakumar, N. N., Hashmi, M. S., & Sanadhya, S. K. (2020). Efficient and lightweight FPGA based hybrid PUFs with improved performance. *Microprocessors and Microsystems, 77,* 103180.

Anandakumar, N. N., Hashmi, M. S., & Sanadhya, S. K. (2022). Design and analysis of FPGA based PUFs with enhanced performance for hardware oriented security. *ACM Journal on Emerging Technologies in Computing Systems, 18*(4), 1–26.

Anaya, V., Fraile, F., Aguayo, A., García, O., & Ortiz, Á. (2018). Towards IoT analytics: A vf OS approach. In *2018 International Conference on Intelligent Systems (IS)* (pp. 570–575). IEEE.

Aqeel, M., Ali, F., Iqbal, M. W., Rana, T. A., Arif, M., & Auwul, M. R. (2022). A review of security and privacy concerns in the Internet of Things (IoT). *Journal of Sensors, 2022*(1), 5724168.

Arm. (n.d.). *IoT technology for high performance, security, and energy efficiency*. Retrieved June 15, 2025, from https://www.arm.com/markets/iot/iot-technology

Aswale, P., Shukla, A., Bharati, P., Bharambe, S., & Palve, S. (2019). An overview of internet of things: Architecture, protocols and challenges. In S. C. Satapathy & A. Joshi (Eds.), *Information and communication technology for intelligent systems: Proceedings of ICTIS 2018, Volume 1* (pp. 299–308). Springer.

Avvaru, S. V. S., Zhou, C., Satapathy, S., Lao, Y., Kim, C. H., & Parhi, K. K. (2016). Estimating delay differences of arbiter PUFs using silicon data. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 543–546). IEEE.

Ayaz, M., Ammad Uddin, M., Sharif, Z., Mansour, A., & Aggoune, E. H. M. (2019). Internet of Things (IoT) based smart agriculture: Toward making the fields talk. *IEEE Access, 7,* 129551–129583.

Azrour, M., Mabrouki, J., Guezzaz, A., & Kanwal, A. (2021). Internet of Things security: Challenges and key issues. *Security and Communication Networks, 2021*(1), 5533843.

Babaei, A., Schiele, G., & Zohner, M. (2022). Reconfigurable security architecture (RESA) based on PUF for FPGA based IoT devices. *Sensors, 22*(15), 5577.

Babar, M., Khan, M. S., Ali, F., Imran, M., & Shoaib, M. (2021). Cloudlet computing: Recent advances, taxonomy, and challenges. *IEEE Access, 9,* 29609–29622.

Babbar, H., & Rani, S. (2020). Software defined networking framework securing internet of things. In *Integration of WSN and IoT for smart cities* (pp. 1–14). Springer.

Baird, I., Wadhaj, I., Ghaleb, B., Thomson, C., & Russell, G. (2025). Evaluating the energy costs of SHA 256 and SHA 3 (KangarooTwelve) in resource constrained IoT devices. *IoT, 6*(3), 40.

Balan, A., Balan, T., Cirstea, M., & Sandu, F. (2020). A PUF based cryptographic security solution for IoT systems on chip. *EURASIP Journal on Wireless Communications and Networking, 2020*(1), 231.

Baniya, P., Agrawal, A., Abid, K., Nath, J., Chaudhary, B. K., & Kunwar, B. (2024). The Internet of Things: Security challenges and opportunities. In *2024 3rd International Conference on Power Electronics and IoT Applications in Renewable Energy and its Control (PARC)* (pp. 153–158). IEEE.

Bassham, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Leigh, S. D., … Banks, D. L. (2010). *A statistical test suite for random and pseudorandom number generators for cryptographic applications* (NIST Special Publication 800 22). National Institute of Standards and Technology.

Bathalapalli, V. K., Mohanty, S. P., Kougianos, E., Iyer, V., & Rout, B. (2023). PUFchain 4.0: Integrating PUF based TPM in distributed ledger for security by design of IoT. In *Proceedings of the Great Lakes Symposium on VLSI 2023* (pp. 231–236). ACM.

Bayılmış, C., Ebleme, M. A., Çavuşoğlu, Ü., Küçük, K., & Sevin, A. (2022). A survey on communication protocols and performance evaluations for Internet of Things. *Digital Communications and Networks, 8*(6), 1094–1104.

Benjamin, M., & Yik, S. (2019). Precision livestock farming in swine welfare: A review for swine practitioners. *Animals, 9*(4), 133.

Bergfalck, L., & Engstrom, J. (2021). *Designing a Physical Unclonable Function for cryptographic hardware*. Linköping University.

Bhuiyan, M. N., Rahman, M. M., Billah, M. M., & Saha, D. (2021). Internet of Things (IoT): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities. *IEEE Internet of Things Journal, 8*(13), 10474–10498.

Bin Rabiah, A., Ramakrishnan, K. K., Richelson, S., Liri, E., & Kar, K. (2021). Haiku: Efficient authenticated key agreement with strong security guarantees for IoT. In *Proceedings of the 22nd International Conference on Distributed Computing and Networking* (pp. 196–205). ACM.

Bodagala, H., & H, P. (2022). Security for IoT using federated learning. In *2022 International Conference on Recent Trends in Microelectronics, Automation, Computing and Communications Systems (ICMACC)* (pp. 131–136). IEEE.

Boeckmann, L., Kietzmann, P., Lanzieri, L., Schmidt, T., & Wählisch, M. (2022). Usable security for an IoT OS: Integrating the zoo of embedded crypto components below a common API. *arXiv preprint,* arXiv:2208.09281.

Borcoci, E., & Obreja, S. (2018). Edge computing architectures – A survey on convergence of solutions. In *Proceedings of the FUTURE COMPUTING*.

Cabrera Gutiérrez, J., Castillo, E., Escobar Molero, A., Álvarez Bermejo, J. A., Morales, D. P., & Parrilla, L. (2022). Integration of hardware security modules and permissioned blockchain in industrial IoT networks. *IEEE Access, 10,* 114331–114345.

Cao, K., Liu, Y., Meng, G., & Sun, Q. (2020). An overview on edge computing research. *IEEE Access, 8,* 85714–85728.

Cao, Y., Xu, J., Wu, J., Wu, S., Huang, Z., & Zhang, K. (2023). Advances in Physical Unclonable Functions based on new technologies: A comprehensive review. *Mathematics, 12*(1), 77.

Cetintav, I., & Sandikkaya, M. T. (2025). A review of lightweight IoT authentication protocols from the perspective of security requirements, computation, communication, and hardware costs. *IEEE Access, 13,* 37703–37723.

Chaintoutis, C., Akriotou, M., Mesaritakis, C., Komnios, I., Karamitros, D., Fragkos, A., & Syvridis, D. (2019). Optical PUFs as physical root of trust for blockchain-driven applications. *IET Software, 13*(3), 182–186.

Chang, Z., Liu, S., Xiong, X., Cai, Z., & Tu, G. (2021). A survey of recent advances in edge computing powered artificial intelligence of things. *IEEE Internet of Things Journal, 8*(18), 13849–13875.

Chebbi, A., Hamza, R., Abdennadher, N., & Mendonça, F (2022). Combining FPGA and Edge-to-Cloud solutions to build AI-based self-adaptive applications.

Chiuchisan, I., & Dimian, M. (2015). Internet of Things for e Health: An approach to medical applications. In *2015 International Workshop on Computational Intelligence for Multimedia Understanding (IWCIM)* (pp. 1–5). IEEE.

Chuang, K. K. H., Chen, H. M., Wu, M. Y., Yang, E. C. S., & Hsu, C. C. H. (2021). Quantum tunneling PUF: A chip fingerprint for hardware security. In *2021 International Symposium on VLSI Technology, Systems and Applications (VLSI TSA)* (pp. 1–2). IEEE.

Chung, M. K., Kim, M. U., Han, J. W., Yang, J. S., Kim, B. J., Jo, M. S., … Yoon, J. B. (2024). Contribution of MEMS to Physical Unclonable Functions (PUFs): Random configuration of PDMS nano structure for optical PUF. In *2024 IEEE 37th International Conference on Micro Electro Mechanical Systems (MEMS)* (pp. 521–524). IEEE.

Cogniteq. (2023, August 10). *How IoT powers smart water management: Key solutions and benefits*. https://www.cogniteq.com/blog/how-iot-powers-smart-water-management-key-solutions-and-benefits

Colombier, B., Bossuet, L., Fischer, V., & Hély, D. (2017). Key reconciliation protocols for error correction of silicon PUF responses. *IEEE Transactions on Information Forensics and Security, 12*(8), 1988–2002.

Comeagă, A. M., & Marin, I. (2023). Memory management strategies for an Internet of Things system. In *2023 International Symposium on Fundamentals of Electrical Engineering (ISFEE)* (pp. 1–6). IEEE.

Conti, M., Dehghantanha, A., Franke, K., & Watson, S. (2018). Internet of Things security and forensics: Challenges and opportunities. *Future Generation Computer Systems, 78,* 544–546.

CYSEC. (2024). *Securing edge devices: The crucial role of root of trust in a connected world.* Retrieved from https://www.cysec.com/iot-edge/

Datta, M., & Raman, R. (2024). AI and ML in retail: IoT sensors and augmented reality for competitive strategies using IoT and linear regression. In *2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)* (pp. 1–5). IEEE.

Davies, R. (2015, May). *The Internet of Things: Opportunities and challenges* (EPRS Briefing No. PE 557.012). European Parliamentary Research Service. https://www.europarl.europa.eu/RegData/etudes/BRIE/2015/557012/EPRS_BRI(2015)557012_EN.pdf

Deep, S., Zheng, X., Jolfaei, A., Yu, D., Ostovari, P., & Bashir, A. K. (2022). A survey of security and privacy issues in the Internet of Things from the layered context. *Transactions on Emerging Telecommunications Technologies, 33*(6), e3935.

Deng, Z., Torim, A., Yahia, S. B., & Bahsi, H. (2025). Generative AI in intrusion detection systems for Internet of Things: A systematic literature review. *IEEE Open Journal of the Communications Society.*

Diro, A. A., & Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for Internet of Things. *Future Generation Computer Systems, 82,* 761–768.

Domínguez Bolaño, T., Campos, O., Barral, V., Escudero, C. J., & García Naya, J. A. (2022). An overview of IoT architectures, technologies, and existing open source projects. *Internet of Things, 20,* 100626.

Dragomir, D., Gheorghe, L., Costea, S., & Radovici, A. (2016). A survey on secure communication protocols for IoT systems. In *2016 International Workshop on Secure Internet of Things (SIoT)* (pp. 47–62). IEEE.

Dritsas, E., & Trigka, M. (2025). A survey on cybersecurity in IoT. *Future Internet, 17*(1), 30.

Du, C., & Zhu, S. (2012). Research on urban public safety emergency management early warning system based on technologies for the Internet of Things. *Procedia Engineering, 45,* 748–754.

Dubrova, E., Näslund, O., Degen, B., Gawell, A., & Yu, Y. (2019). CRC PUF: A machine learning attack resistant lightweight PUF construction. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (pp. 264–271). IEEE.

Dumitru, M. C., Moisescu, M. A., Pietraru, R. N., & Pop, E. (2024). Extending Internet of Things systems with fog and edge computing. In *2024 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)* (pp. 1–6). IEEE.

Ehret, A., Moore, P., Stojkov, M., & Kinsy, M. A. (2023). Hardware root of trust support for operational technology cybersecurity in critical infrastructures. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1–7). IEEE.

Ejaz, M., Kumar, T., Ylianttila, M., & Harjula, E. (2020). Performance and efficiency optimization of multi layer IoT edge architecture. In *2022 6G Wireless Summit (6G SUMMIT)* (pp. 1–5). IEEE.

Ekolama, S. M., & Ebregbe, D. (2023). Application of artificial intelligence (AI) model to mitigate security threats of Internet of Things (IoT): A review. *Journal of Engineering, Emerging Technologies and Applied Sciences, 1*(2), 88–96.

El Gharbaoui, O., Kiyadi, I., & El Boukhari, H. (2024). Evaluating AI and ML in network security: A comprehensive literature review. *Procedia Computer Science, 251,* 727–733.

El Sofany, H., El Seoud, S. A., Karam, O. H., & Bouallegue, B. (2024). Using machine learning algorithms to enhance IoT system security. *Scientific Reports, 14*(1), 12077.

Fathima, K. M. M., & Santhiyakumari, N. (2021). A survey on evolution of cloud technology and virtualization. In *2021 Third International Conference on*

Intelligent Communication Technologies and Virtual Mobile Networks (ICICV) (pp. 428–433). IEEE.

Fazeldehkordi, E., & Grønli, T. M. (2022). A survey of security architectures for edge computing based IoT. *IoT, 3*(3), 332–365.

Fedorkow, G., & Hardjono, T. (2024). Mind your roots of trust! *Authorea Preprints.*

Feng, H., Guo, S., Zhu, A., Wang, Q., & Liu, D. (2020). Energy efficient user selection and resource allocation in mobile edge computing. *Ad Hoc Networks, 107,* 102202.

Friha, O., Ferrag, M. A., Shu, L., Maglaras, L., & Wang, X. (2021). Internet of Things for the future of smart agriculture: A comprehensive survey of emerging technologies. *IEEE/CAA Journal of Automatica Sinica, 8*(4), 718–752.

Gassend, B., Clarke, D., van Dijk, M., & Devadas, S. (2002). Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (pp. 148–160). ACM.

Gebali, F., & Mamun, M. (2022). Review of physically unclonable functions (PUFs): Structures, models, and algorithms. *Frontiers in Sensors, 2,* 751748.

Georgiou, K., Xavier de Souza, S., & Eder, K. (2018). The IoT energy challenge: A software perspective. *IEEE Embedded Systems Letters, 10*(3), 53–56.

Gopika Rajan, J., & Ganesh, R. S. (2022). Hardware based data security techniques in IoT: A review. In *Proceedings of the 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)* (pp. 408–413). IEEE.

Greenvolve Project. (n.d.). *Smart environment.* Retrieved June 15, 2025, from https://greenvolve-project.eu/smart-environment/

Grzesik, P., & Mrozek, D. (2024). Combining machine learning and edge computing: Opportunities, challenges, platforms, frameworks, and use cases. *Electronics, 13*(3), 640.

Gu, C., Liu, W., Cui, Y., Hanley, N., O'Neill, M., & Lombardi, F. (2021). A flip flop based arbiter physical unclonable function (APUF) design with high entropy and uniqueness for FPGA implementation. *IEEE Transactions on Emerging Topics in Computing, 9*(4), 1853–1866.

Guleria, C., Das, K., & Sahu, A. (2021). A survey on mobile edge computing: Efficient energy management system. In *2021 Innovations in Energy Management and Renewable Resources (52042)* (pp. 1–4). IEEE.

Gupta, C., & Varshney, G. (2023). A lightweight and secure puf-based authentication and key-exchange protocol for iot devices. *arXiv preprint arXiv:2311.04078.*

Gupta, R., & Tewari, A. (2025). Artificial intelligence driven cybersecurity system for Internet of Things. *Scientific Reports, 15*(1), 12345.

Gutierrez, S. V. (2024). *Methodologies for the design, the modelling, and the quality assessment of physical unclonable functions (PUFs)* (Doctoral dissertation, Université Grenoble Alpes, France).

HaddadPajouh, H., Dehghantanha, A., Parizi, R. M., Aledhari, M., & Karimipour, H. (2021). A survey on Internet of Things security: Requirements, challenges, and solutions. *Internet of Things, 14,* 100129.

Hamadeh, H., & Tyagi, A. (2019). Physical unclonable functions (PUFs) entangled trusted computing base. In *2019 IEEE International Symposium on Smart Electronic Systems (iSES)* (pp. 177–180). IEEE.

Harris, S., & Harris, D. (2021). *Digital design and computer architecture: RISC V edition.* Morgan Kaufmann.

Hassan, R., Qamar, F., Hasan, M. K., Aman, A. H. M., & Ahmed, A. S. (2020). Internet of Things and its applications: A comprehensive survey. *Symmetry, 12*(10), 1674.

Hazra, A., Rana, P., Adhikari, M., & Amgoth, T. (2023). Fog computing for next generation Internet of Things: Fundamental, state of the art and research challenges. *Computer Science Review, 48,* 100549.

He, Y., Wang, E., Rong, Y., Cheng, Z., & Chen, H. (2025). Security of AI agents. In *2025 IEEE/ACM International Workshop on Responsible AI Engineering (RAIE)* (pp. 45–52). IEEE.

Heidari, A., & Jamali, M. A. J. (2023). Internet of Things intrusion detection systems: A comprehensive review and future directions. *Cluster Computing, 26*(6), 3753–3780.

Hemalatha, R., Amulya, G., & Lalitha, C. S. N. S. (2024). IoT enabled smart retail environments. In *2024 Second International Conference Computational and Characterization Techniques in Engineering & Sciences (IC3TES)* (pp. 1–5). IEEE.

Herder, C., Yu, M. D., Koushanfar, F., & Devadas, S. (2014). Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE, 102*(8), 1126–1141.

Huber, B., & Kandah, F. (2024). Zero Trust+: A trusted based zero trust architecture for IoT at scale. In *2024 IEEE International Conference on Consumer Electronics (ICCE)* (pp. 1–6). IEEE.

Humayun, M., Alsaqer, M. S., & Jhanjhi, N. (2022). Energy optimization for smart cities using IoT. *Applied Artificial Intelligence, 36*(1), 2037255.

Humayun, M., Jhanjhi, N., Hamid, B., & Ahmed, G. (2020). Emerging smart logistics and transportation using IoT and blockchain. *IEEE Internet of Things Magazine, 3*(2), 58–62.

Humayun, M., Tariq, N., Alfayad, M., Zakwan, M., Alwakid, G., & Assiri, M. (2024). Securing the Internet of Things in artificial intelligence era: A comprehensive survey. *IEEE Access, 12*, 25469–25490.

Hussain, M., Pal, S., Jadidi, Z., Foo, E., & Kanhere, S. (2024). Federated zero trust architecture using artificial intelligence. *IEEE Wireless Communications, 31*(2), 30–35.

IBM. (n.d.). Zero trust. Retrieved August 16, 2025, from https://www.ibm.com/think/topics/zero-trust

Imran, A., & Shah, W. (2023). AI powered cyber security for IoT: Enhancing network resilience and privacy. *TechScience Press*.

Intel. (n.d.). *Introduction to Intel FPGAs.* Retrieved August 16, 2025, from https://www.intel.com/content/www/us/en/docs/programmable/683475/19-4/introduction-to.html

Intel. (n.d.). *What is a trusted platform module?* Retrieved August 16, 2025, from https://www.intel.com/content/www/us/en/learn/what-is-a-trusted-platform-module.html

International Telecommunication Union. (2012). *Overview of the Internet of Things* (Recommendation ITU T Y.4000/Y.2060). International Telecommunication Union.

Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N. S., & Mahmoudi, C. (2018). *Fog computing conceptual model* (NIST Special Publication 500 325). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.500-325

Ishak, M. H., Mispan, M. S., Chiew, W. Y., Kamaruddin, M. R., & Korobkov, M. A. (2022). Secure lightweight obfuscated delay based physical unclonable function design on FPGA. *Bulletin of Electrical Engineering and Informatics, 11*(2), 1075–1083.

Islam, M. M., Nooruddin, S., Karray, F., & Muhammad, G. (2022). Internet of Things: Device capabilities, architectures, protocols, and smart applications in healthcare domain. *IEEE Internet of Things Journal, 10*(4), 3611–3641.

Jahangeer, A., Bazai, S. U., Aslam, S., Marjan, S., Anas, M., & Hashemi, S. H. (2023). A review on the security of IoT networks: From network layer's perspective. *IEEE Access, 11,* 71073–71087.

Jaušovec, N., & Pahor, A. (2017). *Increasing intelligence.* Academic Press.

Javed, A., Malhi, A., Kinnunen, T., & Främling, K. (2020). Scalable IoT platform for heterogeneous devices in smart environments. *IEEE Access, 8,* 211973–211985.

Jayalaxmi, P., Saha, R., Kumar, G., Kumar, N., & Kim, T. H. (2021). A taxonomy of security issues in Industrial Internet of Things: Scoping review for existing solutions, future implications, and research challenges. *IEEE Access, 9,* 25344–25359.

Jeba, N., & Rathi, S. (2021). Effective data management and real time analytics in Internet of Things. *International Journal of Cloud Computing, 10*(1 2), 112–128.

Joshi, S., Mohanty, S. P., & Kougianos, E. (2017). Everything you wanted to know about PUFs. *IEEE Potentials, 36*(6), 38–46.

Jouini, Z. C., Danger, J. L., & Bossuet, L. (2011). Performance evaluation of physically unclonable function by delay statistics. In *2011 IEEE 9th International New Circuits and Systems Conference* (pp. 482–485). IEEE.

Kahlert, T., & Giza, K. (2016). *Visual Studio Code tips & tricks* (Vol. 1). Microsoft Deutschland GmbH.

Kareem, H., & Dunaev, D. (2021). Physical unclonable functions based hardware obfuscation techniques: A state of the art. In *Proceedings of the 2021 16th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1–6). IEEE.

Kaviarasan, R., Balamurgan, G., Kalaiyarasan, R., & Reddy, Y. V. R. (2023). Effective load balancing approach in cloud computing using inspired lion optimization algorithm. *E Prime – Advances in Electrical Engineering, Electronics and Energy, 6,* 100326.

Kenaza, R., Khemane, A., Bendjenna, H., Meraoumia, A., & Laimeche, L. (2022). Internet of Things (IoT): Architecture, applications, and security challenges. In *2022 4th International Conference on Pattern Analysis and Intelligent Systems (PAIS)* (pp. 1–5). IEEE.

Khan, A., Jhanjhi, N., Abdulhabeb, G. A. A., Ray, S. K., Ghazanfar, M. A., & Humayun, M. (2025). Securing IoT devices using generative AI techniques. In *Reshaping cybersecurity with generative AI techniques* (pp. 219–264). IGI Global.

Khan, M., Hatami, M., Zhao, W., & Chen, Y. (2024). A novel trusted hardware-based scalable security framework for IoT edge devices. *Discover Internet of Things, 4*(1), 4.

Khan, U. Y., & Soomro, T. R. (2018). Applications of IoT: Mobile edge computing perspectives. *In 2018 12th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)* (pp. 1–7). IEEE.

Khan, W. Z., Rehman, M. H., Zangoti, H. M., Afzal, M. K., Armi, N., & Salah, K. (2020). Industrial Internet of Things: Recent advances, enabling technologies and open challenges. *Computers & Electrical Engineering, 81,* 106522.

Kheddar, H., Dawoud, D. W., Awad, A. I., Himeur, Y., & Khan, M. K. (2024). Reinforcement-learning-based intrusion detection in communication networks: A review. *IEEE Communications Surveys & Tutorials, 27*(4), 2420-2469.

Knowledge Center. (2024). Root of trust: Trusted environment for secure functions. SemiEngineering. https://semiengineering.com/knowledge_centers/semiconductor-security/root-of-trust/

Kolias, C., Kambourakis, G., Stavrou, A., & Gritzalis, S. (2017). DDoS in the IoT: Mirai and other botnets. *Computer, 50*(7), 80–84.

Kong, L., Tan, J., Huang, J., Chen, G., Wang, S., Jin, X., & Das, S. K. (2022). Edge computing driven Internet of Things: A survey. *ACM Computing Surveys, 55*(8), 1–41.

Kong, X., Wu, Y., Wang, H., & Xia, F. (2022). Edge computing for Internet of Everything: A survey. *IEEE Internet of Things Journal, 9*(23), 23472–23485.

Korona, M., Giermakowski, R., Biernacki, M., & Rawski, M. (2024). Lightweight strong PUF for resource constrained devices. *Electronics, 13*(2), 351.

Kovačić, M., Mutavdžija, M., & Buntak, K. (2022). E Health application, implementation and challenges: A literature review. *Business Systems Research: International Journal of the Society for Advancing Innovation and Research in Economy, 13*(1), 1–18.

Kroeger, T., Cheng, W., Guilley, S., Danger, J. L., & Karimi, N. (2022). Assessment and mitigation of power side channel based cross PUF attacks on arbiter PUFs and their derivatives. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 30*(2), 187–200.

Kujo, J. (2023). *Implementing zero trust architecture for identities and endpoints with Microsoft tools.* Microsoft Press.

Kujanov, V., & Perepelitsyn, A. (2024). A survey of FPGA integration techniques in IoT infrastructure for AI/ML tasks. In *2024 14th International Conference on Dependable Systems, Services and Technologies (DESSERT)* (pp. 1–6). IEEE.

Kulkarni, A., Niraula, A., Bhattarai, H., & Niamat, M. (2024). A zero trust architecture employing blockchain and ring oscillator physical unclonable function for Internet of Things. In *2024 IEEE International Conference on Electro Information Technology (eIT)* (pp. 508–513). IEEE.

Kultan, J., Meruyert, S., Danara, T., & Nassipzhan, D. (2025). Application of computer vision methods for information security. *R&E SOURCE,* 161–177.

Laghari, A. A., Wu, K., Laghari, R. A., Ali, M., & Khan, A. A. (2021). A review and state of art of Internet of Things (IoT). *Archives of Computational Methods in Engineering, 29*(3), 1395–1413.

Lata, K., & Cenkeramaddi, L. R. (2023). FPGA based PUF designs: A comprehensive review and comparative analysis. *Cryptography, 7*(4), 55.

Lee, I., & Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons, 58*(4), 431–440.

Lee, U., & Park, C. (2020). SofTEE: Software based trusted execution environment for user applications. *IEEE Access, 8,* 121874–121888.

Lee, B., Lee, I. G., & Kim, M. (2022). Design and implementation of secure cryptographic system on chip for Internet of Things. *IEEE Access, 10,* 18730–18742.

Lee, S., Oh, M. K., Kang, Y., & Choi, D. (2019). RC PUF: A low cost and an easy to design PUF for resource constrained IoT devices. In *International Workshop on Information Security Applications* (pp. 275–285). Springer.

Leong, W. (2025). Internet of Things for enhancing public safety, disaster response, and emergency management. *Engineering Proceedings, 92*(1), 61.

Li, F., & Li, T. (2022). Intelligent logistics enterprise management based on the Internet of Things. *Mathematical Problems in Engineering, 2022*(1), 1621082.

Li, H., Cao, W., Wang, C., Zhu, X., Liao, G., & He, Z. (2023). FOM CDS PUF: A novel configurable dual state strong PUF based on feedback obfuscation mechanism against modeling attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 106*(10), 1311–1321.

Li, H., Ge, L., & Tian, L. (2024). Survey: Federated learning data security and privacy preserving in edge Internet of Things. *Artificial Intelligence Review, 57*(5), 130.

Li, Y., Zuo, Y., Song, H., & Lv, Z. (2022). Deep learning in security of Internet of Things. *IEEE Internet of Things Journal, 9*(22), 22133–22146.

Libelium. (n.d.). *Top 50 IoT sensor applications ranking*. Retrieved June 15, 2025, from https://www.libelium.com/libeliumworld/top-50-iot-sensor-applications-ranking/

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., & Zhao, W. (2017). A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal, 4*(5), 1125–1142.

Lin, L., Liao, X., Jin, H., & Li, P. (2019). Computation offloading toward edge computing. *Proceedings of the IEEE, 107*(8), 1584–1607.

Lin, R., Xie, T., Luo, S., Zhang, X., Xiao, Y., Moran, B., & Zukerman, M. (2022). Energy efficient computation offloading in collaborative edge computing. *IEEE Internet of Things Journal, 9*(21), 21305–21322.

Lingishetty, S. K., Moharir, C., & Kumar, M. (2025). AI based encryption techniques for securing data transmission in telecommunication systems. *Journal of Network and Computer Applications, 225,* 103876.

Liu, D., Yan, Z., Ding, W., & Atiquzzaman, M. (2019). A survey on secure data analytics in edge computing. *IEEE Internet of Things Journal, 6*(3), 4946–4967.

Liu, S., Yu, Y., Guo, L., Yeoh, P. L., Vucetic, B., & Li, Y. (2023). Adaptive delay energy balanced partial offloading strategy in mobile edge computing networks. *Digital Communications and Networks, 9*(6), 1310–1318.

Liu, Y., Qu, H., Chen, S., et al. (2025). Energy efficient task scheduling for heterogeneous multicore processors in edge computing. *Scientific Reports, 15,* 11819.

Liu, Y., Xie, Y., Bao, C., & Srivastava, A. (2017). A combined optimization theoretic and side channel approach for attacking strong physical unclonable functions. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 26*(1), 73–81.

Lloret, J., Tomas, J., Canovas, A., & Parra, L. (2016). An integrated IoT architecture for smart metering. *IEEE Communications Magazine, 54*(12), 50–57.

Lombardi, M., Pascale, F., & Santaniello, D. (2021). Internet of Things: A general overview between architectures, protocols and applications. *Information, 12*(2), 87.

López Escobar, J. J., Díaz Redondo, R. P., & Gil Castiñeira, F. (2022). In depth analysis and open challenges of mist computing. *Journal of Cloud Computing, 11*(1), 81.

Lu, S., Lu, J., An, K., Wang, X., & He, Q. (2023). Edge computing on IoT for machine signal processing and fault diagnosis: A review. *IEEE Internet of Things Journal, 10*(13), 11093–11116.

Lu, X., Hong, L., & Sengupta, K. (2018). CMOS optical PUFs using noise immune process sensitive photonic crystals incorporating passive variations for robustness. *IEEE Journal of Solid State Circuits, 53*(10), 2709–2721.

Ma, D., Lan, G., Hassan, M., Hu, W., & Das, S. K. (2020). Sensing, computing, and communications for energy harvesting IoTs: A survey. *IEEE Communications Surveys & Tutorials, 22*(2), 1222–1250.

Ma, X., Wang, P., Li, G., & Zhou, Z. (2023). Machine learning attacks resistant strong PUF design utilizing response obfuscates challenge with lower hardware overhead. *Microelectronics Journal, 142,* 105977.

Magyari, A., & Chen, Y. (2023). Integrating Lorenz hyperchaotic encryption with ring oscillator physically unclonable functions (RO PUFs) for high throughput Internet of Things (IoT) applications. *Electronics, 12*(23), 4929.

Mahmood, K., Shamshad, S., Saleem, M. A., & Kharel, R. (2023). Blockchain and PUF based secure key establishment protocol for cross domain digital twins in Industrial Internet of Things architecture. *Journal of Advanced Research, 62*(5), 194–210.

Mahmood, Y., Kama, N., Azmi, A., & Ya'acob, S. (2020). An IoT based home automation integrated approach: Impact on society in sustainable development perspective. *International Journal of Advanced Computer Science and Applications, 11*(1), 240–250.

Maiti, A., Gunreddy, V., & Schaumont, P. (2012). A systematic method to evaluate and compare the performance of physical unclonable functions. In P. Athanas, D. Pnevmatikatos, & N. Sklavos (Eds.), *Embedded systems design with FPGAs* (pp. 245–267). Springer.

Makhdoom, I., Abolhasan, M., Ni, W., & Lipman, J. (2019). Blockchain's adoption in IoT: The challenges, and a way forward. *Journal of Network and Computer Applications, 125,* 251–279.

Malhotra, P., Singh, Y., Anand, P., Bangotra, D. K., Singh, P. K., & Hong, W. C. (2021). Internet of Things: Evolution, concerns and security challenges. *Sensors, 21*(5), 1809.

Manan, A. (2018). Efficient 16 nm SRAM design for FPGA's. In *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)* (pp. 457–461). IEEE.

Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A. T., Siddiqa, A., & Yaqoob, I. (2017). Big IoT data analytics: Architecture, opportunities, and open research challenges. *IEEE Access, 5,* 5247–5261.

Martínez Rodríguez, M. C., Rojas Muñoz, L. F., Camacho Ruiz, E., Sánchez Solano, S., & Brox, P. (2022). Efficient RO PUF for generation of identifiers and keys in resource constrained embedded systems. *Cryptography, 6*(4), 51.

Marton, K., & Suciu, A. (2015). On the interpretation of results from the NIST statistical test suite. *Science and Technology, 18*(1), 18–32.

Mavrovouniotis, S., & Ganley, M. (2013). Hardware security modules. *In Secure smart embedded devices, platforms and applications* (pp. 383-405). New York, NY: Springer New York.

Mazhar, T., Talpur, D. B., Al Shloul, T., Ghadi, Y. Y., Haq, I., Ullah, I., … Hamam, H. (2023). Analysis of IoT security challenges and its solutions using artificial intelligence. *Brain Sciences, 13*(4), 683.

Mendiboure, L., Chalouf, M. A., & Krief, F. (2019). Edge computing based applications in vehicular environments: Comparative study and main issues. *Journal of Computer Science and Technology, 34*(4), 869–886.

Miller, T., Durlik, I., Kostecka, E., Kozlovska, P., Łobodzińska, A., Sokołowska, S., & Nowy, A. (2025). Integrating artificial intelligence agents with the Internet of Things for enhanced environmental monitoring: Applications in water quality and climate data. *Electronics, 14*(4), 696.

Mishra, S., Sagban, R., Yakoob, A., & Gandhi, N. (2021). Swarm intelligence in anomaly detection systems: An overview. *International Journal of Computers and Applications, 43*(2), 109–118.

Mishra, S. R., Shanmugam, B., Yeo, K. C., & Thennadil, S. (2025). SDN enabled IoT security frameworks – A review of existing challenges. *Technologies, 13*(3), 121.

Mohanty, S. P., Plusquellic, J., Rose, G. S., Zhang, W., & Michael, M. K. (2021). Introduction to the special issue on hardware assisted security for emerging Internet of Things. *ACM Journal on Emerging Technologies in Computing Systems, 18*(1), 1–3.

Moraes, P. F., & Martins, J. S. (2019, July). A pub/sub SDN-integrated framework for IoT traffic orchestration. In *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems* (pp. 1-9).

Mosenia, A., & Jha, N. K. (2017). A comprehensive study of security of Internet of Things. *IEEE Transactions on Emerging Topics in Computing, 5*(4), 586–602.

Mukherjee, A., De, D., Ghosh, S. K., & Buyya, R. (2021). Introduction to mobile edge computing. In *Mobile Edge Computing* (pp. 3-19). Cham: Springer International Publishing.

Mukhtar, N., Mehrabi, A., Kong, Y., & Anjum, A. (2023). Edge enhanced deep learning system for IoT edge device security analytics. *Concurrency and Computation: Practice and Experience, 35*(13), e6764.

Muniswamaiah, M., Agerwala, T., & Tappert, C. C. (2021). Fog computing and the Internet of Things (IoT): A review. In *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)* (pp. 10–12). IEEE.

Muñoz, A. (2022). Secure mobile agents on embedded boards: A TPM based solution. In Proceedings of the 17th International Conference on Availability, *Reliability and Security* (pp. 1–7). ACM.

Muthanna, M. S. A., Alkanhel, R., Muthanna, A., Rafiq, A., & Abdullah, W. A. M. (2022). Towards SDN enabled, intelligent intrusion detection system for Internet of Things (IoT). *IEEE Access, 10,* 22756–22768.

National Institute of Standards and Technology. (n.d.). *Guide to the statistical tests*. https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software/guide-to-the-statistical-tests

National Instruments. (2025, February 14). *FPGA fundamentals: Basics of field programmable gate arrays.* Retrieved May 4, 2025, from https://www.ni.com/en/shop/fpga.html

Neethirajan, S. (2017). Recent advances in wearable sensors for animal health management. *Sensing and Bio Sensing Research, 12,* 15–29.

Neethirajan, S., Ragavan, K. V., & Weng, X. (2018). Agro defense: Biosensors for food from healthy crops and animals. *Trends in Food Science & Technology, 73*, 25–44.

Nguyen, D. C., Ding, M., Pathirana, P. N., Seneviratne, A., Li, J., & Poor, H. V. (2021). Federated learning for Internet of Things: A comprehensive survey. *IEEE Communications Surveys & Tutorials, 23*(3), 1622–1658.

Noaman, M., Khan, M. S., Abrar, M. F., Ali, S., Alvi, A., & Saleem, M. A. (2022). Challenges in integration of heterogeneous Internet of Things. *Scientific Programming, 2022*(1), 8626882.

Ojo, M. O., Viola, I., Baratta, M., & Giordano, S. (2021). Practical experiences of a smart livestock location monitoring system leveraging GNSS, LoRaWAN and cloud services. *Sensors, 22*(1), 273.

Oye, E., Peace, P., & Owen, J. (2024). The role of natural language processing in cybersecurity. *Journal of Cybersecurity Research, 15*(2), 45–62.

Pandey, S., & Bhushan, B. (2024). Recent lightweight cryptography (LWC) based security advances for resource constrained IoT networks. *Wireless Networks, 30*(4), 2987–3026.

Pang, H., & Tan, K. L. (2004). Authenticating query results in edge computing. In *Proceedings of the 20th International Conference on Data Engineering (ICDE)* (pp. 560–571). IEEE.

Patel, P., Ali, M. I., & Sheth, A. (2017). On using the intelligent edge for IoT analytics. *IEEE Intelligent Systems, 32*(5), 64–69.

Paul, E. M., Mmaduekwe, U., Kessie, J. D., & Dolapo, M. (2024). Zero trust architecture and AI: A synergistic approach to next generation cybersecurity frameworks. *International Journal of Science and Research Archive, 13*(2), 4159–4169.

Perifanis, N. A., & Kitsios, F. (2022). Edge and fog computing business value streams through IoT solutions: A literature review for strategic implementation. *Information, 13*(9), 427.

Pescosolido, L., Berta, R., Scalise, L., Revel, G. M., De Gloria, A., & Orlandi, G. (2016). An IoT inspired cloud based web service architecture for e Health applications. In *2016 IEEE International Smart Cities Conference (ISC2)* (pp. 1–4). IEEE.

Pillai, A. S., Chandraprasad, G. S., Khwaja, A. S., & Anpalagan, A. (2021). A service oriented IoT architecture for disaster preparedness and forecasting system. *Internet of Things, 14,* 100076.

Plusquellic, J. (2022). Shift register, reconvergent fanout (SIRF) PUF implementation on an FPGA. *Cryptography, 6*(4), 59.

Plusquellic, J., Howard, J., MacKinnon, R., Hoffman, K., Tsiropoulou, E. E., & Chan, C. (2024). A physical layer analysis of entropy in delay based PUFs implemented on FPGAs. *arXiv preprint, arXiv:2409.00825.*

Prawiyogi, A. G., Purnama, S., & Meria, L. (2022). Smart cities using machine learning and intelligent applications. *International Transactions on Artificial Intelligence, 1*(1), 102–116.

Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., & Rana, O. (2019). Fog computing for the Internet of Things: A survey. *ACM Transactions on Internet Technology, 19*(2), 1–41.

Qazi, F., Kwak, D., Khan, F. G., Ali, F., & Khan, S. U. (2024). Service level agreement in cloud computing: Taxonomy, prospects, and challenges. *Internet of Things, 25,* 101126.

Radhakrishnan, V., & Wu, W. (2018). IoT technology for smart water system. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (pp. 1491–1496). IEEE.

Raj, P., & David, P. E. (2020). *The digital twin paradigm for smarter systems and environments: The industry use cases* (Vol. 117). Academic Press.

Raji, M. A., Olodo, H. B., Oke, T. T., Addy, W. A., Ofodile, O. C., & Oyewole, A. T. (2024). Real time data analytics in retail: A review of USA and global practices. *GSC Advanced Research and Reviews, 18*(3), 059–065.

Ram, S. K., Sahoo, S. R., Das, B. B., Mahapatra, K., & Mohanty, S. P. (2022). Securing things: A novel CRO applicable in PUF and recycled IC detection (Version 1). Research Square.

Rao, P. M., & Deebak, B. D. (2023). Security and privacy issues in smart cities/industries: Technologies, applications, and challenges. *Journal of Ambient Intelligence and Humanized Computing, 14*(8), 10517–10553.

Rehman, A., Saba, T., Kashif, M., Fati, S. M., Bahaj, S. A., & Chaudhry, H. (2022). A revisit of Internet of Things technologies for monitoring and control strategies in smart agriculture. *Agronomy, 12*(1), 127.

Riesgo, T., Torroja, Y., & De la Torre, E. (2002). Design methodologies based on hardware description languages. *IEEE Transactions on Industrial Electronics, 46*(1), 3–12.

Rojas Muñoz, L. F., Sánchez Solano, S., Martínez Rodríguez, M. C., Camacho Ruiz, E., Navarro Torrero, P., Karmakar, A., … Brox, P. (2024). Cryptographic security through a hardware root of trust. In *International Symposium on Applied Reconfigurable Computing* (pp. 106–119). Springer.

Roy, A., Kokila, J., Ramasubramanian, N., & Begum, B. S. (2024). OTK based PUF CRP obfuscation for IoT device authentication. *Microelectronics Journal, 144,* 106070.

Rozlomii, I., Yarmilko, A., & Naumenko, S. (2024). Data security of IoT devices with limited resources: Challenges and potential solutions. *Information Security, 3666,* 85–96.

Rudra, B. (2020). Impact of Blockchain for internet of Things Security. *Cryptocurrencies and Blockchain Technology Applications,* 99-127.

Rührmair, U., Sölter, J., & Sehnke, F. (2009). On the foundations of physical unclonable functions. *Cryptology ePrint Archive.*

Rührmair, U. (2016). On the security of PUF protocols under bad PUFs and PUFs-inside-PUFs attacks. *Cryptology ePrint Archive.*

Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., & Schmidhuber, J. (2010). Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (pp. 237–249). ACM.

Rupanetti, D., & Kaabouch, N. (2024). Combining edge computing assisted Internet of Things security with artificial intelligence: Applications, challenges, and opportunities. *Applied Sciences, 14*(16), 7104.

Ruth, J. A., Vijayalakshmi, G. V., Visalakshi, P., Uma, R., & Meenakshi, A. (Eds.). (2024). *Innovative machine learning applications for cryptography*. IGI Global.

Saddi, V. R., Gopal, S. K., Mohammed, A. S., Dhanasekaran, S., & Naruka, M. S. (2024). Examine the role of generative AI in enhancing threat intelligence and

cyber security measures. In *2024 2nd International Conference on Disruptive Technologies (ICDT)* (pp. 537–542). IEEE.

Sasaki, Y. (2022). A survey on IoT big data analytic systems: Current and future. *IEEE Internet of Things Journal, 9*(2), 1024–1036.

Sattari, A., Ehsani, R., Leppänen, T., Pirttikangas, S., & Riekki, J. (2020). Edge supported microservice based resource discovery for mist computing. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)* (pp. 462–468). IEEE.

Schaub, A., Danger, J. L., Rioul, O., & Guilley, S. (2020). The big picture of delay PUF dependability. In *2020 European Conference on Circuit Theory and Design (ECCTD)* (pp. 1–4). IEEE.

Schrijen, G. J., & Garlati, C. (2018). Physical unclonable functions to the rescue. In *Proceedings of the Embedded World.*

Sefati, S., Mousavinasab, M., & Zareh Farkhady, R. (2022). Load balancing in cloud computing environment using the grey wolf optimization algorithm based on the reliability: Performance evaluation. *The Journal of Supercomputing, 78*(1), 18–42.

Shafiq, D. A., Jhanjhi, N. Z., & Abdullah, A. (2022). Load balancing techniques in cloud computing environment: A review. *Journal of King Saud University – Computer and Information Sciences, 34*(7), 3910–3933.

Shah, N., Chatterjee, D., Sapui, B., Mukhopadhyay, D., & Basu, A. (2021). Introducing recurrence in strong PUFs for enhanced machine learning attack resistance. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 11*(2), 319–332.

Shah, S. W., Syed, N. F., Shaghaghi, A., Anwar, A., Baig, Z., & Doss, R. (2021). LCDA: Lightweight continuous device to device authentication for a zero trust architecture (ZTA). *Computers & Security, 108,* 102351.

Shahraki, A., Geitle, M., & Haugen, Ø. (2020). A comparative node evaluation model for highly heterogeneous massive-scale Internet of Things-Mist networks. *Transactions on Emerging Telecommunications Technologies, 31*(12), e3924.

Shaukat, K., Luo, S., Varadharajan, V., Hameed, I. A., & Xu, M. (2020). A survey on machine learning techniques for cyber security in the last decade. *IEEE Access, 8,* 222310–222354.

Sheik, S. A., & Muniyadi, A. P. (2023). Secure authentication schemes in cloud computing with glimpse of artificial neural networks: A review. *Cyber Security and Applications, 1,* 100002.

Shi, W., Pallis, G., & Xu, Z. (2019). Edge computing [Scanning the issue]. *Proceedings of the IEEE, 107*(8), 1474–1481.

Shinde, N. K., Seth, A., & Kadam, P. (2023). Exploring the synergies: A comprehensive survey of blockchain integration with artificial intelligence, machine learning, and IoT for diverse applications. In *Machine learning and optimization for engineering design* (pp. 85–119). Springer.

Singh, P., Basit, A., Kumar, N. C., & Venkaiah, V. C. (2019). Towards a hybrid Public Key Infrastructure (PKI): A review. *Cryptology ePrint Archive.*

Singh, R. P., Barhaiva, H., Sahu, S., Kumar, J., & Rathore, D. S. (2024). A review on edge computing: Security considerations and challenges. In *Proceedings of the 2024 First International Conference on Data, Computation and Communication (ICDCC)* (pp. 605–609). IEEE.

Sodiya, E. O., Umoga, U. J., Obaigbena, A., Jacks, B. S., Ugwuanyi, E. D., Daraojimba, A. I., & Lottu, O. A. (2024). Current state and prospects of edge computing within the Internet of Things (IoT) ecosystem. *International Journal of Science and Research Archive, 11*(1), 1863–1873.

Song, Y., Yu, F. R., Zhou, L., Yang, X., & He, Z. (2020). Applications of the Internet of Things (IoT) in smart logistics: A comprehensive survey. *IEEE Internet of Things Journal, 8*(6), 4250–4274.

Srirama, S. N. (2024). A decade of research in fog computing: Relevance, challenges, and future directions. *Software: Practice and Experience, 54*(1), 3–23.

Statista. (2025). Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2034, by vertical. Retrieved June 15, 2025, from https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/

Stolojescu Crisan, C., Crisan, C., & Butunoi, B. P. (2021). An IoT based smart home automation system. *Sensors, 21*(11), 3784.

Su, H. (2021). *Novel design in mixed signal and machine learning resilient architecture physical unclonable functions* (Doctoral dissertation, University of Southampton).

Syed, N. F., Shah, S. W., Shaghaghi, A., Anwar, A., Baig, Z., & Doss, R. (2022). Zero trust architecture (ZTA): A comprehensive survey. *IEEE Access, 10,* 57143–57179.

Szpilko, D., Naharro, F. J., Lăzăroiu, G., Nica, E., & de la Torre Gallegos, A. (2023). Artificial intelligence in the smart city – A literature review. *Sustainable Cities and Society, 99,* 104904.

Talpur, D. B. (2023). Analysis of IoT security challenges and its solutions using artificial intelligence. *Electronics, 13*(24), 4965.

Tariq, U., Ahmed, I., Bashir, A. K., & Shaukat, K. (2023). A critical cybersecurity analysis and future research directions for the Internet of Things: A comprehensive review. *Sensors, 23*(8), 4117.

Tian, L., & Zhong, X. (2022). A case study of edge computing implementations: Multi access edge computing, fog computing and cloudlet. *Journal of Computing and Information Technology, 30*(3), 139–159.

Torğul, B., Şağbanşua, L., & Balo, F. B. (2016). Internet of Things: A survey. *International Journal of Applied Mathematics Electronics and Computers,* (Special Issue 1), 104–110.

Ullah, A., Anwar, S. M., Li, J., Nadeem, L., Mahmood, T., Rehman, A., & Saba, T. (2024). Smart cities: The role of Internet of Things and machine learning in realizing a data centric smart environment. *Complex & Intelligent Systems, 10*(1), 1607–1637.

Uprety, A., & Rawat, D. B. (2020). Reinforcement learning for IoT security: A comprehensive survey. *IEEE Internet of Things Journal, 8*(11), 8693–8706.

ur Rehman, M. H., Yaqoob, I., Salah, K., Imran, M., Jayaraman, P. P., & Perera, C. (2019). The role of big data analytics in industrial Internet of Things. *Future Generation Computer Systems*, *99*, 247-259.

V7 Labs. (n.d.). *Performance metrics in machine learning.* Retrieved August 16, 2025, from https://www.v7labs.com/blog/performance-metrics-in-machine-learning

Vailshery, L. S. (2025). Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2034, by vertical. Statista. Retrieved June 15, 2025, from https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/

Van den Abeele, F., Hoebeke, J., Moerman, I., & Demeester, P. (2015). Integration of heterogeneous devices and communication models via the cloud in the constrained Internet of Things. *International Journal of Distributed Sensor Networks, 11*(10).

Van Leemput, D., Sabovic, A., Hammoud, K., Famaey, J., Pollin, S., & De Poorter, E. (2023). Energy harvesting for wireless IoT use cases: A generic feasibility model and tradeoff study. *IEEE Internet of Things Journal, 10*(17), 15025–15043.

Vuyyuru, M., Annapurna, P., Babu, K. G., & Ratnam, A. S. K. (2012). An overview of cloud computing technology. *International Journal of Soft Computing and Engineering, 2*(3), 244–247.

Wang, Y., Zhang, G., Mei, X., & Gu, C. (2024). A high-reliability, non-CRP-discard arbiter PUF based on delay difference quantization. *IEEE Transactions on Circuits and Systems I: Regular Papers, 72*(2), 573-585.

Wang, Y., Xi, X., & Orshansky, M. (2020). Lattice PUF: A strong physical unclonable function provably secure against machine learning attacks. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (pp. 273–283). IEEE.

Wanga, K., Shi, J., Lai, W., He, Q., Xu, J., Ni, Z., … Yang, D. (2024). All silicon multidimensionally encoded optical physical unclonable functions for integrated circuit anti counterfeiting. *Nature Communications, 15*(1), 3203.

Wena, F. (2024). The new trend of the integration of artificial intelligence and blockchain in network security. *Academic Journal of Computing & Information Science, 7*(3), 38–42.

Whaiduzzaman, M., Barros, A., Chanda, M., Barman, S., Sultana, T., Rahman, M. S., … Fidge, C. (2022). A review of emerging technologies for IoT based smart cities. *Sensors, 22*(23), 9271.

Wu, L., Hu, Y., Zhang, K., Li, W., Xu, X., & Chang, W. (2022). Flam PUF: A response feedback based lightweight anti machine learning attack PUF. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 41*(11), 4433–4444.

Xiao, Y., Jia, Y., Liu, C., Cheng, X., Yu, J., & Lv, W. (2019). Edge computing security: State of the art and challenges. *Proceedings of the IEEE, 107*(8), 1608–1631.

Xu, C., Jiang, S., Luo, G., Sun, G., An, N., Huang, G., & Liu, X. (2020). The case for FPGA based edge computing. *IEEE Transactions on Mobile Computing, 21*(7), 2610–2619.

Xu, J., Palanisamy, B., Ludwig, H., & Wang, Q. (2017). Zenith: Utility aware resource allocation for edge computing. In *2017 IEEE International Conference on Edge Computing (EDGE)* (pp. 47–54). IEEE.

Xu, Y., Lao, Y., Liu, W., Zhang, Z., You, X., & Zhang, C. (2020). Mathematical modeling analysis of strong physical unclonable functions. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 39*(11), 4426–4438.

Yang, K., Liu, J., Chen, J., & Vasilakos, A. V. (2017). A survey on security and privacy issues in Internet of Things. *IEEE Internet of Things Journal, 4*(5), 1250–1258.

Yaqoob, I., Salah, K., Imran, M., Jayaraman, P. P., & Perera, C. (2019). The role of big data analytics in industrial Internet of Things. *arXiv.*

Yavuz, S., Naroska, E., & Daniel, K. (2024). Vulnerabilities and challenges in the development of PUF based authentication protocols on FPGAs: A brief review. In *2024 IEEE Conference on Dependable and Secure Computing (DSC)* (pp. 58–65). IEEE.

Yu, M. D., Sowell, R., Singh, A., M'Raïhi, D., & Devadas, S. (2012). Performance metrics and empirical results of a PUF cryptographic key generation ASIC. In *2012 IEEE International Symposium on Hardware Oriented Security and Trust* (pp. 108–115). IEEE.

Yuyuan, L., & Xueqiang, S. (2024). A highly reliable reconfigurable SRAM PUF based on error correction codes and capacitor preselection. *Authorea Preprints.*

Zaheeruddin, & Gupta, H. (2020). Foundation of IoT: An overview. In *Internet of Things (IoT) concepts and applications* (pp. 3–24). Springer.

Zahoor, A., Mahmood, K., Shamshad, S., & Saleem, M. A. (2023). An access control scheme in IoT enabled smart grid systems using blockchain and PUF. *Internet of Things, 22*(2), 100787.

Zenatix. (n.d.). *Smart energy meter using IoT: The future of energy analytics*. Retrieved June 15, 2025, from https://zenatix.com/smart-energy-meter-using-iot-the-future-of-energy-analytics/

Zenodo. (n.d.). Smart environment datasets. Retrieved June 15, 2025, from https://zenodo.org/records/10501101

Zhang, J. L., Wu, Q., Ding, Y. P., Lv, Y. Q., Zhou, Q., Xia, Z. H., … Wang, X. W. (2023). Techniques for design and implementation of an FPGA specific physical unclonable function. *Journal of Computer Science and Technology, 31*(1), 124–136.

Zhang, J., & Shen, C. (2020). Set based obfuscation for strong PUFs against machine learning attacks. *IEEE Transactions on Circuits and Systems I: Regular Papers, 68*(1), 288–300.

Zhang, X., Cao, Z., & Dong, W. (2020). Overview of edge computing in the agricultural Internet of Things: Key technologies, applications, challenges. *IEEE Access, 8,* 141748–141761.

Zhang, Y., Deng, R. H., & Liu, J. K. (2019). Cloud computing security: Foundations and challenges. *IEEE Cloud Computing, 6*(5), 68–74.

Zhang, Y., Deng, R. H., & Weng, J. (2018). Secure and efficient data communication in Internet of Things: Principles and applications. *IEEE Internet of Things Journal, 5*(3), 2130–2145.

Zhou, Y., Zhang, D., & Xiong, N. (2017). Post cloud computing paradigms: A survey and comparison. *Tsinghua Science and Technology, 22*(6), 714–732. https://doi.org/10.23919/TST.2017.8195348

Zhou, K., Liang, H., Jiang, Y., Huang, Z., Jiang, C., & Lu, Y. (2019). FPGA-based RO PUF with low overhead and high stability. *Electronics Letters, 55*(9), 510–513.

Zhu, B., Jiang, X., Huang, K., & Yu, M. (2023). A response feedback based strong PUF with improved strict avalanche criterion and reliability. *Sensors, 24*(1), 93.

Zhuang, Y., Mursi, K. T., & Gaoxiang, L. (2021). A challenge obfuscating interface for arbiter PUF variants against machine learning attacks. *arXiv preprint arXiv:2103.12935.*

Zimmer, V., & Krau, M. (2016, August). Establishing the root of trust. UEFI Forum. https://uefi.org/sites/default/files/resources/UEFI%20RoT%20white%20paper_Final%208%208%2016%20%28003%29.pdf.

Zubaydi, H. D., Varga, P., & Molnár, S. (2023). Leveraging blockchain technology for ensuring security and privacy aspects in Internet of Things: A systematic literature review. *Sensors, 23*(2), 788.

Zyrianoff, I., Heideker, A., Silva, D., & Kamienski, C. (2018). Scalability of an Internet of Things platform for smart water management for agriculture. In *2018 23rd Conference of Open Innovations Association (FRUCT)* (pp. 432–439). IEEE.

Żyliński, M., Nassibi, A., Rakhmatulin, I., Malik, A., Papavassiliou, C. M., & Mandic, D. P. (2024). Deployment of artificial intelligence models on edge devices: A tutorial brief. *IEEE Transactions on Circuits and Systems II: Express Briefs, 71*(3), 1738–1743.

APPENDIX I

I. VHDL MODEL OF ARBITER PUF

I.1 Top module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use STD.TEXTIO.ALL;
use ieee.std_logic_textio.all;
entity TOP_ArbiterPUF is
   generic (
      N : integer := 16;  -- Number of challenge bits
                  OUTPUT_FILE_NAME : string := "C:/Users/Admin/Desktop/Intel
Softwares/My Work/10 Mar 2025/CRP.txt"  -- Output file name
   );
   Port (
      --challenge : in  STD_LOGIC_VECTOR(N-1 downto 0);
      clk    : in  STD_LOGIC;
                  reset   : in  STD_LOGIC;
                  enable          : in  STD_LOGIC;
                  chal    : out STD_LOGIC_VECTOR(N-1 downto 0);
                  resp    : out STD_LOGIC
   );
end TOP_ArbiterPUF;

architecture Structural of TOP_ArbiterPUF is
signal challenge : STD_LOGIC_VECTOR(N-1 downto 0);
signal response : STD_LOGIC;
--signal reg: std_logic_vector(15 downto 0);
```

```
--signal dclk: std_logic;
component ArbiterPUF
   Generic (
      N : integer := 16  -- Number of challenge bits
   );
   Port (
      challenge : in  STD_LOGIC_VECTOR(N-1 downto 0);
      clk       : in  STD_LOGIC;
      response  : out STD_LOGIC
   );
end component;
component PRBS
   generic (N : integer := 16);
   port (
      clk    : in  STD_LOGIC;
      rst    : in  STD_LOGIC;
      enable : in  STD_LOGIC;
      rnd    : out STD_LOGIC_VECTOR(N-1 downto 0)
   );
end component;
begin
--process(clk)
--begin
--        if (reset = '0') then
--                reg <= X"0000";
--                dclk<= '0';
--        elsif rising_edge(clk) then
--                if (reg = X"FFFF") then
--                        dclk <= not clk;
--                end if;
--                        reg <= reg + '1';
--        end if;
--end process;
PRBS_INST: PRBS generic map (n => 16) port map (clk, reset, enable, challenge);
```

```
ArbiterPUF_INST0: ArbiterPUF generic map (n => 16) port map(challenge, clk,
response);
chal <= challenge;
resp <= response;
process (challenge, response)
file output_file : text open write_mode is OUTPUT_FILE_NAME;
variable line_out : line;   -- Line buffer for writing
begin
        write(line_out, challenge);
        write(line_out, string'(","));
        write(line_out, response);
  writeline(output_file, line_out);
 end process;
end Structural;
```

I.2 PRBS generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PRBS is --PRBS generator
   generic (
      N : integer := 16  -- Set N to the desired bit-width
   );
   port (
      clk    : in  STD_LOGIC;
      rst    : in  STD_LOGIC;
      enable : in  STD_LOGIC;
      rnd    : out STD_LOGIC_VECTOR(N-1 downto 0) --output of prbs
   );
end entity PRBS;
architecture Behavioral of PRBS is
         signal seed : STD_LOGIC_VECTOR(N-1 downto 0) := X"F97C";
```

```
        -- seed is the Initial value for the LFSR
    signal lfsr : STD_LOGIC_VECTOR(N-1 downto 0);
begin
process(clk, rst)
    begin
        if rst = '0' then
            lfsr <= seed;  -- Load the seed value on reset
        elsif rising_edge(clk) then
            if enable = '1' then
                -- XOR feedback based on a primitive polynomial (example for N=8: x^8 +
x^6 + x^5 + x^4 + 1)
                lfsr <= lfsr(N-2 downto 0) &
                (lfsr(N-1) xor lfsr(5) xor lfsr(4) xor lfsr(3) xor lfsr(0));
            end if;
        end if;
    end process;
  rnd <= lfsr;
end architecture Behavioral;
```

I.3 Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity MUX2x1 is
    Port (
        a, b   : in  STD_LOGIC;
        sel    : in  STD_LOGIC;
        outMux : out STD_LOGIC
    );
end MUX2x1;
architecture Behavioral of MUX2x1 is
begin
```

```
with sel select
        outMux <= a when '0',
                  b when '1',
                  'X' when others;
end Behavioral;
```

I.4 Arbiter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Arbiter is
    Port (
        clk   : in  STD_LOGIC;
        d_in  : in  STD_LOGIC;
        q_out : out STD_LOGIC
    );
end Arbiter;
architecture Behavioral of Arbiter is
signal reg : std_logic:='0';
begin
    process(clk)
    begin
        if rising_edge(clk) then
            reg <= d_in;
                    else
            reg <= reg;
        end if;
    end process;
q_out <= reg;
end Behavioral;
```

I.5 Arbiter PUF

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use STD.TEXTIO.ALL;
use ieee.std_logic_textio.all;
entity ArbiterPUF is
    Generic (
        N : integer := 16  -- Number of challenge bits
    );
    Port (
        challenge : in  STD_LOGIC_VECTOR(N-1 downto 0);
        clk       : in  STD_LOGIC;
        response  : out STD_LOGIC
    );
end ArbiterPUF;
architecture Structural of ArbiterPUF is
    component MUX2x1
        Port (
            a, b   : in  STD_LOGIC;
            sel    : in  STD_LOGIC;
            outMux : out STD_LOGIC
        );
    end component;
    component Arbiter
        Port (
            clk   : in  STD_LOGIC;
            d_in  : in  STD_LOGIC;
            q_out : out STD_LOGIC
        );
    end component;
    signal path_A, path_B : STD_LOGIC_VECTOR(N downto 0);
```

```
    signal arbiter_out    : STD_LOGIC;
    signal mux_out_A, mux_out_B : STD_LOGIC_VECTOR(N-1 downto 0);
begin
    path_A(0) <= '0';  -- Initial signal propagation
    path_B(0) <= '1';
        MUX_STAGE: for i in 0 to N-1 generate
        MUX_A: MUX2x1 port map (
            a => path_A(i),
            b => path_B(i),
            sel => challenge(i),
            outMux => path_A(i+1)
        );
        MUX_B: MUX2x1 port map (
            a => path_B(i),
            b => path_A(i),
            sel => challenge(i),
            outMux => path_B(i+1)
        );
    end generate;
     -- Arbiter to generate response
    ARBITER_INST: Arbiter port map (
        clk => clk,
        d_in => path_A(N),
        q_out => response
    );
end Structural;
```

I.6 Testbench wrapper

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ArbiterPUF_TB is
```

```
end ArbiterPUF_TB;
architecture testbench of ArbiterPUF_TB is
   constant N : integer := 64;
   signal challenge : STD_LOGIC_VECTOR(N-1 downto 0);
   signal clk       : STD_LOGIC := '0';
   signal response  : STD_LOGIC;
    -- Clock period definition
   constant clk_period : time := 10 ns;
    -- DUT (Device Under Test) Component
   component ArbiterPUF
      Generic (
         N : integer := 64
      );
      Port (
         challenge : in  STD_LOGIC_VECTOR(N-1 downto 0);
         clk       : in  STD_LOGIC;
         response  : out STD_LOGIC
      );
   end component;
   begin
   -- Instantiate DUT
   DUT: ArbiterPUF
   generic map (
      N => N
   )
   port map (
      challenge => challenge,
      clk       => clk,
      response  => response
   );
   -- Clock Process
   process
   begin
      while now < 1000 ns loop
```

```
            clk <= '0';
            wait for clk_period / 2;
            clk <= '1';
            wait for clk_period / 2;
        end loop;
        wait;
    end process;
    -- Stimulus Process
    process
    begin
        -- Apply different challenge vectors
        challenge <= (others => '0');
        wait for 20 ns;
        challenge <= (others => '1');
        wait for 20 ns;
        challenge <=
"10101010101010101010101010101010101010101010101010101010101010";
        wait for 20 ns;
        challenge <=
"01010101010101010101010101010101010101010101010101010101010101";
        wait for 20 ns;
        wait;
    end process;
    end testbench;
```

I.7 RO PUF

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ro_puf is
    Port (
        clk       : in  STD_LOGIC;
        reset     : in  STD_LOGIC;
```

```
        enable     : in  STD_LOGIC;
        challenge  : in  STD_LOGIC_VECTOR(2 downto 0);  -- select 2 out of 4
oscillators
        response   : out STD_LOGIC
    );
end ro_puf;
architecture Behavioral of ro_puf is
    -- Declare custom types for arrays
    type ro_count_array is array (0 to 3) of unsigned(15 downto 0);
    type ro_div_array   is array (0 to 3) of unsigned(3 downto 0);
    type ro_clk_array   is array (0 to 3) of STD_LOGIC;
    -- Signal declarations
    signal counter   : unsigned(15 downto 0) := (others => '0');
    signal ro_count  : ro_count_array := (others => (others => '0'));
    signal ro_clk    : ro_clk_array := (others => '0');
    signal ro_div    : ro_div_array := (others => (others => '0'));
    signal done      : STD_LOGIC := '0';
begin
    -- Simulated ring oscillators using counters
    process(clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                for i in 0 to 3 loop
                    ro_div(i)   <= (others => '0');
                    ro_clk(i)   <= '0';
                    ro_count(i) <= (others => '0');
                end loop;
                counter <= (others => '0');
                done <= '0';
            elsif enable = '1' and done = '0' then
                -- simulate oscillators with slightly different toggle rates
                for i in 0 to 3 loop
                    ro_div(i) <= ro_div(i) + 1;
```

```
                if ro_div(i)(i) = '1' then  -- unique toggling per oscillator
                    ro_clk(i) <= not ro_clk(i);
                    ro_count(i) <= ro_count(i) + 1;
                end if;
            end loop;
            counter <= counter + 1;
            if counter = 50000 then  -- observation window
                done <= '1';
            end if;
        end if;
    end if;
end process;
-- Challenge-response comparison
process(clk)
    variable idx_a : integer range 0 to 3;
    variable idx_b : integer range 0 to 3;
begin
    if rising_edge(clk) then
        if done = '1' then
            idx_a := to_integer(unsigned(challenge(2 downto 1)));
            idx_b := to_integer(unsigned(challenge(1 downto 0)));
            if idx_a = idx_b then
                response <= '0';  -- avoid comparing the same oscillator
            elsif ro_count(idx_a) > ro_count(idx_b) then
                response <= '1';
            else
                response <= '0';
            end if;
        end if;
    end if;
end process;
end Behavioral;
```

I.8 Inverter

```
-- Inverter entity
library ieee;
use ieee.std_logic_1164.all;
entity inv is
   port (
      input  : in  std_logic;
      output : out std_logic
   );
end inv;
architecture behavioral of inv is
   attribute keep : string;
   attribute keep of output : signal is "true";
begin
   output <= not input after 1 ns; -- Simulate delay for synthesis
end behavioral;
```

APPENDIX II

II. PYTHON SCRIPTS

II.1 Intra Hamming Distance analysis of PUF CRP

```
import numpy as np
import csv
import matplotlib.pyplot as plt
import csv
import matplotlib.pyplot as plt
# Function to calculate Hamming distance
def hamming_distance(str1, str2):
    return sum(el1 != el2 for el1, el2 in zip(str1, str2))
# Read the CSV file containing challenge-response pairs
def read_puf_data(file_path):
    challenges = []
    responses = []
    with open(file_path, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            challenges.append(row['challenge'])
            responses.append(row['response'])
        return responses
# Compute pairwise Hamming distances
def compute_hamming_distances(responses):
    distances = []
    for i in range(len(responses)):
        for j in range(i+1, len(responses)):
            distance = hamming_distance(responses[i], responses[j])
            distances.append(distance)
    return distances
# Main function to read data, compute Hamming distances, and plot
def plot_intra_hamming_distance(file_path):
```

```
    responses = read_puf_data(file_path)
    # Compute pairwise Hamming distances
    distances = compute_hamming_distances(responses)
    # Plot the distribution of intra Hamming distances
    plt.hist(distances, bins=10, edgecolor='black')
    plt.title("Intra Hamming Distance of PUF Responses")
    plt.xlabel("Hamming Distance")
    plt.ylabel("Frequency")
    plt.show()
# Example usage
file_path = 'C:\Users\Admin\Desktop\Intel Softwares\My Work\10 Mar
2025\CRP.csv'
plot_intra_hamming_distance(file_path)
```

```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Load and preprocess CRP data (example placeholder; adjust as per original code)
data = pd.read_csv('ArbiterCRP.csv')
X = data.iloc[:, :-1].values  # Challenges
y = data.iloc[:, -1].values   # Responses
indices = np.random.permutation(len(X))
X, y = X[indices], y[indices]
test_size = 1000
trainsize = 20000
X_test, y_test = X[:test_size], y[:test_size]
X_train_pool, y_train_pool = X[test_size:trainsize], y[test_size:trainsize]
# Define training sizes
train_sizes = np.linspace(50, len(X_train_pool), 10, dtype=int)
accuracies = []
# Training loop with Linear SVM
```

```
for train_size in train_sizes:
    X_train = X_train_pool[:train_size]
    y_train = y_train_pool[:train_size]
    # Initialize and train the Linear SVM model
    svm_model = LinearSVC(max_iter=1000)
    svm_model.fit(X_train, y_train)
    # Predict and evaluate
    y_pred = svm_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    print(f"Training size: {train_size}, Accuracy: {accuracy:.4f}")
# Plot results
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, accuracies, 'bo-', label='Linear SVM Accuracy')
plt.title('Linear SVM Accuracy vs. Number of CRPs')
plt.xlabel('Training Size')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.savefig('puf_svm_accuracy.png')
```

II.2 LR ML attack on an Arbiter PUF

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# Parameters
test_size = 90000
step_size = 10000
# Read CRP data from CSV
df = pd.read_csv('ArbiterCRP.csv', header=None)
# Check that all challenge bits and responses are 0 or 1
```

```
assert ((df.iloc[:, :-1] == 0) | (df.iloc[:, :-1] == 1)).all().all(), "Challenge bits must be 0 or 1"
assert ((df.iloc[:, -1] == 0) | (df.iloc[:, -1] == 1)).all(), "Responses must be 0 or 1"
# Shuffle the data
df_shuffled = df.sample(frac=1, random_state=42)
# Convert to NumPy arrays
X_all = df_shuffled.iloc[:, :-1].values
y_all = df_shuffled.iloc[:, -1].values
# Split into test set and training pool
X_test = X_all[:test_size]
y_test = y_all[:test_size]
X_train_pool = X_all[test_size:]
y_train_pool = y_all[test_size:]
# Determine max_train_size
max_train_size = len(X_train_pool)

# Set train_sizes
potential_train_sizes = range(step_size, max_train_size + step_size, step_size)
train_sizes = [min(size, max_train_size) for size in potential_train_sizes]
train_sizes = sorted(set(train_sizes))  # Remove duplicates, if any
# Lists to store results
accuracies = []
# Train LR model for each training set size
for train_size in train_sizes:
    X_train = X_train_pool[:train_size]
    y_train = y_train_pool[:train_size]
  # Initialize and train Logistic Regression model
    lr_model = LogisticRegression(max_iter=1000, solver='lbfgs')
    lr_model.fit(X_train, y_train)
    # Predict on test set and calculate accuracy
    y_pred = lr_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    print(f"Training size: {train_size}, Accuracy: {accuracy:.4f}")
```

```
# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, accuracies, 'bo-', label='Accuracy')
plt.title('Logistic Regression Accuracy vs. Number of CRPs')
plt.xlabel('Number of CRPs in Training Set')
plt.ylabel('Accuracy on Test Set')
plt.grid(True)
plt.legend()
plt.savefig('puf_lr_accuracy.png')
print("Plot saved as 'puf_lr_accuracy.png'")
```

II.3 GB ML attack on an Arbiter PUF

```
import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
# Step 1: Load the CSV file
data = pd.read_csv('ArbiterCRP.csv', header=None)
# Step 2: Separate challenges and responses
X = data.iloc[:, :-1].values  # all columns except last = challenge
y = data.iloc[:, -1].values   # last column = response
# Step 3: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Step 4: Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(
    n_estimators=200,     # number of boosting rounds (trees)
    learning_rate=0.1,    # shrinkage
    max_depth=5,          # tree depth
    random_state=42
```

```
)
# Step 5: Train the model
gb_model.fit(X_train, y_train)
# Step 6: Evaluate
y_pred = gb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\n Gradient Boosting Accuracy: {accuracy * 100:.2f}%")
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples')
plt.title("Confusion Matrix - Gradient Boosting PUF Model")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
# Classification Report
print("\n Classification Report:\n")
print(classification_report(y_test, y_pred))
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
```

II.4 Multiple ML attack on an Arbiter PUF

```
# Models
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
# Load CSV
data = pd.read_csv('trimmed_file.csv')
# Split features and labels
```

```
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Define models
models = {
   "SVM": SVC(kernel='rbf', gamma='scale'),
   "Logistic Regression": LogisticRegression(max_iter=1000),
   "MLP": MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=1000,
random_state=42),
   "KNN": KNeighborsClassifier(n_neighbors=5)
}
# Store results
accuracies = {}
conf_matrices = {}
P = np.mean(y)
H = -P * np.log2(P) - (1 - P) * np.log2(1 - P) if 0 < P < 1 else 0.0
print(f"Bitwise Probability P (response=1): {P:.4f}")
print(f"Bitwise Entropy H: {H:.4f} bits\n")
# Train, predict, evaluate
for name, model in models.items():
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   acc = accuracy_score(y_test, y_pred)
   cm = confusion_matrix(y_test, y_pred)
     accuracies[name] = acc
   conf_matrices[name] = cm
     print(f"{name} Accuracy: {acc * 100:.2f}%")
# Plot accuracy comparison
plt.figure(figsize=(7, 4))
plt.bar(accuracies.keys(), accuracies.values(), color=['skyblue', 'orange', 'green',
'purple'])
plt.ylabel('Accuracy')
```

```
plt.ylim(0, 1)
plt.title('Prediction Accuracy Comparison (SVM, LR, MLP, KNN)')
plt.grid(True, axis='y', linestyle='--', alpha=0.6)
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()
# Plot confusion matrices
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
axs = axs.flatten()
for i, (name, cm) in enumerate(conf_matrices.items()):
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
    disp.plot(ax=axs[i], cmap='Blues', colorbar=False)
    axs[i].set_title(f"{name} Confusion Matrix")
plt.tight_layout()
plt.show()
```

II.5 ML attack on an RO-PUF

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
from imblearn.over_sampling import RandomOverSampler
# ============================
# LOAD DATA
# ============================
file_path = "ROCRP.csv"
data = pd.read_csv(file_path)
```

```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# Convert to numeric and remove NaNs
X = X.apply(pd.to_numeric, errors="coerce")
y = pd.to_numeric(y, errors="coerce")
df = pd.concat([X, y], axis=1).dropna()
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].astype(int).values
print("Dataset shape:", X.shape)
print("Class distribution:", dict(zip(*np.unique(y, return_counts=True))))
# ============================
# TRAIN TEST SPLIT
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=42
)
# ============================
# SCALING
# ============================
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# ============================
# OVERSAMPLING (FIX IMBALANCE)
# ============================
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)
print("After Oversampling:", dict(zip(*np.unique(y_train, return_counts=True))))
# ============================
# FAST MLP MODEL
# ============================
```

```
mlp = MLPClassifier(
    hidden_layer_sizes=(64, 32),
    activation="relu",
    solver="adam",
    learning_rate_init=0.001,
    max_iter=200,
    early_stopping=True,
    validation_fraction=0.1,
    n_iter_no_change=10,
    random_state=42
)
mlp.fit(X_train, y_train)
# ============================
# PREDICT
# ============================
y_pred = mlp.predict(X_test)
# ============================
# ACCURACY
# ============================
acc = accuracy_score(y_test, y_pred)
print("\nAccuracy:", round(acc * 100, 2), "%")
# ============================
# CONFUSION MATRIX
# ============================
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap="Blues")
plt.title("MLP Confusion Matrix")
plt.show()
```

APPENDIX III

III.1 QUARTUS PRIME SOFTWARE

The Intel Quartus Prime software provides an intuitive GUI that supports easy design entry, fast design processing, straightforward device programming, and seamless integration with industry-standard EDA tools. Each logic circuit, or sub-circuit, created in Quartus Prime is organized as a project. The software works on one project at a time, storing all related information in a dedicated directory within the file system. A project encapsulates design files, hierarchy, libraries, constraints, and project settings, ensuring structured design management.

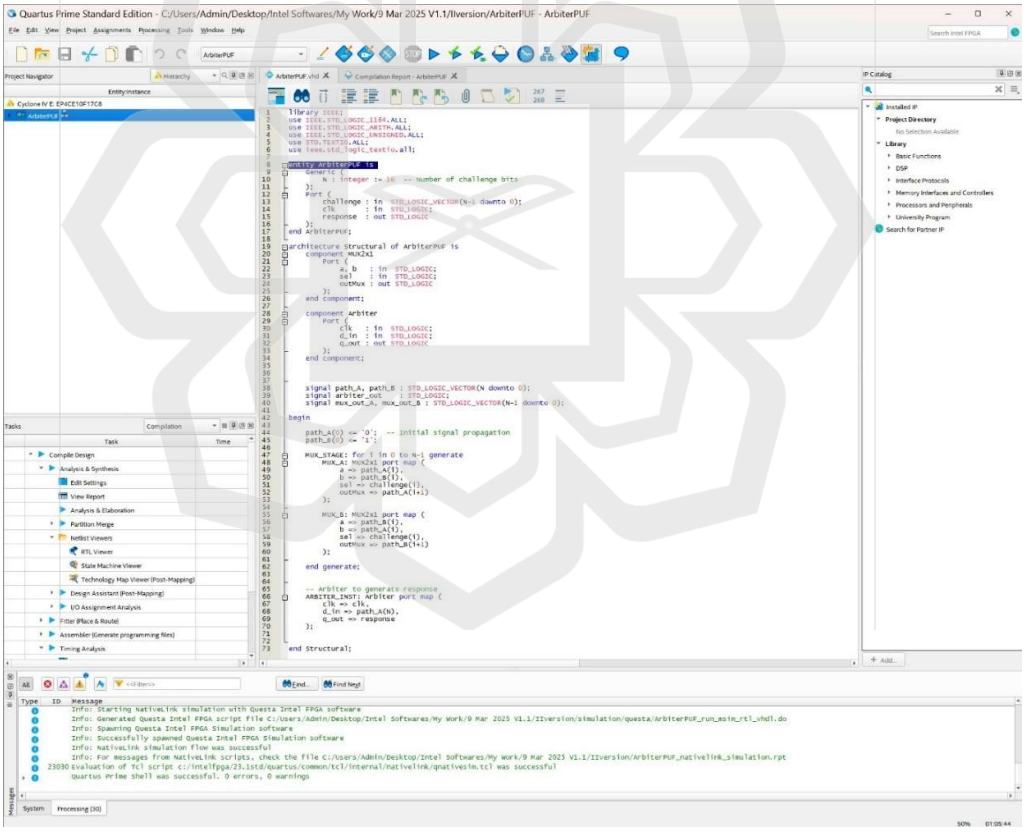


Figure III.1 Quartus Prime Environment

The Intel Quartus Prime Standard Edition serves as a comprehensive integrated development environment (IDE) for field-programmable gate array (FPGA) design. It

provides a unified suite of tools for design entry, synthesis, optimization, functional verification, and simulation, thereby streamlining and accelerating the overall development process. The environment incorporates advanced features such as the Power Analyzer for evaluating energy efficiency, the Platform Designer for system-level integration, and the Design Assistant for design rule guidance. In addition, it offers robust timing analysis and debugging capabilities, which support all stages of the FPGA design flow and contribute to improved design reliability and performance. Figure III.1 illustrates that the basic information about the intended project is displayed in the Tasks pane, Project Navigator, Report panel, and Messages window.

III.1.1 Managing Intel Quartus Prime Projects (Intel, 2025)

- The Project Navigator (View > Utility Windows > Project Navigator) presents significant information about the elements of the project, such as the design files, IP components, and project hierarchy (after elaboration). The Project Navigator organizes information on the Files, Hierarchy, Design Units, and IP Components tabs. Right-click items on the Project Navigator are used to locate or perform actions on the elements of the project.
- The Compilation Report panel is dynamically updated to display detailed reports during project processing. To access Compilation Reports, click (Processing > Compilation Report).
- The Messages window (View > Utility Windows > Messages) displays information, warning, and error messages about Intel Quartus Prime processes. A right-click on the messages tab is used to locate the source or get message help.

  Processing tab: It displays messages from the most recent process.

  System tab: It displays messages unrelated to design processing.

  Search: It locates specific messages.
- When a new project is created in the GUI, the Intel Quartus Prime software automatically creates an Intel Quartus Prime Project File (.qpf) for that project. The .qpf references the Intel Quartus Prime Settings File (.qsf). The .qsf lists the project's design, constraint, and IP files, and stores project-

wide and entity-specific settings specified in the GUI. The .qsf stores the settings for each project version. Optimizing the project settings helps the Compiler generate programming files that fulfill the intended specifications.

- The targeted Intel device or board to be used in the project is specified in the Device dialog box. By clicking the Device and Pin Options button in the dialog, you can select preferences for the device configuration scheme, programming file generation, I/O timing, voltage, and other options.
- The Intel Quartus Prime software includes full-featured schematic and text editors, as well as HDL templates to accelerate the design process. It supports VHDL Design Files (.vhd), Verilog HDL Design Files (.v), SystemVerilog (.sv), and schematic Block Design Files (.bdf). It also supports Verilog Quartus Mapping (.vqm) design files generated by other design entry and synthesis tools. In addition, logic design files with Intel and third-party IP core design files can be integrated, including components into a Platform Designer (Standard) system (.qsys).
- The Fitter tool is used to optimize the placement of logic in the device to meet expected timing and routing constraints. Specify timing constraints in the Timing Analyzer (Tools > Timing Analyzer), or in an .sdc file. Specify constraints for clock characteristics, timing exceptions, and external signal setup and hold times before running analysis. The Timing Analyzer reports detailed information about the design's performance compared to the constraints in the Compilation Report panel.
- Optional support for integrating EDA design entry, synthesis, simulation, physical synthesis, and formal verification tools into the Intel Quartus Prime design flow allows netlist files from other EDA design entry and synthesis tools.

III.1.2 FPGA Implementation Considerations

- Designers should consider the target device characteristics to plan for I/O interfaces, integration of IP, on-chip debugging tools, and the use of other EDA tools.

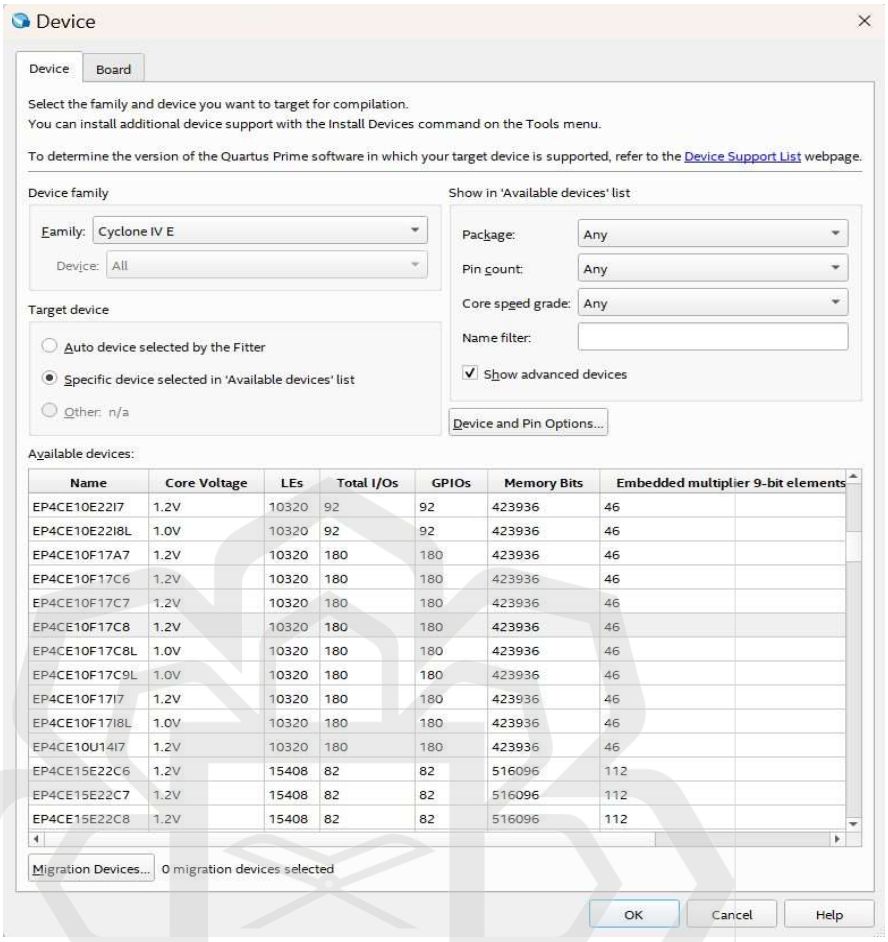


Figure III.2 Device Dialog Box

- Consider the power consumption of the target device and the available programming methods when planning the layout.
- Intel Quartus Prime software settings can be adjusted to optimize any aspect of design, such as performance, routing layouts, area, or power utilization. Hence, the design priorities and trade-offs must be considered. For example, certain device features, density, and performance requirements can increase system cost. Signal integrity and board issues can impact I/O pin locations. Power, timing performance, and area utilization all affect one another. Compilation time is affected when optimizing these priorities.
- Early decision and clarity on design priorities help in choosing the most suitable devices, tools, features, and methodologies to adopt.

- A summary of the target device's resources is available for quick reference by selecting a device in the Device dialog box (Assignments > Device) shown in Figure III.2.
- The configuration scheme is defined on the Configuration tab of the Device and Pin Options dialog box shown in Figure III.3. The Intel Quartus Prime software uses the settings for the configuration scheme, configuration device, and configuration device voltage to enable the appropriate dual-purpose pins as regular I/O pins.

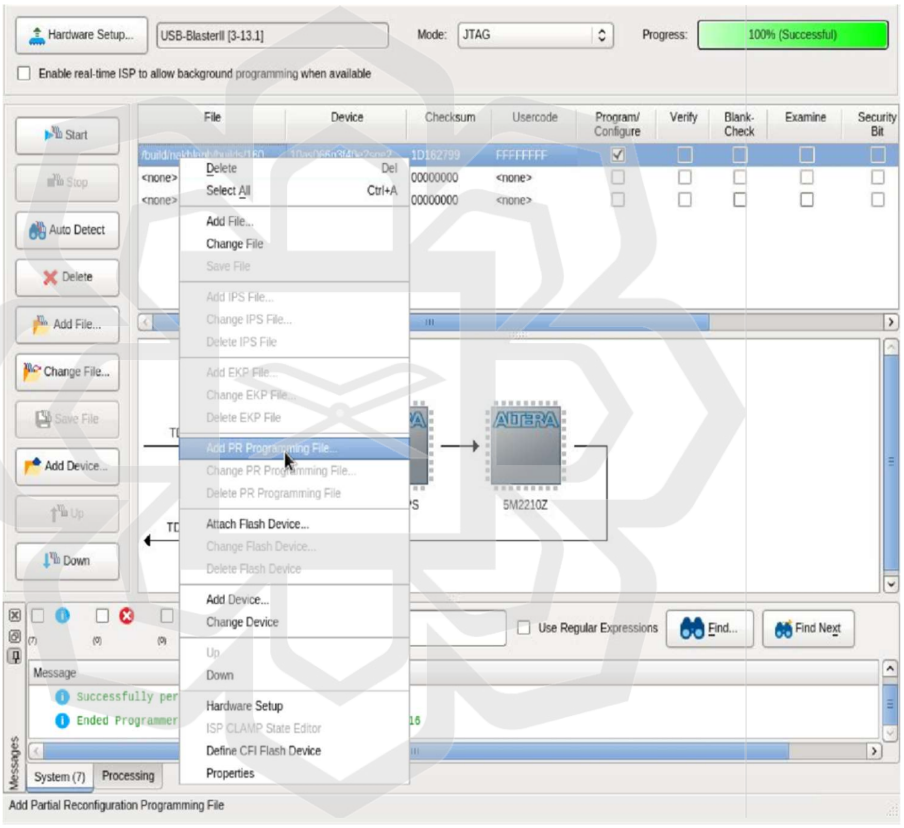


Figure III.3 Intel Quartus Prime Programmer

III.2 FPGAs

Field Programmable Gate Arrays (FPGAs) serve as an effective platform for implementing PUF based hardware security primitives in IoT edge environments. Their reconfigurable nature allows designers to exploit inherent manufacturing variations present in configurable logic blocks, routing paths, and input output structures to

generate unique device specific fingerprints. Unlike microcontrollers or application specific integrated circuits, FPGAs provide fine grained control over delay paths and circuit structures, which is essential for constructing reliable PUF architectures such as Arbiter PUF and Ring Oscillator PUF. The architectural flexibility of FPGAs enables precise placement and routing of logic elements, allowing controlled extraction of process induced variations that form the basis of challenge response behavior. This capability supports the design of strong PUFs with large challenge response spaces while maintaining adaptability to environmental variations through calibration and error correction techniques. Additionally, FPGA based PUF implementations benefit from parallelism, enabling rapid response generation and efficient integration with cryptographic protocols for authentication and key generation at the edge.

Furthermore, the reconfigurability of FPGAs introduces an additional security dimension by allowing dynamic modification of PUF structures, making modeling and cloning attacks more difficult. At the same time, careful design considerations are required to ensure stability, uniqueness, and resistance to machine learning based attacks, especially when deployed in resource constrained IoT edge systems. Thus, FPGAs provide a versatile and secure hardware substrate specifically suited for practical and scalable PUF implementations (Manan, 2018).

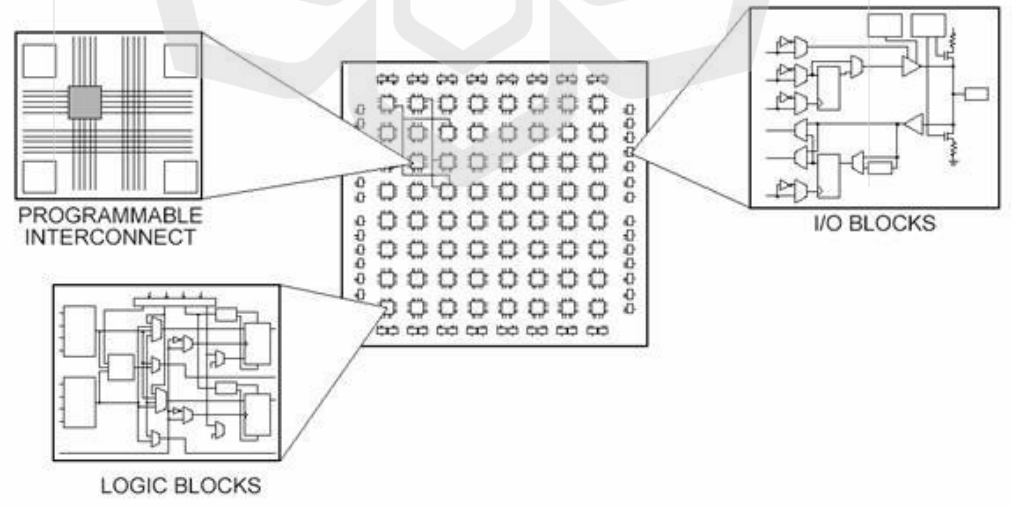


Figure III.4 FPGA layout (National Instruments, 2025)

From a PUF oriented perspective, FPGA resource specifications become important not for general computation, but for how effectively they support the creation of reliable and secure hardware fingerprints. As illustrated in Figure III.4, configurable logic blocks (CLBs) are the most critical resource because they contain lookup tables and flip flops that are directly used to construct delay based PUF structures. Variations in propagation delay across LUTs and routing paths form the fundamental entropy source in designs such as Arbiter PUFs and Ring Oscillator PUFs. The number and distribution of CLBs determine how many independent challenge response pairs can be generated, as well as the scalability of strong PUF implementations. Additionally, the internal structure of slices affects routing symmetry, which is crucial for minimizing bias and improving uniqueness.

Fixed function blocks such as multipliers are generally less relevant to core PUF generation, but they can support auxiliary operations such as post processing, hashing, or lightweight cryptographic integration of PUF responses. Similarly, embedded block RAM is not directly involved in entropy generation, but it plays an important role in storing helper data, error correction codes, and challenge response datasets required for reliable key reconstruction. At the edge level, FPGA based PUFs benefit from low latency and direct hardware interaction, enabling fast challenge response evaluation without processor overhead. This is particularly useful for authentication protocols where rapid response time is essential. Moreover, the ability of FPGAs to directly interface with sensors can be leveraged to design context aware or environmental PUFs, although this requires careful handling to avoid instability. In higher level platforms such as edge servers or cloud accelerators, FPGA resources can support large scale PUF evaluation, enrollment, and even machine learning based attack analysis. However, from a security standpoint, careful design is required to ensure that the same flexibility and accessibility of FPGA resources do not unintentionally expose side channel information or make the PUF more vulnerable to modeling attacks.

Overall, when viewed through the PUF lens, FPGA resource specifications are primarily evaluated based on how well they support controlled delay variation, reliable response generation, and secure integration with authentication mechanisms rather than purely computational performance.

III.2.1 Intel Cyclone FPGA

The Cyclone IV device family is engineered for high volume, cost constrained applications, providing an efficient balance between performance, power consumption, and silicon area. Fabricated using a low power process technology, these devices are optimized to support increasing bandwidth and system integration requirements while maintaining minimal cost overhead. The architecture is tailored to deliver adequate logic density, embedded memory, and digital signal processing capabilities suitable for embedded and edge oriented designs. The Cyclone IV family is divided into two primary variants, each targeting distinct application domains. The Cyclone IV E variant is optimized for general purpose logic intensive applications, offering a cost efficient solution with a balanced distribution of logic elements and memory resources. In contrast, the Cyclone IV GX variant integrates high speed transceiver blocks with 3.125 Gbps transceivers, enabling support for high bandwidth communication protocols and data intensive applications requiring fast serial connectivity. Cyclone IV E—lowest power, high functionality, with the lowest cost

Table III.1 Cyclone IV E (EP4CE10) device resources.

Resources	#
Logic elements (LEs)	10,320
Embedded memory (Kbits)	414
Embedded 18 × 18 multipliers	23
General-purpose PLLs	2
Global clock networks	10
User I/O banks	8
Maximum user I/O	179

Initially, FPGAs were primarily employed for proof-of-concept (PoC) designs and prototype development. Over time, however, their utility has extended to applications that require high performance, low latency, and real-time adaptability. For instance, Cyclone IV FPGAs are composed of logic elements (LEs), which include four-input lookup tables (LUTs), memory blocks, and multipliers, as summarized in Table

III.1. The M9K memory blocks provide 9 Kbits of embedded SRAM, which can be configured in various ways, including single-port, simple dual-port, or true dual-port RAM, as well as FIFO buffers or ROM. This configurability enables the efficient implementation of a wide range of digital circuits. In the context of delay-based PUFs, such as arbiter PUFs implemented on FPGAs, the propagation delays of signals through LUTs play a critical role in determining the PUF's output response. Moreover, the physical placement of LUTs within the FPGA fabric and the routing paths between them further influence these propagation delays. Consequently, subtle variations in LUT placement and interconnect routing can affect the behaviour of the arbiter PUF, introducing inherent unpredictability that is essential for security applications. The internal architecture of logic elements (LEs) in Altera Cyclone IV FPGAs is illustrated in Figure III.5.

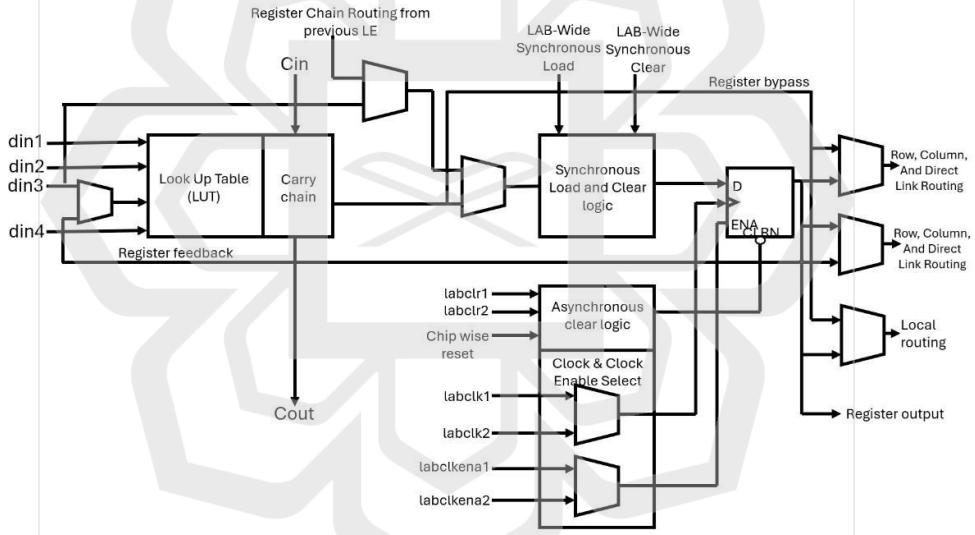


Figure III.5 Internal structure of the FPGA CLB (Altera Corporation, 2009)

Each LE consists of a flip-flop, a lookup table (LUT) that can be configured as either a four-input, one-output or a three-input, one-output LUT, and fast carry logic for efficient arithmetic operations. Sixteen LEs are grouped to form a Logic Array Block (LAB), and depending on the specific Cyclone IV family, the FPGA may contain up to 8,000 LABs, providing extensive logic resources. Each LE includes a configurable register capable of operating as a D, T, JK, or SR flip-flop. The clock and clear signals for these registers can be sourced from the global clock network, general-purpose I/O pins, or internal logic, providing flexibility for timing and control in complex designs.

LEs can operate in two primary modes. In the normal mode, they implement general-purpose combinational logic, supporting up to four data inputs from the local interconnect within the LAB. In the arithmetic mode, the LEs are optimized for arithmetic functions such as adders, counters, accumulators, and comparators. Here, each LE functions as a 2-bit full adder with a dedicated carry chain, and the output is available both as a registered and unregistered signal from the LUT. This versatile architecture allows designers to efficiently implement a wide range of digital logic and arithmetic operations, enabling high-performance, resource-efficient designs. By supporting both combinational and arithmetic functions within the same LE structure, Cyclone IV FPGAs offer significant flexibility for optimizing the implementation of complex digital systems.

Artificial intelligence (AI) enabled edge devices hold significant potential for performing computationally intensive tasks such as advanced signal filtering, event detection, communication optimization, and data compression. These capabilities enhance device performance, enable sophisticated on-chip process control, and improve energy efficiency. The interconnectivity within an FPGA is often compared to the neural wiring of the human brain. The programmable logic fabric of an FPGA is highly interconnected, which makes it an excellent platform for implementing neural networks and other AI workloads. Its bit-level dynamic programmability, both at the logic and interconnect levels, mirrors the brain's ability to shift attention and adapt to specific tasks. Furthermore, FPGA external I/O offers exceptional flexibility unmatched by most hardware architectures, allowing seamless integration with diverse sensors, including radar, audio, vibration, and vision. Like the brain, FPGAs can intelligently manage how signals enter and exit the device in real time.

The architecture and development environment of edge devices differ significantly from high-end computing servers, making rigorous optimization of AI models essential. Edge devices face constraints such as limited memory capacity, reduced computational resources, and the need for energy efficiency to minimize power consumption. To address these challenges, two widely used strategies for reducing model weight size are (i) Pruning: removing connections with small weights, typically those below a predefined threshold, and (ii) Quantization: lowering the precision of weight values, for example, reducing from 32-bit to 8-bit representations (Żyliński et

al., 2024). Some of the common FPGA frameworks for the deployment of AI models on FPGAs are listed below:

- Xilinx Vitis AI
  - Supports Xilinx FPGAs.
  - Provides a pre-optimized library of AI models.
  - Includes tools for quantization and model compilation.
- Intel OpenVINO
  - Usually customized for Intel FPGAs.
  - This system supports seamless integration of AI models trained on popular frameworks like TensorFlow or PyTorch.
  - Supports heterogeneous inference across CPUs, GPUs, and FPGAs.
- OpenCL for FPGAs
  - Designers can describe AI accelerators using high-level programming languages.
  - Enables portability across FPGA platforms.
- Custom HDL Design
  - Verilog or VHDL modeling language for customized AI accelerators.
  - This approach offers flexibility but requires deep expertise in FPGA design.

III.3 Design Simulation

The NativeLink Integration feature of Intel Quartus Prime software enables a seamless integration between Quartus and third-party EDA tools such as ModelSim-Intel FPGA Edition, Questa Advanced Simulator, Synopsys VCS, or Cadence Xcelium for simulation, synthesis, and timing analysis. Instead of manually exporting netlists, compiling files, and launching simulators, NativeLink automates the process so that simulation can be launched directly from the Quartus environment. Quartus provides different simulation options, both built-in and through third-party tools.

- RTL Simulation (Functional Simulation): RTL simulation is a functional verification of Register Transfer Level (RTL) design before synthesis and implementation. It verifies the logical correctness of the HDL code (VHDL,

Verilog, or SystemVerilog) by simulating design behaviour in response to input stimuli, without considering propagation delays or timing information. A testbench or Quartus Waveform Editor is used to apply inputs and observe outputs. RTL simulation usually runs on integrated tools like ModelSim-Intel FPGA Edition via Quartus NativeLink.

- Gate-Level Simulation (Post-Synthesis Simulation): It is performed after synthesis to verify the correctness of the design at the netlist level, where the HDL code has already been converted into logic gates. It can be run with or without timing information. It verifies only the logical equivalence without timing. In contrast, timing (provided in Standard Delay Format, SDF) accounts for gate and interconnect delays to check for setup/hold violations and timing hazards. It is typically performed on ModelSim-Intel FPGA Edition or third-party simulators integrated through Quartus NativeLink.
- Timing Simulation (Post-Fitting Simulation): Timing simulation is performed after the design has been fully synthesized, placed, and routed (fitted) onto the FPGA device. Unlike functional or gate-level simulation, it accounts for actual propagation delays, interconnect delays, and timing constraints, making it the most accurate representation of how the circuit will behave in hardware. The process uses back-annotated delay information provided in a Standard Delay Format (SDF) file, which is applied to the post-fitting netlist. Timing simulation is typically carried out using ModelSim-Intel FPGA Edition or other third-party simulators through Quartus' NativeLink interface.

The complete summary of the HDL-based FPGA design flow is illustrated in Figure III.6. The synthesizable VHDL/Verilog/System Verilog source code of the logic design must be compiled and optimized using a logic synthesis tool to produce a lower-level netlist, like a gate-level netlist. This netlist is then mapped to FPGA internal resources, which are subsequently fitted during the place and route steps to produce a post-fitting netlist. In the final step, a corresponding bitstream file is generated for the configuration of the FPGA.

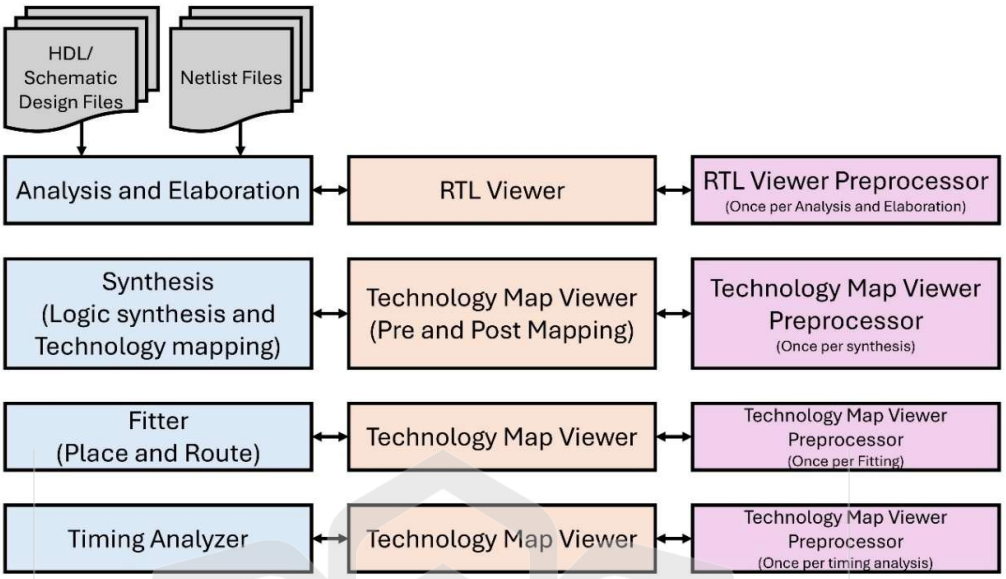


Figure III.6 Design Flow using Quartus Prime tools

Figure III.7 (a) illustrates the basic steps for simulating a design in ModelSim. The basic steps for simulating a design in ModelSim are enumerated below.

a. Creating the working library

   In ModelSim, all designs are compiled into a library. A new simulation typically begins with the creation of a working library named ‘work’, which serves as the default destination for compiled design units such as entities, architectures, and modules.

b. Compiling Design Files

   Once the working library is created, all the design units are compiled into it. The compilation process checks the source code for syntax errors, resolves dependencies between modules or entities, and stores the compiled design units in the library. The ModelSim library format is compatible across all supported platforms.

c. Loading the Simulator with Design and Running the Simulation

   Compilation of the design is followed by loading the simulator with the design by invoking the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL). Once the design loads successfully, the simulation time is set to zero, and the run command is asserted to begin the simulation.

d. Debugging

The debugging stage in ModelSim enables designers to detect and fix functional and timing errors in their HDL designs. During this stage, signals can be monitored in the waveform window, and simulation can be controlled using breakpoints or step-by-step execution.

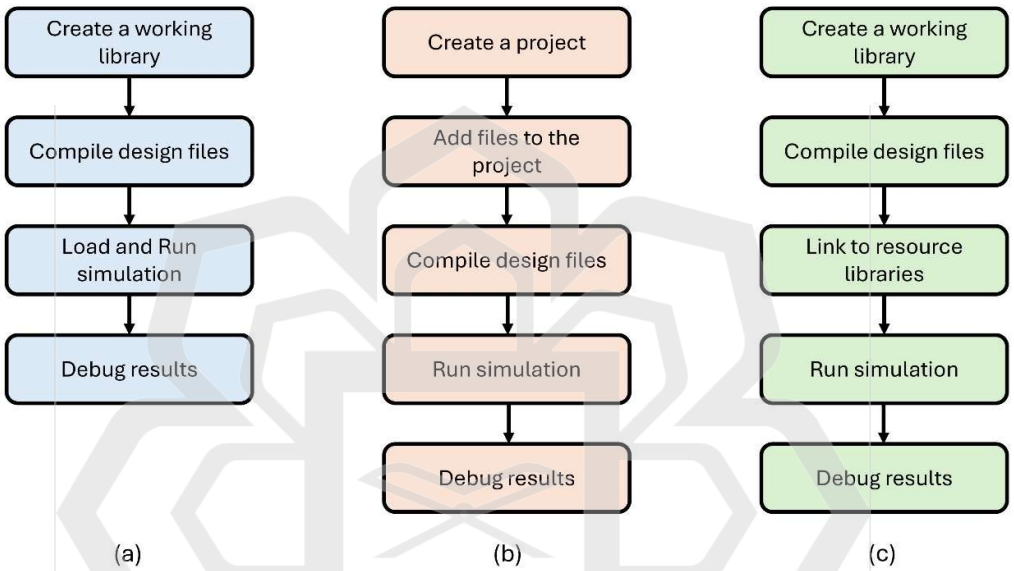


Figure III.7 ModelSim Simulation(a) Basic Simulation Flow (b) Project Flow (c) Multiple Library Flow

The flow shown in Figure III.7(b) is like the basic simulation flow in Figure 3.8(a), with a key difference: the working library is created automatically, and projects are persistent. ModelSim uses libraries in two ways: (i) as a local working library, which contains the compiled version of the design, and (ii) as a resource library, which is typically static and serves as a source of design components. The contents of the working library change as design units are updated or recompiled. In contrast, resource libraries, often supplied by other design teams or third-party vendors, must be specified during compilation. Rules must be defined to determine the order in which these libraries are searched. A common scenario involves compiling the gate-level design and testbench into the working library while referencing gate-level models from a separate

resource library. Figure III.7(c) illustrates the basic steps for simulating a design using multiple libraries (Kahlert & Giza, 2016).

III.4 Visual Studio Code

Visual Studio Code (VS Code) is a free open-source code editor for development and debugging modern cloud and web applications, available for Linux, OS X, and Windows. VS Code supports more than 30 different programming, markup, and database languages, including JavaScript, C#, C++, PHP, Java, HTML, R, CSS, SQL, Markdown, TypeScript, Less, Sass, JSON, XML, and Python. Extensions in Visual Studio Code (VS Code) are add-ons that expand the editor's capabilities. They enable developers to customize the environment to fit their specific needs, whether for coding, debugging, or managing projects. Visual Studio Code features a clean and intuitive layout designed to maximize the editor space while still providing easy access to project context and navigation. Its interface is divided into several key sections:

- *Editor:* he main area for writing and editing code, with support for up to three editors open side by side.
- *Sidebar:* provides views such as the Explorer for file navigation and project management.
- *Status Bar:* displays information about the active project, files, and tasks.
- *The Activity Bar,* located on the far left, allows switching between views and shows context-specific indicators, such as pending Git changes.
- *Panel:* hosts integrated terminals, debugging tools, and other utilities.

Additionally, the Command Palette gives quick access to commands, while settings and extensions allow for extensive workspace customization. Each time VS Code restarts, it restores the previous state, including open folders, layouts, and files. The user interface (UI) of Visual Studio Code layout illustrated in Figure III.8 is divided into six main areas:

- *Editor:* This is the main area to edit files. It is possible to open as many editors as needed side by side vertically and horizontally.

- *Primary Side Bar:* It contains different views, like the Explorer, to assist while working on project.

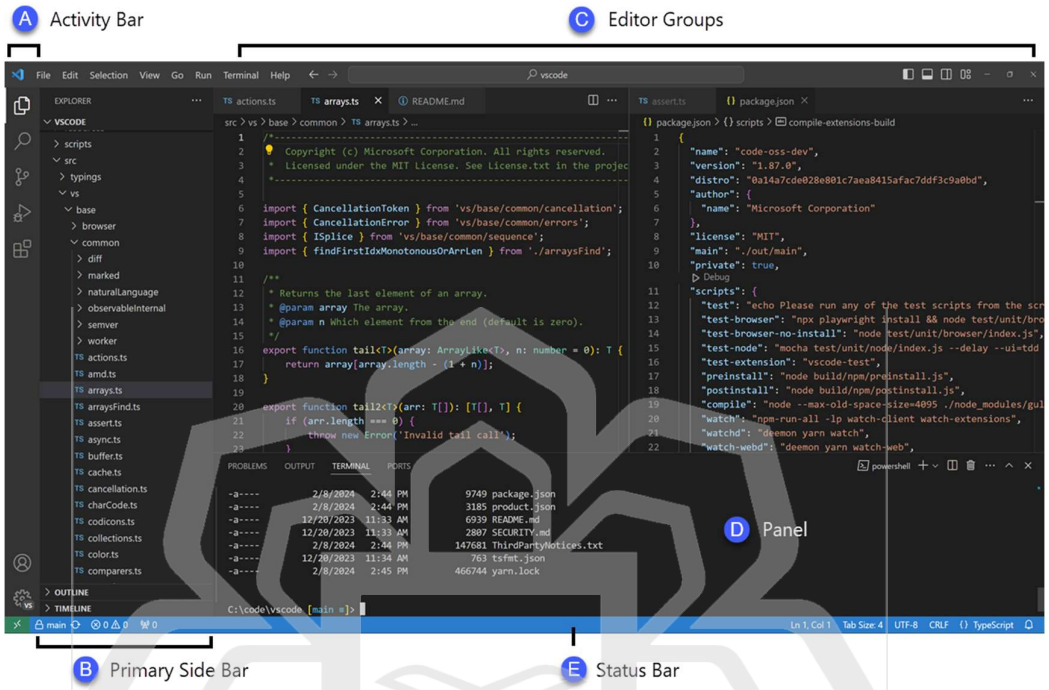


Figure III.8 Visual Studio Code layout

- *Secondary Side Bar:* Opposite the Primary Side Bar. By default, it contains the Chat view. Drag and drop views from the Primary Side Bar to the Secondary Side Bar to move them.
- *Status Bar:* Information about the project open and the edited files. At the very bottom, this bar displays contextual information about projects and the current file, like the language mode and any errors.
- *Activity Bar:* Located on the far left-hand side. It lets users switch between views and provides additional context-specific indicators, like the number of outgoing changes when Git is enabled. It is possible to change the position of the Activity Bar.
- *Panel:* An additional space for views below the editor region. By default, it contains output, debug information, errors, and warnings, and an integrated terminal. The Panel can also be moved to the left or right for more vertical space. Positioned at the bottom of the window, the Panel is used for other

views, such as the Integrated Terminal for running commands, or the Debug console.

- *Command Palette:* Accessed by Ctrl+Shift+P (Windows/Linux) or Cmd+Shift+P (macOS), this palette allows users to search for and run any command within VS Code.
- *Settings:* Allows customization of the VS Code environment by accessing the settings menu to change themes, keybindings, and other behaviors.
- *Extensions:* The VS Code Marketplace offers a vast library of extensions to add support for new languages, debugging tools, and other features.

To work with Python inside VS Code, the Python extension is required, which brings many useful features, such as code completion with IntelliSense, debugging, and unit testing support.

APPENDIX IV

LIST OF PUBLICATIONS

Journals:

Sheikh, A. M., Islam, M. R., Habaebi, M. H., Zabidi, S. A., Bin Najeeb, A. R., & Kabbani, A. (2025). Integrating Physical Unclonable Functions with Machine Learning for the Authentication of Edge Devices in IoT Networks. *Future Internet*, *17*(7), 275.

Sheikh, A. M., Islam, M. R., Habaebi, M. H., Zabidi, S. A., Bin Najeeb, A. R., & Kabbani, A. (2025). A survey on edge computing (EC) security challenges: Classification, threats, and mitigation strategies. *Future Internet*, *17*(4), 175.

Sheikh, A. M., Islam, M. R., Habaebi, M. H., Kabbani, A., Zabidi, S. A., & bin Najeeb, A. R. (2024). Securing the IoT Edge Devices Using Advanced Digital Technologies. *Asian Journal of Electrical and Electronic Engineering*, *4*(2), 52-60.

Sheikh AM, Islam MR, Habaebi MH, Zabidi SA, Najeeb ARb, Baloch M. Securing IoT Networks Using Machine Learning-Resistant Physical Unclonable Functions (PUFs) on Edge Devices. Network. 2026; 6(1):6. https://doi.org/10.3390/network6010006

Conference:

Sheikh, A. M., Islam, M. R., Habaebi, M. H., Zabidi, S. A., Najeeb, A. R. B., & Basahel, A. (2023, August). Machine Learning (ML) assisted Edge security framework on FPGAs. In *2023, the 9th International Conference on Computer and Communication Engineering (ICCCE)* (pp. 155-160). IEEE.