

April 2021

ENABLING IOT AUTHENTICATION, PRIVACY AND SECURITY VIA BLOCKCHAIN

Md Nazmul Islam
University of Massachusetts Amherst

Follow this and additional works at: https://scholarworks.umass.edu/dissertations_2



Part of the [VLSI and Circuits, Embedded and Hardware Systems Commons](#)

Recommended Citation

Islam, Md Nazmul, "ENABLING IOT AUTHENTICATION, PRIVACY AND SECURITY VIA BLOCKCHAIN" (2021). *Doctoral Dissertations*. 2108.
https://scholarworks.umass.edu/dissertations_2/2108

This Open Access Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

University of Massachusetts Amherst

ScholarWorks@UMass Amherst

Doctoral Dissertations

Dissertations and Theses

ENABLING IOT AUTHENTICATION, PRIVACY AND SECURITY VIA BLOCKCHAIN

Md Nazmul Islam

Follow this and additional works at: https://scholarworks.umass.edu/dissertations_2



Part of the [VLSI and Circuits, Embedded and Hardware Systems Commons](#)

ENABLING IOT AUTHENTICATION, PRIVACY AND SECURITY VIA BLOCKCHAIN

A Dissertation Presented

by

MD NAZMUL ISLAM

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2021

Electrical and Computer Engineering

© Copyright by Md Nazmul Islam 2021

All Rights Reserved

ENABLING IOT AUTHENTICATION, PRIVACY AND SECURITY VIA BLOCKCHAIN

A Dissertation Presented

by

MD NAZMUL ISLAM

Approved as to style and content by:

Sandip Kundu, Chair

Wayne P. Burleson, Member

Daniel E. Holcomb, Member

Brian N. Levine, Member

Christopher V. Hollot, Department Chair
Electrical and Computer Engineering

DEDICATION

To my parents.

ACKNOWLEDGMENTS

First and foremost, I am grateful to my advisor Prof. Sandip Kundu for his guidance and support during my stay at UMass. I am forever indebted to the research skills and the engineering knowledge he has passed through his supervision. His research and teaching dedication has encouraged me immensely during my Ph.D. and will always be a great source of inspiration.

I want to thank Prof. Wayne Burleson, Prof. Dan Holcomb, and Prof. Brian Levine for agreeing to be part of my committee. I am greatly indebted to Prof. Levine for providing vital feedback as part of my committee. I want to thank Prof. Burleson and Prof. Holcomb for their constructive inputs, which have played a significant part in shaping this research. Special thanks to Prof. Holcomb for always giving great feedback whenever I asked for any guidance.

Also, I would like to thank my lab-mate Vinay Patil, for his contributions to various projects. Technical discussions with him have been a great learning curve. Thanks to all of our current and past lab members for creating a great work culture and ambiance. I will forever cherish my experiences and the friends I made at Amherst. Without their support, my Amherst experience would not be the same.

Finally, my graduate life would not have been possible without the sacrifices of my parents. I will forever be thankful for their unconditional love and support.

ABSTRACT

ENABLING IOT AUTHENTICATION, PRIVACY AND SECURITY VIA BLOCKCHAIN

FEBRUARY 2021

MD NAZMUL ISLAM

B.Sc., BANGLADESH UNIVERSITY OF ENGINEERING & TECHNOLOGY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Sandip Kundu

Although low-power and Internet-connected gadgets and sensors are increasingly integrated into our lives, the optimal design of these systems remains an issue. In particular, authentication, privacy, security, and performance are critical success factors. Furthermore, with emerging research areas such as autonomous cars, advanced manufacturing, smart cities, and building, usage of Internet of Things (IoT) devices is expected to skyrocket [61]. A single compromised node can be turned into a malicious one that brings down whole systems or causes disasters in safety-critical applications. This dissertation addresses the critical problems of *(i)* device management, *(ii)* data management, and *(iii)* service management in IoT systems. In particular, we propose an integrated platform solution for IoT device authentication, data privacy, and service security via blockchain-based smart contracts. We ensure IoT device authentication by blockchain-based IC traceability system, from its fabrication to its end-of-life, allowing both the supplier and a potential customer to verify an IC's provenance. Results show that our proposed consortium blockchain framework implementation in Hyperledger Fabric for IC traceability achieves a throughput of 35 transactions per second (tps). To corroborate the blockchain information, we authenticate the IC securely and uniquely with an embedded Physically Unclonable Function (PUF). For reli-

able Weak PUF-based authentication, our proposed accelerated aging technique reduces the cumulative burn-in cost by $\sim 56\%$. We also propose a blockchain-based solution to integrate the privacy of data generated from the IoT devices by giving users control of their privacy. The smart contract controlled trust-base ensures that the users have private access to their IoT devices and data. We then propose a remote configuration of IC features via smart contracts, where an IC can be programmed repeatedly and securely. This programmability will enable users to upgrade IC features or rent upgraded IC features for a fixed period after users have purchased the IC. We tailor the hardware to meet the blockchain performance. Our on-die hardware module design enforces the hardware configuration's secure execution and uses only 2,844 slices in the Xilinx Zedboard Zynq Evaluation board. The blockchain framework facilitates decentralized IoT, where interacting devices are empowered to execute digital contracts autonomously.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	v
ABSTRACT	vi
LIST OF TABLES	xiii
LIST OF FIGURES	xiv
 CHAPTER	
1. INTRODUCTION	1
1.1 Blockchain	1
1.1.1 Blockchain Classification	2
1.1.2 Smart Contracts	3
1.1.3 Security of Blockchain and Smart Contracts	3
1.2 PUFs	4
1.2.1 Strong PUFs	4
1.2.2 Weak PUFs	4
1.3 Scope of this Work	5
1.4 Dissertation Outline	6
1.5 Collaborators	6
 2. ENABLING IC TRACEABILITY VIA BLOCKCHAIN PEGGED TO EMBEDDED PUF	7
2.1 Introduction	7
2.2 Related Works	9
2.2.1 RFID-based Traceability	10
2.2.2 Package ID-based Traceability	10
2.2.3 Chip ID-based Traceability	11
2.2.4 Blockchain-based Traceability	11
2.3 Motivation	12

2.3.1	Transparency and End-to-end Visibility with Blockchain	12
2.3.2	Blockchain vs Private Databases for IC Traceability	13
2.3.3	Blockchain vs Centralized Database for IC Traceability	13
2.4	Proposed IC Traceability Protocol based on Customized Blockchain Transactions	13
2.4.1	Approach	14
2.4.2	Transaction and Blockchain Creation	15
2.4.2.1	Ownership addresses and keys	15
2.4.2.2	Transaction customization	16
2.4.2.3	Incorporating a transaction to a block and creating a blockchain	17
2.4.3	Protocol for Ownership Transfer	18
2.4.4	Protocol Demonstration and Discussion	20
2.5	Proposed IC Traceability Protocol based on Smart Contracts	20
2.5.1	System Requirements and Smart Contract Implementation	21
2.5.1.1	Ownership Keys and Addresses	22
2.5.1.2	Smart Contract Implementation by the Consortium	22
2.5.2	Proposed IC Traceability Protocol.....	25
2.5.2.1	Enrollment of a Device by the Manufacturer.....	25
2.5.2.2	Procedure for Ownership Transfer.....	25
2.5.3	Authentication by Strong and Weak PUFs	26
2.5.3.1	Authentication via Strong PUFs	27
2.5.3.2	Authentication via Weak PUFs	28
2.5.4	Counterfeit Detection by the Proposed Traceability Protocol.....	30
2.5.4.1	Recycled & Remarked ICs	30
2.5.4.2	Overproduced ICs.....	30
2.5.4.3	Cloned ICs and Tampered ICs	31
2.5.5	Protocol Demonstration in Ethereum Blockchain	32
2.5.6	Protocol Demonstration in Hyperledger Fabric	33
2.5.6.1	Blockchain Network Model	33
2.5.6.2	Implementation of the Chaincode and Access Control Policies	33
2.5.6.3	Performance Evaluation	34
2.5.6.4	Operational Cost of Consortium Blockchain	36
2.6	Analysis of the Protocols	36

2.6.1	Security	36
2.6.2	Privacy	38
2.6.3	Reliability	38
2.6.4	Performance	39
2.6.5	Practicality of the Proposed Protocol	40
2.7	Limitations and Discussion	42
2.8	Concluding Remarks	42
3.	PRESERVING IOT PRIVACY IN SHARING ECONOMY VIA SMART CONTRACTS	43
3.1	Introduction	43
3.2	Threat Model and Motivation	44
3.2.1	Threat Model	44
3.2.2	Motivation	45
3.3	Related Works	46
3.4	Proposed Methodology	47
3.4.1	Implementing Smart Contract	47
3.4.2	Transferring Tenancy to a Tenant	47
3.4.3	Establishing a Shared Encryption Key	49
3.4.4	Encrypting IoT Data with the Shared Encryption Key	50
3.4.5	Change of Encryption Key after Tenancy Period	51
3.5	Hardware Collateral for the Smart Contract	51
3.6	Protocol Demonstration and Discussion	51
3.7	Limitations and Discussion	52
3.8	Concluding Remarks	53
4.	IMPROVING RELIABILITY OF WEAK PUFs VIA ACCELERATED AGING	54
4.1	Introduction	54
4.2	Background and Motivation	55
4.2.1	Weak PUF	55
4.2.2	PUF Reliability	56
4.2.3	Temporal Majority Voting	57
4.2.4	Negative Bias Temperature Instability	58
4.2.5	Burn-In (Accelerated Aging)	60
4.2.6	PUF Reliability using accelerated aging	61
4.3	Methodology	61
4.3.1	Weak PUF System Design	61
4.3.2	Process Variation and Error Rate	63

4.4	Burn-in time reduction	66
4.4.1	Weak PUF Designs	66
4.4.2	Thermal Noise Errors	69
4.4.3	Error Rate vs Mismatch	69
4.4.4	Modeling Process Variation	71
4.4.4.1	Planar MOSFET	71
4.4.4.2	FinFET	72
4.4.5	Heterogeneous Error Model	73
4.4.6	Cumulative Burn-in Time	73
4.5	Concluding Remarks	76
5.	PMU-TROJAN: ON EXPLOITING POWER MANAGEMENT SIDE CHANNEL FOR INFORMATION LEAKAGE	77
5.1	Introduction	77
5.2	Related Works	80
5.3	Background	81
5.3.1	Hardware Trojans	81
5.3.2	Dynamic Voltage and Frequency Scaling (DVFS)	82
5.3.3	Power Management Unit (PMU)	83
5.3.4	Remote Server Maintenance: Integrated Management Card	83
5.4	Proposed Methodology	84
5.4.1	Threat Model	84
5.4.2	Trojan Insertion	84
5.4.3	Trojan Activation	85
5.4.4	Trojan Operation	87
5.5	Experimental Results	88
5.6	Trojan Detection	88
5.7	Concluding Remarks	89
6.	REMOTE CONFIGURATION OF INTEGRATED CIRCUIT FEATURES VIA SMART CONTRACTS	90
6.1	Introduction	90
6.2	Related Works	93
6.3	Motivation	93
6.3.1	Motivation for Post-production IC Customization	93
6.3.2	Motivation for Smart Contract-based Solution	94
6.4	Proposed Protocol and Smart Contract Implementation	96

6.4.1	System Design	96
6.4.2	Implementation of Smart Contracts	96
6.4.2.1	Register Device	96
6.4.2.2	Upgrade Configuration	98
6.4.2.3	Query Configuration	99
6.5	Proposed Hardware-software Co-design for Remote Configuration	100
6.5.1	Software	100
6.5.2	Hardware	101
6.5.2.1	Hardware configuration module (HCM)	101
6.5.2.2	Timestamp module	102
6.5.2.3	Crypto module	103
6.6	Implementation and Protocol Demonstration	103
6.6.1	Implementation of the Smart Contract	104
6.6.2	Implementation of Feature Configuration by Hardware	104
6.6.2.1	Target device and the Hardware Architecture	104
6.6.2.2	Gateway	105
6.6.2.3	Protocol Demonstration	105
6.7	Results and Evaluation	106
6.8	Security Analysis of the Protocol	107
6.9	Limitations and Discussion	109
6.9.1	Performance	109
6.9.2	Vulnerabilities in Smart Contracts	111
6.9.3	Transactional Privacy	112
6.10	Concluding Remarks	112
7.	CONCLUSION	113
7.1	Summary of Contributions	113
7.2	Future Works	114
 APPENDICES		
A.	MULTI-HOST HYPERLEDGER FABRIC IMPLEMENTATION	115
B.	SAMPLE CODE SNIPPET FOR TRIGGERING THE PMU-TROJAN	129
BIBLIOGRAPHY		130

LIST OF TABLES

Table	Page
2.1 Transaction format for ownership transfer. The Size is for a single IC ownership transfer.	16
2.2 Structure of a Block	18
2.3 Operation cost for a chip in supply chain.	32
2.4 Comparative Performance Analysis of the Proposed IC Traceability Protocol.	40
3.1 Operation cost for the proposed smart contract transactions.	52
4.1 Implementation details for various Weak PUF design configurations	67
4.2 Results for reduction in Cumulative Burn-in time for various PUF configurations	74
5.1 Frequency and voltage level change	88
6.1 Estimates of transaction fees for various operations.	104
6.2 Resource utilization of our hardware modules.	107
6.3 Device utilization summary.	107
B.1 Code Snippet for Triggering the PMU-Trojan	129

LIST OF FIGURES

Figure	Page
2.1 Detailed diagram of the ownership transfer protocol. *For the manufacturer, this transaction will be the <i>genesis transaction</i>	19
2.2 Proposed approach for IC traceability from manufacturer to the end-user. OEM: Original Equipment Manufacturer.	21
2.3 Smart contract implementation for IC traceability.	22
2.4 Detailed diagram of the ownership transfer protocol. Transactions requiring payment of fees are drawn with solid black lines.	26
2.5 Hardware authentication module for generating cryptographic key from PUF embedded in an SoC [91].	28
2.6 (a) Device enrollment and (b) authentication process via Weak PUFs.	29
2.7 Chip activation using active metering [181] before enrollment of authentication data in blockchain. Green arrow indicates secure link established by the designer and manufacturer’s public key infrastructure. ATE: Automated Test Equipment.	31
2.8 Hyperledger Fabric network model for our proposed protocol.	33
2.9 Hyperledger Caliper benchmark result.	34
2.10 Throughput and latency of the <code>registerDevice()</code> transaction in the proposed blockchain implementation using Hyperledger Fabric.	35
2.11 Schematic of typical in-vehicle network architecture of a modern automobile. All inter-bus communication is done exclusively only over the gateway ECU. CAN - Controller Area Network, MOST - Media Oriented Systems Transport.	41
3.1 Threats associated with accessing indoor IP camera by home-owner from remote location in a home-sharing economy scenario.	45
3.2 Block diagram of a Trusted Platform Module (green block) embedded into the camera’s software architecture [8].	46

3.3	Detailed diagram showing the role of all entities in the proposed IoT privacy protection protocol.	48
3.4	Detailed diagram of the tenancy transfer protocol.	49
3.5	Detailed diagram of the privacy protection protocol: (1) the smart contract notifies the IP camera about the tenancy change, (2) IP camera computes the symmetric key from tenant's public key and encrypts video data, (3) tenant calculates the symmetric key and decrypts the video data. The dotted lines indicate inaccessible data.	50
4.1	SRAM-like cross-coupled inverter PUF cell (<i>Ref</i>) [213]	56
4.2	Error rate reduction due to Temporal Majority Voting	59
4.3	Block Diagram of the proposed reliability enhancement scheme (from [112], our earlier paper)	62
4.4	Flowchart illustrating the operation of <i>Burn-in Optimizer</i>	64
4.5	SRAM-like cross-coupled inverter PUF cell (<i>Ref</i>) [213]	65
4.6	Error rate correlation with PUF cell threshold voltage mismatch.	66
4.7	Modified parallel active loads-based PUF design (<i>D1</i>) [166]	68
4.8	Modified current mirror-based PUF design (<i>D2</i>) [166]	68
4.9	Error rate correlation with PUF cell threshold voltage mismatch for alternate Weak PUF designs ($\{D1, D2\}$) based on [166]	70
4.10	Error rate correlation with PUF cell threshold voltage mismatch for <i>Ref</i> under nominal and boosted supply voltage (1.2 V)	71
4.11	Error rate correlation with PUF cell threshold voltage mismatch for planar MOSFET (<i>Ref</i>) and FinFET (<i>F1</i>) designs	72
5.1	Minimalistic hardware Trojan example.	81
5.2	Block diagram of MPSoC embedded with PMU [81].	82
5.3	MPSoC infected with hardware Trojan.	85
5.4	Flow chart illustrating the proposed methodology.	86
5.5	An example attack scenario at data center.	87

6.1	An example application that stores a user's credit card information is installed in the user's device and vendor repository. Before upgrading the chip configuration, the device and the vendor repository identify each other. Then, the credit card stored by the cloud service is charged after the user upgrades the device configuration.	91
6.2	User's device uses smart contracts in Ethereum to rent upgraded feature configurations.	95
6.3	Proposed protocol for remote configuration of IC features using smart contracts. Transactions requiring payment of fees are drawn with solid black lines.	97
6.4	Proposed hardware design for remote configuration of IC features (FC: Functional Component).	102
6.5	Experimental setup for demonstrating feature configuration using proposed hardware design. The Raspberry Pi works as the gateway.	105
6.6	HTTP POST Content sent from the Zedboard via ESP8266-01.	106
6.7	Authentication of the manufacturer, integrity and confidentiality of the configuration in our protocol.	109
6.8	Transaction times of <code>upgradeConfiguration()</code> for increasing <code>gas price</code> in the Main Ethereum network with a <code>gas limit</code> of 83500.	110
6.9	Cumulative density function (CDF) for the <code>upgradeConfiguration()</code> transaction times in the Main Ethereum network with a <code>gas price</code> of 40×10^{-9} Ether and a <code>gas limit</code> of 83500.	111

CHAPTER 1

INTRODUCTION

The increasing availability of computing and communication infrastructure has allowed electronic devices surrounding us to be interconnected to work smartly, collectively called the Internet of Things (IoT). The intelligent, interconnected IoT devices bring significant and indispensable convenience and intelligence in every aspect of our lives. However, IoT is still confronting several challenges and manifesting a series of issues that we need to address urgently. Counterfeit hardware, data privacy, software faults, system management difficulties, security issues during communication, and remote device service management are vital issues for current IoT infrastructure. Although several works have proposed new standards, lightweight protocols, and novel frameworks, some IoT challenges remain unresolved.

To address the challenges of implementing IoT infrastructure, a holistic new approach and technical solution are necessary. The inherently distributed nature of IoT necessitates the design of distributed solutions and novel architecture models. In this dissertation, we propose that the blockchain technology can address some of the weaknesses mentioned above. In particular, we propose an integrated platform solution for IoT device authentication, data privacy, and service security via blockchain. This chapter first explains a few backgrounds and the motivation behind the work. Derived from this motivation, we present the scope of the work. Finally, we give an overview of the structure of this dissertation.

1.1 Blockchain

Blockchains are distributed ledgers that maintain a continuously growing list of ordered records called *blocks* [157]. A distributed ledger is a type of shared database, replicated, and synchronized among the members of a decentralized network. Each block contains a list of transactions, timestamp, nonce, and a link to the previous block, forming a chronological chain. A transaction is a transfer of any asset with a value from the current owner to a new

owner. To submit transactions to the blockchain, each node uses a pair of private and public keys. First, the node constructs and signs a transaction and broadcasts it to the blockchain network. Each blockchain node validates any transactions it receives before broadcasting them to its peers, dropping invalid transactions. The miner nodes in the network construct a new block to record these valid transactions and broadcast it to their peers, who verify it before appending it to the ledger. This process repeats continuously. To resolve different states, or “forks” in the network, each blockchain employs a mechanism, known as *consensus* [206]. Figure 2.2 illustrates a blockchain formed from a sequence of blocks, each containing multiple transactions.

Often introduced as the technology behind Bitcoin [157, 191], blockchain has potential applications in numerous industries beyond financial services [123, 202]: from real estate [156, 204] and health-care [122] to utilities [37, 131], the government sector [215], and IoT [72, 100]. These applications have been possible because of its decentralized nature. Applications that could previously run only through a trusted intermediary can now operate without a central authority and achieve the same functionality. The distributed ledger facilitates business networks wherever anything of value needs to be tracked and traded without requiring central management.

1.1.1 Blockchain Classification

There are three types of blockchain: public, private, and consortium. Public blockchains are accessible to every Internet user. The public nature stems from the fact that everyone in the blockchain network can freely and unconditionally participate in the *consensus* process. A private blockchain is a blockchain where write permissions are kept centralized to one organization [54].

In a consortium blockchain, a pre-selected group of organizations control the consensus process instead of allowing any user within a blockchain network to participate in the consensus process or allowing a single organization to have full control. The right to read the blockchain can be made public or restricted to the consortium blockchain participants. A hybrid approach is also possible that allows public members to make a limited number of queries [54].

1.1.2 Smart Contracts

Smart contracts are blockchain-powered autonomous computer programs that, once started, execute automatically the conditions defined beforehand, such as the verification, facilitation, or enforcement of the negotiation or performance of a contract [203]. Smart contracts give us distributed trustworthy computations on a blockchain platform. They translate the existing contractual clauses into embedded hardware and software so that it can self-verify that conditions have been met to execute the contract [55]. Smart contracts contain code functions and interact with other contracts, make decisions, store data, and send tokens/money to others.

The main benefit of deploying smart contracts in a blockchain is the blockchain’s assurance that the contract terms cannot be modified. The blockchain makes it extremely expensive to modify or tamper the contract terms. It generates the confidence and security necessary to automate the declarative phrases without resorting to a third party.

1.1.3 Security of Blockchain and Smart Contracts

Despite the widespread adoption of blockchain-based solutions, the blockchain systems encounter several security threats. The two most fundamental attacks against blockchain systems are the *double-spend* attack [164] and the *selfish mining* attack [48]. In a *double-spend* scenario, an attacker creates a transaction that moves funds to a merchant’s address. After the transaction appears in the newest block on the main branch, the attacker takes possession of the purchased goods. The attacker then releases two blocks immediately, using his mining power, with a transaction in the first that transfers the funds to a second attacker-owned address. In this way, the attacker can have the goods and his coin back.

In a *selfish mining* attack, the *selfish miner* keeps discovered blocks secret and continues to mine on top of them, hoping to gain a larger lead on the public chain, and only publishes the selfish chain to claim the rewards when the public chain approaches the length of the selfish chain. Though risking some secret blocks’ rewards, once the selfish chain is longer than its competitor, the selfish miner can securely invalidate honest miners’ competing blocks. Accordingly, the overall expectation of the selfish miner’s relative revenue increases.

Apart from the *double-spend* attack and the *selfish mining* attack, some of the well-known attacks on blockchains include the eclipse attack [48], private key leakage, vulnerabilities in smart contracts [35]. Several solutions have been proposed in the literature to counter these threats that increase the security, efficiency, and transparency of blockchain systems [47, 163] and ensure smart contracts' security [46] and privacy [127, 225].

1.2 PUFs

Physical unclonable functions (PUFs) harness the intrinsic disorder in an IC introduced during the fabrication process and provide a set of *unique* input to output mappings, called challenge-response pairs (CRPs) [65, 200]. Based on the implementation, a PUF that provides a limited set of CRPs is classified as a *Weak* PUF, while a design that can produce an exponential number of CRPs is called a *Strong* PUF.

1.2.1 Strong PUFs

Strong PUFs can be leveraged for an authentication mechanism to uniquely identify an IC and detect tampering, impersonation, or substitution of such components. The salient features of a Strong PUF are the *uniqueness* of responses across different PUFs to the same challenge, ensuring the *reliability* of a response in the presence of noise, and the *unclonability* in the form of being resistant to model-building attacks [183]. Even if a Strong PUF exhibits ideal uniqueness and unclonability [212], it can still be susceptible to noise. Hence, authentication using a Strong PUF requires acquiring multiple CRPs to check against a database and applying a threshold to account for noise. The number of CRPs required depends on the total population of possible devices that need to be distinguished and the noise that may affect the system [175].

1.2.2 Weak PUFs

Weak PUFs can be leveraged for secure cryptographic key generation to combat semiconductor device counterfeiting, theft of service, and tampering. Weak PUFs rely on intrinsic process variations to produce repeatable and unique fingerprints. This fingerprint is further processed to generate a unique cryptographic key. For generating a reliable key, the fin-

gerprint needs to be reproducible over time, even under changing environmental conditions. However, noise in the system can affect the fingerprint and introduce errors. To alleviate noise and generate stable keys, literature works have proposed several solutions, such as error-correcting codes [64], accelerated device aging [112, 113], built-in self-test [43], etc.

1.3 Scope of this Work

The Internet of Things, refers to the billions of physical devices worldwide connected to the Internet, all collecting and sharing data. Centralized approaches to building an Internet of hundreds of billions of things are expensive, do not scale, lack privacy, pose security challenges to large-scale enterprises, and are not designed for business model endurance [111]. The use of blockchain as a reliable, distributed ledger of transactions and peer-to-peer communication among participating nodes can offer greater scalability and security for the IoT. It can offer multidimensional reinforcements for the IoT infrastructure. In this work, we propose blockchain-based solutions for building smart mechanisms in IoT systems contributing to *device management*, *data management*, and *service management*.

We propose a blockchain-based IC traceability system for IoT device *authentication*. Our proposed traceability system allows both the supplier and a potential customer to verify an IC’s provenance, from its fabrication to its end-of-life. To corroborate the blockchain information, we authenticate the IC securely and uniquely by embedded Physically Unclonable Function (PUF). We propose a method to enhance the reliability of Weak PUFs via post-Silicon accelerated aging.

IoT devices send out sensitive information that must be protected from unauthorized access, usage, or disclosure. We propose a blockchain-based protocol that integrates the *privacy of data* generated from the IoT devices by giving users control of their privacy. In an exemplary solution, we show how smart contracts can facilitate efficient IoT devices by automating their operations and decision making in sharing economies. This autonomous decision-making capabilities of blockchain-enabled IoT devices can eliminate privacy threats.

When using blockchain, the user’s private key is regarded as the identity and security credential. However, users often rely on third-party hosted nodes to access the wallet and check balances, initiate transactions, and more. Such reliance can be exploited by a side-channel

attack to extract the private key successfully. We propose a method where a co-tenant thread monitors the power management side-channel information from a thread affected by a hardware Trojan. Such a Trojan can leak private keys and disrupt digital transactions.

Blockchain and smart contracts provide transparency, longevity, and allow applications to minimize the need for a trusted arbiter. Facilitation of these three aspects can enable remote, *secure device service* management. This work proposes a protocol for remote configuration of IC features, where a user can program an IC repeatedly and securely using smart contracts. Our proposed integrated platform solution facilitates proof of data integrity, device authentication, and secure features reconfiguration. A fundamental primitive to achieve this goal is establishing a trust-base that cannot be tampered with. Storing data by using blockchain secured with PUF derived keys provides an assurance that data has not been tampered with, in addition to providing traceability and transparent auditing capabilities.

1.4 Dissertation Outline

This dissertation document is organized as follows. Chapter 2 discusses existing IC authentication methods and presents a method of IC authentication via blockchain pegged to embedded PUFs. In Chapter 3, we propose a methodology to eliminate privacy threats from IoT-enabled telematics devices via blockchain-based smart contracts. In Chapter 4, we present a technique to improve the reliability of PUFs by accelerated aging. In Chapter 5, we propose a hardware Trojan that can leak private keys using a power management side channel and a software solution to suppress the side channel. In Chapter 6, we propose a remote configuration of IC features via smart contracts. We conclude the dissertation in Chapter 7 with insights for future work.

1.5 Collaborators

All five studies were conducted under the supervision of Professor Sandip Kundu. The study featured in Chapter 2 was a collaboration with Vinay Patil. Chapter 3 was done in collaboration with Arman Pouraghily and Professor Tilman Wolf. The study featured in Chapter 4 was done in collaboration with Professor Daniel Holcomb and Vinay Patil.

CHAPTER 2

ENABLING IC TRACEABILITY VIA BLOCKCHAIN PEGGED TO EMBEDDED PUF

2.1 Introduction

The globalization of semiconductor manufacturing has increased the risk of tampered and counterfeit products from manufacturing to distribution and field use. Malicious actors can counterfeit, tamper with, or re-package ICs, and introduce compromised ICs into the supply chain [90, 103]. This compromised supply chain exacts financial loss to legitimate suppliers [168] and poses a security risk for safety-critical applications, like defense [30, 99, 104]. In the effort to stem the tide of counterfeit electronics, *traceability* has long been established and promoted [154]. Traceability plays a crucial role in securing the supply chain by counterfeit electronic parts avoidance, detection, mitigation, and disposition. Hence, it is necessary to develop a secure IC *traceability* system that can allow both the supplier and a potential customer to verify its provenance.

Traceability refers to the combination of the ability to know the current possession of a product at all times (*track*) and the ability to find the origin, ownership history, time spent at each point (*trace*), by means of recorded identifications [114, 117]. *Tracing* allows a customer to establish a product's provenance with a high degree of confidence. *Tracking* can help suppliers minimize counterfeit products and offer value-added services, like a product recall, during the life-cycle of a product.

Prior works proposed implementing a centralized system with a governing third party to empower supply chain traceability to ensure data and transaction transparency over a product's lifetime. The governing third party is commissioned to create a centralized data storage to enable a flow of trusted information [94, 159]. However, relying on a governing third party to broker all data about every product's supply chain may create a single point of weakness and inherent bias, fraud in the system. If the party were the brand itself, or the

most powerful actor in the supply chain, it would ultimately be responsible for only its bottom line; this could lead to selective disclosure or, worse, extortion. If a third party gathered the supply chain data, it would have to be totally unbiased and adequately incentivized to deliver the system’s technical capability.

However, third parties like NGOs or industry associations would become a single point of weakness; this would make them and their operations a vulnerable target for bribery, social engineering, or targeted hacking. For example, GS1 is a standards organization trying to solve supply chain problems by uniquely identifying and accurately capturing information about products. However, according to recent Data Quality pilots, data accuracy was a growing problem — about 50% of the data surveyed was inaccurate [7]. Besides, GS1 has, of late, been restricting access to the centralized product database, excluding consumers and prioritizing corporations [24]. Moreover, although few supply chains are integrated, their supporting information systems can be heavily fragmented [158]. Hence, there is a need to come up with a more robust approach for developing IC traceability. Potential solutions must also account for legitimate re-sale of devices during the product lifetime.

In this chapter, we propose novel IC traceability protocols via a blockchain-based ownership management system. Blockchains can enable the IC ownership transfer information to be verified, recorded and make it infeasible for any malicious party to alter or challenge the legitimacy of the information recorded. Thus, this verifiable and shared ledger can enable an IC’s identification and traceability throughout the supply chain and its deployment lifetime.

However, including just the ownership and simple IC information in the blockchain is not sufficient enough. To corroborate a record in the blockchain against a physical device, the device must be authenticated securely and uniquely. For this purpose, we utilize physical unclonable functions (PUFs) as the hardware root-of-trust. We propose securely and uniquely identifying an IC using both Strong and Weak PUFs to corroborate the blockchain information in this work. This work’s primary focus is specifically on securing the IC supply chain, although we can apply the methodology to any electronic device supply chain. The major contributions of this work are:

- Proposing a blockchain-based open traceability protocol that IC suppliers can use to *track* and detect any counterfeits in supply chain.
- Enhancing the traceability protocol, such that, customers can *trace* and verify IC's provenance.
- Proposing a smart contract which automates self-execution of the enrollment, authentication, and ownership transfer of an IC.
- Developing a comprehensive ownership management system which enables each party along the supply chain to prove the ownership of PUF-embedded IC and transfer it to a new owner.
- Demonstrating the proposed solution in Ethereum blockchain. The code has been made publicly accessible in Github [2].

We outline this chapter as follows. Section 2.2 discusses the related works on traceability solutions for counterfeit detection in the IC supply chain. After that, we describe our motivation behind a blockchain-based solution for IC traceability in Section 2.3. In the following two sections, we propose two novel IC traceability protocols via blockchain pegged to embedded PUFs. The first protocol is based on customized blockchain transactions (Section 2.4), while the second protocol is based on smart contracts (Section 2.5). Section 2.6 presents a detailed analysis of the proposed protocols regarding security, privacy, reliability, performance, and scalability. Finally, we discuss the limitations of the proposed protocols in 2.7 and conclude the chapter in Section 2.8.

2.2 Related Works

Several traceability solutions have been proposed in the literature for counterfeit detection in the IC supply chain. These solutions resort to RFIDs, various kinds of chip IDs, package IDs, blockchain, etc. In this section, we present the literature works which proposed various IC traceability solutions.

2.2.1 RFID-based Traceability

RFID-based counterfeit detection has been proposed in [75, 190, 197, 223]. Schuster *et al.* proposed an RFID-based track and trace solution using EPC (Electronic Product Code) Network infrastructure [126]. Staake *et al.* proposed an extension of EPC Network infrastructure with an EPC Product Authentication Service to provide secure authentication functionalities [197]. Shi *et al.* proposed a Batch Clone Detection (BCD) scheme, which performs the clone tag detection at a batch level [190]. The proposed scheme reduces storage and computational overhead on the centralized detection server side. Elkhayaoui *et al.* proposed a new protocol, CHECKER, for counterfeit detection in RFID-based supply chains through on-site checking [75]. In this protocol, RFID readers can verify product genuineness by checking the product's path's validity, while RFID-equipped products travel through the supply chain. However, RFID-based technologies cannot provide truly secure solutions for supply chain traceability because an adversary can easily copy one RFID tag's unique identifier to another tag.

2.2.2 Package ID-based Traceability

Several traceability solutions proposed affixing a unique ID in package (SHIELD [62]), on the package (DNA marking [154], nanorods [128], QR codes) of each component to track it as it moves throughout the supply chain. For verifying the trustworthiness of an electronic component, DARPA initiated a program called SHIELD (Supply Chain Hardware Integrity for Electronics Defense) [62]. The program proposes to embed a dielet into host packages of legitimately produced ICs, without disrupting or harming the system. A secure remote server stores the information for identification and authentication, such as cryptographic key and serial ID for each dielet. Passive sensors inside the dielet can record any malicious behavior as tampering evidence. Jin *et al.* proposed an improved protocol that resists the try-and-check attack in DARPA's example authentication protocol [118]. The proposed two-phase activation secures the untrusted transit between a trusted fabrication facility and an assembly facility. A true random number generator (TRNG) inside the dielets efficiently generates their cryptographic keys and serial IDs in parallel during a trusted fabrication process.

2.2.3 Chip ID-based Traceability

Several works have proposed traceability solutions using various kinds of chip IDs, such as PUFs or ECIDs [179]. Skudlarek *et al.* proposed a hardware root-of-trust based solution that enables connection of SoCs to a secure server [193] for tracking them at each step in the supply chain. An SRAM PUF provides each chip with its unique ID and DNA. The chip connects to a secure server at important stages in the supply chain. The server authenticates the chip and records that event, establishing a reliable audit trail for the chip’s progress and providing proof of provenance. Several Strong PUF-based RFID ICs have also been proposed previously for supply chain traceability [65, 176, 208]. In these methods, the back-end server stores a list of challenge-response pairs for RFID tags with embedded PUF. When the RFID tag communicates with a reader, the back-end server sends a challenge and the tag’s PUF reconstructs, and transmits the corresponding response. If the response matches the one stored by the back-end server, the tag is verified. In all these methods, RFID monitoring schemes need to have either a persistent online connection between supply chain partners and the back-end database or a local database on each partner site to perform authentication appropriately. However, this approach is not secure from a man-in-the-middle attack. ECID-based chip tracking solutions [179, 222] are not secure. As it is static and readable, it is easy to clone an ECID. Moreover, the chip vendors are silos. Individual companies do not share data across corporate boundaries and cannot build cross-party trust relationships.

2.2.4 Blockchain-based Traceability

Blockchains have been successfully deployed to enable the supply chain integrity of various commodities. For example, Walmart has proposed monitoring the food supply chain by tracking the temperature variations and the time taken to transit food commodities [185]. However, it is not straightforward to apply it to the electronics supply chain as the semiconductor industry has some unique characteristics compared to other industries. It would not be a secure solution to enable electronic products’ authentication and integrity only by the shipping time and packaging appearance.

In literature [89], the authors proposed a method to protect the information flow in IoT devices with blockchain. The proposed solution uses blockchain for storage, transparency,

auditability of the IoT data. The proposed solution explores its benefits to multi-factor authentication, software integrity, and continuous authentication of IoT devices. However, it does not facilitate traceability solutions and ownership management to ensure the integrity of the supply chain, from fabrication to the end of product-life, that can allow a customer to verify the provenance of a device or a system. Hence, a more comprehensive protocol is required to enable IC traceability, authenticate a device, and transfer the ownership for a secure supply chain.

2.3 Motivation

This work’s key novelty is developing a blockchain-based traceability protocol for ensuring that the IC supply chain can be fully tracked and traced from the time of fabrication to the end of product-life. Several works have been done to deploy blockchain across a wide span of industries: from finance [123] and health-care [122] to utilities [131], real estate [156], government sector [215], and IoT [72, 100]. To the best of our knowledge, no other works have used blockchain for IC traceability and supply chain integrity. In the next section, we describe why we need a blockchain instead of a centralized database for IC traceability.

2.3.1 Transparency and End-to-end Visibility with Blockchain

For secure IC authentication, it is required that (i) data transparency and trust is preserved, (ii) there is end-to-end traceability of ICs produced from an untrusted supply chain (originated from diverse suppliers who might be dispersed throughout the globe), and (iii) the chip identity is bound to the device. A blockchain-based traceability solution can increase transparency, add end-to-end visibility, maintain a single version of the truth about the supply chain, and provide cyber-attack resilience for data storage via no single point of failure. Blockchain establishes a trusted environment for all participants — manufacturers, distributors, retailers, and consumers, who can gain permissioned access to known and trusted information regarding their transactions’ origin and state.

2.3.2 Blockchain vs Private Databases for IC Traceability

If IoT device authentication data is kept into private databases by the designers (e.g., the scenario that each vendor maintains the authentication data of hardware components), various security and trust issues may arise. Firstly, these databases are maintained and updated by a database administrator (DBA). If a competitor can bribe the DBA, they can compromise the database. Secondly, the vendor performs authentication within a private network that is not interoperable with any other network. Thirdly, a system integrator may use a large number of different chips from different manufacturers. It would be inconvenient to validate the authenticity of all chips from various companies.

2.3.3 Blockchain vs Centralized Database for IC Traceability

If IoT device authentication data is kept in a centralized database, it is vulnerable to being modified by malicious insiders without being noticed. The corruption of the administrator will violate the trust and integrity of the whole network. As a result, to record the authentication data of ICs and build a secure and trustworthy authentication infrastructure, a database accessible to all the supply chain participants should be maintained. In this work, we explore an alternative solution to perform IoT authentication using blockchain.

We focus on the supply chain integrity from a different angle — an end-to-end framework to provide a comprehensive solution from the device fabrication stage to systems' end-of-life. Blockchain ensures the management of all necessary information in a trusted and decentralized manner. This decentralization enables all supply chain participants to track, verify, and authenticate devices. Time-stamped tracking information provides tamper-resistance and evidence. Blockchain also creates opportunities for more supply chain participants within the vertical to join the network.

2.4 Proposed IC Traceability Protocol based on Customized Blockchain Transactions

In this section, we present our proposed protocol for IC traceability via customized blockchain transactions [110]. First, we describe our approach and the key system requirements of the protocol. After that, we present the methodology of creating our customized

transaction in the blockchain for IC traceability. Finally, we explore ownership transfer details, which is the key part of our overall blockchain protocol.

2.4.1 Approach

The blockchain will contain a record of the relevant IC ownership transfer information, termed as a *transaction*, and PUF data, used for authentication, for each point of transfer over the device’s lifetime. This record also enables proof of ownership without an explicit need for a trusted intermediary. Furthermore, authorized parties can utilize the blockchain to authenticate, track, analyze, and provision chips.

Before detailing the protocol, we specify the key requirements for creating the blockchain and explain their necessity for enabling reliable tracing of ICs.

1. Only the legitimate IC manufacturers registered with a designated consortium can claim the initial ownership and write the relevant PUF data on the blockchain.
2. Only the IC’s current owner can create a new transaction for transferring the ownership to a new owner.

The first requirement prevents unauthorized parties, like counterfeiters, from falsely claiming ownership of an IC. This requirement can be ensured by verifying that the *first* transaction in the blockchain for ownership transfer (the *genesis transaction*) was created only by the registered IC manufacturer. We propose *consortium blockchain* for the verification in our protocol. The consortium can be formed of multiple semiconductor organizations, each of which operates a node in the blockchain. Examples of such organizations can be Semiconductor Industry Association (SIA), Joint Electron Device Engineering Council (JEDEC), etc. Validation of a transaction requires a set number of nodes to sign-off on it.

The second requirement prevents an unauthorized party from hijacking an IC’s ownership by creating a fraudulent transaction. Our protocol grants only the current owner the ability to create a new transaction. The information in the last transaction recorded in the blockchain establishes the current owner’s identity, and PUF authentication verifies the owner’s physical possession of the IC.

2.4.2 Transaction and Blockchain Creation

We leverage Bitcoin's idea [157], in which the possession of a user's balance can be proven in the blockchain. In particular, by borrowing the ideas presented in the "proof of possession of balance" used in Bitcoin, we introduce here the concept of "proof of possession of IC". In this section, we describe how our protocol customizes the blockchain transaction and transfers the ownership in a step by step process.

2.4.2.1 Ownership addresses and keys

All the potential owners of an IC are assigned their addresses, used during ownership transfer. Owners generate their addresses from ECDSA (Elliptical Curve Digital Signing Algorithm) public/private key pair. First, a random 256-bit private key is generated. Each transaction is supplemented with a digital signature created using the private key. The signature uniquely identifies the current owner as of the *seller*.

$$K_{priv} \in \{0, 1\}^{256}$$

Next, a 512-bit public key is generated from the private key using the ECDSA algorithm. The public key is used for verification of the transaction signature. The public key is not revealed until a transaction is signed, unlike most systems where the public key is made public.

$$K_{pub} = ECDSA_{512}(K_{priv})$$

Since the 512-bit public key is large, it is converted to a smaller address shared with others and utilized as a part of the blockchain transaction. The 512-bit public key is hashed using SHA-256, further hashed using RIPEMD-160 to generate the 160-bit address.

$$K_{address} = RIPEMD_{160}(SHA_{256}((K_{pub})))$$

The consortium plays a key role in authenticating participants, and it maintains a directory that binds the identity information of an associated party with a transaction address. Any blockchain-based identity management system can ensure the reliability of the user's entity information [36, 139, 178].

Table 2.1: Transaction format for ownership transfer. The Size is for a single IC ownership transfer.

	Field	Description	Size (Bytes)
Input(s)	<i>icCount</i>	Number of ICs	variable
	<i>serialNumber</i>	Identifier of an IC	variable
	<i>prevTxID</i>	The previous transaction reference	32
	<i>icInfo</i>	Other necessary information related to IC	variable
	<i>challenges</i>	Challenge to the PUF	variable
	<i>hashResponse</i>	Hash of the Responses from PUF	variable
	<i>signature</i>	Seller's signature	71
	<i>publicKey</i>	Seller's public key for verifying the signature	64
Output(s)	<i>value</i>	Transaction Value	1
	<i>publicKeyHash</i>	Buyer's Address	20

2.4.2.2 Transaction customization

The general structure of a transaction in our protocol is shown in Table 2.1. We denote the current and the new owners as the *seller* and *buyer*, respectively. A transaction input contains the reference to the previous transaction, seller's signature, seller's public address, and IC information. A transaction output contains the buyer's address and transaction value. The salient features of a transaction are described as follows:

- Our proposed protocol facilitates the seller to sell multiple ICs in the same transaction. The seller needs to specify the number of ICs to be sold in the *icCount* field.
- The *serialNumber* in the format serves as an identifier for an IC. This identifier (e.g., Electronic Product Code, or EPC) can be used to enable the verifier to look up the correct transaction for the IC being queried among a collection of ICs. Here, the serial number is being used to *identify*, and not as the primary means to *authenticate*. For each IC, the corresponding previous transaction hash is referenced in the *prevTxID* field. For a *genesis transaction*, this is set to *none*. Other important IC information is included in the *icInfo* field.

- Each IC's PUF challenge-response data is included in *challenge* and *hashResponse* field, respectively. For achieving reliable authentication on-field, the manufacturer includes the most stable CRPs in the transaction. The manufacturer can also generate *helper data* for error correction and achieving greater reliability. Literature works have shown that the helper data can be constructed to become known to an adversary without compromising the PUF response's secrecy, i.e., it does not need to be kept secret [69, 182]. The manufacturer can include it in the transaction during device enrollment or at the device side in insecure one-time programmable (OTP) non-volatile memory (NVM).
- A transaction includes the seller's signature and the public key in *signature* and *publicKey* field, respectively. The seller generates the public key from the private key. The hash of this public key must match the hash given in the previous transaction output (*publicKeyHash*). The public key is also used to verify the seller's signature. The signature is an ECDSA signature over a hash of a raw version of the transaction. The signature, combined with the public key, proves that the real owner created the transaction.
- The transaction output nominates the buyer's address in *publicKeyHash* field. Any future transaction by the buyer will require the relevant public key and signature.

2.4.2.3 Incorporating a transaction to a block and creating a blockchain

After signing a transaction, the seller sends it into the consortium blockchain network, where the nodes pick up the transaction and verify the signature cryptographically. After verification, the blockchain node ensures that the referenced transaction has not been spent in a different transaction to prevent *double spending*. Finally, all the verified transactions that are considered to have happened at the same time are placed in groups called blocks. Table 2.2 presents the structure of a block.

Each block has a reference to the previous block, and this is what places one block after another, forming a chronological chain. A consensus mechanism ensures that the creation and modification of data are agreed upon by all the nodes or a majority of the nodes. Each

Table 2.2: Structure of a Block

	Field	Description	Size (Bytes)
Header	prevBlockHash	The hash value of the previous block used as a pointer	32
	timeStamp	A Unix timestamp recording when this block was created	4
	nonce	The block-specific nonce to allow variations of the header and compute different hashes	4
Transactions	txnCount	Number of transactions in block	1
	transaction	List of verified transactions	variable

transaction in a block is tied to a unique identifier ($TxID$), a double SHA256 hash of the verified, signed transaction. A $TxID$ is used to look up a transaction in the blockchain and reference for future transactions.

2.4.3 Protocol for Ownership Transfer

Figure 2.1 illustrates the detailed system model of the proposed ownership transfer protocol. The protocol consists of several phases, which we describe as follows:

Sending previous Transaction ID to the buyer To allow the buyer to trace the IC supply chain information and authenticate it, the seller sends him the previous transaction ID ($TxID$) in the blockchain. The latest transaction information of an IC contains the current owner's address ($publicKeyHash$). Using the previous transaction ID, the potential buyer can prove the genuineness of the seller.

IC Verification by the buyer With the received transaction ID ($TxID$), the buyer can query the blockchain's previous transaction information. The buyer can verify the genuineness of the ownership and authenticate the IC from the blockchain information. The occurs in the following two steps:

a) Verifying the ownership information: The buyer can verify the genuineness of the current seller from the referenced previous transaction. Also, using $TxID$, the buyer can obtain the transaction information which has reference to its previous transaction, which

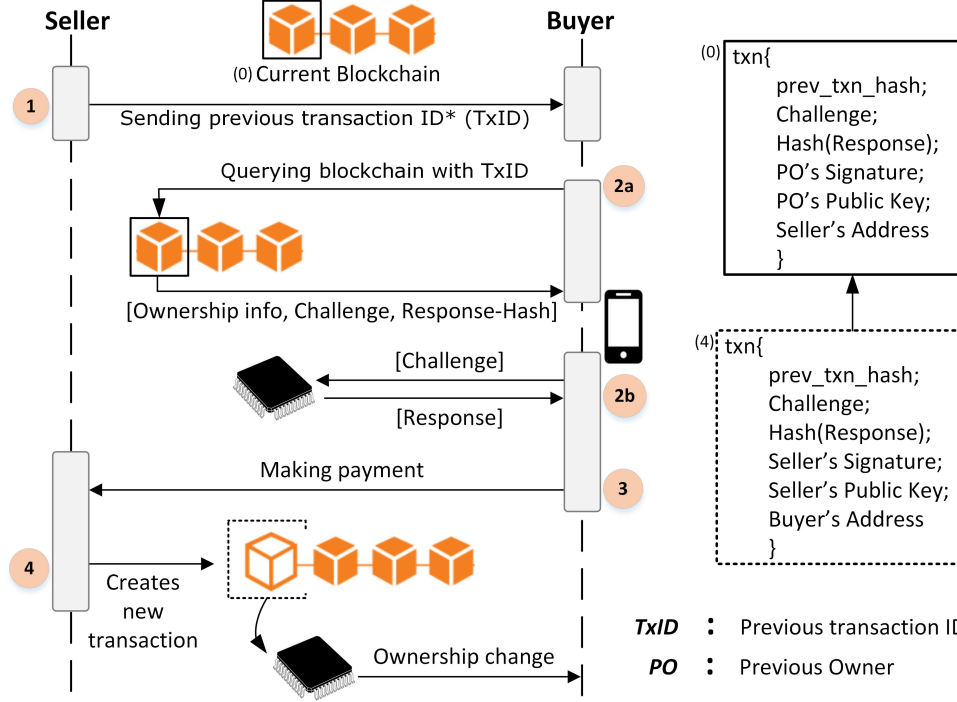


Figure 2.1: Detailed diagram of the ownership transfer protocol. *For the manufacturer, this transaction will be the *genesis transaction*.

again has reference to its previous information, and so on. In this way, the buyer can trace back along the supply chain to the IC's provenance.

b) Authenticating the IC: The buyer can retrieve challenges from the blockchain and apply them to the IC. The PUF embedded in the IC will give the corresponding responses. If the calculated hash of the responses matches the hash recorded in the blockchain, the IC is authenticated. The authenticator can be an API (blockchain accessor) or a complete program, which a user needs to run during the verification. For online/remote sales, the buyer/seller can designate a trusted agent/broker.

Making payment Once the IC is authenticated, and the seller's IC ownership is verified, the buyer makes a payment to the seller. The buyer can make the payment in blockchain-based cryptocurrency.

Creating a transaction After receiving the payment confirmation, the seller is now ready to issue the ownership transfer transaction. In a similar way described in section 2.4.2.2, the seller creates a transaction. Seller puts the buyer's address in the output field

of the transaction. If the transaction is validated and added to the blockchain, the buyer gets the IC ownership.

2.4.4 Protocol Demonstration and Discussion

We implemented the proposed protocol in Bitcoin *Testnet3* [1] and made the code available online [2]. We send the ownership information and the challenge-response pairs (CRPs) in the *Metadata* field of the *Testnet3* transaction. In our emulated blockchain, a *genesis transaction* can be created by only selected private-public keys. Any other private-public key pairs can create other transactions. Any transaction created with a valid private-public key pair (whose corresponding address is the recipient of the referenced previous transaction) is verified and incorporated into a block.

The number of PUF challenges required to distinguish 1 trillion ICs is ~ 1024 for a Hamming distance threshold of 10% [175]. For practical implementation, using, e.g., a 64-bit arbiter PUF [200], we take the set of challenges, and apply SHA-1 hash function to the corresponding responses. SHA-1 produces a hash digest of 160 bits. The *helper data* is also added to this CRPs for error correction and achieving greater reliability. Hiller *et al.* showed that the required *helper data* size for 165 PUF bits with the probability of 10^{-5} is 36 bits [96]. This PUF data adds a total of 260 bits (64-bit challenge, plus the 160-bit hash of the responses, and 36-bit *helper data*).

2.5 Proposed IC Traceability Protocol based on Smart Contracts

Our previous protocol proposed a blockchain solution for IC traceability, where we used customized transactions for logging supply chain information. We can enhance the proposed solution to facilitate transaction automation in an already existing blockchain platform. For automated execution of enrollment, authentication, ownership traceability, and ownership transfer of an IC, we propose a novel IC traceability protocol based on smart contracts [107]. The main advantage of deploying smart contracts in a blockchain is the blockchain's assurance that the contract terms cannot be modified. In our proposed smart contract, the contractual clauses regarding enrollment, authentication, and ownership transfer of an IC

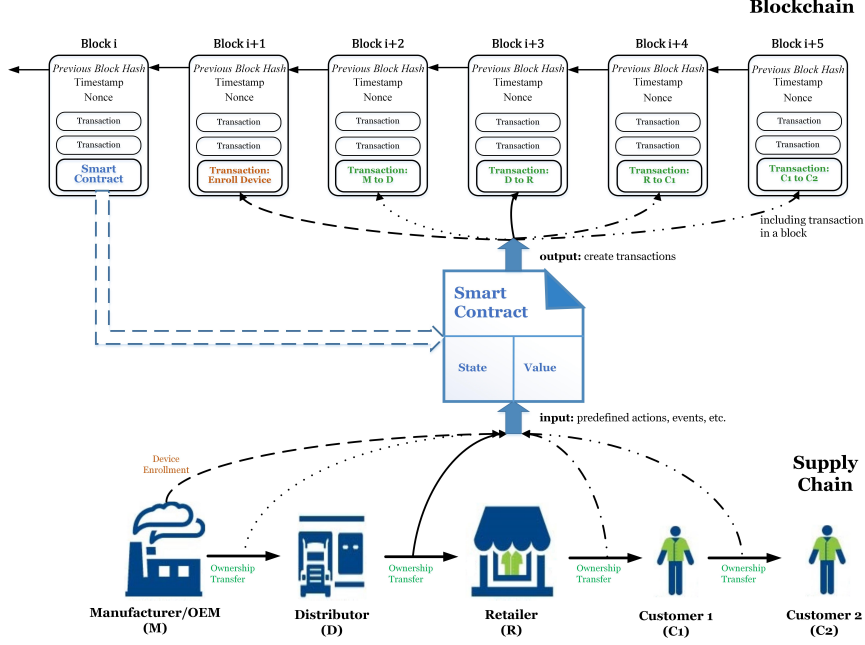


Figure 2.2: Proposed approach for IC traceability from manufacturer to the end-user. OEM: Original Equipment Manufacturer.

are translated into embedded hardware and software. The embedded hardware and software can self-verify that conditions have been met to execute the contract.

In this section, we present our proposed protocol for IC traceability via smart contracts. First, we introduce the system requirements and implemented smart contracts for our proposed protocol. Next, we present our proposed methodology to establish IC supply chain traceability using blockchain and embedded PUF. After that, we detail the authentication method used in our traceability protocol and discuss how the proposed protocol performs counterfeit avoidance and detection. Finally, we outline a demonstration of the protocol and discuss the practicality, limitations, and future directions.

2.5.1 System Requirements and Smart Contract Implementation

The key system requirements for IC traceability remains the same as described in Section 2.4.1. Both of the system requirements for IC traceability can be fulfilled by implementing a smart contract in the blockchain. Figure 2.2 presents our proposed approach for IC traceability using smart contracts. In our proposed protocol, a smart contract defines the condition for enrolling a new device by a manufacturer and transferring a device to a buyer

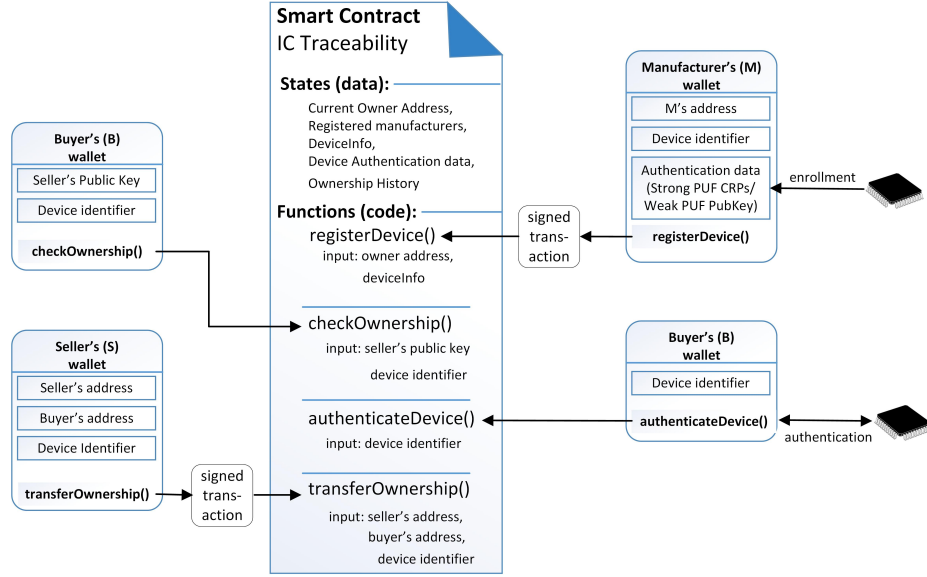


Figure 2.3: Smart contract implementation for IC traceability.

by the current owner. It also defines the function for checking the ownership and authenticating the device. In this section, we describe all the elements needed for implementing a smart contract.

2.5.1.1 Ownership Keys and Addresses

All the supply chain participants are registered with consortium blockchain and have their own private-public key pairs and addresses. This unique private-public key pairs and addresses are necessary for uniquely identifying a supply chain participant and preventing any forgery. The method for generating private-public keys and addresses is described in section 2.4.2.1. A user can manage personal keys and addresses using a digital wallet that can be used to perform any transaction. A digital wallet is a software and hardware, or specifically designed hardware, that holds the private-public keys and the address [79].

2.5.1.2 Smart Contract Implementation by the Consortium

To enable IC traceability, we have implemented smart contract, namely *Ownership Contract* (OC). The contract is created by the consortium to maintain uniform applicability and usability for all supply chain participants. After creating the smart contract, it is sent to the blockchain network as a transaction that assigns an address to the contract. After this

ALGORITHM 1: Pseudo-code of `registerDevice()` for registering a device claimed by a manufacturer.

Inputs: Manufacturer's address (*addrManufacturer*), and device information (*deviceInfo*)

```
if Message sender is in manufacturer's list then
    Specify owner of the device as addrManufacturer
    Register deviceInfo on the blockchain
else
    Do nothing
end
```

ALGORITHM 2: Pseudo-code of `checkOwnership()` for verifying the ownership information of a device.

Inputs : Seller's public key (*sellerPubKey*), and device identifier (*deviceIdentifier*)

Output: A boolean **True** or **False**

```
if (hash(sellerPubKey) == blockchain[identifierDevice].owner) then
    return True
else
    return False
end
```

initial transaction, the contract becomes a part of the blockchain forever, and its address never changes. Figure 2.3 presents our proposed smart contract for IC traceability. The smart contract, OC, provides the services for device registration, ownership information verification, device authentication, and ownership transfer. Next, we explain these services and how our proposed smart contract implements these.

Device Registration: For introducing a device into the supply chain, the device must be registered first. We implement this registration by the smart contract function `registerDevice()`. Algorithm 1 presents the pseudo-code of function `registerDevice()`. The function enrolls a device if the message sender is one of the manufacturers registered with the consortium. The *deviceInfo* in the function includes data for *identification* and *authentication*. The identification data can be a serial number for the device (e.g., Electronic Product code, or EPC). It is used to look up the targeted device being queried among a collection of devices. Here, the device identifier is being used to *identify*, and not as the primary means to *authenticate*. For the *authentication* purpose, the manufacturer includes PUF data as input to the function `registerDevice()`.

Ownership Verification: Before buying a device, a potential buyer needs to verify the ownership to confirm that the owner is a genuine one. We implement this verification by the smart contract function `checkOwnership()`. Algorithm 2 presents the pseudo-code of

ALGORITHM 3: `authenticateDevice()` for authenticating a device.

Inputs : Identifier of the device to be transferred (*deviceIdentifier*)

Output: A boolean `True` or `False`

set *deviceChallenge* = `blockchain[deviceIdentifier].Challenge`

get *deviceResponse* from the device after applying *deviceChallenge*

if (*deviceResponse* == `blockchain[deviceIdentifier].Response`) **then**
| return `True`

else
| return `False`

end

ALGORITHM 4: `transferOwnership()` for transferring the ownership of a device from seller to buyer.

Inputs: Buyer's address (*addrBuyer*), and device identifier (*deviceIdentifier*)

if (*addrMessageSender* == `blockchain[identifierDevice].owner`) **then**

| set `blockchain[identifierDevice].owner` = *addrBuyer*

else
| Do nothing

end

function `checkOwnership()`. This function verifies the ownership of the device against the seller's address. If the device with the provided identifier is owned by the seller, it returns `True`. Any potential buyer invokes this function when he/she wants to verify the ownership of the device. Additionally, the function can return the ownership history to the provenance.

Device Authentication: In our protocol, a potential buyer must authenticate a device before buying to confirm that it is authentic, not a counterfeit one. We implement this authentication capability by the smart contract function `authenticateDevice()`. Algorithm 3 presents the pseudo-code of `authenticateDevice()`. This function starts the device authentication process for a given device identifier. The process includes getting the challenge-response data for a particular *deviceIdentifier*, applying the challenge to the device via the verifier's wallet, calculates the response, and matching the challenge-response pairs.

Ownership Transfer: For transferring the ownership of a device, the smart contract implements the function `transferOwnership()`. Algorithm 4 presents the pseudo-code of `transferOwnership()`. This function transfers the device (*deviceIdentifier*) from the seller (*addrSeller*) to the buyer (*addrBuyer*). First, the function checks whether the message sender is the owner of the device with *deviceIdentifier*. If that is true, then the function assigns the *addrBuyer* as the new owner of the device.

2.5.2 Proposed IC Traceability Protocol

In this section, we present the details of the algorithmic procedures necessary to realize the proposed protocol. We outline the procedure for enrolling a device by the manufacturer, followed by the procedure for transferring the ownership, which is the key part of our overall traceability protocol.

2.5.2.1 Enrollment of a Device by the Manufacturer

When a manufacturer (M) wants to register/enroll a device, it sends a transaction `registerDevice()` to the smart contract OC (Figure 2.3). The manufacturer also sends necessary device information, such as the device’s identifier, authentication information (PUF data) in the transaction. We name the first transaction for the enrollment of a device as *genesis transaction*. The manufacturer can also enroll N number of devices in the same transaction by including all the devices’ corresponding information. This process facilitates the scalability of the protocol. As the transactor digitally signs all the blockchain transactions, any counterfeiter cannot illegally claim to be a non-authorized manufacturer.

2.5.2.2 Procedure for Ownership Transfer

Figure 2.4 illustrates the detailed system model of the proposed ownership transfer protocol. In this work, we denote the current and the new owners as the *seller* and *buyer*, respectively. The protocol consists of several phases described as follows.

Verifying the ownership information: To verify the current ownership information, the buyer invokes `checkOwnership()` with the device identifier. If the return is `True`, the buyer can verify the genuineness of the current seller. Additionally, the buyer can also trace back all the way along supply chain to IC’s provenance.

Authenticating the IC: Verifying the ownership of a device is not sufficient enough. A malicious seller can replace the original device with a counterfeit one and sell it to the buyer. Therefore, the buyer also needs to authenticate the device with the stored blockchain information. We propose two implementations for authentication of the device. We present the details of the implementation in section 2.5.3.

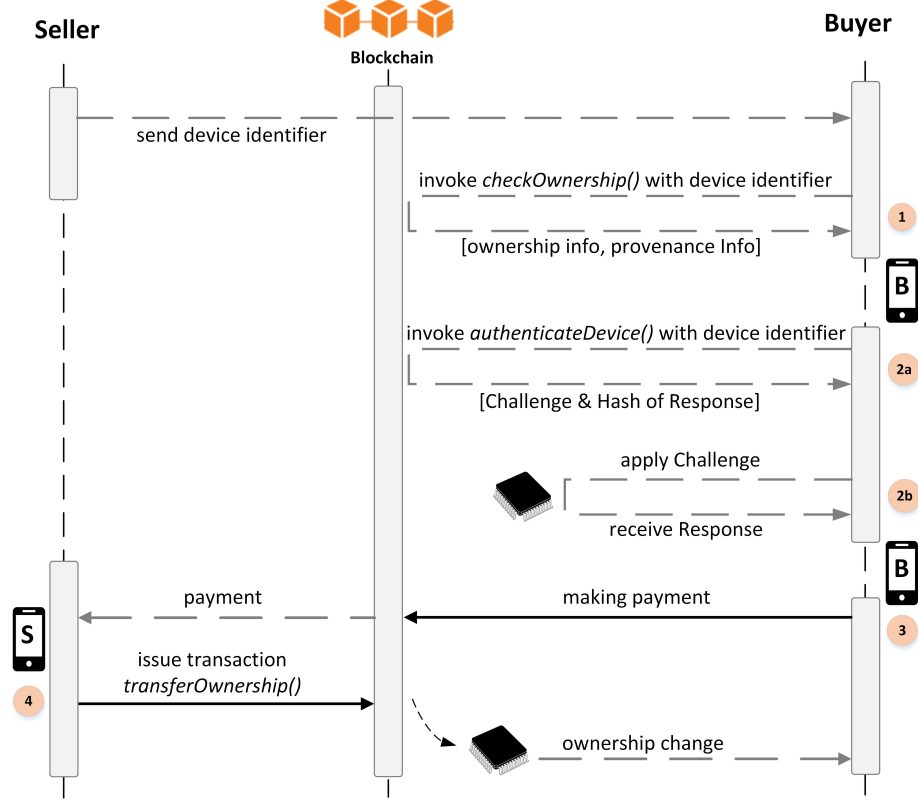


Figure 2.4: Detailed diagram of the ownership transfer protocol. Transactions requiring payment of fees are drawn with solid black lines.

Making payment: Once the IC is authenticated, and the seller’s IC ownership is verified, the buyer makes a payment to the seller. The buyer can make the payment in blockchain-based cryptocurrency. However, details of secure payment are out of our focus in this work.

Issuing transaction for ownership transfer: In the last step of the proposed protocol, the seller issues a transaction `transferOwnership()` with the device identifier and the buyer’s address. This function transfers the device from the seller to the buyer.

2.5.3 Authentication by Strong and Weak PUFs

During ownership transfer, the buyer needs to authenticate an IC with the stored blockchain information. The authenticator can be an API (smart contract front end or blockchain accessor) or a complete program, which a user needs to run during the verification. We propose the following two implementations for authenticating the device.

2.5.3.1 Authentication via Strong PUFs

When a manufacturer registers a device, it sends a transaction `registerDevice()` to the smart contract OC. This transaction includes necessary device information, such as the device’s identifier and Strong PUF CRPs for future authentication. At any point in the supply chain over the device’s lifetime, a potential buyer can authenticate a device by invoking `authenticateDevice()` with the respective *deviceIdentifier*. This invocation retrieves challenges from the blockchain and applies the challenges to the IC. The PUF embedded in the IC will give the corresponding responses. If the calculated hash of the responses matches the hash recorded in the blockchain, the IC is authenticated. Hash is a compact representation of the entire digital content, which is easy to compute and verify. For example, while pushing software updates, software vendors supply a hash (also known as digest), which is digitally signed by the software vendor. This property allows the devices to verify the software update’s authenticity by computing hash and verifying it using the vendor’s public key. Inspired by this software attestation technique, we propose compressing the CRP table as a digest, which can similarly attest to the authenticity of PUF CRPs supplied by the manufacturer.

To achieve pervasive authentication, it is necessary that the authentication functionality be easily integrated with modern electronic devices, such as smartphones, and be easy to use. As illustrated in Figure 2.4, one such example shows an IC verification process using Near Field Communication (NFC). Several PUF NFC tags are already used in commercial products [224]. For an easy authentication process, a silicon PUF can be integrated into a form of NFC. With the recent emergence of NFC-enabled smartphones, custom wired authentication system setup is no longer necessary for many use cases. This authentication system facilitates anyone in the supply chain to authenticate ICs using a smartphone with the necessary application. Other possible authentication procedures can also be implemented based on the needs of the system.

Here, we note that if the distributors and retailers have no authenticating infrastructure, they can still look up the blockchain using device identifiers and verify the chips’ provenance. However, in this case, they cannot authenticate the chips. An end customer can always verify the provenance and authenticate a chip.

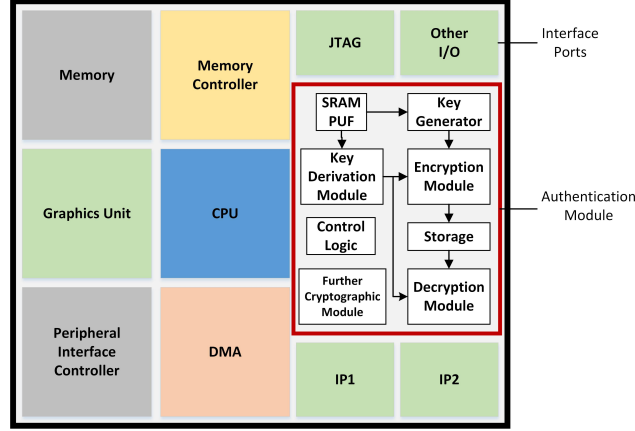


Figure 2.5: Hardware authentication module for generating cryptographic key from PUF embedded in an SoC [91].

2.5.3.2 Authentication via Weak PUFs

Several Weak PUF based public-key cryptography for authentication have been proposed in the literature [32, 43, 88, 136]. Figure 2.5 presents a hardware authentication module to generate a cryptographic key from SRAM-based Weak PUF [91]. A manufacturer first records the device’s Weak PUF data in the blockchain by issuing a transaction `registerDevice()`. At any stage of the supply chain, any buyer can authenticate the device by invoking `authenticateDevice()`. We describe the components of the authentication module and their functions as follows.

Key Generation Module: During the enrollment phase, the key generator in the authentication module generates a private-public key pair. Using the private key, a device can create the signature of a message protecting the message’s integrity and proving its authenticity. The recipient can verify the digital signature for authenticity using the public key corresponding to the private key. The private-public key pair may be keys for RSA, DSA, Schnorr, El Gamal, Elliptic Curve based public-key cryptosystems, etc. For the illustration purpose, we describe RSA public-key cryptography here. First, the key generator finds two prime numbers from a seed derived from PUF output, e.g., the contents of an SRAM. Finding prime numbers can be done on an appropriately programmed device or Hardware Security Module (HSM). Once two prime numbers of appropriate sizes are found, an RSA private-

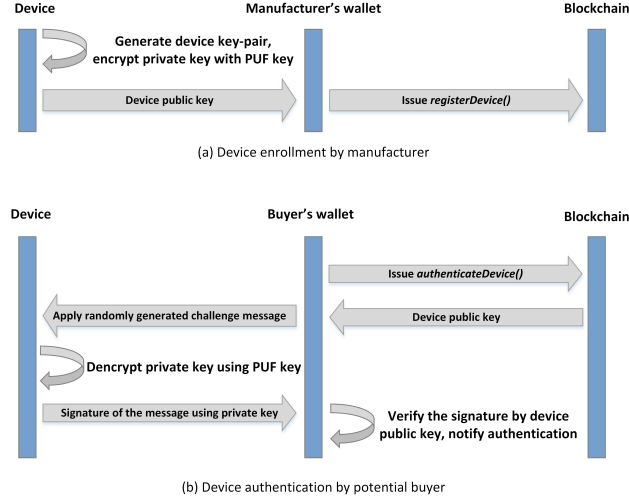


Figure 2.6: (a) Device enrollment and (b) authentication process via Weak PUFs.

public key pair is constructed. The RSA key pair generation is a computationally-intensive process and done only during the enrollment phase.

Encryption Module: The private key is encrypted with a second key (encrypting key) generated from the same PUF and stored in a non-volatile memory. This encryption is done using a symmetric encryption algorithm, like AES, and the encrypting key is obtained from any random 128-bit string from the PUF. The manufacturer records the public key in the blockchain by issuing a transaction `registerDevice()` for the device (Figure 2.6(a)).

Decryption Module: During the authentication phase, the encrypting key can be generated instantly from the PUF output. Using the encrypting key, the decryption module generates the private key of the device. When a potential buyer wants to authenticate the device, he invokes `authenticateDevice()` with the device identifier. The smart contract sends the device's public key to the buyer's wallet. The buyer's wallet then sends a challenge message generated with a pseudo-random number generator to the device. The device's further cryptographic module creates the device signature using the private key. Using the public key, the buyer's wallet can verify the digital signature for the device's authenticity, as shown in Figure 2.6(b).

2.5.4 Counterfeit Detection by the Proposed Traceability Protocol

Various methods of counterfeiting ICs have been described in the literature [90]. In this section, we present how our proposed traceability protocol prevents various counterfeiting techniques.

2.5.4.1 Recycled & Remarked ICs

Recycled and remarked components jointly contribute more than 80% of counterfeit incidents [124]. The recycled ICs are taken from used printed circuit boards (PCBs), repackaged and remarked, and then sold in the market as new. Our proposed protocol defeats this malicious approach in the following two ways:

If an unregistered, malicious manufacturer wants to introduce a recycled/remarked IC as a legitimate *new* product, he will need to create a valid *genesis transaction* for the IC. However, such a transaction can only be created by the original IP owner registered with the consortium and easily monitored. The false *genesis transaction* would immediately be flagged and would incriminate the counterfeiter. Also, the prevention of *double spending* [164] ensures that the counterfeiter cannot introduce multiple forged ICs by pretending to be a legitimate buyer and then attempting a future sale.

If a registered manufacturer wants to introduce a recycled/remarked IC as a legitimate product, it must include the device identifier information in the transaction. However, an already registered IC in the blockchain with the same device identifier would prevent the recycled IC to re-enter into the supply chain as a new one.

2.5.4.2 Overproduced ICs

In overproduction, an untrusted foundry, assembly, or test site with access to a designer's IP, may overproduce the original IP design outside the contract and the IP's owner knowledge. They can then sell the overproduced ICs in the open market as the original owner. Our proposed protocol defeats this malicious approach by requiring that the registering manufacturer sign the transaction, `registerDevice()` using its private key. As the counterfeit manufacturer cannot get the original manufacturer's key, it cannot register the overproduced ICs as the original manufacturer.

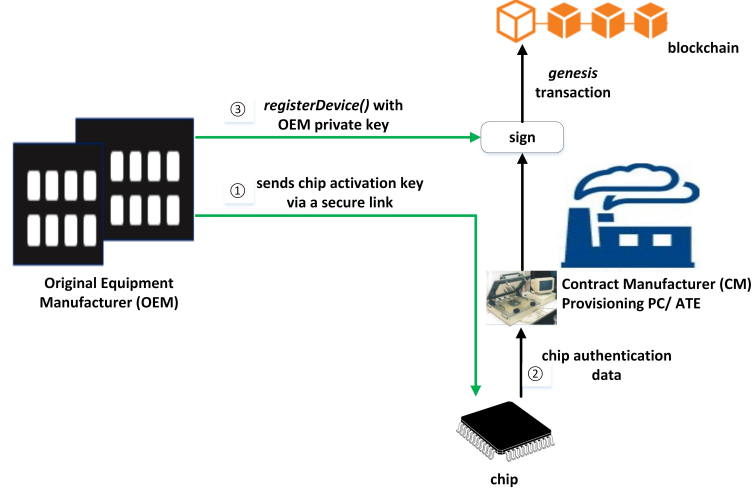


Figure 2.7: Chip activation using active metering [181] before enrollment of authentication data in blockchain. Green arrow indicates secure link established by the designer and manufacturer's public key infrastructure. ATE: Automated Test Equipment.

If the counterfeiter decides to use his brand (and own private key), there is still a need to protect the original designer's Intellectual Property. In this case, IC metering [31], obfuscation/locking [181], and/or Secure Split Test (SST) [173] based methods can complement our proposed approach. These approaches use an activation key to activate each chip uniquely. Figure 2.7 shows how these approaches can complement our proposed scheme. After the manufacturer fabricates a chip, the designer sends the chip activation key via a secure link established by the designer and manufacturer's public-key infrastructure (step 1 in Figure 2.7). After the chip gets the activation key, it is unlocked for testing and enrollment of PUF authentication data (step 2). Finally, the designer sends a transaction `registerDevice()` to the smart contract OC by signing the chip authentication data with its private key (step 3). Recently, logic-locking techniques have been targeted by SAT attacks [221]. These SAT attacks against logic-locking techniques present a case for a better solution.

2.5.4.3 Cloned ICs and Tampered ICs

The current owner cannot clone an IC with the same ID and provenance information due to the embedded PUF whose data is recorded in the blockchain. Each PUF device has a unique and unpredictable way of mapping challenges to responses, even if it was manufactured with the same process as a similar device, and it is infeasible to clone a PUF

Table 2.3: Operation cost for a chip in supply chain.

Operation	Gas limit	Gas price	Cost (in ETH)	Cost (in USD)
<code>registerDevice()</code>	121478	10.89×10^{-9}	0.0013229	0.463
<code>transferOwnership()</code>	30365	10.89×10^{-9}	0.00033067	0.116

with the same Challenge-Response behavior as another given PUF. Sufficient PUF data is assumed to be available in the blockchain to assure the authenticity of the IC. Furthermore, our proposed method makes the tampering with the IC very costly due to the embedded PUF. Various PUF tampering techniques, such as Focused Ion Beam (FIB), are costly and would have to be performed on a per chip basis to obtain multiple clones [181].

2.5.5 Protocol Demonstration in Ethereum Blockchain

We implemented the proposed smart contract in **Solidity** programming language, deployed it in Ethereum blockchain, and evaluated it in terms of its operational cost. We made the code publicly accessible in Github [2]. In particular, we estimated the total cost by measuring the total gas amount (execution fee for every operation made on Ethereum) for all of the functions involved in the process, that is, (i) registering device (`registerDevice()`), and (ii) transferring ownership (`transferOwnership()`), and then converting it into USD. As the amount of gas is fixed for each operation in Ethereum, e.g., a SHA3 calculation costs 20 gas, the total gas amount for executing a function is also fixed. In particular, we have used the Ethereum’s test environment tool, *testrpc* [115], to measure the gas amount since it can automatically count the gas amount. With the current value of 1 ETH = 350USD and 1 Gas = 10.89×10^{-9} ETH (10.89 Gwei), the operation cost for a chip in the supply chain is calculated in Table 2.3. Finally, the total cost of maintaining the identity of a chip in a supply chain with N entities is:

$$\text{total cost} = \text{cost of enrollment} + (N - 1) \times \text{cost of ownership transfer}$$

Using the formula, the total cost of maintaining a chip’s identity in the supply chain presented in Figure 2.2 with five entities (manufacturer, distributor, retailer, consumer 1, consumer 2) is $0.463\text{USD} + 0.116 \times 4 = 0.927\text{USD}$.

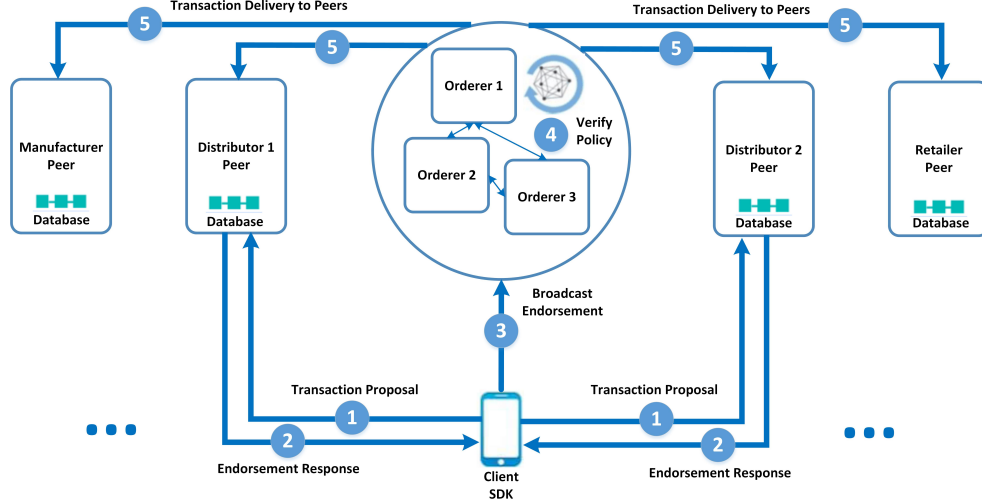


Figure 2.8: Hyperledger Fabric network model for our proposed protocol.

2.5.6 Protocol Demonstration in Hyperledger Fabric

To implement our proposed consortium blockchain and prove the proposed protocol’s applicability, we select the Hyperledger Fabric framework [33]. Hyperledger Fabric, introduced and maintained by IBM, is becoming one of the most prominent blockchain platforms. Its permissioned architecture and non-resource intensive consensus algorithm ideally match the requirements of implementing our proposed blockchain. A customer could also be registered with an identity in the blockchain. In this way, the blockchain can record the post-sale traces as well.

2.5.6.1 Blockchain Network Model

Figure 2.8 presents the Hyperledger blockchain network model, where the manufacturers, distributors, and retailers are the major members of the blockchain. They are registered as *nodes* in the blockchain. Each node creates and maintains an identity (i.e., account, address, or participant identity) in the system. Any addition (new member) or replacement of identities must be notified and accepted by all the major members identified in the chain.

2.5.6.2 Implementation of the Chaincode and Access Control Policies

The chaincode (smart contract) implements the underlying functionalities that provide data storage and management. All the blockchain’s major members verify the creation,

maintenance, and deprecation of the chaincode. In our traceability protocol, the required chaincode functions are described in 2.5.1.2. For regulated and secure chaincode functionalities in the blockchain system, we need access control policies in our prototype infrastructure. The blockchain system enforces the policies to give access to the operations; otherwise, it denies the operations. The `registerDevice()` function implements the organization identity-based access control to enforce that only a manufacturer can register a new device. The `transferOwnership()` function implements the user identity-based access control to enforce that only the owner can transfer a device's ownership.

2.5.6.3 Performance Evaluation

We set up an experimental environment with a single machine equipped with 8-core, 3.6Ghz CPU and 16GB RAM. As depicted in Figure 2.8, we created a Hyperledger Fabric 1.4.4 network with four organizations (one manufacturer, two distributors, one retailer) in a single *channel* with *CouchDB* state databases using docker containers [13]. We set the *block size* to 80 and *batch timeout* to 400ms. Our *endorsement policy* adopts the default ‘N of N’ policy, meaning that a transaction needs to be endorsed by all four organizations. Since Hyperledger Fabric adopted the Raft *ordering service* as the new consensus mechanism, we deployed 3 RAFT orderers on the machine. Finally, we created one client to send the transactions.

We used Hyperledger Caliper as the blockchain performance benchmark framework [12]. We continuously send transactions from the client and observe the network's throughput

```

2020.11.11-15:45:43.819 info [caliper] [report-builder]      ### Test result ###
2020.11.11-15:45:43.824 info [caliper] [report-builder]
+-----+-----+-----+-----+-----+-----+-----+-----+
| Name | Succ | Fail | Send Rate (TPS) | Max Latency (s) | Min Latency (s) | Avg Latency (s) | Throughput (TPS) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| open | 400  | 0    | 70.2           | 6.93            | 1.32            | 4.81            | 35.7              |
+-----+-----+-----+-----+-----+-----+-----+-----+

2020.11.11-15:45:43.826 info [caliper] [round-orchestrator] Finished round 1 (open) in 11.212 seconds
2020.11.11-15:45:43.826 info [caliper] [report-builder]      ### All test results ###
2020.11.11-15:45:43.827 info [caliper] [report-builder]
+-----+-----+-----+-----+-----+-----+-----+-----+
| Name | Succ | Fail | Send Rate (TPS) | Max Latency (s) | Min Latency (s) | Avg Latency (s) | Throughput (TPS) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| open | 400  | 0    | 70.2           | 6.93            | 1.32            | 4.81            | 35.7              |
+-----+-----+-----+-----+-----+-----+-----+-----+

2020.11.11-15:45:43.835 info [caliper] [caliper-flow] Generated report with path /hyperledger/caliper/workspace/report.html
2020.11.11-15:45:43.937 info [caliper] [message-handler] Handling "exit" message
2020.11.11-15:45:43.937 info [caliper] [round-orchestrator] Benchmark finished in 22.41 seconds. Total rounds: 1. Successful rounds: 1. Failed rounds: 0.
2020.11.11-15:45:43.937 info [caliper] [caliper-engine] Network configuration attribute "caliper.command.end" is not present, skipping end command
2020.11.11-15:45:43.937 info [caliper] [cli-launch-master] Benchmark successfully finished
(base) [nazmul@ecs-mst-140-02 ContainerisingCaliperForFabricBenchmark-master]$

```

Figure 2.9: Hyperledger Caliper benchmark result.

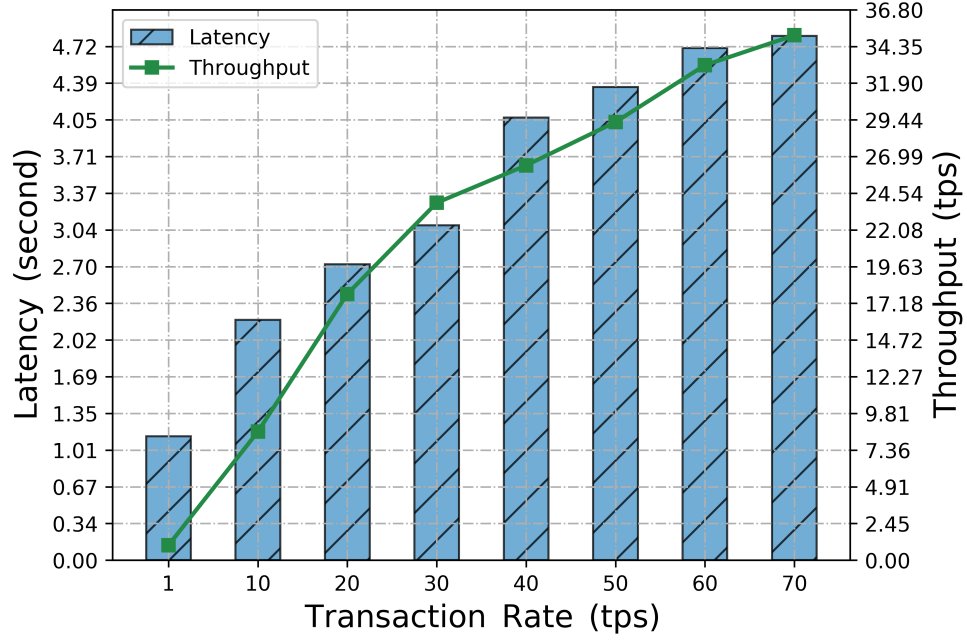


Figure 2.10: Throughput and latency of the `registerDevice()` transaction in the proposed blockchain implementation using Hyperledger Fabric.

and latency. Figure 2.9 presents a sample Linux terminal output after running Caliper. Figure 2.10 presents the throughput and latency of the proposed blockchain implementation at various transaction rates. We observed the maximum throughput of 35 tps. With the increasing transaction rate, the latency keeps increasing, as the accumulated and queued transactions need to be held at the orderer nodes. The performance of the Hyperledger Fabric network depends on various factors, such as chaincode execution time, endorsement delay, network delay, consensus delay among multiple orderers, and block validation delay.

It is important to note here that the hardcore performance evaluation of the Hyperledger Fabric platform is not the main objective of this experiment. Several works have measured the performance evaluation of the Hyperledger framework with detailed metrics and comprehensive analysis. Here, we measure the performance metrics (throughput and latency) of the prototype system to prove our proposed framework’s applicability. We also show our proposed framework’s applicability in a multi-host scenario for a similar blockchain network model, detailed in Appendix A.

2.5.6.4 Operational Cost of Consortium Blockchain

Costs linked with the system activity, such as the system setup, execution, or maintenance, are shared within the consortium according to a pre-established governance scheme. The consortium’s incentive to build and maintain blockchain infrastructure is to detect counterfeits in the IC supply chain. Individual companies can leverage the same resources for various other blockchain applications. Additionally, for regular system maintenance, a consortium node is rewarded a small fee (e.g., some ETH) when it validates any transaction committed by a customer for ownership transfer. Selecting appropriate values for transfer reward will depend on the consortium members’ actual investment for the implementation. However, such a topic is outside the scope of our current research, and thus it will not be considered further.

2.6 Analysis of the Protocols

In this section, we analyze the security, privacy, and reliability issues of the proposed blockchain-based IC traceability protocol; and discuss our proposed protocol’s performance in terms of scalability and resource requirements.

2.6.1 Security

Double-spending by the seller: Once the transaction is complete and added to the blockchain by a seller, the buyer becomes the only person authorized to update the blockchain for a particular IC. Hence, the seller cannot falsely attempt another sale. A malicious buyer can also mount a *double-spend* attack. In this attack scenario, the attacker makes a payment transaction that moves funds to the seller’s address. After the transaction appears in the newest block on the main branch, the attacker takes possession of the IC. The attacker then releases two blocks immediately, using his mining power, with a transaction in the first that transfers the funds to a second attacker-owned address. Now the attacker has the IC and his funds back. However, this type of *double-spend* attack is a fundamental security issue in any blockchain system, and several solutions have been proposed in the literature [47, 163] to counter these threats, as described in Section 1.1.3.

Seller’s refusal to transfer IC: Following the previous point, the buyer can use the blockchain to take legal action against the seller if the seller refuses to transfer the IC to the buyer. A supply chain participant’s integrity and legitimacy are ensured during registration when it joins the system and gets its private/public keys (Section 2.4.2.1).

Possible man-in-the-middle attack: An attacker may mount a man-in-the-middle attack during ownership transfer. First, a malicious party gets valid CRPs of an authentic device from the blockchain and hardcodes those CRPs into his counterfeit device. Then, he proves to a potential buyer that his device (a counterfeit one) shows correct CRPs. Our proposed protocol resists against such attacks. In our proposed protocol, a potential buyer performs two-step authentication: (i) verifying the ownership information, (ii) authenticating the IC with Challenge-Response Pairs (CRPs). Even if the attacker can prove the valid CRPs from his counterfeit device, he cannot prove his ownership of the device because the blockchain record will show the genuine owner’s information corresponding to that CRP authentication data.

De-centralized verification and authentication: In a traditional authentication process, a centralized Verification Authority (or Verifier) possesses a large set of CRPs for each IC, enrolled prior to sale. The Verifier can only perform authentication of a device in the field. Our protocol provides the flexibility for any potential buyer to authenticate a chip.

Modeling attack resistance: An IC’s buyer verifies CRPs on the actual physical device against the records in blockchain tracing back to the manufacturer. In this instance, a software model [184] of the PUF is irrelevant and cannot be used to mount an attack. Unless the chip is provisioned with an Artificial Neural Network (ANN), it cannot clone the PUF. Provisioning a chip with ANN, solely for cloning increases die area and accrues additional cost. If an attacker chooses this route, there will be a cost explosion since each chip’s software model has to be learned separately.

Security against malicious foundries: If a third party foundry (contract manufacturer) or assembly wants to behave maliciously, our proposed approach gives no incentive to do that. For example, if a foundry creates a bad *genesis transaction* that is not authenticated by the on-chip PUF, it cannot sell that part. If the foundry creates a *genesis transaction*,

which is authenticated by the PUF, but the chip is a bad one, then the buyer precisely knows who is responsible. Besides, if the manufacturer provides incorrect PUF data, the IP owner cannot sell the chip and loses trust in the manufacturer. Moreover, if the manufacturer leaks the PUF data, still it is infeasible to build hardware with a cloned PUF. Any effort by a malicious foundry to overproduce ICs is defeated by the locking scheme described in Section 2.5.4.2. Thus, the manufacturer has no incentive to become an attacker.

Consortium blockchain security: The consortium members control the consensus process and write the consortium blockchain with transactional data. Since the customers cannot participate in the consensus process, they cannot compromise the consortium blockchain’s security and privacy.

2.6.2 Privacy

Transparency vs privacy: The proposed protocol shows that an entity performs all the transactions using a single address. This address re-use may raise some privacy concerns for the customers. One possible approach to address the issue is to assign multiple addresses to each participant.

Consortium blockchain vs public blockchain: There are several advantages of a consortium blockchain over a public blockchain. First, if the read permissions are restricted, consortium blockchain can provide a greater level of privacy [54]. Moreover, the consortium can, if desired, change the rules of the blockchain. Since the transactions need to be verified only by a select number of nodes, transactions are cheaper in validation overheads. Any accidental fault can be quickly fixed, as the nodes are very well connected.

2.6.3 Reliability

Authentication by Strong PUFs: The *genesis block* is assumed to have enough PUF data to disambiguate the IC from millions of other devices [175]. To increase the number of authentication events, the PUF can be re-mined during each ownership transfer event, and the new data can be included in *challenge* and *hashResponse* fields of the transaction format. The hashing of PUF responses prevents their exposure in the blockchain and ensures only the IC’s physical owners can re-generate the hash. The re-mining process can also check the

reliability of the PUF, which can be affected by aging over the product lifetime, and update the blockchain accordingly.

Authentication by Weak PUFs: To generate stable keys from the Weak PUF, several literature works have proposed solutions, such as accelerated device aging [112,113], built-in self-test [43], Helper Data Algorithm, also known as a Fuzzy Extractor [64]. The helper data can be generated during the enrollment phase and stored in a non-volatile memory present in the chip. It can be constructed to become known to an adversary without compromising the secrecy of the PUF response, i.e., it does not need to be kept secret [69,182]. During the authentication stage, the secret key can be reconstructed, combining the response from the PUF and helper data from the non-volatile memory. Alternatively, a non-volatile memory-based PUF can be used for authentication, which does not require any helper data [58]. If PUF is sufficiently reliable, then producing helper data may be omitted.

2.6.4 Performance

Scalability: Our proposed protocols allow the registration and ownership transfer of multiple ICs in a single transaction between two parties by utilizing the multiple inputs option in the transaction format and the smart contract functions. The consortium blockchain eliminates the cost of transaction fees and improves the efficiency by using a non-resource intensive consensus algorithm. As a result, a consortium blockchain could minimize the cost of the supply chain’s daily operations, which is ideal for supply chain traceability. Table 2.4 presents the comparative performance analysis of our traceability protocol implementations in Ethereum and Hyperledger Fabric.

Low power and low cost authentication: The PUF consumes dynamic power only during the authentication process - when it generates the response. The power required during authentication is small. Also, the PUF is a low-cost, lightweight root-of-trust, enabling easier integration.

Simple, robust authentication: PUFs enable fast, secure authentication of the ICs due to their unclonability and light footprint. Furthermore, the PUF information can be updated in the blockchain during each transfer and provides a future buyer with more data points to verify IC authenticity.

Table 2.4: Comparative Performance Analysis of the Proposed IC Traceability Protocol.

	Ethereum	Hyperledger Fabric
Permissions	Permissionless	Permissioned
Consensus mechanism	PoW	PBFT
Access to Data	Public	Restricted or Public
Node Scalability	High	Low
Block Generation Time	15 seconds	0.4 seconds
Throughput	15 tps	35 tps
Benchmark Tools	EVMremix	Caliper
Programming	Solidity	Go
Cryptocurrency	Ethereum	None

Flexible consensus mechanisms: Permissioned platforms have semi-trusted members where only known participating nodes that are part of a consortium are verified and registered. These groups are expected to be small in number and, therefore, can employ alternative consensus mechanisms. Achieving consensus in a distributed system has known solutions in the research literature, e.g., Paxos [133], RAFT [162], and various Byzantine Fault Tolerance algorithms [56]. Permissioned blockchain platforms have primarily adopted these consensus algorithms.

2.6.5 Practicality of the Proposed Protocol

Our proposed authentication and the overall traceability protocol are highly suitable for a computing system where the IC is part of a system, such as System on Chips (SOCs), computer motherboard systems, automotive ECUs, etc. These systems can be connected to the Internet, and using our blockchain protocol allows all the ICs of the system to be authenticated simultaneously.

Figure 2.11 presents how automotive security can be ensured during any ownership transfer using our IC traceability protocol via blockchain pegged to embedded Weak PUFs. Modern automobiles contain more than 70 Electronic Control Units (ECUs) networked together [186]. The overall safety of the vehicle relies on the authenticity of various ECUs. The security of a car can be compromised by connecting counterfeit ECU in a vehicle

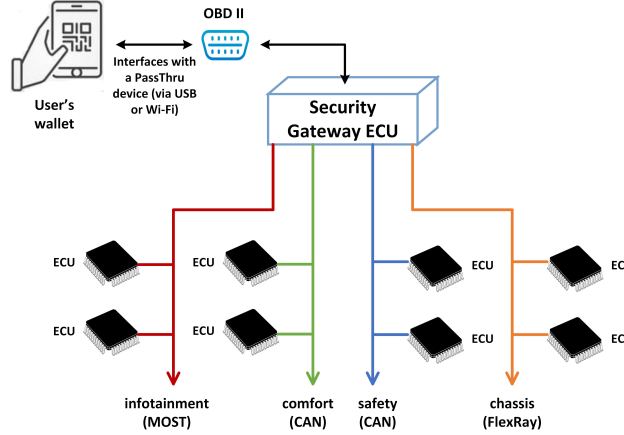


Figure 2.11: Schematic of typical in-vehicle network architecture of a modern automobile. All inter-bus communication is done exclusively only over the gateway ECU. CAN - Controller Area Network, MOST - Media Oriented Systems Transport.

communication network. A counterfeit ECU from any malicious party can threaten the functioning of steering, brakes, airbags, windows, headlights, etc [217]. This threat gives rise to the safety concern of people in and around the car. To circumvent this, we propose our traceability protocol, by which a buyer can authenticate the provenance of all ECUs during any ownership transfer.

In the proposed protocol, a centralized gateway ECU connects all existing bus systems. During device enrollment, all accredited OEMs (Original Equipment Manufacturer) of the respective vehicle enroll the ECUs by issuing `registerDevice()`. The vehicle manufacturer enlists *device IDs* of all ECUs in the gateway security ECU. At any stage of the supply chain, a potential buyer can verify the ownership and the provenance of the ECUs by invoking `checkOwnership()` with the list of *device IDs* obtained from the gateway security ECU. He can authenticate all the ECUs connected in the vehicle network by invoking `authenticateDevice()`.

For authentication, the blockchain sends the public keys of all ECUs to the buyer's wallet. The buyer's wallet sends a challenge message to the gateway security ECU via a PassThru device [186]. The gateway ECU collects the signatures from all ECUs, including its own, and sends back to the buyer's wallet. The buyer's wallet can verify the digital signature for the authenticity of the device.

2.7 Limitations and Discussion

Cloning PUFs: Recent literature work has shown that PUFs can be tampered by Focused Ion Beam (FIB) to create clones [93]. However, FIB based tampering is costly and would have to be performed on a per chip basis to obtain multiple clones. The authors in [181] conjectured that the FIB attack might not be possible for the ICs with $32nm$ or smaller feature nodes.

Authenticity of Analog and Mixed-signal Circuits: The number of counterfeit analog ICs has been increasing at an alarming rate [25]. Our proposed solution is for ICs with embedded PUFs and is agnostic to the device’s type, whether analog, digital or mixed-signal. The solution is easily extended to analog and mixed-signal circuits (with analog PUFs [66], or package IDs [62], [154], [128]) but is outside the scope of the current work. Since the blockchain is pegged to PUF for authentication, legacy devices without any authentication mechanism are not supported by this solution.

Transparency vs Privacy: An open research question regarding our proposed protocol is how both the transparency requirement for supply chain tracking and anonymity requirements for maintaining users’ privacy can be satisfied. This requirement is of particular importance to IP owners selling millions of devices as it can be infeasible to create millions of public-private key pairs. In future works, we plan to investigate possible solutions to address this issue.

2.8 Concluding Remarks

In this chapter, we outline a novel methodology to establish IC traceability via blockchain technology and PUFs. The blockchain allows legitimate parties to track an IC over its entire lifetime. We use embedded PUFs to provide a simple, secure, and robust authentication process at every point-of-sale. The protocol automates the entire process of authentication, ownership transfer by using the smart contract. Our approach does not impose onerous restrictions on the participants in the blockchain; instead, the underlying technology guarantees the integrity of the system even in the face of dishonesty or idleness. This approach provides a technological solution to a supply chain problem.

CHAPTER 3

PRESERVING IOT PRIVACY IN SHARING ECONOMY VIA SMART CONTRACTS

3.1 Introduction

For widespread adoption of the ever-expanding IoT, privacy and anonymity must be integrated into its design by giving users control of their privacy. Privacy is the right of individuals or cooperative users to maintain confidentiality and control over their information when disclosed to another party. In IoT applications, privacy threats can arise from the perspective of IoT devices' users and their data. Any unauthorized access could unexpectedly initiate privacy threats and attacks. One good example of this is the 'sharing economy'. Sharing economy platforms such as Airbnb have recently flourished in the tourism industry. In a sharing economy platform, a centralized third-party usually provides the technical infrastructure, user interfaces, and the guidance/monitoring process.

However, relying on a centralized third-party sharing platform inevitably leads to a single point of weakness, higher fees, lack of trust, and governance issues for both users and service providers [28]. It creates an inherent bias, fraud, and a single point of weakness in the system. Such intermediaries charge a large service fee (up to 15% for guests and up to 5% commission of the homeowner), can arbitrarily change the terms and conditions [28].

Moreover, sharing any IoT-devices enabled smart houses poses a severe threat to user's privacy. Airbnb hosts prefer to know what is going on in their rentals. Because of this, hosts may opt to have surveillance cameras in key places. This surveillance allows Airbnb hosts to spy on guests, which is a severe infringement of the guests' privacy expectations. Similarly, by accessing the smart door lock, an intrusive homeowner can compromise the security system. By accessing the stored credentials on connected devices, hosts can take control of the IoT devices' sensors and can even disable an apartment's control of HVAC systems [153].

In this chapter, we propose smart contracts to eliminate *(i)* distrust in the third-party controlled home-sharing economy by decentralization and *(ii)* privacy threats from IoT-enabled telematics devices in a sharing house [106,109,171]. In our proposed solution, which is based on the Ethereum blockchain network, the shared IoT devices are directly connected to the blockchain and are controlled by a smart contract through which they receive and update their security parameters and the serving user’s information. The proposed method ensures exclusive access to the IoT devices seamlessly through blockchain smart contracts.

We organize the chapter as follows. Section 3.2 presents a threat model in a conventional home-sharing economy and then describes our proposed solution’s motivation. Section 3.3 gives an overview of the related literary works on sharing resources in different IoT ecosystems and enforcing supervised access to those resources. Section 3.4 describes our proposed protocol to preserve IoT privacy in the sharing economy via smart contracts, and Section 3.5 outlines the hardware collateral required for the protocol. Section 3.6 presents a demonstration of the proposed protocol, and finally, Section 3.8 concludes the chapter.

3.2 Threat Model and Motivation

In this section, we present the security, trust, and privacy threats in a conventional home-sharing economy and then describe our motivation for the proposed solution.

3.2.1 Threat Model

Figure 3.1 presents the trust, and privacy threats in a conventional home-sharing economy, such as Airbnb. Here, firstly, renting out a unit on Airbnb requires multiple levels of trust. Both host and guest have to trust Airbnb’s ability and integrity regarding booking and payment processes. In such a scenario, Airbnb is the provider of the user interfaces, technical infrastructure, and the guidance/monitoring process. Moreover, it is also responsible for providing services such as insurance and the user’s reputation management. By doing so, Airbnb is the only responsible for establishing and maintaining trust among users. Unfortunately, current mechanisms cannot cope with malicious behaviors, high service and transaction fees, strategic lies, and the formation of deceiving coalitions. Therefore, there is a need for a critical technological innovation since no centralized entity nor an intermediary

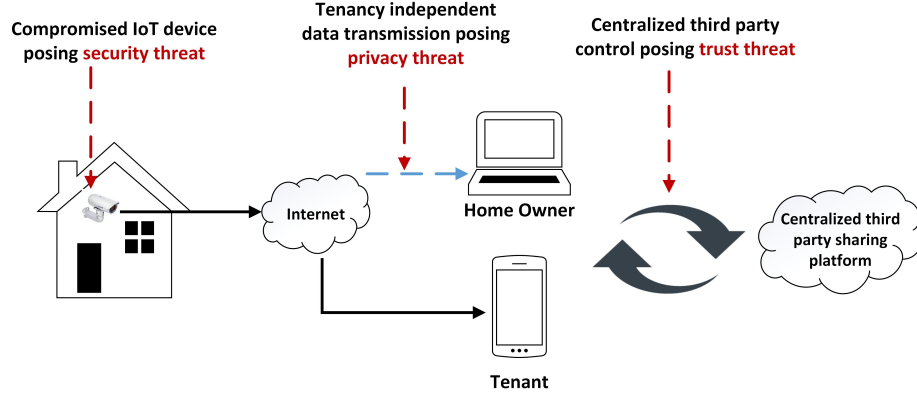


Figure 3.1: Threats associated with accessing indoor IP camera by home-owner from remote location in a home-sharing economy scenario.

can address these problems [155]. Additionally, an intrusive homeowner can compromise the privacy of the tenant by accessing the connected IoT devices. For example, the data collected by the IP camera in a rental property cannot be shared with the owner of the property while it is occupied by a tenant.

3.2.2 Motivation

Protection of Trust The sharing economy is a case of a consumer to consumer (C2C) business model. Contrary to other business models, where companies sell their products or services to other businesses (B2B) or consumers (B2C), companies in a sharing economy are only middlemen who can be cut out of the process by establishing an alternative source of trust between consumers. Thus, the sharing economy is uniquely suited to decentralization via the blockchain. For IoT in the home-sharing economy, blockchain can provide an infrastructure for direct, safe, and secure transfers between devices, without the need for a centralized authority. Smart contracts can translate the existing contractual clauses into embedded hardware and software so that it can self-verify that conditions have been met to execute the contract. Smart contracts contain code functions and interact with other contracts, make decisions, store data, and send tokens/money to others.

Protection of Privacy Smart contracts can also facilitate efficient IoT devices by automating their operations and decision making. We can achieve this automation by allowing IoT devices to interact with smart contracts and make decisions defined by the fixed con-

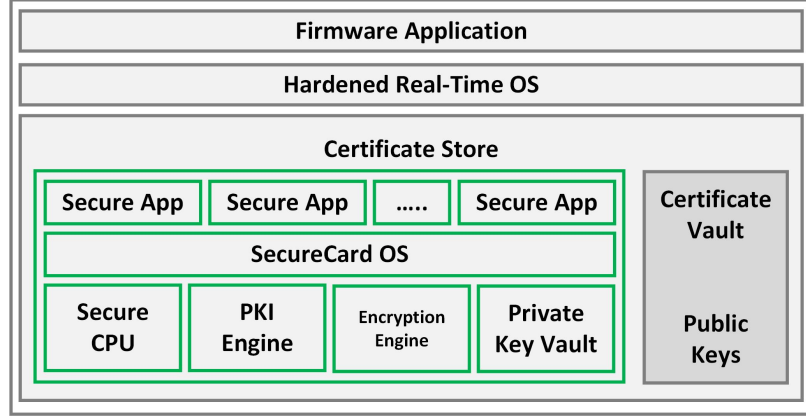


Figure 3.2: Block diagram of a Trusted Platform Module (green block) embedded into the camera’s software architecture [8].

tract logic. For example, in order to safeguard surveillance data, modern IP cameras are equipped with an onboard security chip, Trusted Platform Module (TPM) [8]. Using a symmetric cryptographic key, the TPM encrypts the data stream (Figure 3.2). In this work, we leverage the TPM to change its encryption key whenever a tenancy change is recorded in the smart contract. This key can only be computed using the device and the tenant’s private key so that no one else can access the surveillance data other than the current tenant.

3.3 Related Works

Privacy is a concern whenever common resources are shared. Wolf *et al.* [218] proposed a new IoT architecture, which facilitates the horizontal integration of different IoT ecosystems. The proposed work assumes that the owner controls the resource being shared at all times, and only supervised access to that resource can be granted to the requester by the owner. The main drawback of such a solution is that there is no provisioning in it by which the requester can ensure the privacy of its access to that resource. Although this privacy concern is not a significant problem in various IoT applications, such as sharing the data collected from temperature sensors deployed in a farm, it could be crucial for other types of applications, such as home-sharing, car-sharing.

3.4 Proposed Methodology

Figure 3.3 presents our proposed protocol showing the role of all entities (the manufacturer, owner, tenant, IoT device, blockchain) to preserve IoT privacy in the home-sharing economy. The protocol consists of the following steps.

3.4.1 Implementing Smart Contract

In the first step of our proposed protocol, the manufacturer of an IoT device (an IP camera, for example, in our case) creates a smart contract (**possessionContract** in Figure 3.3). This contract offers functions for managing the possession transfer and polling the possession of the device. The manufacturer then deploys the contract in blockchain and embeds the address of the contract in the device. The IP camera may either ship with the smart contract's address baked into it, or the blockchain nodes can find out about it via a discovery service [76].

The smart contract consists of *code* (its functions) and *data* (its state). The contract functions are **setDeviceInfo**, **sendDeposit**, **transferTenancy**, **pollTenancy**, and the state variables are owner, owner's public key, tenant, tenant's public key, device identifier, device public key, rental status (*rented/unrented*), rent initiation date, rental period. Once the smart contract is deployed in the blockchain, the IoT device executes according to the smart contract's states. The IP camera queries the contract, finds the tenant and changes the video data encryption key. The manufacturer sets the first buyer as the owner of the device.

3.4.2 Transferring Tenancy to a Tenant

To fulfill this protocol, the smart contract stores an internal list of IoT devices, along with their owner information and their respective rental prices. New devices are added to this list, and the owner can adjust the prices in this list by calling **setDeviceInfo**, the function to set the minimum deposit, prices, and types. When a prospective tenant creates a transaction **sendDeposit** to send a deposit, it includes the device information so that the smart contract can keep a list of tenants and their deposits for each device, along with the prices at the time of deposit.

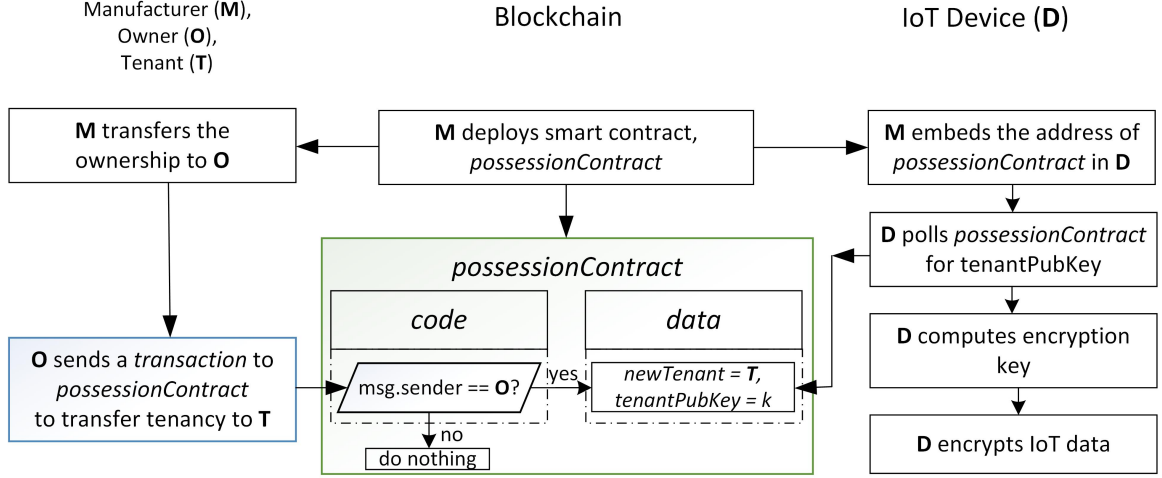


Figure 3.3: Detailed diagram showing the role of all entities in the proposed IoT privacy protection protocol.

The owner can check this list to ensure that a deposit has been made before providing access to his/her property. To transfer the tenancy to a tenant, the owner sends a transaction `transferTenancy()` to the `possessionContract`. Algorithm 5 shows the contract function `transferTenancy()` used in our proposed methodology. This transaction defines all the necessary information related to a tenancy transfer, such as new tenant information, tenancy starting time, tenancy period. The transaction includes the tenant's public key, which will be used by the IoT device for computing the data encryption key. If the transaction's sender is the current owner of the device, the smart contract updates its new tenant and the new tenant public key.

ALGORITHM 5: Pseudo-code of `transferTenancy()` for transferring tenancy to the tenant.

Inputs: Device identifier (*deviceIdIdentifier*) Tenant's public key (*pubKeyTenant*), and tenancy period (*tenancyPeriod*)

```

if msg.sender is the owner AND deviceIdIdentifier is available then
    | Set pubKeyTenant as the target feed in the deviceIdIdentifier
    | Change Subscription time to tenancyPeriod
    | Change deviceIdIdentifier state to rented
else
    | Throw error
end
  
```

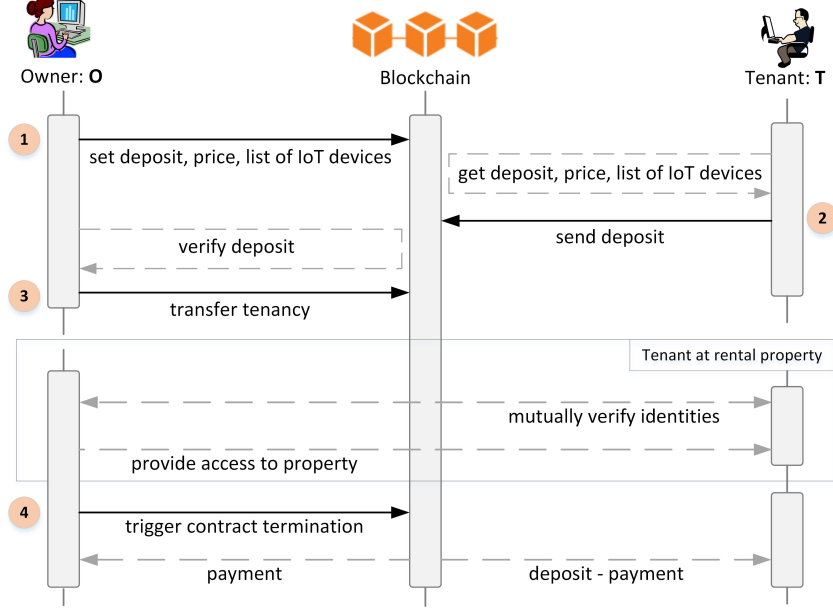


Figure 3.4: Detailed diagram of the tenancy transfer protocol.

3.4.3 Establishing a Shared Encryption Key

The device interacts with the blockchain network and polls the smart contract periodically by querying the function `pollTenancy()`. We present the details of device interaction methods with blockchain in Chapter 6. The `pollTenancy()` function returns the tenant's public key to the requesting device. For video stream encryption purposes, the IP camera establishes a symmetric key using the Diffie-Hellman protocol [67]. On the one hand, the device calculates the symmetric key using its private key stored secretly inside the device and the tenant's public key from the smart contract. On the other hand, the tenant calculates the symmetric key using his private key and the device public key. The computation is based on a pre-established large prime number (p) and a generator (g), which is a primitive root modulo p .

After the tenant includes the public key in the smart contract, the IP camera can encrypt the video data with the tenant's public key, and the tenant can decrypt the data with a private key using any standard Public Key Cryptography (PKC) algorithm. However, for large data like video stream, encryption/decryption with asymmetric PKC (e.g., RSA, ECC) is very slow. Instead, we choose symmetric key cryptography (AES) for encryption/decryption, which is several orders of magnitude faster.

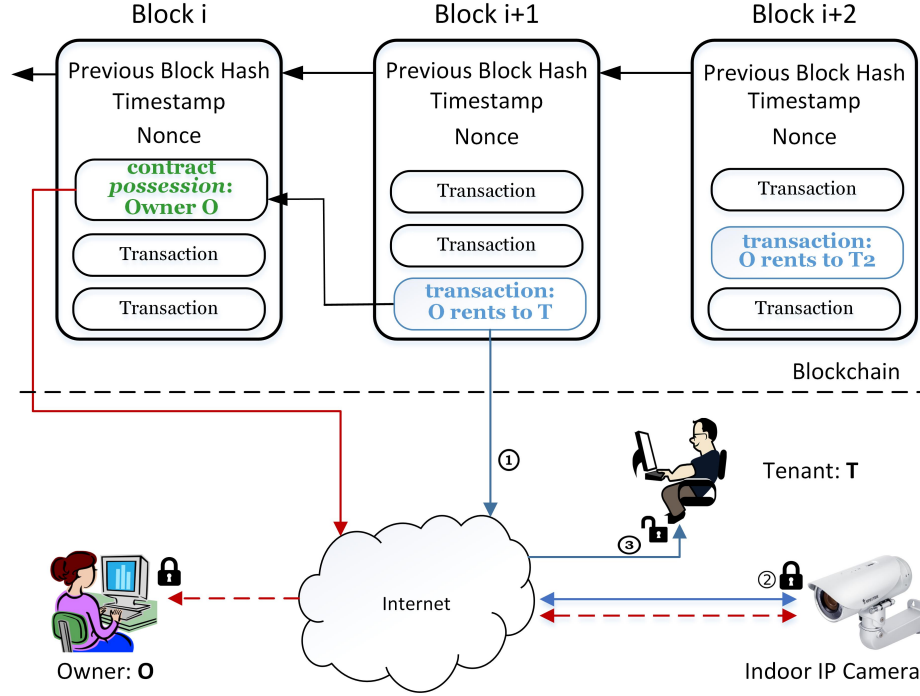


Figure 3.5: Detailed diagram of the privacy protection protocol: (1) the smart contract notifies the IP camera about the tenancy change, (2) IP camera computes the symmetric key from tenant's public key and encrypts video data, (3) tenant calculates the symmetric key and decrypts the video data. The dotted lines indicate inaccessible data.

Tenant's public key, $T = g^t \bmod p$,

Device public key, $D = g^d \bmod p$

Next, the public keys are exchanged via smart contract. The tenant and the device computes a shared symmetric key (s) using their private keys.

Tenant computes $s = D^t \bmod p = g^{dt} \bmod p$,

Device computes $s = T^d \bmod p = g^{dt} \bmod p$

3.4.4 Encrypting IoT Data with the Shared Encryption Key

The encryption engine of the TPM changes the encryption key to the newly computed symmetric key. Then it encrypts all the video data stream or other payload with encryption key (Figure 3.5). The tenant can also decrypt the video data with the shared key. On the other hand, the owner can no longer decrypt the surveillance data as the key has been changed.

3.4.5 Change of Encryption Key after Tenancy Period

The `transferTenancy` function in `possessionContract` defines that during the tenancy period, the tenant will have the possession of the device. After the tenancy period, the original owner will have the device's possession, and the encryption key will be changed according to the original owner's public key. As the smart contracts are reactive entities, in order to terminate them, one should trigger them. The owner does this termination once the rental period is over. The trigger message is only accepted and processed by the contract if it comes from the owner and if the contract period is over. Upon receiving the trigger message, if the rental period is over, the contract reverts the device's rental state to *unrented*, which means that the camera will no longer use the renter's information to send its feed. The camera then uses the owner's public key to calculate the symmetric key and redirects the encrypted stream to the owner's address. With the trigger message, the payment and change (deposit minus payment) are also sent to the owner and the tenant, respectively.

3.5 Hardware Collateral for the Smart Contract

For implementing the proposed smart contract, the IP camera's system-on-chip (SoC) needs to be equipped with an encryption engine, such as AES, DES, 3DES etc. Most of the modern IP cameras are equipped with such security engines for safeguarding the data [8]. Similar privacy protection methodology can be applied to any other IoT devices present in a home. Alternatively, a Smart Home Hub [187] can perform the privacy protection for all the IoT devices connected to it.

3.6 Protocol Demonstration and Discussion

We implemented the proposed protocol in `Solidity` programming language and evaluated in terms of its operational cost. The code is publicly accessible in Github repository [27]. In particular, we calculated the total cost by measuring the total gas amount (execution fee for each operation made in Ethereum) for all of the functions involved in the process, that is, (i) registering device (`registerDevice()`), (ii) transferring ownership (`transferOwnership()`), and (iii) transferring tenancy (`transferTenancy()`) and then converting it into USD (Table 3.1). For practical implementation, the protocol can be imple-

Table 3.1: Operation cost for the proposed smart contract transactions.

Operation	Gas limit	Gas price	Cost (in ETH)	Cost (in USD)
<code>registerDevice()</code>	121478	10.89×10^{-9}	0.0013229	0.463
<code>transferOwnership()</code>	30365	10.89×10^{-9}	0.00033067	0.116
<code>transferTenancy()</code>	23365	10.89×10^{-9}	0.00023067	0.08

mented in an application and all the steps, like creating transaction, verifying ownership, authenticating the device, transferring the tenancy can be done by using the application. Participants can manage their keys and addresses using a digital wallet.

3.7 Limitations and Discussion

Data Storage on the Blockchain In a decentralized sharing economy platform, the data need to be securely stored on the blockchain. However, storing all data on a blockchain database distributed across thousands of nodes is hugely inefficient. Therefore, not all data should be stored on the blockchain – only items that are sensitive and need to be stored securely like transactions, identities, core property information, reviews, etc. Items like property descriptions, indexes for searching, and photos change regularly and should not be stored within the blockchain. More centralized applications could be built on a secondary layer above the blockchain, which can act as a window into the system – but keep its core USP (User Services Platform) the same.

Reduced Operational Governance Control One major limitation of a blockchain-based sharing economy is reduced operational governance control of the provider’s actual service level. In general, federal, state, or local authorities always regulate businesses offering rental services. In the sharing economy, however, unlicensed individuals offering rental services may not obey these regulations. For example, there are reports of users who find their bikes provided by the owners of Mobike severely destroyed, with a little chance of compensation.

Listing All IoT Devices The hosts may have hidden video or audio surveillance equipment to monitor the guests. These hidden remotely-connected IoT devices violate the

guests' expectations of privacy. Our proposed protocol necessitates that the owner has to enlist all blockchain-enabled IoT devices in the smart contract for preserving privacy.

3.8 Concluding Remarks

Sharing IoT devices introduces new opportunities for innovation, but at the same time, it makes it challenging to ensure private access to the resources being shared. To avoid the cost overhead of a trusted third party supervising the private access to the resource, we proposed the alternative solution of using blockchain technology. Our proposed protocol preserves the IoT privacy by facilitating the change of encryption key via smart contracts.

CHAPTER 4

IMPROVING RELIABILITY OF WEAK PUFs VIA ACCELERATED AGING

4.1 Introduction

In the previous chapters, we have used Weak PUFs for device-unique key generation and authentication. Among the Weak PUFs, SRAM-like PUFs have become prominent in the industry as part of secure key/ID generation functions [98]. SRAM-like Weak PUFs utilize the mismatch in the device characteristics of the cross-coupled inverters, due to manufacturing process variation, to settle into a particular logical state upon power-on. These power-on states are random across cells. Ideally, a Weak PUF should reproduce the same output behavior during each power-on operation, i.e., the PUF should be *reliable*. However, due to environmental variations, noise or aging, a Weak PUF output can become erroneous. The problem is exacerbated if the inherent mismatch between the cross-coupled inverters in SRAM-like Weak PUF is small. Hence, the intended use of Weak PUFs for key/ID generation is negatively impacted.

Among the prior approaches to improve PUF reliability, accelerating device aging or burn-in has received increasing attention [42, 84, 148]. In accelerated aging, devices are subjected to temperature and voltage stress in a burn-in chamber. For SRAM-like Weak PUFs, the burn-in process can increase the inherent mismatch between the cross-coupled inverters so that a PUF output attains stronger immunity to noise.

While burn-in is beneficial, it can significantly inflate production costs due to long baking times to maximize the number of reliable integrated circuits (ICs). The straightforward way to determine the bake times is to account for the worst-case design corners. This worst-case design consideration can prove detrimental to the utilization of PUFs in *low-cost* applications, like Smart Cards, and for high volume manufacturing. Hence, there is a compelling need to reduce burn-in time. Integrating a mechanism for the IC to provide certain in-

formation to the manufacturer to aid in calculating the *minimum* burn-in time, without compromising the security of the PUF, can prove advantageous and offer a considerable increase in manufacturing throughput.

In this chapter, we present a method to reduce the burn-in time by quantifying the minimum burn-in requirements for each Weak PUF cell [113]. We use a low-cost proxy to represent the inherent cross-coupled inverter mismatch in terms of the PUF error rate. The error rates of the instantiated PUFs in an IC are measured and used to decide the burn-in requirements. We present a *low-overhead* architecture to automate the collection of necessary data. Also, the effect of alternate SRAM-like PUF designs and different transistor technology implementations on the burn-in process are analyzed.

4.2 Background and Motivation

In this section, we discuss some relevant background with regards to device aging and burn-in. We also discuss the temporal majority voting (TMV) technique in detail as we will utilize this in our methodology. We also present the prior research regarding previous techniques for improving Weak PUF reliability.

4.2.1 Weak PUF

In the hardware security context, literature works have proposed SRAM cells for use as Weak PUFs [98]. Due to process variation, the transistor parameters of an SRAM cell are skewed from their nominal value, making the cell asymmetrical. The random nature of process variation results in a unique start-up pattern of an SRAM memory array, which can be used as a device fingerprint. The most common design of an SRAM cell is the 6T structure consisting of two cross-coupled inverters and two access transistors. SRAM-based Weak PUFs generate their random bits by amplifying the mismatches in process variations in two (or more) transistors using a positive feedback structure. However, when these mismatches are small, different environmental variations, ambient noise, or aging can often affect the PUF outputs resulting in some bits of the raw PUF response being unreliable. Recent hardware studies have shown that various environmental variations can result in the unreliability of 6 – 8 % in the PUF response bits [142].

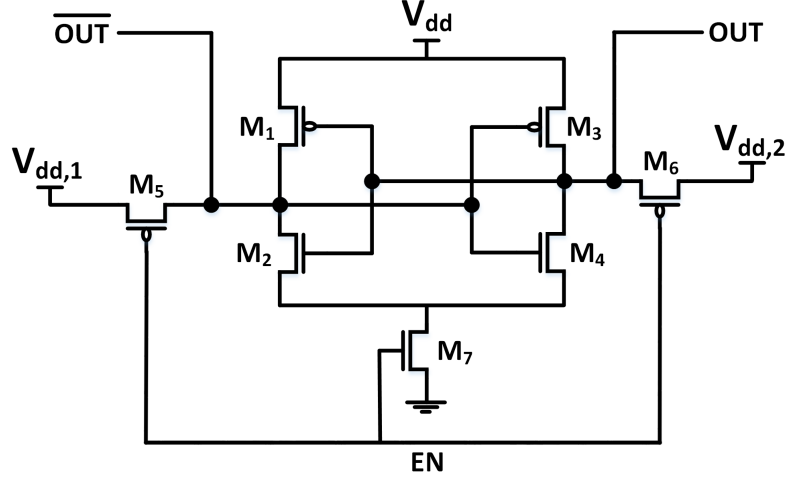


Figure 4.1: SRAM-like cross-coupled inverter PUF cell (*Ref*) [213]

To circumvent the need for multiple power-ups of the SRAM PUFs, we modify the basic cell, as shown in Figure 4.1, to efficiently allow multiple evaluations of a cell's output in the presence of noise [213]. The circuit, termed as *Ref*, works using pre-charge and discharge cycles. During pre-charge ($EN = 0$), the output nodes are raised to supply level ($V_{dd,1}$ and $V_{dd,2}$) and M_7 (footer) prevents short-circuit current consumption. In the evaluation phase ($EN = 1$), the process variation of the transistors creates discharge paths of different strengths. This allows the circuit to settle to either *logic-1* or *logic-0*. The effect of noise on the outputs is simulated by adding noise voltages to $V_{dd,1}$ and $V_{dd,2}$. This design is similar to changes required to enable Temporal Majority Voting (TMV) in related work [148] and to Sense-Amplifier PUF [41].

4.2.2 PUF Reliability

Fuzzy extraction has been explored to derive stable keys from biometric data and, for authentication of such data [70]. For improving the reliability of Weak PUFs, application of error-correction codes (ECCs) and fuzzy extractors requires generating and utilizing helper data [50, 64, 143, 144, 146], which is made public. For ECC implementations, a stable key of the desired length needs a large number of initial Weak PUF bits, even with helper data [50]. The intrinsic error rate of a Weak PUF cell increases as the process variation reduces, which is the case as a technology node matures. An increase in PUF error rate increases the

initial bits required to derive a stable key of the same length and increases the helper data needed for fuzzy extraction and error correction.

Circuit and device-level solutions can improve the Weak PUF cell reliability, providing alternatives to costly ECC. Pre-selecting SRAM-based Weak PUF cells that exhibit a greater degree of transistor threshold voltage mismatch has been proposed by Hofer *et al.* to improve the reliability of the system [97]. Layout techniques and resettable logic were shown to reduce the influence of systematic process variations by Su *et al.* [199]. Jang and Ghosh [116] proposed extensions over a conventional 6T SRAM cell by incorporating PMOS latches and using Magnetic Tunnel Junction (MTJ) devices to protect against environmental fluctuations. Alternate configurations of the inverters in the SRAM cell have been explored by Ganta and Nazhandali to reduce the variations with respect to temperature and hence, decrease the number of unreliable bits that need error correction [83]. Bucci and Luzzi [52] constructed a differential circuit to capture the process mismatch and amplify it to reduce the effect of noise on the PUF cell output. Patil *et al.* [166] further explored various configurations of SRAM-like Weak PUFs that exhibit higher sensitivity to process variations, resulting in greater resilience to thermal noise. The higher sensitivity allows the PUF cells to be immune to noise even in mature technology nodes. Cortez *et al.* [60] proposed adapting voltage ramp-up time to ambient temperature to reduce the error rate of memory PUFs. However, the auxiliary circuits needed for voltage ramp-up accrue a large area overhead, and shaping the supply voltage becomes complex in large circuits.

The effect of aging on PUF reliability has been studied extensively by Garg *et al.* [84], and Maes *et al.* proposed techniques to counter the effects [145]. Bhargava *et al.* [42] utilized aging via Hot carrier injection (HCI) aided PUF reliability.

4.2.3 Temporal Majority Voting

The area overhead from implementing traditional ECC blocks and the number of initial PUF bits required scales superlinearly as the error rate increases. A simple, circuit-based way to reduce the error rate using Temporal Majority Voting (TMV) has been explored by Mathew *et al.* [148] and Xiao *et al.* [220]. Mathew *et al.* [148] also explored burn-in and dark bits evaluation to reduce error rates. Design changes were required to enable voting,

and synchronous design helped improve uniqueness. However, the approach can only correct error rates of $< 8\%$, and additional techniques are necessary for practical applications.

In this work, we will assume a simple counter-based TMV for error correction during the regular operation of the PUF. For example, a simple 4-bit counter-based TMV counts from 0 to 15 and can be used for 15-way voting. If the resultant value after 15 evaluations of the cell's response is greater than 8, then the final value can be classified as 1, or else it can be classified as 0. The mathematical model of the TMV is a binomial counting process, and hence, the reduction in error rate can be calculated analytically. For example, a PUF cell whose error rate is $1 - p$ produces a final error rate of $P_e(N)$ given by,

$$P_e(N) = \sum_{m=k}^N \binom{N}{m} p^m (1-p)^{N-m} \quad (4.1)$$

where $k = (N + 1)/2$ (as N is odd) when an N -way voting is used [132]. The circuit implementation of the TMV typically consists of an n -bit counter where $N = 2^n - 1$. The counter is incremented by 1 if the response from the PUF cell is 1.

Using (4.1), we plot the initial and final error rates for 4-bit (15-way), 5-bit (31-way) and 6-bit (63-way) TMVs in Figure 4.2. The TMV selection would be based on the maximum error rate that the system needs to correct (final error rate $\leq 10^{-6}$). We utilize a 4-bit TMV for regular operation for this work, noting that it can correct a maximum error of 6%.

4.2.4 Negative Bias Temperature Instability

Transistor aging has become a significant reliability concern for current CMOS technology. Among various aging mechanisms, Bias Temperature Instability (BTI) is considered the dominant aging mechanism, causing the threshold voltage of the transistor to increase. There are two BTI mechanisms: (i) Negative BTI (NBTI) affecting the PMOS transistors and (ii) Positive BTI (PBTI) affecting the NMOS transistors. NBTI results from continuous trap generation in $Si-SiO_2$ interface when a negative voltage is applied to the PMOS gate (stress). Under stressed operating conditions (i.e., On-state, negative gate bias at elevated temperature and supply voltage), $Si-H$ bonds near the interface continue to break and gen-

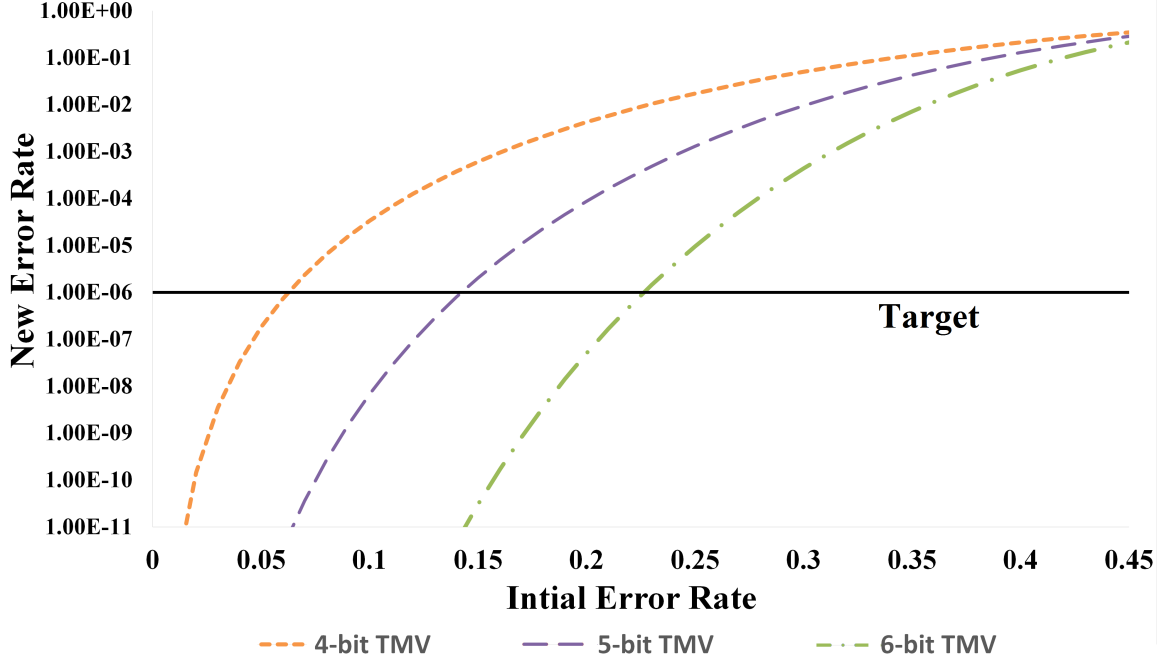


Figure 4.2: Error rate reduction due to Temporal Majority Voting

erate interfacial traps that contribute to an increase in V_{th} . Due to the V_{th} degradation, NBTI results in poor drive current, lower noise margin, and shorter device lifetime.

The NBTI-induced threshold voltage shift is a function of supply voltage, temperature, and many technology parameters. Various models have been proposed in the literature to accurately estimate the threshold voltage degradation of ΔV_{th} due to NBTI. Kang *et al.* proposed a compact threshold voltage degradation model considering the temporal NBTI variation in short channel devices [121]. Paul *et al.* showed that the maximum circuit delay degradation due to NBTI closely follows the same fractional power dependency to time as the V_{th} degradation with an appropriate scale factor [167]. Kumar *et al.* have proposed an efficient AC NBTI model for circuit simulations [130]. Vattikonda *et al.* have proposed a further improved circuit compatible NBTI model to consider AC relaxation effects and technology-dependent parameters [211].

FinFET: The NBTI model for FinFETs is the same as (4.2.4). The effect of NBTI and mitigation techniques for FinFET SRAMs have been explored in detail by Wang *et al.* [216].

According to the Reaction-Diffusion (RD) model [40], the BTI-induced threshold voltage shift is:

$$\Delta V_{th}(t) = \left(\frac{\sqrt{K_v^2 \alpha T_{clk}}}{1 - \beta_t^{1/2n}} \right)^{2n}$$

$$\beta_t = 1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(1 - \alpha)T_{clk}}}{2t_{ox} + \sqrt{Ct}} \quad (4.2)$$

$$K_v = f(V_{dd} - V_{th}, T)$$

where α is the duty cycle, T is the temperature, T_{clk} is the clock cycle, and other parameters are technology parameters previously defined in [40].

4.2.5 Burn-In (Accelerated Aging)

IC designers and manufacturers are concerned about quality and reliability over a product's lifetime. To ensure economic viability, it is desirable to remove defective devices from the population before shipping them to the customer. Consequently, many ICs undergo a *burn-in* process after fabrication to accelerate failures that manifest in early-life, which are primarily caused by process and manufacturing defects. However, under burn-in conditions, increased junction temperature (average temperature of the silicon substrate) increases the leakage current, and increased leakage current further increases the junction temperature. Thus, manufacturers try to control the junction temperature by removing the heat from the IC. If the heat generation rate becomes greater than the rate of heat removal, junction temperature starts increasing. This condition is called *thermal runaway* [209]. It has been shown that the burn-in setup conditions must evolve by reducing either the ambient temperature or the thermal resistance, or a combination of both. For example, in 130 nm process technology, the junction temperature should be kept below 110 °C with a thermal resistance of 0.5 °C/W and an ambient temperature of 80 °C to avoid thermal runaway [209].

In the case of a Weak PUF, the response can be made more reliable by increasing the magnitude of the difference in the threshold voltages of the two PMOS devices in the cross-coupled inverters (M_1, M_3), in Figure 4.5. One such method of improving the reliability is to exploit NBTI aging effects to *reinforce* the desired (or “golden”) response of the PUF cell. This is done by finding the *golden* outputs (OUT, \overline{OUT}) of the PUF cell and forcing the opposite values onto them via the access transistors (M_5, M_6). Increasing the temperature

and/or applying voltage stress for a certain amount of time accelerates the devices' aging in a beneficial manner [42, 84, 148]. Hence, by improving the PUF reliability via burn-in, the number of defective ICs is reduced.

4.2.6 PUF Reliability using accelerated aging

The aging effect on PUF reliability has been studied extensively by Garg *et al.* [84], and Maes *et al.* proposed techniques to counter the effects [145]. Bhargava *et al.* [42] utilized aging via Hot Carrier Injection (HCI) aided PUF reliability.

4.3 Methodology

This section discusses the PUF system design to enable the IC to output the maximum error rate observed. Later, we elaborate on a mechanism to correlate the error rate to the inherent mismatch in the PUF cell that allows us to find the burn-in time required to make all the PUF cells reliable.

4.3.1 Weak PUF System Design

In Figure 4.3, we describe the proposed system to measure the maximum error rate in a PUF. We described this system initially in our paper [112]. The system consists of a PUF cell array connected to an array of multiplexers (*Muxes*). These muxes will direct the PUF outputs to the relevant counters based on the mode of operation. A *Central Control* unit oversees the entire operation of the PUF system. The circuit operates in two modes: (i) Design for Reliability (DfR) mode and (ii) Regular Operation (OP) mode.

(i) *Design for Reliability (DfR) mode*: Before burn-in, the circuit is initialized into (*DfR*) mode to obtain the maximum error rate among the PUF cells. The control unit directs each PUF output via the MUX array to a 10-bit counter and performs 1024 pre-charge and evaluations cycles on each cell. During every evaluation, the counter increments if the output is *logic-1*. The large counter allows us to get an accurate measurement of the error rate of a cell. Also, a single 10-bit counter is enough as this process is carried out before burn-in and is not time intensive like burn-in. Even if a PUF cell takes 1 ns for each evaluation, then each cell error rate is obtained in $\approx 1 \mu s$.

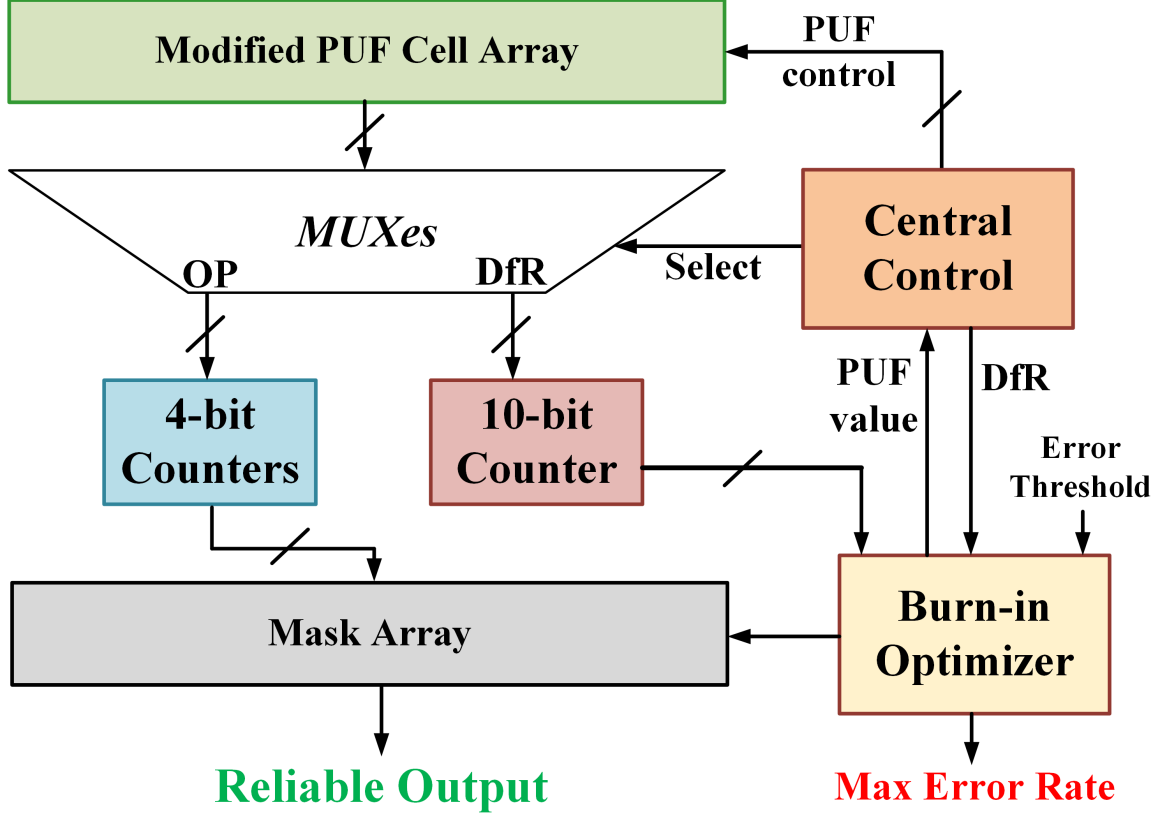


Figure 4.3: Block Diagram of the proposed reliability enhancement scheme (from [112], our earlier paper)

The counter value is fed to the *Burn-in Optimizer* unit that first determines the correct PUF cell output. If the count is below 512 (ideally 0), then the correct output is determined to be logic-0. If the counter is between 0 and 512, this count becomes the error rate for the cell (out of 1023). For logic-1, the count would be > 512 (ideally 1023). To find the error rate, we subtract the current count from 1023. The estimated logic value (PUF value) of the PUF cell is fed back to the Central Control unit, which then writes the opposite value to the PUF cell for burn-in.

The *Burn-in Optimizer* also maintains an internal register that stores the current maximum error rate (initialized to 0 on start-up). Each calculated error rate is compared to the current maximum using a comparator and set as the new maximum if the error rate is higher. All PUF cells are processed to obtain the final maximum error rate. Further optimization is possible by utilizing the maximum error, 6%, that a 4-bit TMV can correct.

The *Error Threshold* can be set to ≈ 62 (6% of 1023), and each calculated error rate is compared against this before comparing with the stored maximum. The final IC output (*Max Error Rate*) is the maximum error only if it is greater than the error threshold, and otherwise, the output is a 0. Such ICs would not need burn-in as the TMV is sufficient.

In some instances, the designer may wish to account for an acceptable yield loss and use more PUF cells than required. Hence, the *Mask Array* can be utilized to select the most reliable PUF cells. The *Burn-in Optimizer* is used to set the mask bits to indicate reliable cells. It can also possess an additional counter to keep track of cells with error rates of $\leq 6\%$ (for TMV). In case the system finds the required number of cells, it can output a 0 max error rate to reduce burn-in requirements further. After burn-in, the *Burn-in Optimizer* can be reused to set the final mask bits. Masking has been shown to help improve the reliability of the Weak PUF system [148]. The Burn-in Optimizer’s entire operation is illustrated as a flowchart, as shown in Figure 4.4.

(ii) *Regular Operation (OP) mode*: In this mode, the control unit queries and directs the PUF outputs to the 4-bit TMVs. We can use multiple TMVs to speed up the PUF evaluations for convenient real-time operation. As the burn-in process will have increased the mismatch of the PUF cells by an appropriate amount, the TMVs should be able to correct any observed errors as they will be well below the TMV threshold.

4.3.2 Process Variation and Error Rate

We utilize the SRAM cell design, whose operation is detailed in Section 4.2.1 and re-illustrated in Figure 4.5 as the reference design, termed *Ref*.

Utilizing the maximum error rate produced by a PUF system, we aim to find the current inherent mismatch, in terms of threshold voltages between the PMOS transistors (M_1 , M_3), in Figure 4.5. This maximum error rate acts as a proxy representing the inherent mismatch between the cross-coupled elements in the actual PUF cell and will help us determine the amount of NBTI aging needed to make the cell reliable.

To correlate the error rate with a threshold voltage mismatch value, we perform a set of SPICE simulations on the Weak PUF cell. Circuit thermal noise is considered as the source of errors in the PUF output. The noise is applied to the circuit in a differential manner at

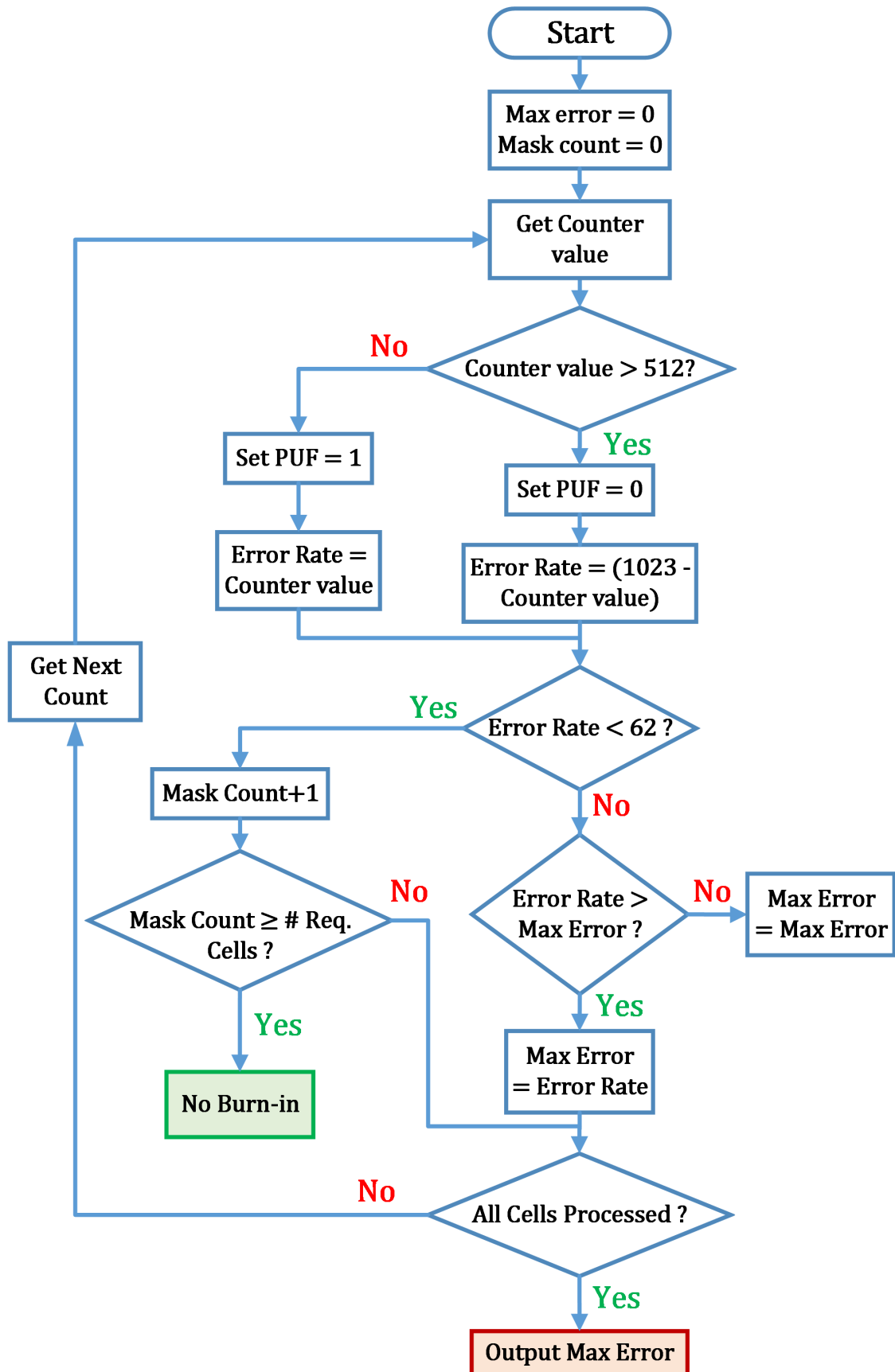


Figure 4.4: Flowchart illustrating the operation of *Burn-in Optimizer*

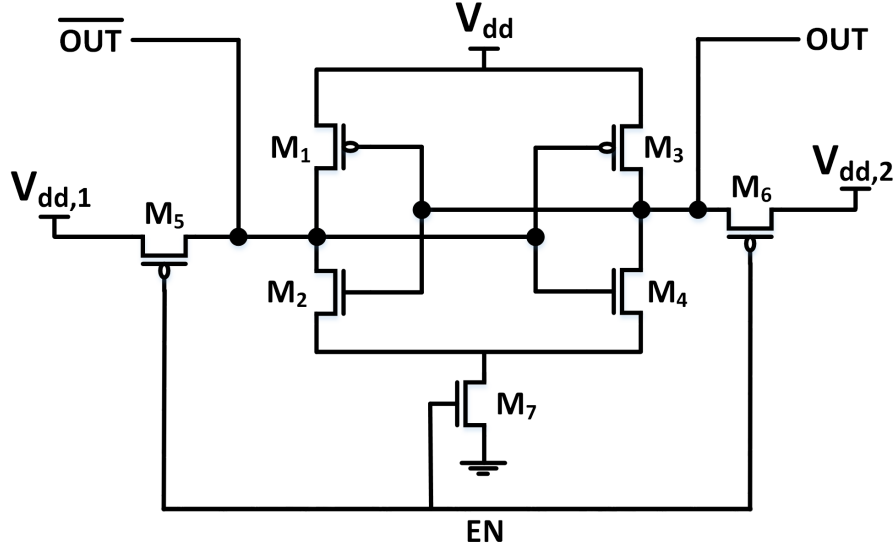


Figure 4.5: SRAM-like cross-coupled inverter PUF cell (*Ref*) [213]

$V_{dd,1}$ and $V_{dd,2}$, as shown Figure 4.5. The inherent mismatch of a PUF cell is *approximated* by varying the threshold voltage in one of the PMOS transistors (M_3 in our case) in steps of 1 mV up to a certain maximum. A large number of evaluations are performed under varying thermal noise for each step, and the error rate is calculated by comparing the values with the output for zero thermal noise.

An example, as shown in Figure 4.6, plots the error rates against the increasing threshold voltage mismatches for a PUF cell, as shown in Figure 4.5, in 45 nm technology [18] using minimum sized transistors. This plot gives us an *approximate* correlation between the error rate of a cell and the inherent PMOS transistors mismatch. Knowing that a 4-bit (15-way) TMV can correct a maximum of 6 % error, we see that any cell with an inherent mismatch of ≥ 19 mV can be handled by the TMV. The cells with lower mismatch are not TMV-correctable and become the target of our burn-in efforts. The worst-case threshold voltage shift, when the maximum error rate is 50 %, for the burn-in process is set at a higher value than 19 mV to account for the approximations made in our analysis. This value becomes the total target mismatch required to create a fully reliable PUF system. We also note that an error rate of 50 % would represent the ideal true random number generator (TRNG). Based on the observed maximum error, which represents the lowest mismatch observed among

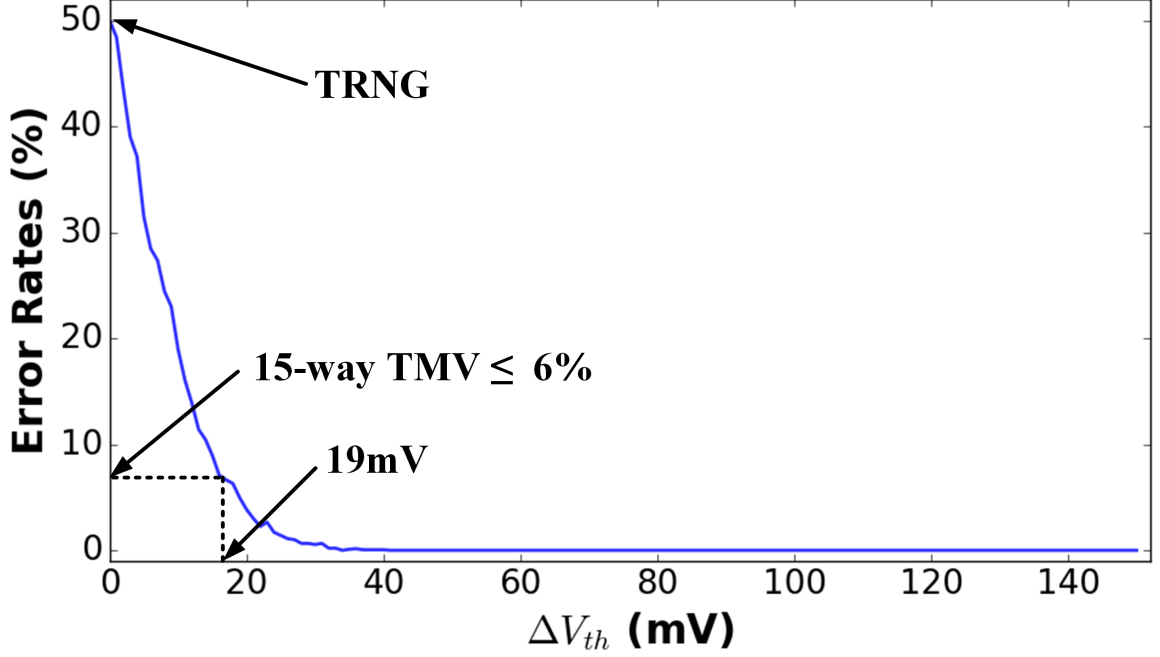


Figure 4.6: Error rate correlation with PUF cell threshold voltage mismatch

the PUF cells of a particular IC, we can calculate the threshold voltage shift required from the burn-in process to reach the target mismatch. In a real-world scenario, the maximum observed error rates are likely to be below 50%, and hence, the needed threshold voltage shift from burn-in will be less than the set target value. This reduced threshold voltage shift translates into significant savings with regards to burn-in time.

4.4 Burn-in time reduction

In this section, we explore the advantage of using the proposed solution presented in Section 4.3 considering Weak PUF systems that need to generate 128 reliable bits. The reported results consider a *cumulative* burn-in time, where we assume that only a single IC undergoes the accelerated aging at a time. In practical situations, different manufacturers can utilize different ways. Hence, it is difficult to assume just one arbitrary process.

4.4.1 Weak PUF Designs

Along with the simple SRAM-like Weak PUF design (*Ref*), we extend the analysis using other PUF designs proposed in our earlier work [166]. It was shown that connecting

Table 4.1: Implementation details for various Weak PUF design configurations

Configuration (<i>Identifier</i>)	Transistor Sizing		2-parameter model values [141]		
	PMOS	NMOS	$\lambda 1$	$\lambda 2$	Cells with 0 error (%)
Simple (<i>Ref</i>)	90 nm	90 nm	0.292	1.906	83
Two Parallel Loads (<i>D1</i>)	180 nm	90 nm	0.188	2.395	96
Current Mirror Load (<i>D2</i>)	180 nm	90 nm	0.191	2.491	97
FinFET (<i>F1</i>)	1 fin	1 fin	0.413	1.595	60

either the pull-up or pull-down transistors of the cross-coupled inverters in the PUF to a bias voltage can significantly benefit reliability. Since we consider NBTI as the aging mechanism of interest for the burn-in process, we modify the circuits so that the pull-down transistors are connected to bias voltages while the pull-up transistors are cross-coupled to form the inverters. For this work, we choose the parallel active loads design with two parallel NMOS transistors (*D1*), as shown in Figure 4.7, and the current mirror-based design with NMOS current mirrors (*D2*), as shown in Figure 4.8. The access transistors and the pre-charge voltages connected to *OUT* and \overline{OUT} are the same as the *Ref* design, shown in Figure 4.5, and are omitted in the circuit diagrams for clarity. The transistors are sized to allow maximum process variation sensitivity except for the footer (M_2), which is sized larger to allow the circuit’s proper operation. V_{bias} was set to 0.5 V for *D1* and 0 V for *D2*.

Conventional CMOS scaling beyond the 45nm technology node is severely constrained by pronounced threshold voltage (V_{th}) fluctuations resulting from Short Channel Effects (SCE) and Random Dopant Fluctuations (RDF) due to process variations [71,180,198,207]. Hence, FinFETs were developed to facilitate the continued scaling of technology nodes. FinFET devices exhibit better electrostatic characteristics with respect to SCE as the gate, or *fin*, wraps around a thin slice of silicon (channel) [177]. The greater control over the channel allows FinFETs to have lower leakage current and power consumption over bulk CMOS. We wished to study the effect on burn-in requirements when using FinFETs as the basis for constructing the Weak PUF. We considered only the *Ref* design, as shown in Figure 4.5,

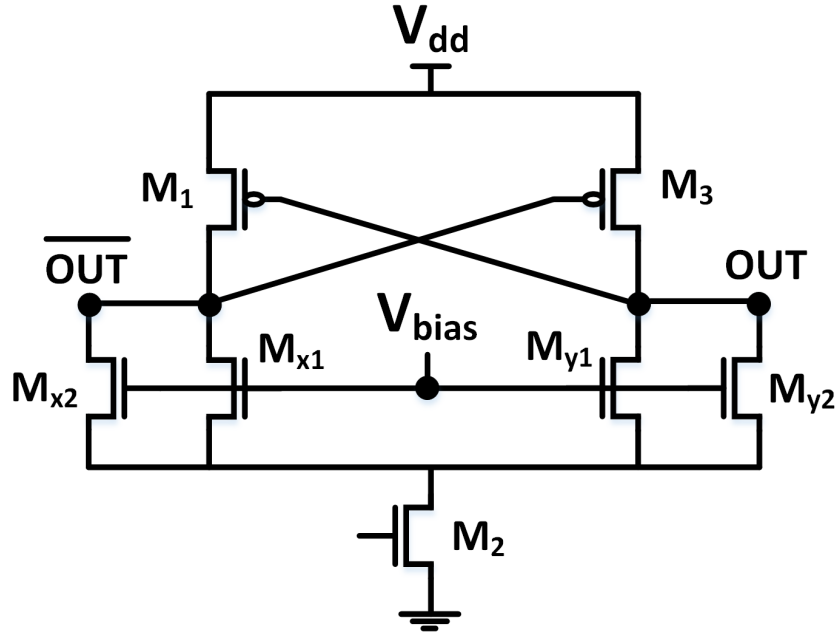


Figure 4.7: Modified parallel active loads-based PUF design ($D1$) [166]

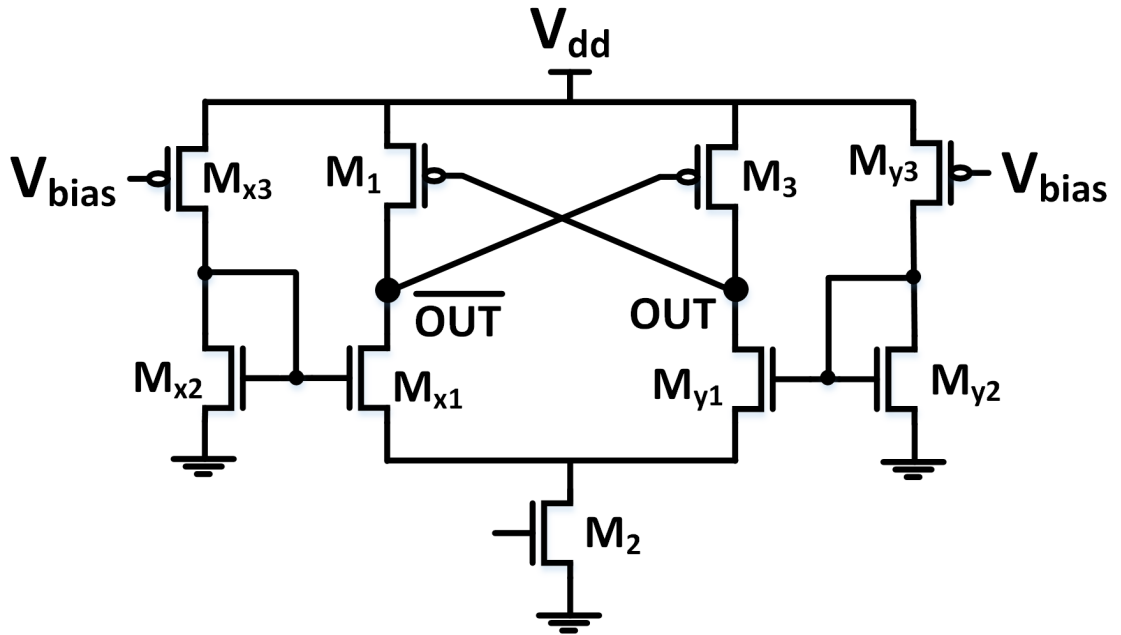


Figure 4.8: Modified current mirror-based PUF design ($D2$) [166]

and instantiated the PUF, termed as $F1$, using 20 nm FinFETs using predictive technology models [19]. We fixed the fin number at 1 for all transistors except the footer (M_7 in

Ref), which had 2 fins for proper current sinking. We set the supply voltage at 0.9 V. The specifications for the various designs are tabulated in Table 4.1.

4.4.2 Thermal Noise Errors

The random motion of the charge carriers from thermal excitation induces thermal noise in transistors and can create random voltage fluctuations in conductors [120, 160]. Thermal noise has a near-uniform power spectral density, and there is no correlation among different samples across time. Short channel effects [87] can exacerbate the effects of thermal noise in advanced CMOS technology nodes, causing a significant impact on transistor noise performance [205]. The thermal noise at any given node can be represented with a normal distribution with 0 mean, and the standard deviation is given by [188],

$$\sigma_{NOISE}(T) = \sqrt{\frac{k_B * T}{C}} \quad (4.3)$$

where k_B is the Boltzmann constant, T is the absolute temperature (Kelvin), and C is the node capacitance (Farads).

We record the node capacitances at *OUT* and \overline{OUT} in each design (*Ref*, *D1*, *D2*, *F1*), under no process variation condition, to determine the amount of thermal noise to be added to $V_{dd,1}$ and $V_{dd,2}$.

4.4.3 Error Rate vs Mismatch

Using the procedure described in section 4.3.2, we seek to find the correlation proxies for each of the designs being considered. For the planar MOSFETs (*Ref*, *D1*, *D2*), we utilized the 45 nm technology [18] to instantiate the cells. The supply voltage (V_{dd}) is set to 1 V. We shift the threshold voltage of one of the PMOS transistors (M_3) in steps of 1 mV up to a maximum of 150 mV. This threshold voltage sweep represents the proxy for the total mismatch and helps simplify further analysis. At each step, we perform 2000 evaluations of the cell under varying thermal noise conditions, using (4.3), and calculate the error rates. Figure 4.9 shows the results which highlight the fact that the alternate designs (*D1* and *D2*) have a greater process sensitivity than *Ref* as every step increase in mismatch reduces the observed error significantly and the error rate reaches 0% with a lower amount of mismatch.

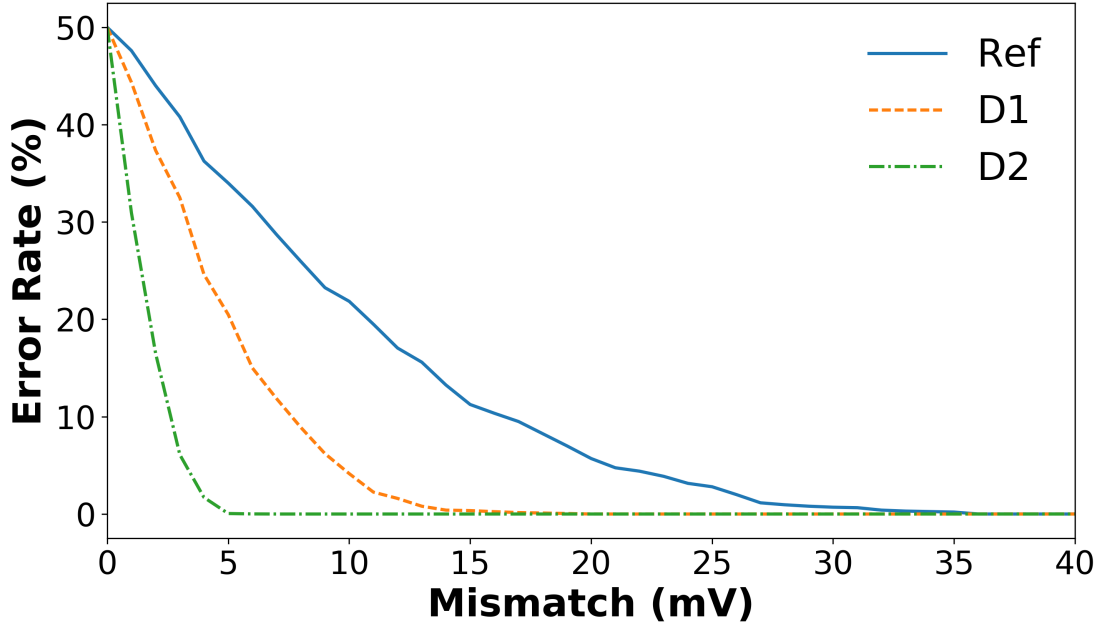


Figure 4.9: Error rate correlation with PUF cell threshold voltage mismatch for alternate Weak PUF designs ($\{D1, D2\}$) based on [166]

In cases where incorporating the alternate designs may not be desired, we sought to explore a technique to improve the performance of the existing SRAM-based PUF (*Ref*). Boosting the supply voltage is beneficial for the operation of an SRAM [169]. However, we must also be aware of an increase in leakage power. For this work, we study the performance of *Ref* when the supply voltage is boosted to 1.2 V. As shown in Figure 4.10, the results indicate a decrease in the error rate with increased mismatch under higher supply voltage. A designer can generate any number of such proxies at different voltages and use the data to adaptively choose what supply voltage needs to be set for the PUF block, which decides the burn-in time. This is because we only output the maximum error rate from an IC that is later used with the correlation data. However, such supply adaptability increases the complexity of the overall system design.

Replacing the planar MOSFET based PUF (*Ref*) with FinFET-based design (*F1*) provides better performance in cases where the inherent mismatch is higher, as shown in Figure 4.11. We note here that the FinFETs are more susceptible to thermal noise (from (4.3)) as the node capacitances are lower due to the smaller technology node (20 nm) compared to

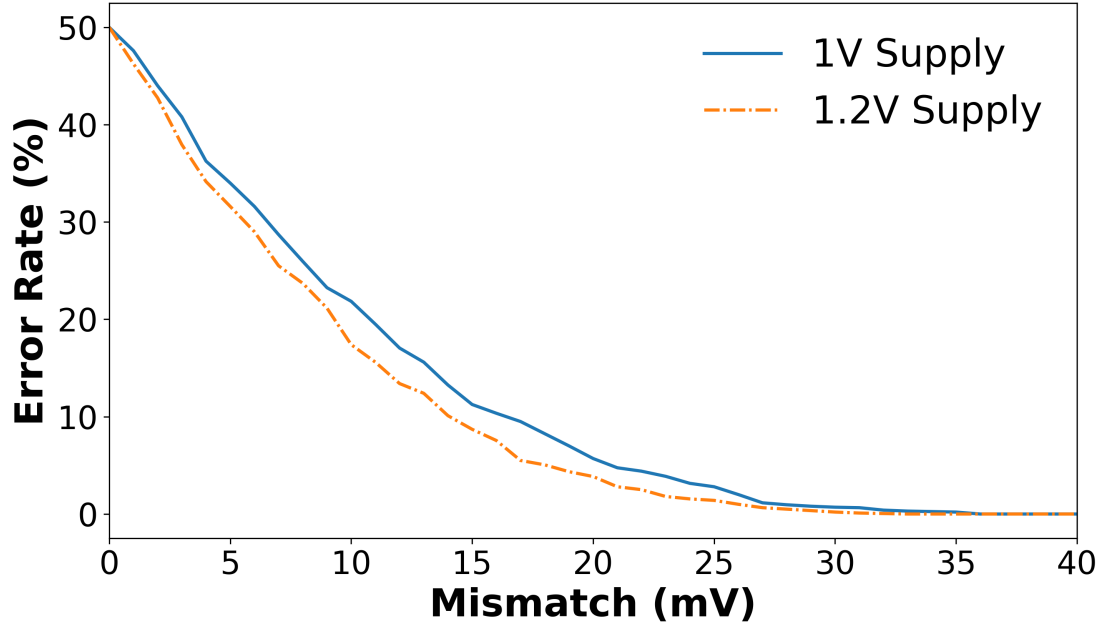


Figure 4.10: Error rate correlation with PUF cell threshold voltage mismatch for *Ref* under nominal and boosted supply voltage (1.2 V)

the planar MOSFET. The results show that FinFET based implementations can be viable as Weak PUFs even with the higher noise.

4.4.4 Modeling Process Variation

4.4.4.1 Planar MOSFET

Manufacturing induced process variations are modeled as *random* parametric variations in the threshold voltage (V_{TH}) and channel length (L) of each transistor in a circuit. The values are obtained from a normal distribution, $N(\mu, \sigma^2)$, where the mean (μ) and standard deviation (σ) are determined based on the technology node used. Transistor geometry impacts the susceptibility of a device to process variations, with larger devices experiencing fewer fluctuations. In terms of threshold voltage, the mean is the default transistor model value, and the standard deviation is given by,

$$\sigma_{V_{TH}} = \frac{\sigma_{V_{TH0}}}{\sqrt{\frac{W*L}{W_{min}*L_{min}}}} \quad (4.4)$$

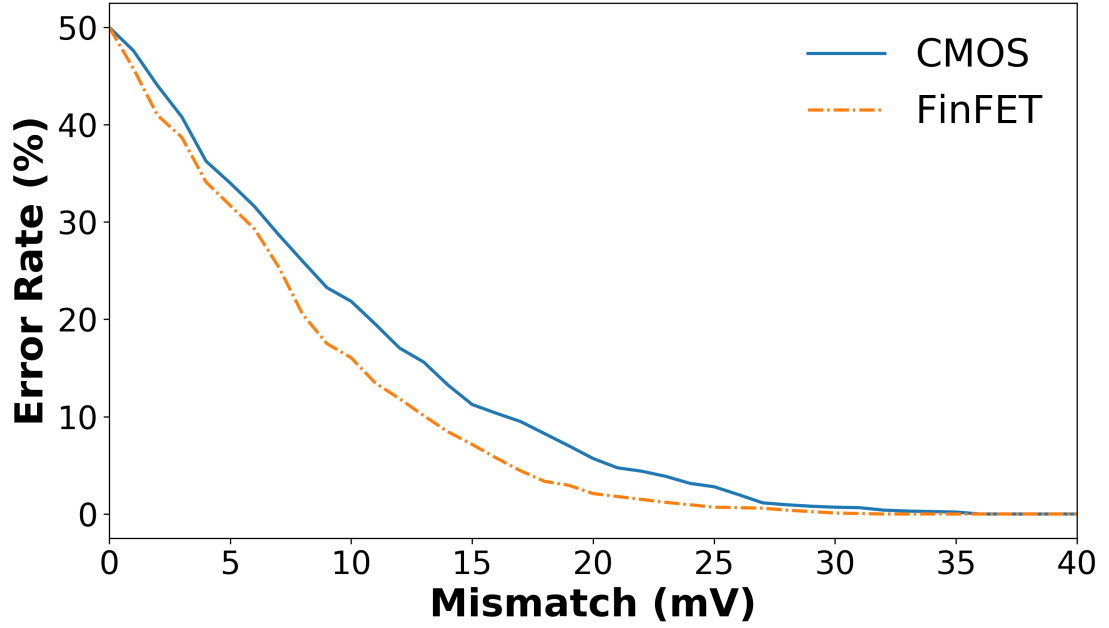


Figure 4.11: Error rate correlation with PUF cell threshold voltage mismatch for planar MOSFET (*Ref*) and FinFET (*F1*) designs

where (W_{min}, L_{min}) are the minimum possible width and length of a device, respectively, and (W, L) are the sizes used in the design. $\sigma_{V_{TH0}}$ is the standard deviation of threshold voltage for the minimum sized device.

In this work, we instantiate the planar MOSFET PUF cell designs (*Ref*, *D1*, *D2*) with 45 nm NCSU FreePDK45 models [18]. Typically, for 45 nm node, a standard deviation of 53 mV for threshold voltage and 10 % channel length variation are considered [16]. We used equation (4.4) to calculate the standard deviation for threshold voltage of any arbitrary sized transistor.

4.4.4.2 FinFET

While FinFETs are not affected by RDF due to undoped channel, they are susceptible to Work Function Variation (WfV) caused by irregularities in *fin* surface from the manufacturing process [149]. Hence, WfV has the most significant impact on threshold voltage variation. For this work, we consider a standard deviation of 30 mV for threshold voltage to represent the process variation in FinFET transistors [149].

4.4.5 Heterogeneous Error Model

In realistic scenarios, a collection of Weak PUF cells would have a distribution of errors. This heterogeneous error model is advantageous in modeling real PUF systems as a homogeneous error model can overestimate the required ECC resources [141]. The homogeneous model would also provide no useful information for driving the burn-in process as all cells would be assumed to have the same error rate and, hence, the same process mismatch.

For this work, 10,000 instances of the PUF cell, for each design, were simulated with the relevant process variation parameters, as described in sections 4.4.4. Each cell was evaluated 1000 times under varying thermal noise (as determined in section 4.4.2) at 25 °C. We obtained the error rate by comparing it with a simulation with no noise. Utilizing the resulting data with the mathematical framework for the heterogeneous error model [141], we can generate error rates for an arbitrary number of PUF cells for each given design. The relevant 2-parameter model details for each PUF design are tabulated in Table 4.1. We also list the percentage of cells among an arbitrary population that would possess 0 % error for each design, given the 2-parameter model data. Due to the higher thermal noise susceptibility, FinFET implementation (*F1*) offers the lowest amount of 60 %. This observation indicates that the FinFET based design will offer fewer savings than a comparable planar MOSFET design in terms of total burn-in time.

4.4.6 Cumulative Burn-in Time

Since we targeted to obtain 128 stable bits from a PUF system, we first considered the system had 128 initial PUF cells, and all the cells needed to be reliable. Next, we considered a scenario where a 2 % yield loss might be acceptable, and *masking* is used. For this case, we reused (4.1) with $\{k = 128, p = 0.98\}$ to find the value of N that would result in $P_e(N) \leq 10^{-6}$ (failure rate of 1 ppm). Results showed that we needed 144 initial PUF cells.

For all the designs considered in this paper (*Ref*, *D1*, *D2*, *F1*), we assume that the burn-in temperature is 100 °C and the stress voltage is 1.1 V and utilize (4.2.4) to calculate the time required. The relevant device parameters for the NBTI equation are obtained from the NCSU FreePDK models [18] for planar devices and from PTM models for the FinFET [19].

Table 4.2: Results for reduction in Cumulative Burn-in time for various PUF configurations

Configuration	# PUF Cells	Cumulative Burn-in Time (hours)			Target Mismatch (mV)	Natural Aging Time
		Worst-case	Optimized	Reduction (%)		
<i>Ref</i>	128	41.52E6	17.93E6	<i>56.82</i>	25	700 days
	144		0.62E6	<i>98.50</i>		
<i>Ref</i> (1.2 V)	128	25.15E6	15.92E6	<i>36.70</i>	23	425 days
	144		0.42E6	<i>98.33</i>		
<i>D1</i>	128	0.49E6	0.114E6	<i>79.73</i>	12	9 days
	144		0	<i>100.00</i>		
<i>D2</i>	128	90.2	43.77	<i>51.48</i>	4	16 mins
	144		0	<i>100.00</i>		
<i>F1</i>	128	10.32E6	8.62E6	<i>16.47</i>	22	295 days
	144		0.18E6	<i>98.25</i>		

For obtaining the cumulative burn-in time, 1 *million* PUF systems were generated for the 128 and 144 cells/system cases for each Weak PUF design using the heterogeneous error model. For each PUF system, the threshold voltage mismatch was found using the methodology described in section 4.3. To find a necessary increase in device mismatch in each system, the data from section 4.4.3 and the mismatch representing the 6 % error rate (for 4-bit TMV) were used. Our target threshold voltage shift (also the worst-case shift) was set by increasing the TMV mismatch found for each design by 30 %. This is to account for the approximations in our methodology for correlating error rate and inherent mismatch. Table 4.2 also lists the amount of time needed to increase the PMOS threshold voltage due to NBTI by the target mismatch value.

For each system, the max error rate indicates the lowest amount of inherent mismatch and decides the threshold voltage shift needed to reach the target. The shift needed is used in (4.2.4) to calculate the burn-in time. The cumulative burn-in time (*Optimized*) for 1 *million* PUF systems with 128 and 144 initial cells are recorded in Table 4.2 for each design. We also note that a manufacturer might need to assume that each IC needs to undergo the maximum burn-in for each design without the proposed solution.

From the results, in Table 4.2, we see that intelligent burn-in offers significant savings compared to a constant worst-case scenario-driven burn-in. Also, using extra bits (144) and masking results in reduced burn-in time compared to using just 128 cells/system. *D2* offers the best results for any PUF system as the burn-in required is negligible compared to other designs. Additionally, combining the alternate designs (*D1* and *D2*) with masking and extra bits can allow us to forgo burn-in entirely. Consequently, these designs are attractive for low-cost applications where dedicated PUF circuits would make more sense regarding resource utilization. The FinFET (*F1*) results show that we need to use extra bits for greater reliability, but the lower cell area for the technology allows us to incorporate more cells easily.

The results discussed here show the *relative* performance of various designs, but are ultimately dependent on the amount of error correction afforded to us by the TMVs used during regular operation. Vijayakumar *et al.* have shown that using an Up-Down counter [213] allows them to correct *twice* the error compared to a similarly sized regular TMV. In

actual designs, using better error correction reduces the target mismatch required by the burn-in process, but a designer must weigh such considerations against various constraints such as available area, power requirements, and so on.

Resource Overheads: We considered a 144-PUFs system and calculated the area overhead from the *Burn-in Optimizer*, *Central Control*, *Mask Array* and 10-bit counter described in section 4.3. The rest of the circuitry for TMV-based reliable bit generation is assumed to be present in the system. Using the 45 nm standard cell library from Nangate [17], the area overhead was found to be $500 \mu m^2$ (Burn-in optimizer costs $149 \mu m^2$). This area is a small overhead as most of the area ($\approx 2500 \mu m^2$) for an efficient implementation is occupied by the TMV counters, PUF cells, and the mux arrays.

The primary testing time overhead is from obtaining the maximum error rate, which entails serially querying each PUF cell 1024 times (10-bit counter) and processing the results. For a 144-PUFs system, this requires $\sim 150,000$ cycles in total. At a test frequency of 10 MHz, for example, each chip will need 15 ms to generate the results. However, in cases where no burn-in is required, the designer can directly proceed to the final key enrollment phase. The Burn-in Optimizer is only used to set the Mask Array, and the system operates at its final intended frequency, which is greater than the test frequency.

4.5 Concluding Remarks

Literature works have extensively proposed Weak PUFs for security applications such as key/ID generation. Such applications require the PUFs to be highly reliable even in the presence of noise. To ameliorate the noise susceptibility of the PUF outputs, many different techniques have been proposed. One such method is to utilize burn-in/accelerated aging for improving PUF reliability. Our work focuses on creating a PUF system that enables the calculation of minimum burn-in time for an IC. We obtain results for various SRAM-like Weak PUF designs, which show a significant reduction in burn-in times, providing large savings during the manufacturing process’s post-silicon stages. Further, results show that FinFET based implementations can also be viable as Weak PUFs even in the presence of significant noise.

CHAPTER 5

PMU-TROJAN: ON EXPLOITING POWER MANAGEMENT SIDE CHANNEL FOR INFORMATION LEAKAGE

5.1 Introduction

In a blockchain network, any entity connected to other entities in a peer-to-peer manner is called a client. All these clients talk to each other and form a network where each client works as a node. Nodes are responsible for verifying and relaying the transactions and blocks on the network [157]. The nodes which download and verify every single block, and therefore every single transaction in each block, are referred to as full nodes. A full node requires time and resources, as the download and validation process is particularly heavy on CPU and disk IO. For example, it is recommended that a full node requires a computer with multi-core CPU, 4GB RAM, and an SSD drive, and at least 200GB free space [26].

Running a full node is the most secure way to interact with the blockchain, making this piece of infrastructure critical to the network. Organizations or individuals run full nodes if they need it for their business. The Miners run full nodes as they get rewards for mining coins in Bitcoin or transaction fees in Ethereum. However, running a full node requires an adequate level of knowledge and resources that most users are understandably unwilling to invest in. As a result, there is no built-in incentive for individual users to run a full node.

Consequently, individual users resort to a third-party centralized infrastructure as an alternative to running full nodes. For example, the most popular software wallets (Metamask, MyEtherWallet, MyCrypto, Jaxx, Exodus, etc.) rely on third-party hosted nodes. These clients connect to a remote node and completely trust their responses. The positive aspect of this is an enhanced user experience as these wallets' users do not need to run their nodes. However, significant security threats may arise when users are forced to trust a third party infrastructure-as-a-service (IaaS) while handling sensitive data such as private keys.

Such Infrastructure-as-a-Service (IaaS) cloud computing supports multiple virtual machines on a hardware platform managed by a virtual machine monitor (VMM) or hypervisor. These co-resident VMs are mutually distrusting, and a malicious VM can pose risks to the confidentiality and integrity of the co-resident VMs. This chapter proposes a method where a co-tenant thread monitors the power management side-channel information from a thread affected by a hardware Trojan. Such a Trojan can leak secret private keys and disrupt digital transactions.

The globalization of semiconductor manufacturing and the phenomenal growth of transistor devices are posing new threats for secure and reliable hardware design. The business model of outsourcing design and fabrication to increase profitability gives an adversary enough scope to tamper the supply chain by inserting hidden, malicious logic, known as Hardware Trojan, into an IC [45]. Hardware Trojan creates a malicious backdoor that can allow an attacker to disable a chip's security, access secret keys, reprogram cryptographic part, access unencrypted configuration bitstream, modify low-level silicon features, or permanently destroy the device [137].

Several emerging trends in hardware integration and user computing practices are creating ample scope of such malicious back-doors. Dynamic Voltage and Frequency Scaling (DVFS) is an intelligent power-management technique that enables frequency scaling-capable processors to change the frequency and voltage of a processor(s) based on system performance requirements [134]. It is a commonly-used technique to save power on a wide range of computing systems, from embedded, laptop, and desktop systems to high-performance server-class systems.

Power Management Unit (PMU) (Power Management Integrated Circuit or PMIC) is a system block for managing DVFS and other power requirements in an SoC device. PMUs offer various services ranging from DVFS, managing power states to dynamic control of power rails. They are the most extensive and fastest-growing part of the analog IC market [4, 22]. They provide flexible power management and increase energy efficiency in high-performance Multiprocessor System-on-Chips (MPSoCs) [195]. The control over the change of voltage and frequency level by PMUs can create a backdoor for malicious attack. Triggering the voltage level change by a hardware Trojan according to the extracted secret key and monitoring the

change level, the secret key can be leaked covertly to an adversary. This chapter proposes PMU-Trojan, a hardware Trojan that exploits the PMU side channel for secret information leakage. This method is easily generalizable to any information leakage. For demonstration purposes, we focus on leaking the Advanced Encryption Standard (AES) key.

The proposed method is particularly useful in data center scenarios where the data center is far away and needs remote maintenance. There are two options for maintaining servers remotely: (i) Integrated management card, (ii) Power Distribution Unit (PDU). While a PDU manages the power supply only, a management interface card allows much more maintenance options for the server. Remote management cards allow administrators to troubleshoot from afar via an interface to the server. Examples of integrated management cards are HPE iLO (Integrated Lights-Out), Dell iDRAC (Integrated Dell Remote Access Card), IBM RSA (Remote Supervisor Adaptor), Lenovo’s ThinkServer EasyManage, etc. These are autonomous management systems built into the server, which offer the remote administrator simplifying server setup, power and thermal optimization, and server health monitoring. As an example scenario, in this work, we demonstrated how a remote administrator could monitor the Trojan infected voltage level change and obtain the secret key using our proposed methodology.

The major contributions of this chapter are:

- Designing a PMU-Trojan to extract and then leak secret key covertly to an adversary.
- Applying the proposed secret key leakage method in a data center application.
- Proposing a Trojan detection method to prevent the secret key leakage.

The chapter is organized as follows. Section 5.2 describes the related works in leaking secret cryptographic keys, mainly the AES key. Section 5.3 briefly presents backgrounds related to our proposed methodology for leaking the AES key. In Section 5.4, we present PMU-Trojan and our proposed methodology to leak AES key using the Trojan. Section 5.5 discusses experimental results and analysis of the proposed method. In section 5.6, the proposed hardware Trojan detection method is presented. Finally, Section 5.7 concludes the chapter.

5.2 Related Works

Several literature works have proposed various methods to leak cryptographic secrets, mainly AES key, from the chip. Physical attacks which target leaking the cryptographic secrets are classified as *observation* and *perturbation* attacks.

Observation or Side-Channel Attack (SCA) consists of observing physical emanations of the system, such as, power [125] or E/H field [172]. Skorobogatov *et al.* proposed a method of extracting the secret key to activate the backdoor and other security keys such as the AES and the Passkey by using Pipeline Emission Analysis (PEA) [192].

Perturbation or Differential Fault Analysis (DFA) consists of injecting faults during the execution of a cryptographic algorithm [49]. Piret *et al.* proposed a technique to break the AES-128 with only two faulty ciphertexts, assuming the fault occurs between the antepenultimate and the penultimate MixColumn [170]. Bhasin *et al.* proposed a hardware Trojan whose principle is to trigger an artificial fault injection to reveal Advanced Encryption Engine (AES) secret key. [44]. Kumaki *et al.* proposed a cipher-destroying and secret-key-emitting hardware Trojan by developing a malicious circuit that connects the encryption and decryption modules in the AES core. [129]

Liu *et al.* presented a silicon implementation of a hardware Trojan, which is capable of leaking the secret key of a wireless cryptographic integrated circuit (IC) consisting of an Advanced Encryption Standard (AES) core and an Ultra-WideBand (UWB) transmitter [138]. Giraud *et al.* presented two fault attacks on the AES: inducing a fault on only one bit of an intermediate result and exploiting faults on bytes. Both require the ability to obtain several faulty ciphertexts originating from the same plaintext [85]. Dusart *et al.* exploited the byte faults occurring after the ShiftRow layer of the 9th round to find AES key [74].

Mayer proposed a side-channeling attack to extract the private key successfully exploiting the SEC (Standards for Efficient Cryptography) Group’s curve *secp256k1*, which is currently used in Bitcoin and Ethereum [151]. Prior information leakage techniques mainly rely on the side-channel analysis or fault injection, subject to noise effects, and the challenge of building an accurate fault model. However, there have been very few works that can leak secret information using hardware Trojan. Our work provides an alternative approach to leaking information using hardware Trojan.

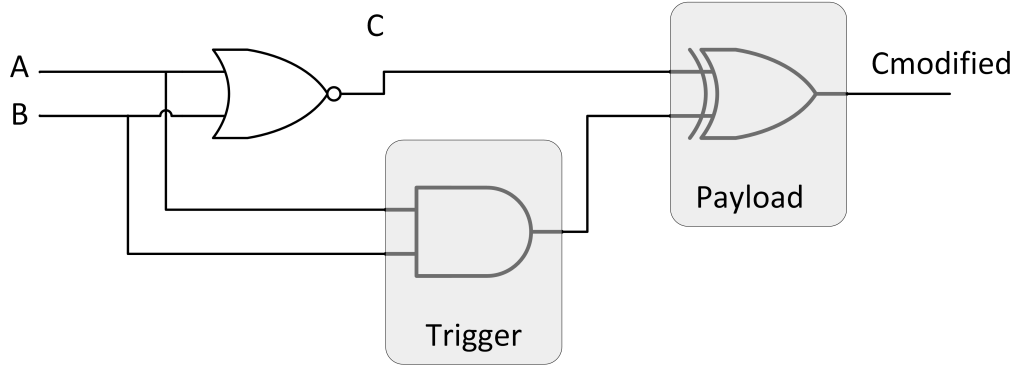


Figure 5.1: Minimalistic hardware Trojan example.

5.3 Background

This section explores some relevant background regarding hardware Trojans, Dynamic Voltage and Frequency Scaling (DVFS), Power Management Unit (PMU), and remote server maintenance.

5.3.1 Hardware Trojans

Hardware Trojans are malicious modifications implanted in an IC, which can be exploited by a knowledgeable adversary to compromise a chip’s security. Trojan circuits, by design, are typically activated under particular conditions (e.g., sensing a specific design signal such as power or temperature, connecting to low-transition probability nets), which makes them unlikely to be activated and detected using random or functional stimuli. A hardware Trojan consists of the following two main components.

- *Trigger*: used for activating a malicious activity.
- *Payload*: used for executing the malicious activity.

Figure 5.1 shows an archetype hardware Trojan. In this minimalistic Trojan, the trigger part is a simple logic-AND gate. When both the inputs A and B are equal to logic-1, the Trojan circuit is activated. The payload part is a logic-XOR gate. When the Trojan is activated, it inverts the intermediate node C. An adversary can design such hardware Trojan to access secret keys, disable or destroy a system at an opportunistic time.

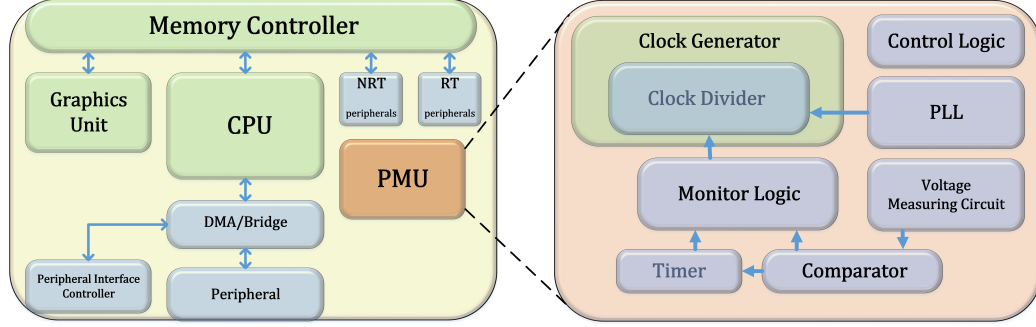


Figure 5.2: Block diagram of MPSoC embedded with PMU [81].

5.3.2 Dynamic Voltage and Frequency Scaling (DVFS)

With higher integration of transistor devices and exponentially increasing demands for processing power and clock-rates, power consumption and thermal performance of integrated circuits have become a limiting factor for modern processor systems. An optimizing power supply solution is required for power-saving benefits and reducing the self-heating of a processor chip. Dynamic Voltage and Frequency Scaling (DVFS) is a method of changing the frequency and operating voltage of a processor(s) based on system performance requirements at a given point in time. DVFS allows operating systems to change operating performance points in real-time without any kernel or user involvement to achieve the lowest power and best performance possible.

In CMOS circuits, most of the dynamic power is consumed in the parasitic capacitance of digital gates. If the dynamic behavior is adjusted to fit the task being executed, a considerable amount of power can be saved. The equation for dynamic power is:

$$P_T = C_{pd} \times V_{cc}^2 \times f_I \times N_{SW}$$

where, P_T is the transient power consumption, C_{pd} is dynamic power dissipation capacitance, V_{cc} is supply voltage, f_I is input frequency, and N_{SW} is number of input bits switching. An intelligent power savings solution reduces operating frequency and simultaneously reduces the supply voltage. Few examples of commercial implementations of DVFS technique are AMD's PowerNow and Intel's SpeedStep.

5.3.3 Power Management Unit (PMU)

DVFS has been realized at both hardware and operating system levels. A Power Management Integrated Unit (PMU) is a system block for managing DVFS and other power requirements in an SoC device [201]. Figure 5.2 illustrates a block diagram of an MPSoC along with an exemplary PMU.

A PMU is configured to transmit a voltage change request to a Voltage Regulator (VR) unit based on the decision made by the DVFS algorithm. For example, Intel processors use a Serial Voltage IDentification (*SVID*) three-wire (clock, data, alert) synchronous interface to transfer the voltage change requests to VR, which in turn decodes the *VID* and supplies the corresponding voltage to the processor through the power rails. Specifically, Intel i7-4650 processor line allows 8-bit *VID* Data signal (00h-FFh) to request for a voltage ranging between $0V$ to $3.04V$, although the specified operating voltage ranges between $1.64V$ to $1.85V$ [14].

An associated monitor circuit is configured to determine whether the requested voltage change has occurred. The monitor circuit detects the voltage change by monitoring the frequency of a VCO (Voltage-Controlled Oscillator) within the monitor circuit coupled to the voltage source. Upon receiving the voltage transition complete signal, the PMU adjusts the processing core's clock frequency within the integrated circuit. PMU supplies each component on an SoC with just enough voltage to deliver the required performance while minimizing power consumption.

5.3.4 Remote Server Maintenance: Integrated Management Card

Today's data centers and server rooms are challenging environments to control and manage. They require 24x7 monitoring of server health and other well-being checks remotely. Since data center security policies are typically complicated, urgent remote server maintenance needs an intelligent solution. Integrated Management Card is a hardware-based management platform that allows administrators to remotely check, configure or reset systems remotely through an interface.

The interface can be simple and Web-based, or a program run on a local computer. Administrators can perform a wide range of operations using the interface menu, such as

monitoring the server’s current state, remote rebooting, performing a power cycle from a distance, or opening a console session to work remotely on the server. Management card provides an array of practical, hardware-level information about a system, including operating frequency, voltage levels, power-supply status, fan speeds, temperatures, etc. Examples of integrated management cards are HPE iLO, Dell iDRAC, IBM RSA, Lenovo ThinkServer EasyManage, etc.

5.4 Proposed Methodology

In this section, we first describe a threat model and then present our proposed methodology for leaking AES key by inserting Trojan and exploiting PMU behavior [105].

5.4.1 Threat Model

Hardware Trojan can be inserted in a chip at various stages from the Register Transfer Level (RTL) code to mask fabrication. An attacker can introduce Trojan by modifying the netlist design or lithographic masks. For an illustrative example, in our work to leak AES key, such modifications can be done by a few key hardware engineers who can insert a malicious circuit during the design phase without affecting the design’s main functionality [214, 219]. At any stage, from designing RTL code to lithographic mask, the soft product design provider can include hidden, malicious functionality that can be turned on at an opportunistic time. The adversary can then leverage such malicious modifications for leaking confidential information, such as a secret cryptographic key.

5.4.2 Trojan Insertion

Figure 5.3 shows an MPSoC block diagram implanted with PMU-Trojan to extract the AES key. The *trigger* part of the Trojan is a checker circuit that checks for a specific sequence of three 8-bit *VID* data signals coming out of PMU. When the specific sequence is matched, it activates the Trojan *payload*. The *payload* circuit consists of a PMU-Trojan key-extractor and a Multiplexer.

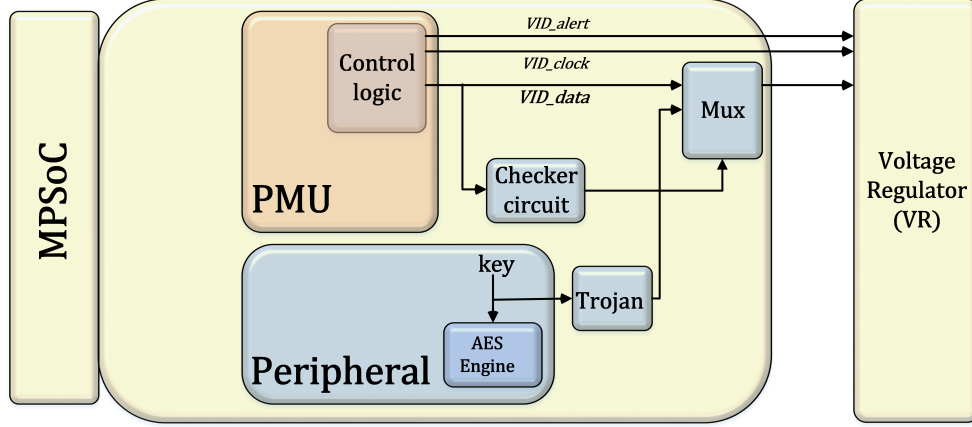


Figure 5.3: MPSoC infected with hardware Trojan.

5.4.3 Trojan Activation

We propose a software-hardware coalition-based trigger that takes advantage of the core processor’s voltage change requests. Similar to the *sequence cheat code* idea presented by Waksman *et al.* [214], we propose a sequence of power event requests for triggering the underlying Trojan. Several OSes deploy some form of DVFS at the operating system level. For example, Linux implements *CPUfreq*, a standard kernel framework to switch between various frequencies and operating voltages. *CPUfreq* monitors the system performance requirements of a processor(s) and takes decisions to increase or decrease operating frequency in order to serve the user’s needs and save power [6].

The rules for adjusting frequencies, whether to a faster or slower clock speed and when to adjust frequencies, are defined by the *CPUfreq* governor. The governor defines the power characteristics of CPU, which in turn affects CPU system performance. There are several implementations of *CPUfreq* governor: (i) *performance*, and (ii) *powersave*, (iii) *ondemand*, (iv) *conservative*, (v) *userspace*, etc.

The *userspace* governor is a customizable governor that allows the user program (or any process running as root) to decide the processor’s specific speed. For triggering PMU-Trojan, a program is written to change the frequency in a particular sequence. When the program runs, the frequency of the processor changes accordingly.

The OS looks-up in the Operating Performance Point (OPP) Library to find the corresponding voltage levels. OPP is the set of discrete tuples consisting of frequency and voltage

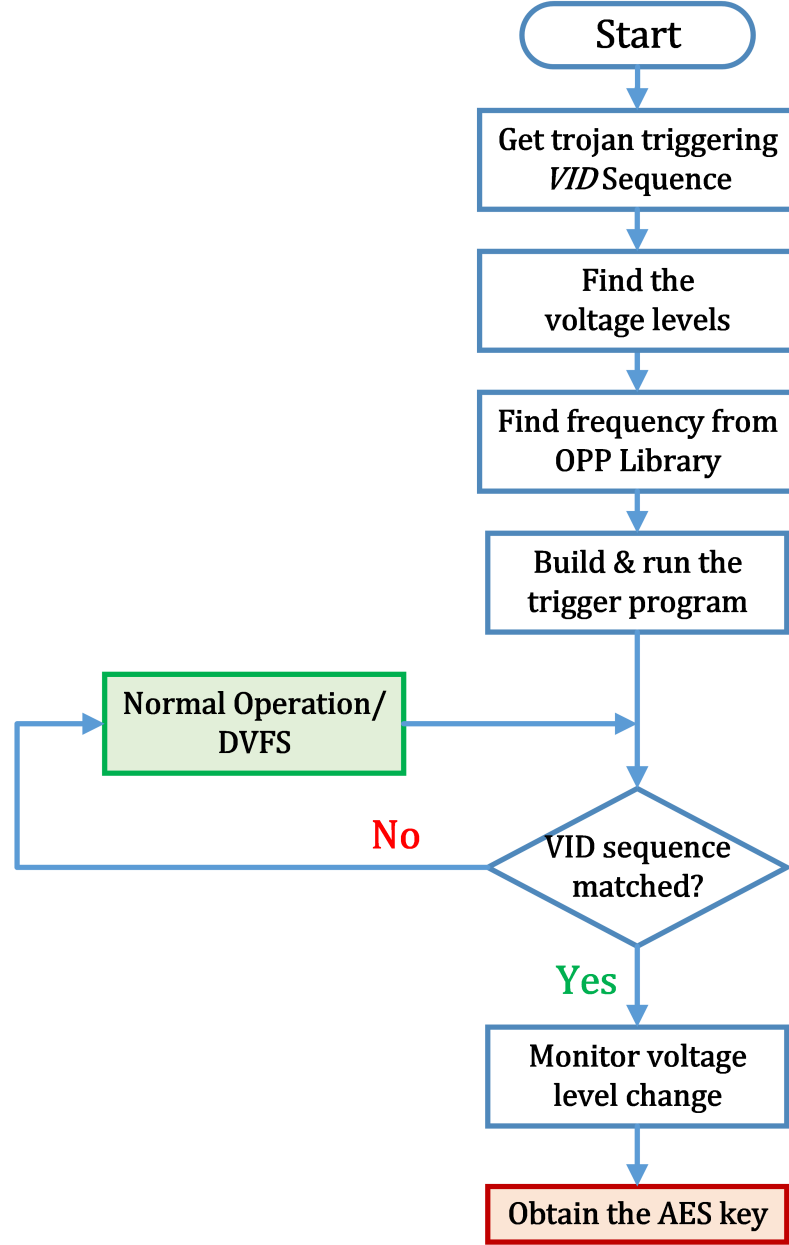


Figure 5.4: Flow chart illustrating the proposed methodology.

pairs that the device supports. OPP library provides a set of helper functions to organize and query the OPP information [210]. The PMU sends the voltage change requests in the form of *VID* signals to the Voltage Regulator (VR). The checker circuit monitors the incoming requests for the complex sequence. When the sequence matches, it selects the AES key extracted by the Trojan. Thus the PMU changes the voltage levels according to the AES key instead of changing according to OPP.

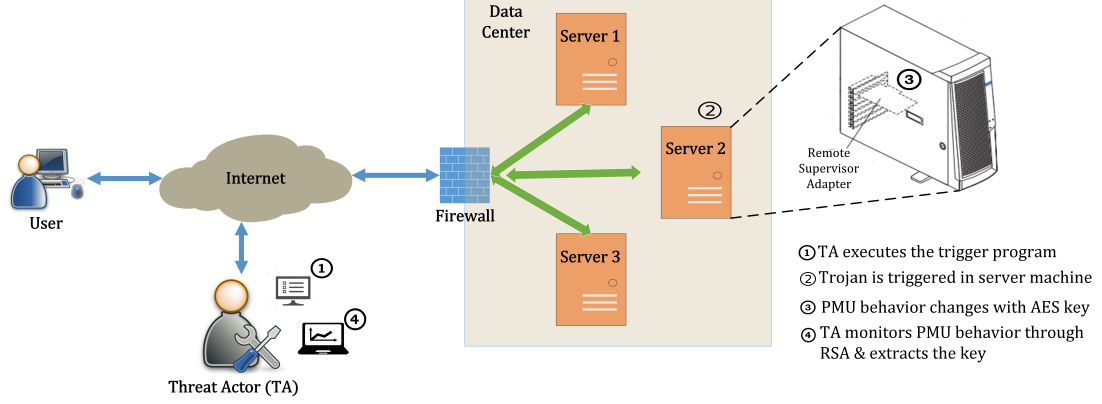


Figure 5.5: An example attack scenario at data center.

5.4.4 Trojan Operation

Once the Trojan is triggered, the processor core voltage changes according to the extracted AES key. An adversary monitors the voltage level change and obtains the key covertly. The Serial VID corresponding to a voltage level is processor-specific and can be obtained from the processor manufacturer’s datasheet. There are several hardware monitoring tools, such as, *Intel Performance Monitoring Unit (PMU)*, *Intel Performance Counter Monitor (PCM)*, *Intel Extreme Tuning Utility* [15], *HWMonitor*, *CPU-Z* [5] for Windows operating system and *i7z*, *c2ctl* for Linux operating system [3]. The methodology is illustrated as a flowchart in Figure 5.4.

In the data center scenario, an adversary can monitor the PMU-Trojan induced voltage level change using the Integrated Management Card. Figure 5.5 illustrates such a case, where a *Threat Actor* can sneak information using IBM RSA management card. A Threat Actor could be anyone with a malicious agenda, such as a rogue Administrator, outside agent, third party vendor, that has gained access to the server network. Apart from administrators, malicious attacks can also be initiated by compromised third party Business Partner if a credentialed account in the Business Partner network is allowed to access servers in the core data center [23].

Table 5.1: Frequency and voltage level change

Frequency (GHz)	<i>FID</i>	Voltage (V)	<i>VID</i>
3.1	8	1.2	32
2.7	7	1.150	28
2.3	6	1.125	26

5.5 Experimental Results

We implemented the Trojan blocks in Verilog RTL and synthesized the RTL with Nan-gate 45nm Open Cell Library [101] using Synopsys Design Compiler to find the Trojan design overheads. The Trojan incurs an area overhead of $192 \mu m^2$. The power overhead is negligible, as PMU-Trojan has a low activity factor. For coarse-grain DVFS with an off-chip voltage regulator, voltage level stabilization takes around $50 \mu s$ [196]. For fine-grain DVFS with a fully integrated on-die voltage regulator, such as Intel Haswell processor, a voltage change can be done once in 500ns [53]. Compared to core frequency, the rate at which voltage can be changed is 2-3 orders of magnitude slower. This results in a low activity factor for the PMU-Trojan, hence low power dissipation overhead.

For testing the frequency and voltage level change in terms of *FID* (Frequency Identification Number) and *VID*, respectively, we ran a program in a Q9450 Core 2 Quad Processor workstation and monitored the *FID* and *VID* signals change using *c2ctl* tool. *c2ctl* is a frequency and voltage monitoring performance utility tool. Appendix B shows a sample code snippet for triggering PMU-Trojan. The imposed frequency change at a specific sequence and the observed voltage level change are shown in Table 5.1. This experiment is just an example of a change of *VID* signals according to a specific frequency level change. A Trojan infected chip will change the *VID* signals according to the AES key similarly.

5.6 Trojan Detection

This section describes the complexity of PMU-Trojan activation during the pre-deployment test and the limitations of traditional Trojan detection methods. We then propose a low complexity technique to detect anomalous behavior inflicted by PMU-Trojan.

Our proposed method increases the magnitude of complexity involved in Trojan detection by a statistical approach during testing [57]. *VID* signals are 6 to 8 bits in modern processors. A sequence of voltage change requests in the form of *VID* signals can thus potentially have enormous state space resulting in enormous test times. Therefore, activating the Trojan during post-silicon tests has an extremely low probability. Our proposed scheme also defeats traditional Differential Power Analysis (DPA) based Trojan detection due to a low information bit rate spread over a long time by a PMU-Trojan. Hardware Trojan detection technique, like information flow tracking [80] tags assets such as cryptographic key bits in the design, then use formal methods such as model checking [174] and theorem proving [119] to analyze how these bits propagate through the design. However, we leak the secret information via a side-channel by monitoring the voltage level change in our proposed work. This information flow via side-channel does not depend on the conditions under which information can safely flow between specific signals in a design.

For detecting PMU-Trojan, a concurrent hardware monitoring thread is written, which continuously checks the voltage level change according to the AES key. If any such event occurs, the monitoring thread will alarm and shut down the system immediately.

5.7 Concluding Remarks

In this chapter, we outline a novel security threat stemming from a hardware Trojan that can intelligently impact the behavior of PMU. We designed PMU-Trojan that can leak confidential information by exploiting the PMU behavior. The proposed Trojan incurs a very marginal area overhead and can be triggered by executing a simple program. The threat is ominous, particularly in the data-center application where a malicious adversary can leak a user’s secret information, such as blockchain private key, with the help of our proposed PMU-Trojan.

CHAPTER 6

REMOTE CONFIGURATION OF INTEGRATED CIRCUIT FEATURES VIA SMART CONTRACTS

6.1 Introduction

CMOS technology has been a driver of growth in the semiconductor industry for over three decades with rapid scaling of transistor feature size. With scaling, the cost of developing a new integrated circuit (IC), its manufacture, debugging and volume production has increased. The cost of developing a new SoC, its debug and design iterations is escalating with the transition to $7nm$ technology and beyond. The semiconductor industry's economic model is typically based on economies of scale, and this model works well for high-volume products, such as smartphones. However, the volume for any single chip is insufficient for smaller, more fragmented markets to justify the high level of investment needed — particularly for advanced process nodes such as FinFET, for example [78].

Consequently, without the ability to customize IC features after production, an IC's lowest-priced application determines its price. This challenge motivates the manufacturers to develop capabilities for post-production IC customization. Today, such customization is limited to one-time programmable (OTP) for predetermined IC bins [38, 68]. This chapter explores how an IC can be programmed repeatedly and securely using blockchain-based smart contracts. This programmability will enable users to upgrade IC features or rent upgraded IC features for a fixed period after users have purchased the IC.

The manufacturer can implement the IC feature configuration based on a centralized architecture [73]. Information is sent from the device to the cloud, where the data is processed using analytics and then sent back to the device. Figure 6.1 shows an example of this centralized system-based remote feature configuration, a smart device automatically paying for its feature configuration upgrade. Such a system encounters four technical challenges that human-centric applications solve with a person in the loop.

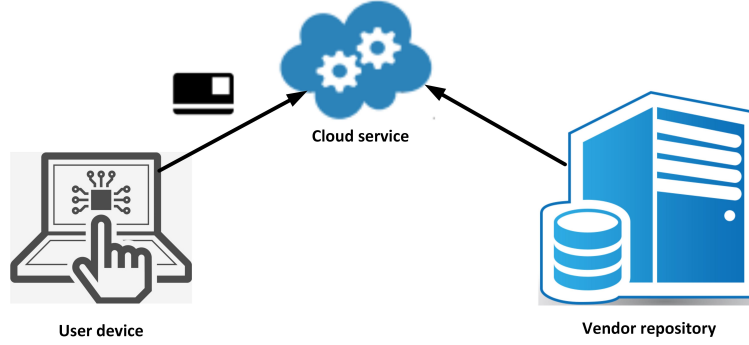


Figure 6.1: An example application that stores a user’s credit card information is installed in the user’s device and vendor repository. Before upgrading the chip configuration, the device and the vendor repository identify each other. Then, the credit card stored by the cloud service is charged after the user upgrades the device configuration.

The first challenge is transparency. In Figure 6.1, the smart device automatically performs actions on a user’s behalf. As the device sends encrypted data to the cloud service, it would be impossible for the user to audit the encrypted information to ensure that the device did not send any private data — a concern that is well justified [34, 161].

The second challenge is longevity. Some of these devices may still be in use long after their vendors stop maintaining them [92]. However, these devices are often vertical silos, where a centralized entity manages application state and communication protocols. As a result these devices cannot function without the cloud services of their vendors [82, 95]. In Figure 6.1, the device cannot pay to the vendor without the cloud service.

The third challenge is trust. In Figure 6.1, the user has to trust the vendor with their credit card information, and the credit card company acts as a trusted third party to help manage funds and resolve disputes in exchange for non-negligible fees. Transactions that involve the exchange of digital or physical assets require trust, which inherently involves risks. Examples of such risks are the vendor leaking credit card information or the credit card company undermining exchange’s fairness by colluding while resolving disputes [86].

The fourth challenge is a single point of failure. Figure 6.1 presents a client-server model for configuration update distribution from the vendor’s repository to the user device. In this model, excessive network traffic may occur when devices request the update files simultaneously. In the IoT environment where tens of millions of devices are possibly required to

be updated simultaneously, this type of centralized client-server model is inappropriate, has very limited scalability, and exposes billions of weak points that may compromise network security.

Each of these challenges can compromise the centralized system based remote device feature configuration. Using blockchain technology is one approach to addressing these challenges. A blockchain guarantees complete transparency of transactions carried out on the chain. Besides, it allows the state to be stored among the nodes in a decentralized network that persists as long as the network of nodes exists. Moreover, independent and autonomous execution of the smart contract logic by each node on the network using the blockchain's data reduces the need for trust and third party involvement in a transaction [108].

In this work, we propose blockchain-based smart contracts for enabling chip features on-field. The smart contract takes the user's feature configuration request as input and returns an encrypted configuration. To support the manufacturer's authorized feature configuration, we propose an on-die hardware module that enforces the hardware configuration's secure execution. The major contributions of this work are:

- Proposing a protocol for remote, secure, and repeated configuration of IC features via smart contracts.
- Proposing smart contracts which define the protocol and stores the state and logic necessary for remote chip feature upgrades.
- Proposing an on-die hardware module that communicates with the smart contract and enforces its functionalities.
- Demonstrating the proposed solution in both software and hardware implementation. The implementation is publicly available [20].

The chapter is organized as follows: Section 6.2 presents the related works on blockchain-based firmware upgrade. Section 6.3 presents the motivation behind blockchain-based remote IC configuration. Section 6.4 presents the system design and implemented smart contracts for our proposed protocol. In Section 6.5, we present our proposed hardware design and

the methodology to chip feature upgrade using blockchain. Section 6.8 discusses the security analysis of the proposed protocol. In Section 6.6, we outline the implementation and demonstration of the protocol. Section 6.7 presents the results, evaluation and Section 6.9 discusses the limitations of this approach. Finally, Section 6.10 concludes the chapter.

6.2 Related Works

Although several blockchain-based firmware update protocols have been proposed in the literature [39, 51, 135], there have been very few works on upgrading the hardware configuration via blockchain. A peer-to-peer framework was proposed by Boudguiga *et al.* to disseminate firmware updates between IoT devices that have restricted Internet access [51]. They investigated how the use of a blockchain framework would meet the requirements of the CIA triad properties, i.e., confidentiality, integrity and availability. Lee *et al.* proposed a blockchain-based firmware update scheme, where an embedded device requests its firmware update to nodes in a blockchain network and gets a response to determine whether its firmware is up-to-date or not [135]. If not latest, the embedded device downloads the latest firmware from a peer-to-peer firmware sharing network of the nodes. Baza *et al.* proposed a firmware update scheme based on blockchain and smart contract for autonomous vehicles [39]. The smart contract ensures the authenticity and integrity of firmware updates, and manages the reputation values of Autonomous Vehicles (AVs) that transfer the new updates to other AVs.

6.3 Motivation

In this section, we present our motivation for post-production IC customization and the smart contract based solution for such customization.

6.3.1 Motivation for Post-production IC Customization

Without the ability to customize chip configuration after production, the price of an IC is determined by its lowest priced application. Chip manufacturers sell the highest end chips as lesser chip parts based on consumer demand. This motivates the manufacturers to develop capabilities for post-production IC customization.

Realizing Economies of Scale Several semiconductor industries apply chip configuration after manufacturing. The commodity microprocessor business offers an example of post-production customization, where a manufacturer can tailor the number of cores, cache size, and frequency of operation for a target market segment after manufacturing a chip [194]. The Intel Skylake micro-architecture, for instance, features a highly configurable design. Intel can meet the various market segment requirements using the same macro cells [147]. The Skylake family consists of five separate actual dies which can be further segmented by disabling various features. For example, GT1 graphics are based on GT2 graphics that have disabled half the execution units [21]. However, such customization techniques are limited to one-time programmable (OTP) for predetermined IC bins presently.

Customizing Feature Upgrades The availability of a post-production customization technique could, for example, allow a buyer to upgrade his processor from *i5* to *i7* after purchase to scale to her computing needs in exchange for a payment made to the manufacturer. During manufacture or in the field, chip features may be enabled or disabled. While chip cost is fixed, the manufacturer can set the chip selling price according to the features enabled and markets being addressed.

6.3.2 Motivation for Smart Contract-based Solution

This section presents how smart contracts address the technical challenges of transparency, longevity, trust, and single point of failure that can be frequently encountered in a centralized system based remote feature configurations like Figure 6.1.

Transparency with Public Logs In Figure 6.1, the vendor may obtain personal information about a user (such as computation pattern, working behavior, or resource usage) that could be sold to advertisers for better advertisement targeting. Users would have no way to directly verify what exact information their device sent if communication to the cloud service was encrypted. The use of smart contracts provides a public, auditable log of communication.

Longevity through Decentralization A user’s device lifetime may outlast the vendor in the example of Figure 6.1. In smart contracts, an application’s core state and logic are fully distributed, which allows an application to continue to operate or be picked up by

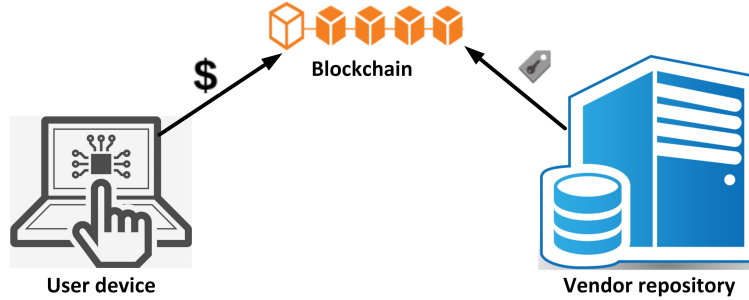


Figure 6.2: User’s device uses smart contracts in Ethereum to rent upgraded feature configurations.

a new vendor long after the original vendor has shut down. As smart contracts are a public interface, anyone can directly interact with the smart contract using their applications with the assurance that the application’s state will be available on the blockchain, rather than being lost with the shutdown of a vendor.

Minimized Trust using Smart Contracts The need to trust vendors or third parties with personal information like credit cards or bank accounts creates worthy targets for attack, many of which have been notoriously exploited [165]. Instead, users can avoid this risk by directly interacting with deterministic business logic defined in smart contracts. Transactors can use smart contracts as reliable escrows for digital assets, creating a platform that supports various applications.

Removing Single Point of Weakness A blockchain can provide an elegant solution to remove the single point of weakness in a centralized scenario. For example, in a blockchain-based solution, the first requests for a configuration update will be served by the manufacturer’s node. The manufacturer’s node also takes part in the network initially, but after the configuration codes have propagated to enough nodes, it can stop serving. As a result, a device that joins the network long after the manufacturer has stopped serving can still retrieve the requested configuration update and be assured that it is the right file. Contrary to this solution, in the centralized scenario in Figure 6.1, the device may poll the manufacturer’s server for an update and get a 404 error [29].

6.4 Proposed Protocol and Smart Contract Implementation

This section outlines the system design and smart contract implementation for our proposed IC feature configuration protocol.

6.4.1 System Design

In our proposed protocol, users can upgrade IC features or rent upgraded IC features for a fixed period after purchasing an IC. We propose implementing this repeated, remote, and secure programming of the IC using a smart contract, as shown in Figure 6.3. The owner of a device requests for remote configuration by sending the list of upgrade-able functional components or IPs, and intended period of usage to the smart contract. The smart contract first verifies the authenticity of the device ownership. According to the requested configurable features, the smart contract sets the configuration state. The user device periodically polls the state of the smart contract and receives the latest configuration code.

Our proposed system design has two main components: *(i)* the smart contract that defines the protocol for renting IC feature configurations and *(ii)* the on-die hardware module that communicates with the smart contract and enforces its functionalities. In this section, we provide the details of our smart contract implementation. In Section 6.5, we present the details of the proposed hardware design.

6.4.2 Implementation of Smart Contracts

In our proposed protocol, a smart contract defines the condition for registering a new device by a manufacturer, upgrading device configuration by a user and querying the current configuration by the device. The smart contract is created by the manufacturer to maintain uniform applicability and usability for all of the owners of all devices. In the following, we describe all the steps in our protocol and how our proposed smart contract implements these services.

6.4.2.1 Register Device

Any new device produced by the manufacturer must be registered first. This requirement is necessary because each device's configuration code will be encrypted with device

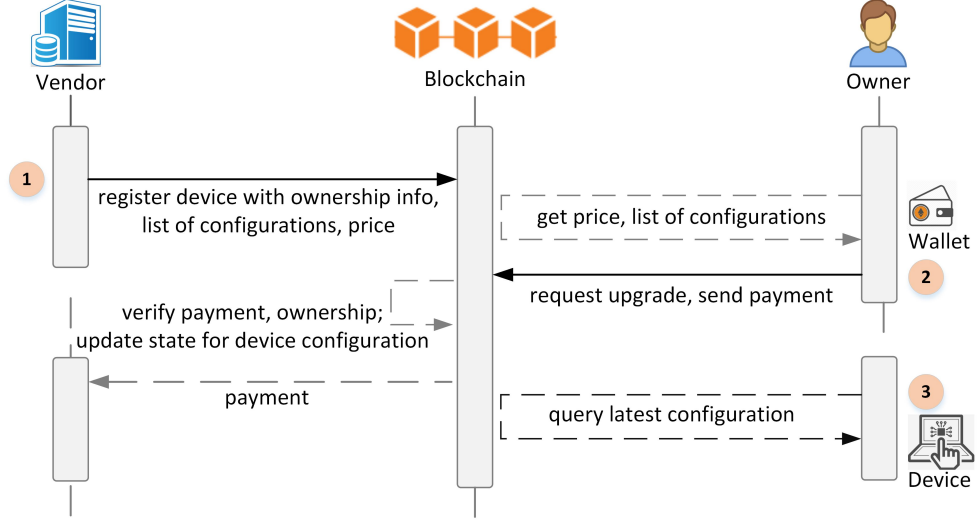


Figure 6.3: Proposed protocol for remote configuration of IC features using smart contracts. Transactions requiring payment of fees are drawn with solid black lines.

public key. During usage, the user can request a specific configuration for a target device. The manufacturer performs the registration by issuing a transaction `registerDevice()` to the smart contract. Algorithm 6 presents the pseudo-code of function `registerDevice()`. This function enrolls a device with all necessary information, if the message sender is the manufacturer. In our proposed protocol, each device is associated with the following three pieces of information:

Device identifier This is a device identification data used to look up the targeted device being queried among many devices. It can be a serial number of the device (e.g., Electronic Product code, or EPC).

Device private key Each device has a unique private key and the corresponding public key. The manufacturer uses this public key to encrypt all possible configurations during the registration phase. The encryption is necessary to make sure that only the target device can decrypt the configuration, and any other device of similar type cannot apply the configuration. The device can later decrypt the encrypted configuration using its private key and apply the configuration. The private-public key pair may be keys for RSA, DSA, Elliptic Curve based public key cryptosystems, etc.

ALGORITHM 6: Pseudo-code of `registerDevice()` for registering a device claimed by a manufacturer.

Inputs: Device information (*deviceInfo*), a list of configurations encrypted with device public key ($K_d^+(config1)$, $K_d^+(config2)$ etc.), pricing, and owner's address (*addrOwner*)

if *Message sender is the Manufacturer* **then**

- Set owner of *deviceInfo* as *addrOwner*
- Set current configuration as $K_d^+(configDefault)$
- Set upgradable configurations as $K_d^+(config1)$, $K_d^+(config2)$

else

- Do nothing

end

All possible device configurations The manufacturer determines all possible configurations in a device and encrypts those configuration codes using the device public key. For example, if the configuration code is `config1`, the manufacturer uses the device public key, K_d^+ to encrypt the code and produce encrypted configuration, $K_d^+(config1)$. Similarly, the manufacturer encrypts and registers all the configurations, such as, $K_d^+(config1)$, $K_d^+(config2)$ etc. These configurations can be the same for all devices of the same type.

6.4.2.2 Upgrade Configuration

The user issues a transaction `upgradeConfiguration()` to the smart contract to upgrade the configuration of a device. This smart contract function defines the feature upgrade policy that enforces its execution without an intermediary, so only authorized devices can request and receive the update. The policy includes the manufacturer's pre-defined Service Level Agreement (SLA) with the customer. The required cost for getting a particular feature upgrade can be set using various policies out of this work's scope.

The smart contract function `upgradeConfiguration()` first verifies if the request sender for the feature upgrade is the device's owner. If this is true, the function then calculates the required transaction fee for that requested feature upgrade. If the requester makes enough payment with the transaction, the function sets the device configuration's current state as the requested configuration. The function also initiates the transfer of the upgrade fee from the contract's address to the manufacturer's wallet address.

ALGORITHM 7: Pseudo-code of `upgradeConfiguration()` for upgrading the configuration of the device.

Input: Device identifier (*deviceIdentifier*), requested configuration (*config*), and *usage period*

```

if (Message sender is the owner of deviceIdentifier) then
    Calculate required price for feature upgrade
    if Message value == required price then
        Set current configuration state of deviceIdentifier as  $K_d^+(config)$ 
        Set current configuration period of deviceIdentifier as usage period
        Transfer the transaction fee from the owners's wallet to manufacturer's wallet
    else
        Do nothing
    end
else
    Do nothing
end

```

6.4.2.3 Query Configuration

The device interacts with the blockchain network and polls the smart contract periodically by querying the function `queryConfiguration()`. This function returns the requested configuration and the upgrade period to the requesting device. The devices either ship with the smart contract's address baked into their blockchain client, or they find it via a discovery service [59]. The device interaction with blockchain and the smart contract can happen in the following two ways.

The device itself runs a blockchain node The device needs a piece of software, known as a *client*, to run a blockchain *full node* or a *light node* (retrieving data live). A *client* is an implementation of a blockchain that verifies all transactions in each block, keeping the network secure and the data accurate. In this case, the device directly receives the query output from the smart contract in blockchain and then processes it. However, resource-constrained devices do not possess enough horsepower to run a software *client* for intensive blockchain calculations.

An edge gateway device runs a blockchain node We propose delegating the task of running a blockchain node to gateways or any other unconstrained device capable of providing this functionality. In this case, the gateway runs the blockchain client. It is the entry point into the blockchain network (main-, test- or private net), capable of running as a full node or a light node. It can be used by the user device as a gateway into the Ethereum network via JSON RPC endpoints exposed on top of HTTP, WebSocket and/or

IPC transports [11]. We propose the gateway solution to be more efficient as the device does not require running a computation-intensive blockchain node. Besides, the gateway can provide the blockchain service to all the devices in the same network. In the next section, we describe how our proposed hardware module processes the configuration once the device receives it.

6.5 Proposed Hardware-software Co-design for Remote Configuration

In our proposed protocol, the device polls the smart contract periodically, receives the requested configuration from the blockchain, and then executes it. We partitioned this execution framework into software and hardware design. In the following, we describe the operations performed in these parts.

6.5.1 Software

The software code is executed on the device processor and handles control tasks such as interaction with the blockchain and communication with the hardware part. This code is flashed on the processor’s write-protected internal ROM during manufacturing and gets executed during or after booting.

Interacting with Blockchain Node The device software interacts with a blockchain node running on a gateway by sending a request through JSON RPC either over HTTP or WebSocket; as a single request. The requests follow this format in Ethereum:

```
{“method” : “eth_call”, “id” : 1, “jsonrpc” : “2.0”, “params” : [{“to” : “..”, “data” : “..”}]}
```

The `method` essentially defines what function an Ethereum node should execute. It either requests data from the node, executes an EVM (Ethereum Virtual Machine) function, returns a response, or transmits data to the Ethereum network (send a transaction). In our design, we use the `eth_call` method to retrieve already mined data from the blockchain about a particular smart contract. The `id` field is an identifier string set by the device and helps make sure the device and the gateway are communicating in the same context. The `jsonrpc`

field specifies the protocol version and must be exactly 2.0, which the protocol expects. The `to` field is the smart contract address.

To query the device configuration using *eth_call*, we use the `queryConfiguration()` function of the contract. The JSON-RPC format expects *eth_call* to have a specific `data` field format. The `data` is the first four bytes of the Keccak-256 hash of the function's signature. The signature is defined as the function name with the parenthesized list of parameter types. The response returned for this request is a hexadecimal number concatenated from two values (encrypted configuration code and the upgrade period), each padded to a 32-byte length with zeroes.

Decrypting the Configuration Code After receiving an appropriate configuration code and the upgrade period for that configuration, the device software first decrypts the encrypted configuration code using the private key. We assume that the private key is hard-coded in the software design for the implementation purpose. After that, the functional component configuration process starts in our proposed on-die hardware module. As processing the configuration is the core part of our remote device management system design, we propose the implementation of this processing in hardware for enhanced security.

6.5.2 Hardware

The custom hardware part aims to secure and accelerate the configuration process. For processing the configuration, we propose a Configuration Controller Module (CCM), as shown in Figure 6.4. Our proposed hardware design consists of the following modules.

6.5.2.1 Hardware configuration module (HCM)

This module acts as the main controller module for the chip. The hardware configuration module (HCM) receives the configuration code from the blockchain module and directs changes to functional component characteristic settings. This module enables conditional activation or deactivation of on-chip functions provided by the on-chip modules in response to the reception of the external reconfiguration request. The chip subsequently utilizes the activated chip features during operation. The HCM forwards the configuration code and the

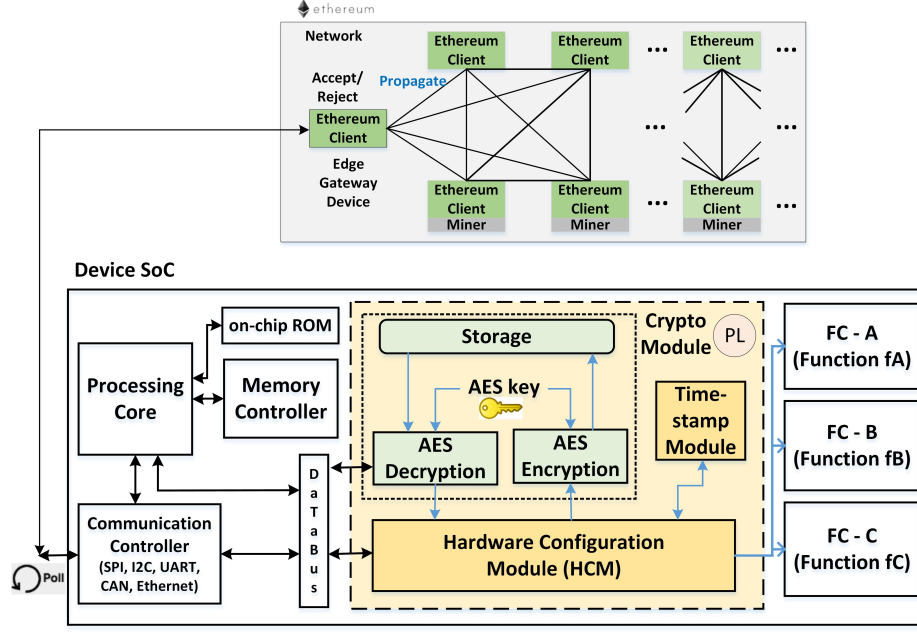


Figure 6.4: Proposed hardware design for remote configuration of IC features (FC: Functional Component).

upgrade period to the encryption module in the crypto engine and the timestamp module, respectively.

The HCM receives the encrypted configuration from the encryption module and stores this updated configuration in an on-chip or off-chip programmable non-volatile storage (EEPROM/flash). The module also sets a configuration flag while storing the new configuration. The software code checks this flag bit during or after the next boot up and configures the hardware accordingly. If the flag bit is on (*logic-1*), the software loads the HCM with the updated configuration (after decrypting it with the decryption module). Otherwise, the software loads the HCM with the default configuration the device is provisioned with during manufacturing.

6.5.2.2 Timestamp module

The timestamp module receives the upgraded reconfiguration period from the HCM and sets the timestamp counter with that value. After the upgrade period, the timestamp module notifies the HCM. The HCM changes the device configuration settings to the default after the upgrade period. The timestamp module also decrements the timestamp counter

value with the usage period to keep track of the remaining upgrade period. The module periodically sends the remaining upgrade period to the HCM. The HCM stores this value in the storage along with the encrypted configuration. The software loads the HCM with the upgraded configuration and the remaining upgrade period during or after the next boot up.

6.5.2.3 Crypto module

The crypto module performs symmetric encryption of the configuration file to prevent any physical attacks from retrieving the configuration. Encrypting the configuration requires the device to have a unique secret key. This module also manages the keys for symmetric key cryptographic operations. We assume that the symmetric key is hard-coded in the design for the implementation purpose. The symmetric key can be generated from a PUF-based system for additional security, as described earlier in Section 2.5.3.2. We propose the use of AES for symmetric encryption.

AES Encryption Module The symmetric key (AES key) encrypts the configuration and stores the encrypted configuration in a persistent storage. For the symmetric encryption algorithm, we choose AES-128 in *counter* (CTR) mode. AES-CTR mode’s advantage is that it uses the only AES encrypt operation (for both encryption and decryption), making the implementation smaller than many other AES modes. This design makes implementations simpler and yields a significant throughput increase in hardware.

AES Decryption Module During device boot-up, the decryption module generates the configuration using the AES key. In AES-CTR mode, this is the same module as the encryption module. To decrypt something, the input states need to be successive values of the same “counter” used for encryption. Then the binary sequence output is XOR-ed to the ciphertext to get the plain text.

6.6 Implementation and Protocol Demonstration

In this section, we present the experimental setup, implementation of smart contract for prototyping our proposed protocol, and evaluation of the experimental results.

Table 6.1: Estimates of transaction fees for various operations.

Transaction	Paying Party	Gas Limit	Gas Price (ETH)	Cost (ETH)	Cost (USD)
Contract Creation	Manufacturer	367191	33×10^{-9}	0.012127	4.25
<code>registerDevice()</code>	Manufacturer	103642	40×10^{-9}	0.004145	1.45
<code>upgradeConfiguration()</code>	User	30184	30×10^{-9}	0.0009055	0.32

6.6.1 Implementation of the Smart Contract

We implemented the proposed smart contract in **Solidity** programming language and tested the functionalities in both Ropsten¹ and Rinkeby² Ethereum Test Networks. After that, we deployed the smart contract in Main Ethereum Network³ and evaluated the protocol in terms of its operational cost. We made the source code publicly accessible in Github [20]. In particular, we estimated the cost of each operation by measuring the gas amount (execution fee for the transaction made on Ethereum) for all of the functions involved in the process, as shown in Table 6.1.

6.6.2 Implementation of Feature Configuration by Hardware

6.6.2.1 Target device and the Hardware Architecture

We implemented the feature configuration using our proposed hardware design. For our experiments, we used the Xilinx Zynq-7000 System-on-Chip platform, which integrates a dual-core ARM Cortex A9 processing system along with Xilinx Virtex-7 family programmable logic. The target device used is Zedboard Zynq Evaluation and development kit (xc7z020clg484-1). The overall system contains the following components: Zynq Processing System (PS), Universal Asynchronous Receiver Transmitter (UART), Timer and our customized IP. The customized IP communicates with the processor via the memory-mapped-register. Reception and the transmission of data happens through the UART.

¹<https://ropsten.etherscan.io/address/0xe790fa82964853ba42695de768a4c4436f6f3022>

²<https://rinkeby.etherscan.io/address/0x4cd8304aa0e8b304cac6ebca95b92ddcf6437638>

³<https://etherscan.io/address/0x990bea57863d1a4de73daf21053c040c9abc8978>

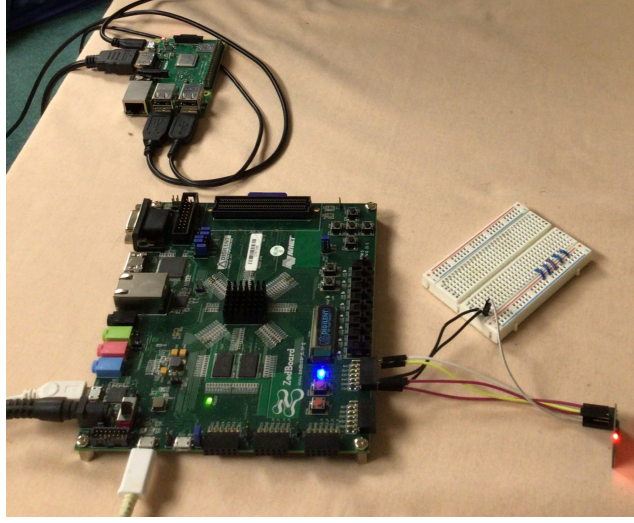


Figure 6.5: Experimental setup for demonstrating feature configuration using proposed hardware design. The Raspberry Pi works as the gateway.

6.6.2.2 Gateway

To interact with the blockchain and receive the smart contract state updates, the device needs to connect to a blockchain client. We use a Raspberry Pi 3 Model B+ that acts as an edge *gateway* device for running the blockchain client. The Raspberry Pi is equipped with a 1.4GHz 64-bit Quad Core ARMv8 processor. Figure 6.5 shows the hardware setup for our experiment. We run the most prominent blockchain client implementation, *go-ethereum* or *geth* [11] on the Raspberry Pi. The Zedboard’s Processing System (PS) uses *geth* as a gateway into the Ethereum network via JSON RPC endpoints exposed on top of HTTP. To implement such a two-way end device-Ethereum proxy, gateway connects to WiFi to communicate with both end-device and an Ethereum client to route data to the blockchain network.

6.6.2.3 Protocol Demonstration

We show the proof-of-concept by enabling and disabling different hardware blocks with the upgraded and the default configuration. In one exemplary implementation, we turn on the board LEDs according to the configuration upgrade requested by the user. The Zedboard communicates with the gateway via the Tx and Rx pins for Universal Asynchronous Receiver/Transmitter (UART). An ESP8266-01 device connected to the Zedboard PMOD

works as the WiFi station and is controlled by AT Commands through UART communication [9]. From the Zedboard, we initiate a TCP connection with the gateway and periodically send an HTTP POST request to the gateway. This POST request contains the query to the smart contract. Figure 6.6 shows the HTTP POST Content sent from the Zedboard via ESP8266-01. The returned output from the query to the smart contract is then processed by the hardware design in Zedboard FPGA.

```

1  {
2      "jsonrpc": "2.0",
3      "method": "eth_call",
4      "params": [
5          {
6              "to": "0x4Cd8304Aa0e8B304CAc6EbCA95B92DdCF6437638",
7              "data": "0x9de4d683"
8          },
9          "latest"],
10     "id": 1
11 }

```

Figure 6.6: HTTP POST Content sent from the Zedboard via ESP8266-01.

6.7 Results and Evaluation

We generated all results using Vivado Design Suite 2017.4. We functionally verified the design using the Vivado simulator and performed the profiling and software implementation in Xilinx SDK 2017.4. We described the design in Verilog at the Register Transfer Level (RTL) for all the modules. We packaged all these Verilog source files for all the modules into an IP. The Zynq Processing System (PS) accesses this IP as a memory-mapped IO (Input/Output). For the timestamp module, we implemented a clock divider and counters to keep track of the usage period and the remaining upgraded configuration period. To access the Zedboard LEDs via the hardware configuration module, we set proper output pin configurations in the design constraint file.

For the crypto module, we implemented the design blocks indicated by the rectangles with sharp corners in Figure 6.4 (the encryption/decryption module). For the design blocks indicated by the rectangles with rounded corners in Figure 6.4, we initialized the design with pre-defined keys — the symmetric key for the hardware part and the private key

Table 6.2: Resource utilization of our hardware modules.

Module	Slice LUTs	Slice Registers	Slice	Frequency (MHz)
Crypto Module (AES)	2245	3602	1087	324.6
Timestamp Module	90	58	54	242.2
Remaining IP Block	482	602	205	242.2

Table 6.3: Device utilization summary.

Resource	Utilization	Available	Utilization %
LUT	2844	53200	5.35
LUTRAM	60	17400	0.34
FF	4336	106400	4.07
IO	10	200	5.00

for the software part. We implemented the encryption module from Opencores⁴ using the AES-128 encryption algorithm in the counter (CTR) mode operation with a pre-defined “counter” value. In AES-CTR mode implementation, the AES hardware cost is reduced by 50% (decryption hardware is not required). This AES implementation features a pipeline architecture with a fully synchronous and synthesizable design. The design has only one clock domain in entire core.

In our implementation, the maximum frequency calculated is 242.2MHz with a 100MHz clock. In Table 6.2, we summarize the resource utilization (in LUTs and Slices) and maximum clock frequency of all building blocks. The maximum frequency of the AES from Opencores is 324.6MHz with the throughput of 1.6Gbytes/second and latency of 21 clock cycles. The device utilization summary is given in Table 6.3.

6.8 Security Analysis of the Protocol

This section considers various system-level and hardware-level threats in our proposed protocol and how our proposed solution can counter them.

⁴https://opencores.org/projects/tiny_aes

Overuse of reconfiguration After paying for the upgrade, the user might want to enjoy the upgraded features in his device even after the upgrade period. Our Hardware Configuration Module (HCM) ensures that the configuration is set back to the previous state after the upgraded reconfiguration period.

Free loading If a user owns multiple devices of same type, he might rent just one upgraded reconfiguration and apply the same upgraded reconfiguration to all of his devices. In our protocol, each reconfiguration is encrypted using the public key (K_d^+) of the device. So, only that specific device can decrypt it with its private key (K_d^-) and install the reconfiguration. This private key is unique to each device and any other device cannot decrypt the firmware and install it.

Arbitrary pricing In a centralized system, the vendor could charge arbitrary fees from the users. In our proposed protocol, as pricing model is pre-determined in the smart contract, the users are safeguarded from any arbitrary pricing.

Incorrect payment by the user The user might not send enough money to get the reconfiguration upgrade. In our protocol, the payment is automated and follows the pre-established rules of the smart contract. If the user pays less than the required amount of money for the upgrade, he/she doesn't get the upgrade.

Physical attack by a malicious user to steal the configuration file An attacker might want to mount a physical attack on the device to get the configuration file and apply the stolen configuration to other devices or the same device later. Our proposed HCM stores the configurations encrypted to prevent any physical attacks by an attacker.

Integrity of the reconfiguration and sender authentication A malicious attacker could modify the reconfiguration en route to the user's device via a man-in-the-middle attack or a compromised gateway. Our proposed protocol provides sender authentication and reconfiguration integrity through digital signatures. During device registration, the manufacturer computes the signature for a configuration c by RSA-encrypting c with the manufacture's private key K_m^- . The manufacturer then appends the signature $K_m^-(c)$ to the configuration c and registers the device with both information. Our scheme does not apply a hash function on the configuration to create the digital signature, as the configuration size would be smaller than the hash digest. Generally, when the hash digest is much smaller

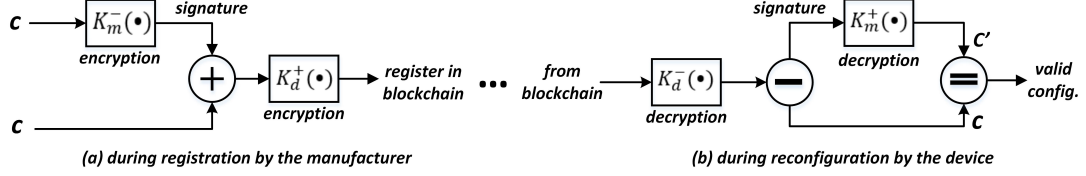


Figure 6.7: Authentication of the manufacturer, integrity and confidentiality of the configuration in our protocol.

than the original message's size, the hash function is applied to reduce the computational effort while creating the digital signature.

Before applying the update, the device receives both the configuration and the digital signature from the blockchain. The device then RSA-decrypts the signature using the manufacturer's public key K_m^+ , yielding c' . If c and c' match, two essential things can be ensured: (i) the manufacturer signed the configuration and (ii) the configuration had not been changed in transit, so that message integrity is given. The manufacturer's public key could be fused at the fabric/hardware level. We note here that the configuration and the signature are encrypted by the corresponding device public key K_d^+ during the registration process and later decrypted by the device private key K_d^- , as described in Section 6.4.2.1. Figure 6.7 presents the overall scheme for the manufacturer authentication and the configuration integrity and confidentiality in our proposed protocol.

6.9 Limitations and Discussion

In this section, we identify the limitations and deployment considerations for our proposed protocol. We comment on possible ways to overcome these issues and highlight the ongoing works that are being done to address them.

6.9.1 Performance

Compared to a properly configured centralized database, a blockchain results in lower transaction throughput and higher latencies. For example, Main Ethereum blockchain's average throughput is 15 transactions per second. In addition, since miners are incentivized to prioritize transactions with higher rewards, the transaction time can be strongly affected by the transaction's **gas price**.

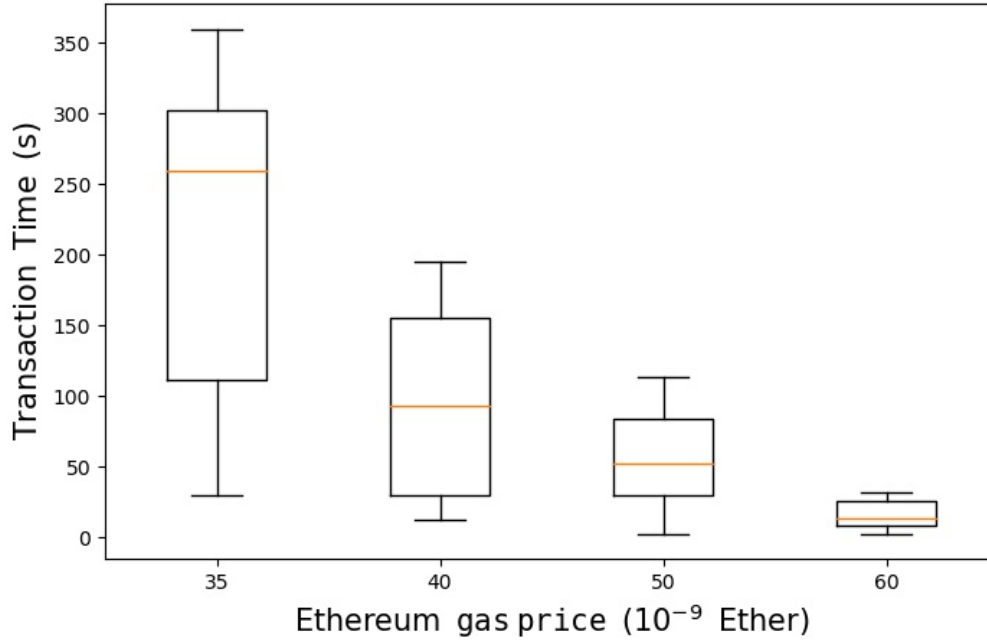


Figure 6.8: Transaction times of `upgradeConfiguration()` for increasing `gas price` in the Main Ethereum network with a `gas limit` of 83500.

In Figure 6.8, we issue transactions `upgradeConfiguration()` to our smart contract, with varying `gas prices` while holding all other variables constant. We measure the time it takes from sending the transaction from a node on the Ethereum Mainnet to seeing that transaction executed and included in a block on the chain. We issued the transaction six times with each `gas price`. As we can see from the Figure 6.8, a higher `gas price` reduces both the mean transaction time and the variance of transaction times, down to an average of 13 seconds with a `gas price` of 60×10^{-9} Ether. However, if we increase the `gas price` more, we observe diminishing returns as eventually we are bottlenecked by the Ethereum network’s throughput.

In Figure 6.9, we use a constant `gas price` and `gas limit` and run the same transaction 10 times over two days. The network load and `gas` value differ with time, resulting in higher tail latencies when a constant `gas price` is used. Our experiments measure a mean latency of 112 seconds and a 95-percentile latency of 190 seconds.

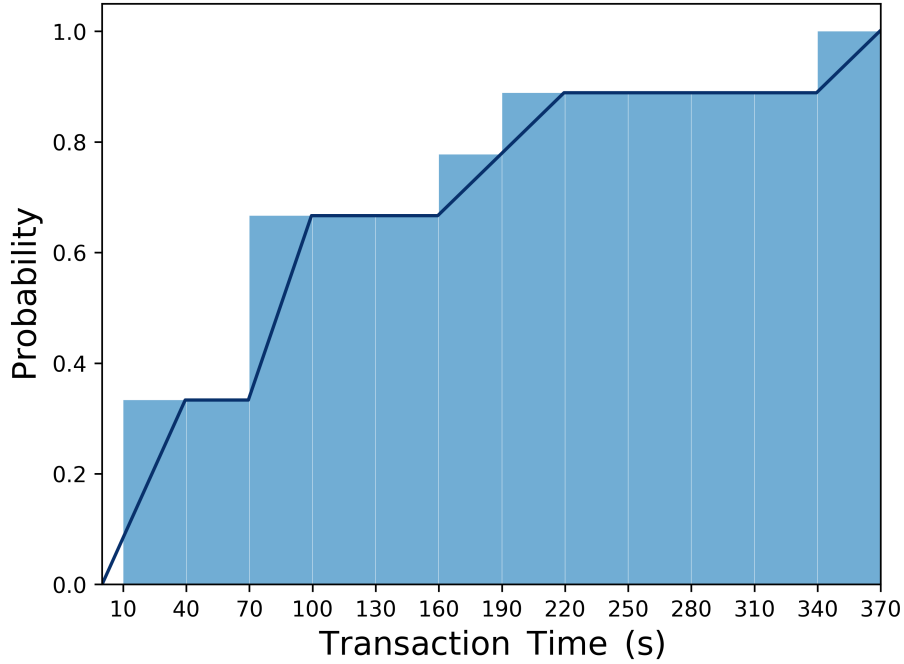


Figure 6.9: Cumulative density function (CDF) for the `upgradeConfiguration()` transaction times in the Main Ethereum network with a gas price of 40×10^{-9} Ether and a gas limit of 83500.

The comparatively lower performance of Ethereum smart contracts implies that applications that use them must design around the possibility of transaction delays (on the order of minutes to hours), or increase the gas price to reduce transaction times (down to the order of tens of seconds). In our protocol, we design the sequence of events so that transaction latency can be hidden from the user by allowing the Ethereum smart contract transactions to occur before the user device receives the configuration code (Figure 6.3). Few newer proposals, such as *sharding* [140], Bitcoin-NG [77], Stellar [152], show promising results to address performance.

6.9.2 Vulnerabilities in Smart Contracts

In contrary to traditional software, smart contracts cannot be directly patched after deployment. If a smart contract contains critical bugs, such as logical errors and lacks a self-destruct, assets can get locked in the contract. Smart contracts may have vulnerabilities at the programming language, bytecode, and blockchain levels that expose them to various kinds of attacks [35, 46]. The most notable example of bugs was in the Distributed

Autonomous Organization (The DAO) smart contracts that allowed a hacker to steal over \$50M USD worth of Ethers out of the \$168M funds invested. The Ethereum community voted to return (or hard fork) the state of the network to one prior to the hack, recovering Ethers back to investors. These challenges impose a unique set of considerations for designing smart contracts. Several solutions have been proposed in the literature to increase smart contracts' security [46, 63]. There are two common strategies to update smart contracts: (i) using a self-destruct function to release the contract's internal states, sending all the funds to a particular address, and then publishing a new contract, or (ii) including a mutable reference to the address of a newly created contract after the old contract is deprecated.

6.9.3 Transactional Privacy

Blockchain transactions contain a source address, a destination address, and the data or assets to be transferred. For validation, the content of every transaction is exposed to every node on the network. Although the addresses are not explicitly tied to any real-world identity, an interested party can monitor and analyze the data to discover patterns about a user's transactions. Several literature works have proposed techniques to attain transactional privacy, such as *zero-knowledge Succinct Non-interactive ARguments of Knowledge* (zkSNARK) [189], *homomorphic commitments* [150]. Other works, such as Hawk [127], explore methods for creating privacy-preserving smart contracts that do not store transactions clearly to maintain transactional privacy from the public.

6.10 Concluding Remarks

In this chapter, we proposed a smart contract-based device management system that can facilitate the configuration of functional components included in a remotely located IC. This system allows individual ICs to be custom configured to a specification mutually agreed by manufacturer and buyer. We partitioned the framework into software and hardware parts based on the analysis result of a pure software solution and the hardware overhead under design constraints. Our small hardware footprint ensures that the configuration code runs securely and that the encryption key is not leaked. Without hardware support, it cannot be guaranteed that the configuration and the secret key is not accessible by malware.

CHAPTER 7

CONCLUSION

In this dissertation, we propose an integrated platform solution for IoT device authentication, privacy, and security via blockchain-based smart contracts. We believe blockchain for IoT has the potential to enable a decentralized trust model for interoperable, digitized identities of devices and transactions between them on a global scale.

7.1 Summary of Contributions

For our first contribution, we ensure IoT device authentication by blockchain-based IC traceability system, from the time of its fabrication to its end-of-life, which allows both the supplier and a potential customer to verify an IC's provenance. To corroborate the blockchain information, we authenticate the IC securely and uniquely by embedded Physically Unclonable Function (PUF).

Secondly, the blockchain also integrates the protection of privacy of data generated from the IoT devices by giving users control of their own privacy. We tailor the hardware to meet the blockchain performance. We propose a side-channel attack to extract the private key successfully by exploiting the users' reliance on third-party hosted nodes to access the wallet, initiate transactions, and more. We propose a method where a co-tenant thread monitors the power management side-channel information from a thread affected by a hardware Trojan. Such a Trojan can leak private keys and disrupt digital transactions.

To conclude this dissertation research, we also propose a secure, remote configuration of IC features using smart contracts. In contrast to conventional enabling/disabling techniques rigidly carried out solely by the manufacturer, our proposed protocol provides flexible techniques by which a user may selectively enable/disable hardware features. The blockchain framework facilitates decentralized IoT where interacting devices are empowered to autonomously execute digital contracts.

7.2 Future Works

Our proposed dissertation research provides many new dimensions that are worth exploring in future work.

Secure data transmission from the device to the blockchain We have looked into requesting data from the blockchain node. In the future, we can look into the secure data transmission from the IoT device to the blockchain node. Our custom hardware design can be utilized for enabling secure data transmission from the device to the blockchain.

Dynamic pricing for remote IC Features configuration Leveraging our custom hardware, this secure data transmission can unearth several new applications. With time, when the device's performance will degrade due to aging, the manufacturer may lower its price accordingly, till its expiry. According to the age of the devices, we can develop a mechanism that can facilitate automated dynamic pricing of a device based on prior protocols defined in the smart contracts. If the device has a Residual Life Meter (RLM) [102, 104], it can send its aging data to the blockchain. We can adopt a dynamic pricing scheme instead of fixed pricing for the remote IC features configuration protocol using this aging data.

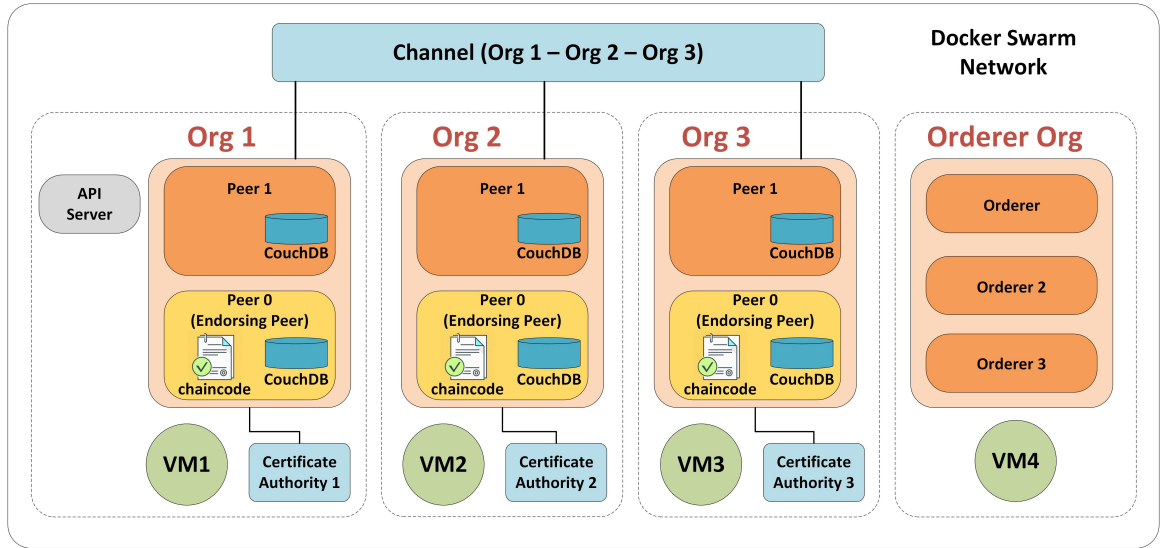
Customizing feature downgrades for a refund to the customer We can utilize our proposed remote configuration protocol to allow a customer to disable unwanted features selectively and subsequently apply for a refund corresponding to the disabled feature. This technique can be value-added for a customer because it can allow flexible changes to inventory (for a distributor) or individual products (for an end-user) to account for market shifts or changes in personal preferences. This technique can also be value-added for the manufacturer because the refund request may give the manufacturer some real-time data regarding customer preferences and market conditions efficiently.

Leveraging AI to create Decentralized Autonomous Organizations We can leverage the Big Data and Machine Learning aspects of Artificial Intelligence (AI) along with blockchain and other technologies. If AI is added to the IoT ecosystem connected to a blockchain network, it creates a Decentralized Autonomous Organization (DAO). DAO refers to an organization that runs without any human intervention.

APPENDIX A

MULTI-HOST HYPERLEDGER FABRIC IMPLEMENTATION

In this section, we describe the multi-host setup for Hyperledger Fabric implementation [10]. It is a setup of a Raft-based ordering service using Docker Swarm as the multi-host container environment. This setup is a more realistic deployment. We started everything from scratch and performed the following steps to bring up this Fabric network in a four-host deployment in the Google Cloud Platform (GCP) environment.

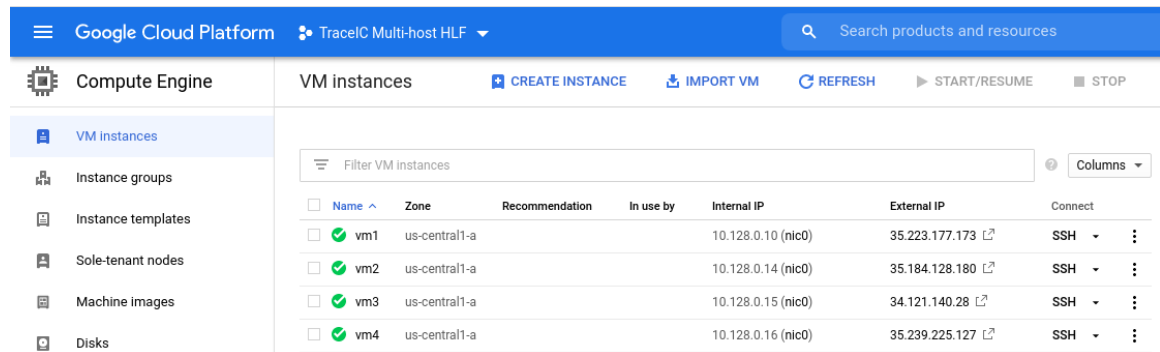


Our sample network comprises three peer organizations (representing the manufacturer, distributor, and retailer) and one orderer organization, as shown in the above diagram. We have a raft-based ordering service cluster of three ordering service nodes (orderers) in the orderer organization. In each peer organization (Organization 1, Organization 2, and Organization 3), there are two peers, Peer 0 and Peer 1. All peers have the CouchDB database, and for all the peer organizations, Peer 0 works as the endorsing peer. Each organization has their own certificate authority instance. The respective certificate authority creates the certificates for all the participants (admin, peer, client) in an organization.

We create a channel (consortium) *mychannel* among the three peer organizations. We deploy the chaincode (smart contract) in this channel, and using the chaincode, we invoke and query transactions. We deploy all the components and services (docker containers) related to Organization 1, 2, and 3 in VM1 (virtual machine 1), VM2, and VM3, respectively. For VM1 (Organization 1), we have a dedicated containerized API server to interact with the Hyperledger Fabric network from outside the Docker Swarm Network (e.g., using our local machine directly from the Postman client). Finally, we deploy all the components and services related to the Orderer organization in VM4.

A.1 Bringing Up All Hosts

We bring up four GCP VM instances representing four hosts (four organizations) and preload the required components (proper fabric prerequisite, tools, images) in the nodes according to the Fabric documentation. The four hosts are running Fabric v2.0 on Ubuntu 18.04 LTS. We create a custom firewall opening all ports and allowing all traffic (all UDP, TCP, and ICMP) for the demonstration purpose.



Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
vm1	us-central1-a			10.128.0.10 (nic0)	35.223.177.173	SSH
vm2	us-central1-a			10.128.0.14 (nic0)	35.184.128.180	SSH
vm3	us-central1-a			10.128.0.15 (nic0)	34.121.140.28	SSH
vm4	us-central1-a			10.128.0.16 (nic0)	35.239.225.127	SSH

A.2 Forming an Overlay Network with Docker Swarm

We use Docker Swarm to form an overlay network and make all the four hosts join. Docker Swarm is a container orchestration tool that provides an overlay network for containers across multiple hosts. Those containers on this overlay network can interact with each other as if they were on a large host. We opened four terminals, one for each host.

From Host 1,

```
ubuntu@vm1:~$ docker swarm init --advertise-addr 35.223.177.173
Swarm initialized: current node (ynqfvw43ntra7n7kz5mscg6dj) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-lprp6oycho0bwf6lqi9fofjgk76hqrewvzcdszjwpd7qtxilvk-lyndts4g14ijryxw00yv8tzb9 35.223.177.173:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

ubuntu@vm1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-lprp6oycho0bwf6lqi9fofjgk76hqrewvzcdszjwpd7qtxilvk-bb5esdxmq3plzoogb6bgozoyz 35.223.177.173:2377
```

Using the token from Host 1, we add other hosts as managers to this swarm. From Host 2, 3, and 4,

```
ubuntu@vm2:~$ docker swarm join --token SWMTKN-1-lprp6oycho0bwf6lqi9fofjgk76hqrewvzcdszjwpd7qtxilvk-bb5esdxmq3plzoogb6bgozoyz 35.223.177.173:2377 --advertise-addr 35.184.128.180
This node joined a swarm as a manager.

ubuntu@vm3:~$ docker swarm join --token SWMTKN-1-lprp6oycho0bwf6lqi9fofjgk76hqrewvzcdszjwpd7qtxilvk-bb5esdxmq3plzoogb6bgozoyz 35.223.177.173:2377 --advertise-addr 34.121.140.28
This node joined a swarm as a manager.

ubuntu@vm4:~$ docker swarm join --token SWMTKN-1-lprp6oycho0bwf6lqi9fofjgk76hqrewvzcdszjwpd7qtxilvk-bb5esdxmq3plzoogb6bgozoyz 35.223.177.173:2377 --advertise-addr 35.239.225.127
This node joined a swarm as a manager.
```

Finally, we add an overlay `artifacts_test`, which will be the network for our demo. We create this overlay network on Host 1 only. If Docker Swarm works correctly, all nodes will have this overlay network.

```
ubuntu@vm1:~$ docker network create --attachable --driver overlay artifacts_test v50hk9kyiwbxpduaisgdpjua
```

A.3 Preparing Fabric Network Materials in Host 1 and Copying to Others

One of the critical steps is to make sure that all components are sharing the same crypto materials. For the sake of simplicity, in this demo, we create all the materials in Host 1 and then copy the whole directory to other hosts. We create a Certificate Authority (CA) container for each organization. Certificates and signing keys for an organization (e.g., org1) are issued and signed by the same CA (ca.org1). For Organization 1, we create the Certificate Authority container (*ca.org1.example.com*) on Host 1.

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1/create-certificate-with-ca$ sudo docker-compose up -d
Creating network "artifacts_test" with the default driver
Creating ca.org1.example.com ... done
```

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1/create-certificate-with-ca$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
241c0ad9ae11	hyperledger/fabric-ca	"sh -c 'fabric-ca-se..."	19 minutes ago	Up 19 minutes	0.0.0.0:7054->7054/tcp

In the following, we generate the required crypto materials for Organization 1 on Host 1. After that, we register and enroll two *peers*, the *admin* and *user*.

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1/create-certificate-with-ca$ ./create-certificate-with-ca.sh
```

Enroll the CA admin

```
2020/11/02 10:03:39 [INFO] Created a default configuration file at /home/ubuntu/FabricMultiHostDeployment/setup1/vm1/crypto-config/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2020/11/02 10:03:39 [INFO] TLS Enabled
2020/11/02 10:03:39 [INFO] generating key: &{A:ecdsa S:256}
2020/11/02 10:03:39 [INFO] encoded CSR
2020/11/02 10:03:40 [INFO] Stored client certificate at /home/ubuntu/FabricMultiHostDeployment/setup1/vm1/crypto-config/peerOrganizations/org1.example.com/msp/signcerts/cert.pem
2020/11/02 10:03:40 [INFO] Stored root CA certificate at /home/ubuntu/FabricMultiHostDeployment/setup1/vm1/crypto-config/peerOrganizations/org1.example.com/msp/cacerts/localhost-7054-ca-org1-example-com.pem
2020/11/02 10:03:40 [INFO] Stored Issuer public key at /home/ubuntu/FabricMultiHostDeployment/setup1/vm1/crypto-config/peerOrganizations/org1.example.com/msp/IssuerPublicKey
2020/11/02 10:03:40 [INFO] Stored Issuer revocation public key at /home/ubuntu/FabricMultiHostDeployment/setup1/vm1/crypto-config/peerOrganizations/org1.example.com/msp/IssuerRevocationPublicKey
```

Register peer0

```
2020/11/02 10:03:40 [INFO] Configuration file location: /home/ubuntu/FabricMultiHostDeployment/setup1/vm1/crypto-config/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2020/11/02 10:03:40 [INFO] TLS Enabled
2020/11/02 10:03:40 [INFO] TLS Enabled
Password: peer0pw
```

Register peer1

```
2020/11/02 10:03:40 [INFO] Configuration file location: /home/ubuntu/FabricMultiHostDeployment/setup1/vm1/crypto-config/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2020/11/02 10:03:40 [INFO] TLS Enabled
```

Similarly, we create the Certificate Authority containers and generate the required crypto materials for Organization 2 and Organization 3 on Host 1. After that, we register and enroll the organizations' *peers*, the *admins* and *users*. Finally, we create the Certificate Authority container for Orderer Organization and generate the required crypto materials on Host 1. This time, we register the three orderers. This way we create all the crypto materials required for the Fabric network on Host 1.

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm4/create-certificate-with-ca$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
32f1cc3e63c1	hyperledger/fabric-ca	"sh -c 'fabric-ca-se..."	5 seconds ago	Up 3 seconds	7054/tcp, 0.0.0.0:9054->7054/tcp
082c567712d4	hyperledger/fabric-ca	"sh -c 'fabric-ca-se..."	47 seconds ago	Up 45 seconds	7054/tcp, 0.0.0.0:10054->7054/tcp
76af7a6e4620	hyperledger/fabric-ca	"sh -c 'fabric-ca-se..."	9 minutes ago	Up 9 minutes	7054/tcp, 0.0.0.0:8054->7054/tcp
241c0ad9ae11	hyperledger/fabric-ca	"sh -c 'fabric-ca-se..."	50 minutes ago	Up 50 minutes	0.0.0.0:7054->7054/tcp

In this way, we create the required crypto materials for all the participants for each organization. With all these crypto materials available now, we run the `create-artifacts.sh` script to create the Genesis block (`genesis.block`), the Channel transaction (`mychannel.tx`) file,

and the *anchor peer* update transaction files for the three peer organizations. We require the Genesis block for bootstrapping the orderers and Channel transaction file for configuring a *channel* among the peer organizations.

```
ubuntu@vm1:~/FabricMultiHostDeployment/artifacts/channel$ ./create-artifacts.sh
chmod: cannot access './crypto-config': No such file or directory
rm: cannot remove 'genesis.block': No such file or directory
rm: cannot remove 'mychannel.tx': No such file or directory
mychannel
2020-11-02 10:22:46.227 UTC [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-11-02 10:22:46.256 UTC [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: e
tcdraft
2020-11-02 10:22:46.256 UTC [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 Orderer.EtcdRaf
t.Options unset, setting to tick_interval:"500ms" election_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_in
terval_size:16777216
2020-11-02 10:22:46.256 UTC [common.tools.configtxgen.localconfig] Load -> INFO 004 Loaded configuration: /home/ubuntu
u/FabricMultiHostDeployment/artifacts/channel/configtx.yaml
2020-11-02 10:22:46.259 UTC [common.tools.configtxgen] doOutputBlock -> INFO 005 Generating genesis block
2020-11-02 10:22:46.259 UTC [common.tools.configtxgen] doOutputBlock -> INFO 006 Writing genesis block
2020-11-02 10:22:46.292 UTC [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-11-02 10:22:46.319 UTC [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/ubuntu
u/FabricMultiHostDeployment/artifacts/channel/configtx.yaml
2020-11-02 10:22:46.319 UTC [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel con
figtx
2020-11-02 10:22:46.322 UTC [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
##### Generating anchor peer update for Org1MSP #####
2020-11-02 10:22:46.354 UTC [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-11-02 10:22:46.380 UTC [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/ubuntu
u/FabricMultiHostDeployment/artifacts/channel/configtx.yaml
2020-11-02 10:22:46.380 UTC [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer u
pdate
2020-11-02 10:22:46.382 UTC [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer upda
te
##### Generating anchor peer update for Org2MSP #####
2020-11-02 10:22:46.414 UTC [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-11-02 10:22:46.441 UTC [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/ubuntu
u/FabricMultiHostDeployment/artifacts/channel/configtx.yaml
```

We have everything on Host 1 now. We copy this directory to all the other hosts. As we cannot copy files across VMs, we first push this directory into a Git repository and pull that repository to the other three VMs. After doing that, all nodes will have the same crypto material and required docker-compose files. We are ready to bring up all containers.

A.4 Bringing Up the Containers in Each Host

We use *docker-compose* to bring up all containers in each host. In Host 1, there is a total of five containers (two *peers*, two *couchDBs* for those two *peers*, a CLI container). We can see the docker containers up and running on Host 1.

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Pulling peer0.org1.example.com (hyperledger/fabric-peer:2.1)...
2.1: Pulling from hyperledger/fabric-peer
cbdbe7a5bc2a: Pull complete
c1275de45a84: Pull complete
533397ecbcb: Pull complete
366c0e7ddacd: Pull complete
66658cc1667a: Pull complete
Digest: sha256:86f4c15607d33bff44f965631a966bc142ebe8e7c18a25a41bea425cefa6dfd7
Status: Downloaded newer image for hyperledger/fabric-peer:2.1
Creating peer1.org1.example.com ... done
Creating cli ... done
Creating couchdb0 ... done
Creating couchdb1 ... done
Creating peer0.org1.example.com ... done
```

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
5b7ffe0e9314	hyperledger/fabric-peer:2.1	peer0.org1.example.com	"peer node start"	5 hours ago	Up 5 hours	0.0.0.0:7051->7051/tcp
fd3162d3709c	hyperledger/fabric-couchdb	couchdb1	"tini -- /docker-ent..."	5 hours ago	Up 5 hours	4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp
9c9a937f1961	hyperledger/fabric-couchdb	couchdb0	"tini -- /docker-ent..."	5 hours ago	Up 5 hours	4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp
202e75eb1acb	hyperledger/fabric-peer:2.1	peer1.org1.example.com	"peer node start"	5 hours ago	Up 5 hours	7051/tcp, 0.0.0.0:8051->8051/tcp
56f7148a8113	hyperledger/fabric-tools	cli	"/bin/bash"	5 hours ago	Up 5 hours	

Similarly, we use *docker-compose* to bring up all containers in Host 2, Host 3, and Host 4. In Host 2 and 3 each, there is a total of four containers (two *peers*, two *couchDBs* for those two *peers*). In Host 4, there is a total of three containers (three *orderers*). The terminal outputs on Host 2 and Host 4 are shown below.

```
ubuntu@vm2:~/FabricMultiHostDeployment/setup1/vm2$ docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Pulling peer0.org2.example.com (hyperledger/fabric-peer:2.1)...
2.1: Pulling from hyperledger/fabric-peer
cbdbe7a5bc2a: Pull complete
c1275de45a84: Pull complete
533397ecbcb: Pull complete
366c0e7ddacd: Pull complete
66658cc1667a: Pull complete
Digest: sha256:86f4c15607d33bff44f965631a966bc142ebe8e7c18a25a41bea425cefa6dfd7
Status: Downloaded newer image for hyperledger/fabric-peer:2.1
Creating couchdb2 ... done
Creating peer1.org2.example.com ... done
Creating couchdb3 ... done
Creating peer0.org2.example.com ... done
ubuntu@vm2:~/FabricMultiHostDeployment/setup1/vm2$ docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
81e2844d9d49	hyperledger/fabric-peer:2.1	peer0.org2.example.com	"peer node start"	3 minutes ago	Up 2 minutes	7051/tcp, 0.0.0.0:9051->9051/tcp
485f545dd659	hyperledger/fabric-peer:2.1	peer1.org2.example.com	"peer node start"	3 minutes ago	Up 2 minutes	7051/tcp, 0.0.0.0:10051->10051/tcp
485b228ca969	hyperledger/fabric-couchdb	couchdb3	"tini -- /docker-ent..."	3 minutes ago	Up 2 minutes	4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp
fe336acedfab	hyperledger/fabric-couchdb	couchdb2	"tini -- /docker-ent..."	3 minutes ago	Up 2 minutes	4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp


```
ubuntu@vm4:~/FabricMultiHostDeployment/setup1/vm4$ docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on
the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Pulling orderer.example.com (hyperledger/fabric-orderer:2.1)...
2.1: Pulling from hyperledger/fabric-orderer
cbdbe7a5bc2a: Pull complete
c1275de45a84: Pull complete
f588f9bba3d6: Pull complete
8f1a3c5e0a14: Pull complete
080b660b2196: Pull complete
1c1cf954d27a: Pull complete
Digest: sha256:163b245e18c5361520e10260093eb9271a091bf71ace8a55e0190658575e3037
Status: Downloaded newer image for hyperledger/fabric-orderer:2.1
Creating orderer2.example.com ... done
Creating orderer3.example.com ... done
Creating orderer.example.com ... done
ubuntu@vm4:~/FabricMultiHostDeployment/setup1/vm4$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
b926c7862171	hyperledger/fabric-orderer:2.1	"orderer"	14 seconds ago	Up 9 seconds	7050/tcp, 0.0.0.0:9050->9050/tcp, 0.0.0.0:8445->8443/tcp
63f43819b030	hyperledger/fabric-orderer:2.1	"orderer"	14 seconds ago	Up 8 seconds	0.0.0.0:7050->7050/tcp, 0.0.0.0:8443->8443/tcp
ec4991efa7dc	hyperledger/fabric-orderer:2.1	"orderer"	14 seconds ago	Up 9 seconds	7050/tcp, 0.0.0.0:8050->8050/tcp, 0.0.0.0:8444->8443/tcp

A.5 Creating Channel and All Peer Nodes Joining It

As we have CLI only on Host 1, we issue all commands from the Host 1 terminal. We first create the channel configuration block for *mychannel* and make the two peers of Organization 1 join the channel using this block.

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ ./createChannel.sh
2020-11-02 16:23:13.501 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:23:13.530 UTC [cli.common] readBlock -> INFO 002 Expect block, but got status: &{NOT_FOUND}
2020-11-02 16:23:13.536 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2020-11-02 16:23:13.738 UTC [cli.common] readBlock -> INFO 004 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-11-02 16:23:13.743 UTC [channelCmd] InitCmdFactory -> INFO 005 Endorser and orderer connections initialized
2020-11-02 16:23:13.945 UTC [cli.common] readBlock -> INFO 006 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-11-02 16:23:13.949 UTC [channelCmd] InitCmdFactory -> INFO 007 Endorser and orderer connections initialized
2020-11-02 16:23:14.153 UTC [cli.common] readBlock -> INFO 008 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-11-02 16:23:14.157 UTC [channelCmd] InitCmdFactory -> INFO 009 Endorser and orderer connections initialized
2020-11-02 16:23:14.359 UTC [cli.common] readBlock -> INFO 00a Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-11-02 16:23:14.364 UTC [channelCmd] InitCmdFactory -> INFO 00b Endorser and orderer connections initialized
2020-11-02 16:23:14.565 UTC [cli.common] readBlock -> INFO 00c Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-11-02 16:23:14.570 UTC [channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
2020-11-02 16:23:14.775 UTC [cli.common] readBlock -> INFO 00e Received block: 0
2020-11-02 16:23:14.850 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:23:15.104 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-11-02 16:23:15.161 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:23:15.373 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-11-02 16:23:15.423 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:23:15.441 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
```

From the Host 2 and Host 3, we fetch the channel configuration block and make their peers join the channel using this block. The terminal output from Host 2 is shown below.

```
ubuntu@vm2:~/FabricMultiHostDeployment/setup1/vm2$ ./joinChannel.sh
2020-11-02 16:27:31.716 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:27:31.724 UTC [cli.common] readBlock -> INFO 002 Received block: 0
2020-11-02 16:27:31.796 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:27:32.162 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-11-02 16:27:32.316 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:27:32.606 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-11-02 16:27:32.668 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-11-02 16:27:32.712 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
```

A.6 Installing and Instantiating TraceIC Chaincode

In this step, we package, install, and approve the *traceic* chaincode for Organization 1 on Host 1. After chaincode approval from Organization 1, we check the commit readiness of the chaincode. As the other two peer organizations have not approved the chaincode yet, we can see only approval from Organization 1 for the chaincode.

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ ./deployChaincode.sh
===== Chaincode is packaged on peer0.org1 =====
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ ./deployChaincode.sh
2020-11-02 16:51:02.575 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:
<status:200 payload:"\nJtraceic 1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f\022\ttraceic 1" >
2020-11-02 16:51:02.575 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Chaincode is installed on peer0.org1 =====
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ ./deployChaincode.sh
Installed chaincodes on peer:
Package ID: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f, Label: traceic_1
PackageID is traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Query installed successful on peer0.org1 on channel =====
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ ./deployChaincode.sh
Installed chaincodes on peer:
Package ID: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f, Label: traceic_1
PackageID is traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Query installed successful on peer0.org1 on channel =====
2020-11-02 16:52:28.980 UTC [chaincodeCmd] ClientWait -> INFO 001 txid [b8eeede927a78baf163b6b267ed28f78e7f499a9b1abdbef612c6411b693d28] committed with status (VALID) at
===== chaincode approved from org 1 =====
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ ./deployChaincode.sh
{
  "approvals": {
    "Org1MSP": true,
    "Org2MSP": false,
    "Org3MSP": false
  }
}
===== checking commit readiness from org 1 =====
```

We choose the chaincode life-cycle *endorsement policy* as *N of N*. So, we have to package, install and approve the chaincode for Organization 2 from Host 2 and Organization 3 from Host 3.

```
===== Chaincode is packaged on peer0.org2 =====
2020-11-02 16:57:07.895 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:
<status:200 payload:"\nJtraceic 1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f\022\ttraceic 1" >
2020-11-02 16:57:07.895 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Chaincode is installed on peer0.org2 =====
Installed chaincodes on peer:
Package ID: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f, Label: traceic_1
PackageID is traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Query installed successful on peer0.org2 on channel =====
Installed chaincodes on peer:
Package ID: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f, Label: traceic_1
PackageID is traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Query installed successful on peer0.org2 on channel =====
2020-11-02 16:57:10.620 UTC [chaincodeCmd] ClientWait -> INFO 001 txid [2ac8901aaede2c3c1bb5a20ea1408e5ba31277e30e8e00f1a8bde92f26ddb47f] committed with status (VALID) at
===== chaincode approved from org 2 =====
{
  "approvals": {
    "Org1MSP": true,
    "Org2MSP": true,
    "Org3MSP": false
  }
}
===== checking commit readiness from org 1 =====
```



```

Finished vendoring Go dependencies
===== Chaincode is packaged on peer0.org3 =====
2020-11-02 17:09:05.157 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:
<status:200 payload:"\nJtraceic 1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f\022\ttraceic 1" >
2020-11-02 17:09:05.157 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identi
fier: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Chaincode is installed on peer0.org3 =====
Installed chaincodes on peer:
Package ID: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f, Label: traceic_1
PackageID is traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Query installed successful on peer0.org3 on channel =====
Installed chaincodes on peer:
Package ID: traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f, Label: traceic_1
PackageID is traceic_1:4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebffc8f
===== Query installed successful on peer0.org3 on channel =====
2020-11-02 17:09:07.804 UTC [chaincodeCmd] ClientWait -> INFO 001 txid [b2b08860dec113519c216e291f51c122ee2d6af1ead93
0d311f651230c97f5a8] committed with status (VALID) at
===== chaincode approved from org 3 =====
{
  "approvals": {
    "Org1MSP": true,
    "Org2MSP": true,
    "Org3MSP": true
  }
}
===== checking commit readiness from org 3 =====
ubuntu@vm3:~/FabricMultiHostDeployment/setup1/vm3$

```

After the Organization 2 and Organization 3 approve the chaincode, we can commit chaincode definition to all the *endorsing peers*. We can do it from any host. The following shows committing the chaincode by Organization 1 from the CLI container on Host 1.

```

ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ docker exec -it cli bash
bash-5.0# export CORE_PEER_LOCALMSPID="Org1MSP"
bash-5.0# export CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/channel/crypto-config/peerOrganizations/org1.example.co
m/peers/peer0.org1.example.com/tls/ca.crt
bash-5.0# export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/channel/crypto-config/peerOrganizations/org1.example.com/us
ers/Admin@org1.example.com/msp
bash-5.0# export CORE_PEER_ADDRESS=peer0.org1.example.com:7051
bash-5.0# export CHANNEL_NAME="mychannel"
bash-5.0# export CC_NAME="traceic"
bash-5.0# export ORDERER_CA=/etc/hyperledger/channel/crypto-config/ordererOrganizations/example.com/orderers/orderer.
example.com/msp/tlsacerts/tlsca.example.com-cert.pem
bash-5.0# export VERSION="1"
bash-5.0# peer lifecycle chaincode commit -o orderer.example.com:7050 --ordererTLSHostnameOverride orderer.example.co
m \
> --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA \
> --channelID $CHANNEL_NAME --name ${CC_NAME} \
> --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganiza
tions/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \
> --peerAddresses peer0.org3.example.com:11051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganiza
tions/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt \
> --peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganiza
tions/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt \
> --version ${VERSION} --sequence ${VERSION} --init-required
2020-11-02 17:24:04.586 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABR
IC LOGGING SPEC environment variable
2020-11-02 17:24:04.590 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use th
e FABRIC LOGGING SPEC environment variable
2020-11-02 17:24:07.309 UTC [chaincodeCmd] ClientWait -> INFO 003 txid [59e66ebd741a26a0451071fc52667d7fbb5b0b25e4765
054cedffb87bd06af56] committed with status (VALID) at peer0.org1.example.com:7051
2020-11-02 17:24:07.313 UTC [chaincodeCmd] ClientWait -> INFO 004 txid [59e66ebd741a26a0451071fc52667d7fbb5b0b25e4765
054cedffb87bd06af56] committed with status (VALID) at peer0.org2.example.com:9051
2020-11-02 17:24:07.350 UTC [chaincodeCmd] ClientWait -> INFO 005 txid [59e66ebd741a26a0451071fc52667d7fbb5b0b25e4765

```

As the CLI container is in the Docker Swarm network, the chaincode container gets created in each host. Thus each organization now has the chaincode container on their respective VM. We can also verify if the chaincode commit is successful by querying it.

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
236ae4b82614	dev-peer0.org1.example.com-traceic_1-4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebf8f-ce02cfd87af1968c99b0c127b19970ca0d456bb7e02cd87c2a85386ef85bce48	"chaincode -peer.add..."	49 seconds ago	Up 47 seconds
4322b4f0c0be50ecf0f101b015d09bebf8f	dev-peer0.org1.example.com-traceic_1-4945b7669228d05fc22c4135a1c			
5b7ffe0e9314	hyperledger/fabric-peer:2.1			
rs	0.0.0.0:7051->7051/tcp	"peer node start"	6 hours ago	Up 6 hours
fd3162d3709c	hyperledger/fabric-couchdb			
rs	4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp	"tini -- /docker-ent..."	6 hours ago	Up 6 hours
9c9a937f1961	hyperledger/fabric-couchdb			
rs	4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp	"tini -- /docker-ent..."	6 hours ago	Up 6 hours
202e75eb1acb	hyperledger/fabric-peer:2.1			
rs	7051/tcp, 0.0.0.0:8051->8051/tcp	"peer node start"	6 hours ago	Up 6 hours
56f7148a8113	hyperledger/fabric-tools			
rs		"/bin/bash"	6 hours ago	Up 6 hours
		cli		

```
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vm1$ ./deployChaincode.sh
Committed chaincode definition for chaincode 'traceic' on channel 'mychannel':
Version: 1, Sequence: 1, Endorsement Plugin: escsc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true, Org3MSP: true]
```

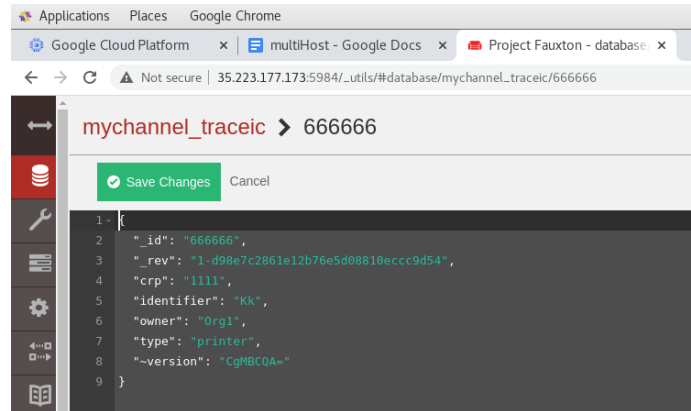
A.7 Invoking and Querying the Chaincode

After we commit the chaincode and create the container for it, the next step is to invoke and query the chaincode. First, we invoke `init` and then invoke `createIC` with some sample arguments to the chaincode from the CLI container.

```
bash-5.0# peer chaincode invoke -o orderer.example.com:7050 \
> --ordererTLSHostnameOverride orderer.example.com \
> --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA \
> -C $CHANNEL_NAME -n ${CC_NAME} \
> --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \
> --peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt \
> --peerAddresses peer0.org3.example.com:11051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt \
> --isInit -c '{"Args":[]}'
2020-11-02 18:12:26.693 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2020-11-02 18:12:26.696 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2020-11-02 18:12:26.850 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003 Chaincode invoke successful. result: status:200
```

```
bash-5.0# peer chaincode invoke -o orderer.example.com:7050 \
> --ordererTLSHostnameOverride orderer.example.com \
> --tls \
> --cafile /etc/hyperledger/channel/crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
> -C mychannel -n traceic \
> --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \
> --peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt \
> --peerAddresses peer0.org3.example.com:11051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-config/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt \
> -c '{"function": "createIC", "Args": [{"666666", "Kk", "printer", "1111", "Org1"}]}'
2020-11-02 18:21:42.097 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2020-11-02 18:21:42.099 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2020-11-02 18:21:42.178 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003 Chaincode invoke successful. result: status:200 payload:{"identifier":"Kk","type":"printer","crp":"1111","owner":"Org1"}
```

We can verify the invocation from Fauxton. Fauxton is a native web-based interface built into CouchDB. It provides a basic interface to most functionalities, including creating, updating, deleting, viewing documents, and designing documents.



After that, we can query an IC record from the CLI container. This query result shows that the fabric network is working well.

```
bash-5.0# peer chaincode query -C $CHANNEL_NAME -n ${CC_NAME} -c '{"function": "queryIC", "Args": ["666666"]}'
2020-11-02 18:27:50.192 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC
IC LOGGING SPEC environment variable
2020-11-02 18:27:50.195 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use th
e FABRIC LOGGING SPEC environment variable
{"crp":"1111","identifier":"Kk","owner":"Org1","type":"printer"}
```

Invoking or querying the chaincode from the CLI container is a little cumbersome process. We create a separate API service container (*artifacts_api_1*) to interact with the chaincode from outside the virtual machine using Postman client. For implementing the API, we use the latest Fabric SDK. This implementation contains a Nodejs API and uses JWT (JSON Web Token) for user identity verification.

```
Successfully built 3fdc51583b39
Successfully tagged api:1.0
WARNING: Image for service api was built because it did not already exist. To rebuild this image you must use `docker
-compose build` or `docker-compose up --build`.
Creating artifacts api_1 ... done
ubuntu@vm1:~/FabricMultiHostDeployment/setup1/vml/api-2.0$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
411726aa0322	api:1.0			
conds	0.0.0.0:4000->4000/tcp	artifacts api_1	11 seconds ago	Up 10 se
236ae4b82614	dev-peer0.org1.example.com-traceic_1-4945b7669228d05fc22c4135a1c4322b4f0c0be50ecf0f101b015d09bebf8f-ce02cfd87af1968c99b0c127b19970ca0d456bb7e02cd87c2a85386ef85bce48	chaincode -peer.add...	2 hours ago	Up 2 hou
rs	4322b4f0c0be50ecf0f101b015d09bebf8f	dev-peer0.org1.example.com-traceic_1-4945b7669228d05fc22c4135a1c		
5b7ffe0e9314	hyperledger/fabric-peer:2.1	peer node start	8 hours ago	Up 8 hou
rs	0.0.0.0:7051->7051/tcp	peer0.org1.example.com		
fd3162d3709c	hyperledger/fabric-couchdb	tini -- /docker-ent...	8 hours ago	Up 8 hou
rs	4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp	couchdb1		
9c9a937f1961	hyperledger/fabric-couchdb	tini -- /docker-ent...	8 hours ago	Up 8 hou
rs	4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp	couchdb0		
202e75eb1acb	hyperledger/fabric-peer:2.1	peer node start	8 hours ago	Up 8 hou
rs	7051/tcp, 0.0.0.0:8051->8051/tcp	peer1.org1.example.com		
56f7148a8113	hyperledger/fabric-tools			
rs		/bin/bash	8 hours ago	Up 8 hou
		cli		

We implement three APIs in the container: (i) register user, (ii) register IC, and (iii) query IC. Firstly, we register a new user, and then with the new user token, we invoke and query the chaincode. As we register and enroll new users for an organization at runtime, we have to ensure that the CA container is up and running for that organization. For the demonstration purpose, we create the API container on Host 1 and use its IP address and port number to register a new user from the Postman client.

We see that the user gets a token. The token gets mapped to an ID, and all crypto materials and the identification (ID) for that user get created inside the container on Host 1.

Using the token created during the registration, the user can issue transactions to the chaincode. First, we invoke `createIC` transaction with some sample arguments (key, identifier, type, CRP, and owner) to the `traceic` chaincode in channel `mychannel`.

POST Register User POST Add IC

▶ Add IC Examples 0 BUILD

POST http://35.223.177.173:4000/channels/mychannel/chaincodes/traceic Send Save

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies C

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beauti

```

2  "fcn": "createIC",
3  "peers": ["peer0.org1.example.com", "peer0.org2.example.com"],
4  "chaincodeName": "traceic",
5  "channelName": "mychannel",
6  "args": ["8888", "LL", "scanner", "2222", "Org1"]

```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 2.74 s Size: 345 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "result": {
3      "message": "Successfully added the ic asset with key 8888"
4    }
5  },

```

We see that the IC is successfully registered. We can also verify the invocation from the CouchDB database in peer 0 of Organization 1 update from the Fauxton as shown below.

Applications Places Google Chrome

Google Cloud Platform multiHost - Google Docs Project Fauxton

Not secure | 35.223.177.173:5984/_utils/#database/mychannel_traceic/_all_docs

mychannel_traceic Document ID Options {} JSON Create Doc

All Documents Run A Query with Mango Permissions Changes Design Documents

id	key	value
owner-key:Org1:666666	owner-key:Org1:666666	{ "rev": "1-dcc50942b196e5e61bb..." }
owner-key:Org1:8888	owner-key:Org1:8888	{ "rev": "1-d1c523e63b900fae739..." }
initialized	initialized	{ "rev": "1-4b7ace2da4833a65ff6..." }
666666	666666	{ "rev": "1-d98e7c2861e12b76e5d..." }
8888	8888	{ "rev": "1-9e66b7aab4383ca8997..." }

Applications Places Google Chrome

Google Cloud Platform multiHost - Google Docs Project Fauxton - database

Not secure | 35.223.177.173:5984/_utils/#database/mychannel_traceic/8888

mychannel_traceic > 8888 Save Changes Cancel

```

1  {
2    "id": "8888",
3    "rev": "1-9e66b7aab4383ca899707753ee43c51c",
4    "crp": "2222",
5    "identifier": "LL",
6    "owner": "Org1",
7    "type": "scanner",
8    "~version": "CgMBCgA="
9  }

```

The CouchDB databases in other organizations' peers also get updated with a successful transaction invocation. The CouchDB database for peer 0 of Organization 2 is shown as below.

id	key	value
owner-key:Org1:666666	owner-key:Org1:666666	{ "rev": "1-d0c50942b136e5e01bb..." }
owner-key:Org1:888888	owner-key:Org1:888888	{ "rev": "1-d1c523e63b600f4e739..." }
initialized	initialized	{ "rev": "1-4b7ace2da483fa6586..." }
666666	666666	{ "rev": "1-d98e7c2861e12b7e5d..." }
8888	8888	{ "rev": "1-9e66b7aab4383ca8997..." }

After we invoke the chaincode, we use query to check again.

POST Register User POST Add IC GET Query By IC ID No Environment

Query By IC ID Examples 0 BUILD

GET http://35.223.177.173:4000/channels/mychannel/chaincodes/traceic?args=["8888"]&peer=peer0.org1.ex... Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

KEY	VALUE	DESCRIPTION
args	["8888"]	
peer	peer0.org1.example.com	
fcn	queryIC	

Body Cookies Headers (7) Test Results Status: 200 OK Time: 287 ms Size: 350 B Save Response

Pretty Raw Preview Visualize JSON

```

2  "result": {
3    "crp": "2222",
4    "identifier": "L1",
5    "owner": "Org1",
6    "type": "scanner"

```

In this demonstration, we implement a RAFT-based orderer cluster with a Fabric network. These containers run on four different hosts. Docker Swarm brings these four hosts together, allowing containers running on separate hosts to communicate. We do not specify static IP on fabric network components, and all containers communicate with each other such that they were on the same host. We use the existing Docker Compose files with minimum modification to run on Docker Swarm.

APPENDIX B

SAMPLE CODE SNIPPET FOR TRIGGERING THE PMU-TROJAN

Table B.1: Code Snippet for Triggering the PMU-Trojan

```
echo -n "userspace" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo -n 3100 /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
openssl speed
echo -n 2700 /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
openssl speed
echo -n 2300 /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
openssl speed
```

BIBLIOGRAPHY

- [1] Bitcoin testnet3. *URL: <https://en.bitcoin.it/wiki/Testnet>.*
- [2] Blockchain for IC Traceability. *URL: <https://github.com/nazmulislam025/IC-Traceability-Blockchain>.*
- [3] c2ctl. *URL: <https://aur.archlinux.org/packages/c2ctl/>.*
- [4] Competitive Landscape: Power Management IC and Power Semiconductor Vendors, 2012. *URL: <https://www.gartner.com/doc/2010815/competitive-landscape-power-management-ic>.*
- [5] CPU-Z & HWMONITOR PRO. *URL: <http://www.cpuid.com/>.*
- [6] Cpufreq. *URL: <https://www.kernel.org/doc/Documentation/cpu-freq/>.*
- [7] Data quality: It's a supply chain issue. *URL: <http://www.dataversity.net/data-quality-its-a-supply-chain-issue/>.*
- [8] Data security — How Bosch secures the camera. *URL: http://resource.boschsecurity.com/documents/WP_TPM_WhitePaper_enUS_9007223261094667.pdf.*
- [9] ESP8266 AT Instruction Set. *URL: https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf.*
- [10] Fabric Multi-host Deployment. *URL: <https://github.com/mni025/FabricMultiHostDeployment>.*
- [11] Go Ethereum. *URL: <https://github.com/ethereum/go-ethereum>.*
- [12] Hyperledger Caliper. *URL: <https://github.com/hyperledger/caliper>.*
- [13] Implementing IC Traceability using Hyperledger Fabric. *URL: <https://github.com/mni025/TraceICHyperledgerFabric>.*
- [14] Intel Core i7-4650U Processor. *URL: <http://ark.intel.com/products/75114/Intel-Core-i7-4650U-Processor-4M-Cache-up-to-3-30-GHz>.*
- [15] Intel Extreme Tuning Utility. *URL: <https://www.intel.com/content/www/us/en/support/processors/processor-utilities-and-programs/intel-extreme-tuning-utility-intel-xtu.html>.*
- [16] International Technology Roadmap for Semiconductor (ITRS). *URL: <http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm>.*
- [17] Nangate Open Cell Library. *URL: <http://www.si2.org/openeda.si2.org/projects/nangate/lib>.*

- [18] NCSU FreePDK 45nm. URL: <http://www.eda.ncsu.edu/wiki/FreePDK45:Contents>.
- [19] Predictive Technology Model (PTM). URL: <http://ptm.asu.edu/>.
- [20] Remote configuration of IC features via smart contracts. URL: <https://github.com/mni025/remote-configuration-smart-contract>.
- [21] Skylake (client) - Microarchitectures - Intel. URL: [https://en.wikichip.org/wiki/intel/microarchitectures/skylake_\(client\)](https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)).
- [22] Sonics Introduces Semiconductor IP Industry's First Power Management Solution Combining Fine-Grain Partitioning and Autonomous Control, 2015. URL: <https://www.gartner.com/doc/2010815/competitive-landscape-power-management-ic>.
- [23] Target Breach. URL: <http://www.itworldcanada.com/post/hacking-of-hvac-supplier-led-to-target-breach-report>.
- [24] Who is hiding behind the barcode? URL: <https://product.okfn.org/2013/10/16/who-is-hiding-behind-the-barcode/>.
- [25] Top 5 counterfeited semiconductors: Analog ICs top the list. URL: <https://electroiq.com/2012/04/top-5-counterfeited-semiconductors-analog-ics-top-the-list/> (2012).
- [26] What are the parity ethereum disk space needs and overall hardware requirements? URL: <https://github.com/openethereum/wiki/blob/master/FAQ.md> (2019).
- [27] Preserving IoT privacy in Sharing Economy. URL: <https://github.com/mni025/IoT-Privacy-via-Blockchain> (August, 2019).
- [28] Decentralizing Airbnb. URL: <https://www.smarthosts.org/posts/Zr4w8SEK5BH42CPZF/airbnb-blockchain-loyalty-travel> (July, 2017).
- [29] HTTP/1.1: Status Code Definitions. URL: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> (March, 2016).
- [30] Inquiry into counterfeit electronic parts in the Department of Defence supply chain. URL: <https://www.armed-services.senate.gov/imo/media/doc/Counterfeit-Electronic-Parts.pdf>. (May 2012).
- [31] Alkabani, Yousra, and Koushanfar, Farinaz. Active hardware metering for intellectual property protection and security. In *USENIX security symposium* (2007), pp. 291–306.
- [32] Alvarez, Anastacia B, Zhao, Wenfeng, and Alioto, Massimo. Static physically unclonable functions for secure chip identification with 1.9–5.8% native bit instability at 0.6–1 v and 15 fj/bit in 65 nm. *IEEE Journal of Solid-State Circuits* 51, 3 (2016), 763–775.
- [33] Androulaki, Elli, Barger, Artem, Bortnikov, Vita, Cachin, Christian, Christidis, Konstantinos, De Caro, Angelo, Enyeart, David, Ferris, Christopher, Laventman, Genady, Manevich, Yacov, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference* (2018), pp. 1–15.

- [34] Angwin, Julia. Own a Vizio Smart TV? It’s Watching You. *URL: <https://www.propublica.org/article/own-a-vizio-smart-tv-its-watching-you>* (2015).
- [35] Atzei, Nicola, Bartoletti, Massimo, and Cimoli, Tiziana. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust* (2017), Springer, pp. 164–186.
- [36] Augot, Daniel, Chabanne, Hervé, Clémot, Olivier, and George, William. Transforming face-to-face identity proofing into anonymous digital identity using the Bitcoin blockchain. *arXiv preprint arXiv:1710.02951* (2017).
- [37] Back, Adam, Corallo, Matt, Dashjr, Luke, Friedenbach, Mark, Maxwell, Gregory, Miller, Andrew, Poelstra, Andrew, Timón, Jorge, and Wuille, Pieter. Enabling blockchain innovations with pegged sidechains. *URL: <http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains>* (2014).
- [38] Baer, David, Sweet, James D, Chen, Iue-Shuenn, Bowers, Heather, and Beach, Jeffrey. Method and system for using one-time programmable (otp) read-only memory (rom) to configure chip usage features, Dec. 20 2007. US Patent App. 11/535,912.
- [39] Baza, Mohamed, Nabil, Mahmoud, Lasla, Noureddine, Fidan, Kemal, Mahmoud, Mohamed, and Abdallah, Mohamed. Blockchain-based firmware update scheme tailored for autonomous vehicles. *arXiv preprint arXiv:1811.05905* (2018).
- [40] Bhardwaj, Sarvesh, Wang, Wenping, Vattikonda, Rakesh, Cao, Yu, and Vrudhula, Sarma. Predictive modeling of the NBTI effect for reliable design. In *IEEE Custom Integrated Circuits Conference 2006* (2006), IEEE, pp. 189–192.
- [41] Bhargava, Mudit, Cakir, Cagla, and MAI, Khanh. Attack resistant sense amplifier based PUFs (SA-PUF) with deterministic and controllable reliability of PUF responses. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on* (2010), IEEE, pp. 106–111.
- [42] Bhargava, Mudit, and Mai, Ken. A high reliability PUF using hot carrier injection based response reinforcement. In *Cryptographic Hardware and Embedded Systems-CHES 2013*. Springer, 2013, pp. 90–106.
- [43] Bhargava, Mudit, and Mai, Ken. An efficient reliable puf-based cryptographic key generator in 65nm cmos. In *Proceedings of the conference on Design, Automation & Test in Europe* (2014), European Design and Automation Association, p. 70.
- [44] Bhasin, Shivam, Danger, Jean-Luc, Guilley, Sylvain, Ngo, Xuan Thuy, and Sauvage, Laurent. Hardware trojan horses in cryptographic ip cores. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on* (2013), IEEE, pp. 15–29.
- [45] Bhunia, Swarup, Hsiao, Michael S, Banga, Mainak, and Narasimhan, Seetharam. Hardware trojan attacks: threat analysis and countermeasures. *Proceedings of the IEEE 102*, 8 (2014), 1229–1247.
- [46] Bissias, George, Levine, Brian, and Kapadia, Nikunj. Securing the assets of decentralized applications using financial derivatives (draft). *arXiv preprint arXiv:1701.03945* (2017).

- [47] Bissias, George, and Levine, Brian N. Bobtail: Improved blockchain security with low-variance mining. In *ISOC Network and Distributed System Security Symposium* (2020).
- [48] Bissias, George, Levine, Brian Neil, Ozisik, A Pinar, and Andresen, Gavin. An analysis of attacks on blockchain consensus. *arXiv preprint arXiv:1610.07985* (2016).
- [49] Blömer, Johannes, and Seifert, Jean-Pierre. Fault based cryptanalysis of the advanced encryption standard (aes). In *Computer Aided Verification* (2003), Springer, pp. 162–181.
- [50] Bösch, Christoph. Efficient Fuzzy Extractors for Reconfigurable Hardware. Master’s thesis, Dept. EECS, Massachusetts Institute of Technology, 2008.
- [51] Boudguiga, Aymen, Bouzerna, Nabil, Granboulan, Louis, Olivereau, Alexis, Quesnel, Flavien, Roger, Anthony, and Sirdey, Renaud. Towards better availability and accountability for iot updates by means of a blockchain. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (2017), IEEE, pp. 50–58.
- [52] Bucci, M., and Luzzi, R. Identification circuit and method for generating an identification bit using physical unclonable functions, Nov. 12 2013. US Patent 8,583,710.
- [53] Burton, Edward A, Schrom, Gerhard, Paillet, Fabrice, Douglas, Jonathan, Lambert, William J, Radhakrishnan, Kaladhar, and Hill, Michael J. Fivr—fully integrated voltage regulators on 4th generation intel® core™ socs. In *Applied Power Electronics Conference and Exposition (APEC), 2014 Twenty-Ninth Annual IEEE* (2014), IEEE, pp. 432–439.
- [54] Buterin, Vitalik. On Public and Private Blockchains. URL: <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/> (August 7, 2015).
- [55] Buterin, Vitalik, et al. A next-generation smart contract and decentralized application platform. *white paper* (2014).
- [56] Castro, Miguel, Liskov, Barbara, et al. Practical byzantine fault tolerance. In *OSDI* (1999), vol. 99, pp. 173–186.
- [57] Chakraborty, Rajat Subhra, Wolff, Francis G, Paul, Somnath, Papachristou, Christos A, and Bhunia, Swarup. Mero: A statistical approach for hardware trojan detection. In *CHES* (2009), vol. 5747, Springer, pp. 396–410.
- [58] Che, Wenjie, Plusquellic, Jim, and Bhunia, Swarup. A non-volatile memory based physically unclonable function without helper data. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design* (2014), IEEE Press, pp. 148–153.
- [59] Christidis, Konstantinos, and Devetsikiotis, Michael. Blockchains and smart contracts for the internet of things. *Ieee Access* 4 (2016), 2292–2303.
- [60] Cortez, M., Hamdioui, S., van der Leest, V., Maes, R., and Schrijen, G.-J. Adapting voltage ramp-up time for temperature noise reduction on memory-based PUFs. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on* (June 2013), pp. 35–40.

- [61] Dakroub, Alain. Understanding the iot platform for service providers. URL: <https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2016/pdf/BRKSPG-2067.pdf> (2018).
- [62] DARPA. Supply chain hardware basic concepts and taxonomy of dependable and secure computing. *Microsystems Technology Office/MTO Broad Agency Announcement* (2014).
- [63] Delmolino, Kevin, Arnett, Mitchell, Kosba, Ahmed, Miller, Andrew, and Shi, Elaine. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International Conference on Financial Cryptography and Data Security* (2016), Springer, pp. 79–94.
- [64] Delvaux, Jeroen, Gu, Dawu, Schellekens, Dries, and Verbauwheide, Ingrid. Helper data algorithms for puf-based key generation: Overview and analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 6 (2015), 889–902.
- [65] Devadas, Srinivas, Suh, Edward, Paral, Sid, Sowell, Richard, Ziola, Tom, and Khandelwal, Vivek. Design and implementation of PUF-based "Unclonable" RFID ICs for anti-counterfeiting and security applications. In *RFID, 2008 IEEE International conference on* (2008), IEEE, pp. 58–64.
- [66] Deyati, Sabyasachi, Chatterjee, Abhijit, and Muldrey, Barry John. Analog push pull amplifier-based physically unclonable function for hardware security, May 4 2017. US Patent App. 15/336,895.
- [67] Diffie, Whitfield, Van Oorschot, Paul C, and Wiener, Michael J. Authentication and authenticated key exchanges. *Designs, Codes and cryptography* 2, 2 (1992), 107–125.
- [68] Dingee, Don. Customized PMICs with OTP in automotive and IoT. URL: <https://www.semiwiki.com/forum/content/6122-customized-pmics-otp-automotive-iot.html> (August, 2016).
- [69] Dodis, Yevgeniy, Ostrovsky, Rafail, Reyzin, Leonid, and Smith, Adam. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing* 38, 1 (2008), 97–139.
- [70] Dodis, Yevgeniy, Ostrovsky, Rafail, Reyzin, Leonid, and Smith, Adam. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM J. Comput.* 38, 1 (Mar. 2008), 97–139.
- [71] Dollfus, Philippe, Bournel, Arnaud, Galdin, Sylvie, Barraud, Sylvain, and Hesto, Patrice. Effect of discrete impurities on electron transport in ultrashort MOSFET using 3D MC simulation. *IEEE Transactions on Electron Devices* 51, 5 (2004), 749–756.
- [72] Dorri, Ali, Kanhere, Salil S, and Jurdak, Raja. Towards an optimized blockchain for iot. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation* (2017), ACM, pp. 173–178.
- [73] Downum, Steven, Phillips, Walter, and Samuel, Balasingh. Firmware update control mechanism using organizational groups, Nov. 6 2018. US Patent App. 15/352,356.

- [74] Dusart, Pierre, Letourneux, Gilles, and Vivolo, Olivier. Differential fault analysis on aes. In *Applied Cryptography and Network Security* (2003), Springer, pp. 293–306.
- [75] Elkhyaoui, Kaoutar, Blass, Erik-Oliver, and Molva, Refik. CHECKER: On-site checking in RFID-based supply chains. In *Proceedings of the fifth ACM conference on security and privacy in wireless and mobile networks* (2012), ACM, pp. 173–184.
- [76] Ethereum. Whisper message for smart contracts. URL: <https://github.com/ethereum/wiki/wiki/Whisper>.
- [77] Eyal, Ittay, Gencer, Adem Efe, Sirer, Emin Gün, and Van Renesse, Robbert. Bitcoinng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)* (2016), pp. 45–59.
- [78] Faulkner, Andrew, et al. Enabling secure semiconductor supply chain management. URL: <https://www.chipestimate.com/Enabling-Secure-Semiconductor-Supply-Chain-Management/Sidense-a-part-of-Synopsys/Technical-Article/2017/09/05> (February, 2017).
- [79] Fay, Thomas, and Paniscotti, Dominick. Systems and methods of blockchain transaction recordation, Oct. 6 2016. US Patent App. 15/086,801.
- [80] Fern, Nicole, San, Ismail, and Cheng, Kwang-Ting Tim. Detecting hardware trojans in unspecified functionality through solving satisfiability problems. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific* (2017), IEEE, pp. 598–504.
- [81] Frank, Michael. Voltage detection, Nov. 18 2014. US Patent 8,892,922.
- [82] Gallagher, S. Internet of \$@!%: Google api change triggers epson printer revolt. URL: <https://arstechnica.com/information-technology/2016/12/internet-of-google-api-change-triggers-epson-printer-revolt/> (2016).
- [83] Ganta, D., and Nazhandali, L. Circuit-level approach to improve the temperature reliability of Bi-stable PUFs. In *Quality Electronic Design (ISQED), 2014 15th International Symposium on* (March 2014), pp. 467–472.
- [84] Garg, A., and Kim, T.T. Design of SRAM PUF with improved uniformity and reliability utilizing device aging effect. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on* (June 2014), pp. 1941–1944.
- [85] Giraud, Christophe. Dfa on aes. *Advanced Encryption Standard–AES* (2005), 571–571.
- [86] Goldfeder, S., et al. Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin. In *International Conference on Financial Cryptography and Data Security* (2017), Springer, pp. 321–339.
- [87] Goo, Jung-Suk, Choi, Chang-Hoon, Abramo, A., Ahn, Jae-Gyung, Yu, Zhiping, Lee, T. H., and Dutton, R. W. Physical origin of the excess thermal noise in short channel MOSFETs. *IEEE Electron Device Letters* 22, 2 (Feb 2001), 101–103.
- [88] Guajardo, Jorge, Kumar, Sandeep S, Schrijen, Geert-Jan, and Tuyls, Pim. Physical unclonable functions and public-key crypto for fpga ip protection. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on* (2007), IEEE, pp. 189–195.

- [89] Guardtime, and IntrinsicID. Internet of things authentication: A blockchain solution using sram physical unclonable functions. URL: https://www.intrinsic-id.com/wp-content/uploads/2017/05/gt_KSI-PUF-web-1611.pdf.
- [90] Guin, Ujjwal, Huang, Ke, DiMase, Daniel, Carulli, John M, Tehranipoor, Mohammad, and Makris, Yiorgos. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proceedings of the IEEE* 102, 8 (2014), 1207–1228.
- [91] Handschuh, Hélène, and Tuyls, Pim Theo. Device and method for obtaining a cryptographic key, Jan. 19 2011. US Patent App. 13/574,311.
- [92] Hartzog, Woodrow, and Selinger, Evan. The internet of heirlooms and disposable things. *NCJL & Tech.* 17 (2015), 581.
- [93] Helfmeier, Clemens, Boit, Christian, Nedospasov, Dmitry, and Seifert, Jean-Pierre. Cloning physically unclonable functions. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on* (2013), IEEE, pp. 1–6.
- [94] Helo, Petri, and Szekely, Bulcsu. Logistics information systems: an analysis of software solutions for supply chain co-ordination. *Industrial Management & Data Systems* 105, 1 (2005), 5–18.
- [95] Hern, Alex. Revolv devices bricked as google’s nest shuts down smart home company. URL: <https://www.theguardian.com/technology/2016/apr/05/revolv-devices-bricked-google-nest-smart-home> (2016).
- [96] Hiller, Matthias, Kürzinger, Ludwig, and Sigl, Georg. Review of error correction for pufs and evaluation on state-of-the-art fpgas. *Journal of Cryptographic Engineering* (2020).
- [97] Hofer, Maximilian, and Boehm, Christoph. An Alternative to Error Correction for SRAM-like PUFs. In *Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems* (Berlin, Heidelberg, 2010), CHES’10, Springer-Verlag, pp. 335–350.
- [98] Holcomb, Daniel E, Burleson, Wayne P, and Fu, Kevin. Power-up sram state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers* 58, 9 (2009), 1198–1210.
- [99] Horvath, Bryan T. Not all parts are created equal: The impact of counterfeit parts in the air force supply chain. Tech. rep., Air War College, Air University Maxwell AFB United States, 2017.
- [100] Huh, Seyoung, Cho, Sangrae, and Kim, Soohyung. Managing iot devices using blockchain platform. In *Advanced Communication Technology (ICACT), 2017 19th International Conference on* (2017), IEEE, pp. 464–467.
- [101] Initiative, Silicon Integration, et al. Nangate open cell library. Available: <http://www.si2.org/openeda.si2.org/projects/nangatelib> (2011).
- [102] Islam, M. N., and Kundu, S. Modeling residual life of an ic considering multiple aging mechanisms. In *2016 IEEE 25th North Atlantic Test Workshop (NATW)* (May 2016), pp. 24–27.

- [103] Islam, Md Nazmul, and Kundu, Sandip. Modeling residual lifetime of an ic considering spatial and inter-temporal temperature variations. In *Asian Test Symposium (ATS), 2016 IEEE 25th* (2016), IEEE, pp. 240–245.
- [104] Islam, Md Nazmul, and Kundu, Sandip. An analytical model for predicting the residual life of an ic and design of residual-life meter. In *VLSI Test Symposium (VTS), 2017 IEEE 35th* (2017), IEEE, pp. 1–6.
- [105] Islam, Md Nazmul, and Kundu, Sandip. Pmu-trojan: on exploiting power management side channel for information leakage. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference* (2018), IEEE Press, pp. 709–714.
- [106] Islam, Md Nazmul, and Kundu, Sandip. Preserving iot privacy in sharing economy via smart contract. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)* (2018), IEEE, pp. 296–297.
- [107] Islam, Md Nazmul, and Kundu, Sandip. Enabling ic traceability via blockchain pegged to embedded puf. *ACM Trans. Des. Autom. Electron. Syst.* **24**, 3 (Apr. 2019), 36:1–36:23.
- [108] Islam, Md Nazmul, and Kundu, Sandip. Remote configuration of integrated circuit features and firmware management via smart contract. In *2019 IEEE International Conference on Blockchain (Blockchain)* (2019), IEEE, pp. 325–331.
- [109] Islam, Md Nazmul, and Kundu, Sandip. *IoT Security, Privacy and Trust in Home-Sharing Economy via Blockchain*. Springer International Publishing, Cham, 2020, pp. 33–50.
- [110] Islam, Md Nazmul, Patil, Vinay C, and Kundu, Sandip. On ic traceability via blockchain. In *VLSI Design, Automation and Test (VLSI-DAT), 2018 International Symposium on* (2018), IEEE, pp. 1–4.
- [111] Islam, Md Nazmul, Patil, Vinay C, and Kundu, Sandip. Determining proximal geolocation of iot edge devices via covert channel. In *Quality Electronic Design (ISQED), 2017 18th International Symposium on* (2017), IEEE, pp. 196–202.
- [112] Islam, Md Nazmul, Patil, Vinay C, and Kundu, Sandip. A guide to graceful aging: How not to overindulge in post-silicon burn-in for enhancing reliability of weak puf. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on* (2017), IEEE, pp. 1–4.
- [113] Islam, Md Nazmul, Patil, Vinay C, and Kundu, Sandip. On enhancing reliability of weak pufs via intelligent post-silicon accelerated aging. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2017).
- [114] ISO. EN ISO 8492.1995. *European Committee for Standardization, Point 3.16*. (1995).
- [115] Iyer, Kedar, and Dannen, Chris. The ethereum development environment. In *Building Games with Ethereum Smart Contracts*. Springer, 2018, pp. 19–36.
- [116] Jang, J. W., and Ghosh, S. Design and analysis of novel SRAM PUFs with embedded latch for robustness. In *Sixteenth International Symposium on Quality Electronic Design* (March 2015), pp. 298–302.

- [117] Jansen-Vullers, Monique H, van Dorp, Christian A, and Beulens, Adrie JM. Managing traceability information in manufacture. *International journal of information management* 23, 5 (2003), 395–413.
- [118] Jin, Chenglu, and van Dijk, Marten. Secure and efficient initialization and authentication protocols for shield. *IEEE Transactions on Dependable and Secure Computing* (2017).
- [119] Jin, Yier, and Makris, Yiorgos. Proof carrying-based information flow tracking for data secrecy protection and hardware trust. In *VLSI Test Symposium (VTS), 2012 IEEE 30th* (2012), IEEE, pp. 252–257.
- [120] Johnson, J. B. Thermal Agitation of Electricity in Conductors. *Phys. Rev.* 32 (Jul 1928), 97–109.
- [121] Kang, Kunhyuk, Park, Sang Phill, Roy, Kaushik, and Alam, Muhammad A. Estimation of statistical variation in temporal nbtj degradation and its impact on lifetime circuit performance. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design* (2007), IEEE Press, pp. 730–734.
- [122] Kar, I. Estonian citizens will soon have the world’s most hack-proof health-care records, 2016.
- [123] Kelly, J, and Williams, A. Forty big banks test blockchain-based bond trading system, 2016.
- [124] Kessler, L. W., and Sharpe, T. Faked parts detection. URL: <http://www.circuitsassembly.com/cms/component/content/article/159/9937-smt>. (2010).
- [125] Kocher, Paul, Jaffe, Joshua, and Jun, Benjamin. Differential power analysis. In *Advances in cryptology—CRYPTO’99* (1999), Springer, pp. 789–789.
- [126] Koh, Robin, Schuster, Edmund W, Chackrabarti, Indy, and Bellman, Attilio. Securing the pharmaceutical supply chain. *White Paper, Auto-ID Labs, Massachusetts Institute of Technology* (2003), 1–19.
- [127] Kosba, Ahmed, Miller, Andrew, Shi, Elaine, Wen, Zikai, and Papamanthou, Charalampos. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)* (2016), IEEE, pp. 839–858.
- [128] Kuemin, Cyrill, Nowack, Lea, Bozano, Luisa, Spencer, Nicholas D, and Wolf, Heiko. Oriented assembly of gold nanorods on the single-particle level. *Advanced Functional Materials* 22, 4 (2012), 702–708.
- [129] Kumaki, Takeshi, Yoshikawa, Masaya, and Fujino, Takeshi. Cipher-destroying and secret-key-emitting hardware trojan against aes core. In *Circuits and Systems (MWSCAS), 2013 IEEE 56th International Midwest Symposium on* (2013), IEEE, pp. 408–411.
- [130] Kumar, Sanjay V, Kim, Chris H, and Sapatnekar, Sachin S. An analytical model for negative bias temperature instability. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design* (2006), ACM, pp. 493–496.

- [131] Lacey, Stephen. The energy blockchain: How bitcoin could be a catalyst for the distributed grid. *GreenTech Media* 26 (2016).
- [132] LAM, Suk Wah Louisa. Theory and application of majority vote: From condorcet jury theorem to pattern recognition. *2nd Int. Conf. mathematics education into the 21st century: Mathematics for Living* (2000).
- [133] Lamport, Leslie. Generalized consensus and paxos.
- [134] Le Sueur, Etienne, and Heiser, Gernot. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems* (2010), pp. 1–8.
- [135] Lee, Boohyung, and Lee, Jong-Hyoun. Blockchain-based secure firmware update for embedded devices in an internet of things environment. *The Journal of Supercomputing* 73, 3 (2017), 1152–1167.
- [136] Lee, Jae W, Lim, Daihyun, Gassend, Blaise, Suh, G Edward, Van Dijk, Marten, and Devadas, Srinivas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on* (2004), IEEE, pp. 176–179.
- [137] Li, He, Liu, Qiang, and Zhang, Jiliang. A survey of hardware trojan threat and defense. *Integration, the VLSI Journal* 55 (2016), 426–437.
- [138] Liu, Yu, Jin, Yier, and Makris, Yiorgos. Hardware trojans in wireless cryptographic ics: silicon demonstration & detection method evaluation. In *Proceedings of the International Conference on Computer-Aided Design* (2013), IEEE Press, pp. 399–404.
- [139] Liu, Yuan, Zhao, Zheng, Guo, Guibing, Wang, Xingwei, Tan, Zhenhua, and Wang, Shuang. An Identity Management System Based on Blockchain. *Conference on Privacy, Security and Trust (PST)* (Aug 2017).
- [140] Luu, Loi, Narayanan, Viswesh, Zheng, Chaodong, Baweja, Kunal, Gilbert, Seth, and Saxena, Prateek. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 17–30.
- [141] Maes, Roel. An Accurate Probabilistic Reliability Model for Silicon PUFs. In *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems* (2013), CHES’13, Springer-Verlag, pp. 73–89.
- [142] Maes, Roel, Rozic, Vladimir, Verbauwhede, Ingrid, Koeberl, Patrick, Van der Sluis, Erik, and van der Leest, Vincent. Experimental evaluation of Physically Unclonable Functions in 65 nm CMOS. In *ESSCIRC (ESSCIRC), 2012 Proceedings of the* (2012), IEEE, pp. 486–489.
- [143] Maes, Roel, Tuyls, Pim, and Verbauwhede, Ingrid. A soft decision helper data algorithm for SRAM PUFs. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on* (2009), IEEE, pp. 2101–2105.

- [144] Maes, Roel, Tuyls, Pim, and Verbauwhede, Ingrid. Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs. In *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 332–347.
- [145] Maes, Roel, and van der Leest, Vincent. Countering the effects of silicon aging on SRAM PUFs. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on* (2014), IEEE, pp. 148–153.
- [146] Maes, Roel, Van Herrewege, Anthony, and Verbauwhede, Ingrid. Pufky: A fully functional puf-based cryptographic key generator. In *Cryptographic Hardware and Embedded Systems-CHES 2012*. Springer, 2012, pp. 302–319.
- [147] Mandelblat, Julius. Technology insight: Intel’s next generation microarchitecture code name skylake. In *Intel Developer Forum, San Francisco* (2015).
- [148] Mathew, Sanu K, Satpathy, Sudhir K, Anders, Mark A, Kaul, Himanshu, Hsu, Steven K, Agarwal, Amit, Chen, Gregory K, Parker, Rachael J, Krishnamurthy, Ram K, and De, Vivek. 16.2 A 0.19 pJ/b PVT-variation-tolerant hybrid physically unclonable function circuit for 100% stable secure key generation in 22nm CMOS. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (2014), IEEE, pp. 278–279.
- [149] Matsukawa, T., Liu, Y., Endo, K., i. O’uchi, S., and Masahara, M. Variability origins of FinFETs and perspective beyond 20nm node. In *IEEE 2011 International SOI Conference* (Oct 2011), pp. 1–28.
- [150] Maxwell, Greg. Confidential transactions. URL: <https://people.xiph.org/greg/confidentialvalues.txt> (Accessed 09/05/2016) (2015).
- [151] Mayer, Hartwig. Ecdsa security in bitcoin and ethereum: a research survey. *Coin-Fabrik, June 28* (2016), 126.
- [152] Mazieres, David. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation 32* (2015).
- [153] Meza, Summer. Airbnb hosts are recording their guests with hidden cameras. URL: <http://www.newsweek.com/airbnb-hidden-cameras-recording-guests-739709> (December, 2017).
- [154] Miller, Mitchell, Meraglia, Janice, and Hayward, James. Traceability in the age of globalization: a proposal for a marking protocol to assure authenticity of electronic parts. Tech. rep., SAE Technical Paper, 2012.
- [155] Mittendorf, Christoph. What trust means in the sharing economy: A provider perspective on airbnb. com.
- [156] Mizrahi, Alex. A blockchain-based property ownership recording system. *A Blockchain-based Property Ownership Recording System* (2015).
- [157] Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system, 2008.
- [158] New, Steve. The transparent supply chain. *Harvard Business Review 88* (2010), 1–5.

- [159] Ninlawan, C, Seksan, P, Tossapol, K, and Pilada, W. The implementation of green supply chain management practices in electronics industry. In *Proceedings of the international multiconference of engineers and computer scientists* (2010), vol. 3, pp. 17–19.
- [160] Nyquist, H. Thermal Agitation of Electric Charge in Conductors. *Phys. Rev.* 32 (Jul 1928), 110–113.
- [161] Oltermann, Philip. German parents told to destroy doll that can spy on children. URL: <https://www.theguardian.com/world/2017/feb/17/german-parents-told-to-destroy-my-friend-cayla-doll-spy-on-children> (2017).
- [162] Ongaro, Diego, and Ousterhout, John K. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference* (2014), pp. 305–319.
- [163] Ozisik, A Pinar, Andresen, Gavin, Bissias, GD, Houmansadr, Amir, and Levine, Brian Neil. A secure, efficient, and transparent network architecture for bitcoin. *UMass Amherst, Tech. Rep. UM-CS-2016-006* (2016).
- [164] Ozisik, A Pinar, and Levine, Brian Neil. An explanation of nakamoto’s analysis of double-spend attacks. *arXiv preprint arXiv:1701.03977* (2017).
- [165] O’Brien, SA. Giant equifax data breach: 143 million people could be affected. URL: <https://money.cnn.com/2017/09/07/technology-/business/equifax-data-breach/> 8 (2017).
- [166] Patil, Vinay C., Vijayakumar, Arunkumar, Holcomb, Daniel E., and Kundu, Sandip. Improving Reliability of Weak PUFs via Circuit Techniques to Enhance Mismatch. In *Hardware-Oriented Security and Trust (HOST), 2017 IEEE International Symposium on* (May 2017), IEEE, p. to be published.
- [167] Paul, Bipul C, Kang, Kunhyuk, Kufluoglu, Haldun, Alam, Muhammad A, and Roy, Kaushik. Impact of nbt on the temporal performance degradation of digital circuits. *IEEE Electron Device Letters* 26, 8 (2005), 560–562.
- [168] Pecht, Michael, and Tiku, Sanjay. Bogus: electronic manufacturing and consumers confront a rising tide of counterfeit electronics. *IEEE spectrum* 43, 5 (2006), 37–46.
- [169] Pille, J., Adams, C., Christensen, T., Cottier, S. R., Ehrenreich, S., Kono, F., Nelson, D., Takahashi, O., Tokito, S., Torreiter, O., Wagner, O., and Wendel, D. Implementation of the Cell Broadband Engine™; in 65 nm SOI Technology Featuring Dual Power Supply SRAM Arrays Supporting 6 GHz at 1.3 V. *IEEE Journal of Solid-State Circuits* 43, 1 (Jan 2008), 163–171.
- [170] Piret, Gilles, and Quisquater, Jean-Jacques. A differential fault attack technique against spn structures, with application to the aes and khazad. In *CHES* (2003), vol. 2779, Springer, pp. 77–88.
- [171] Pouraghily, Arman, Islam, Md Nazmul, Kundu, Sandip, and Wolf, Tilman. Privacy in blockchain-enabled iot devices. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)* (2018), IEEE, pp. 292–293.

- [172] Quisquater, Jean-Jacques, and Samyde, David. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. *Smart Card Programming and Security* (2001), 200–210.
- [173] Rahman, Md Tauhidur, Forte, Domenic, Shi, Quihang, Contreras, Gustavo K, and Tehranipoor, Mohammad. Csst: Preventing distribution of unlicensed and rejected ics by untrusted foundry and assembly. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014 IEEE International Symposium on* (2014), IEEE, pp. 46–51.
- [174] Rajendran, Jeyavijayan, Dhandayuthapany, A, Karri, Ramesh, and Vedula, V. Security verification of 3rd party intellectual property cores for information leakage. In *Proceedings of IEEE VLSI Design* (2016).
- [175] Ramesh, P., Patil, V. C., and Kundu, S. Peer pressure on identity: On requirements for disambiguating PUFs in noisy environment. In *2017 IEEE North Atlantic Test Workshop (NATW)* (May 2017), pp. 1–4.
- [176] Ranasinghe, Damith, Engels, Daniel, Cole, Peter, et al. Security and privacy: Modest proposals for low-cost RFID systems. In *Auto-ID Labs Research Workshop, Zurich, Switzerland* (2004).
- [177] Risch, L. Pushing CMOS beyond the roadmap. *Solid-State Electronics* 50, 4 (2006), 527–535.
- [178] Rizzo, Pete. Blockchain identity startup shocard raises 1.5 million. URL: <http://www.coindesk.com/blockchain-identity-startup-shocard-1-5-million/> (July, 2015).
- [179] Robson, Norm, Safran, John, Kothandaraman, Chandrasekharan, Cestero, Alberto, Chen, Xiang, Rajeevakumar, Raj, Leslie, Alan, Moy, Dan, Kirihata, Toshiaki, and Iyer, Subramanian. Electrically programmable fuse (efuse): From memory redundancy to autonomic chips. In *Custom Integrated Circuits Conference, 2007. CICC'07. IEEE* (2007), IEEE, pp. 799–804.
- [180] Roy, Gareth, Brown, Andrew R, Adamu-Lema, Fikru, Roy, Scott, and Asenov, Asen. Simulation study of individual and combined sources of intrinsic parameter fluctuations in conventional nano-MOSFETs. *IEEE Transactions on Electron Devices* 53, 12 (2006), 3063–3070.
- [181] Roy, Jarrod A, Koushanfar, Farinaz, and Markov, Igor L. Ending piracy of integrated circuits. *Computer* 43, 10 (2010), 30–38.
- [182] Rührmair, Ulrich, and Holcomb, Daniel E. Pufs at a glance. In *Proceedings of the conference on Design, Automation & Test in Europe* (2014), European Design and Automation Association, p. 347.
- [183] Rührmair, Ulrich, Sehnke, Frank, Sölter, Jan, Dror, Gideon, Devadas, Srinivas, and Schmidhuber, Jürgen. Modeling Attacks on Physical Unclonable Functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 237–249.

- [184] Rührmair, Ulrich, Sölter, Jan, Sehnke, Frank, Xu, Xiaolin, Mahmoud, Ahmed, Stoyanova, Vera, Dror, Gideon, Schmidhuber, Jürgen, Burleson, Wayne, and Devadas, Srinivas. Puf modeling attacks on simulated and silicon data. *IEEE Transactions on Information Forensics and Security* 8, 11 (2013), 1876–1891.
- [185] S, Nash, Kim. Wal-mart readies blockchain pilot for tracking u.s produce, china pork. URL: <https://blogs.wsj.com/cio/2016/12/16/wal-mart-readies-blockchain-pilot-for-tracking-u-s-produce-china-pork/>.
- [186] Sagstetter, Florian, Lukasiwycz, Martin, Steinhorst, Sebastian, Wolf, Marko, Bouard, Alexandre, Harris, William R, Jha, Somesh, Peyrin, Thomas, Poschmann, Axel, and Chakraborty, Samarjit. Security challenges in automotive hardware/software architecture design. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2013), EDA Consortium, pp. 458–463.
- [187] Samsung. Samsung smart home hub. URL: <https://www.samsung.com/us/smart-home/how-it-works/>.
- [188] Sarpeshkar, Rahul, Delbruck, Tobias, and Mead, Carver A. White Noise in MOS Transistors and Resistors. *IEEE Circuits and Devices* 9, 6 (1993), 23–29.
- [189] Sasson, Eli Ben, Chiesa, Alessandro, Garman, Christina, Green, Matthew, Miers, Ian, Tromer, Eran, and Virza, Madars. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy* (2014), IEEE, pp. 459–474.
- [190] Shi, Jie, Kywe, Su Mon, and Li, Yingjiu. Batch Clone Detection in RFID-enabled supply chain. In *RFID (IEEE RFID), 2014 IEEE International Conference on* (2014), IEEE, pp. 118–125.
- [191] Shirriff, Ken. Bitcoins the hard way: Using the raw Bitcoin protocol. URL: <http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html> (2014).
- [192] Skorobogatov, Sergei, and Woods, Christopher. In the blink of an eye: There goes your aes key. *IACR Cryptology ePrint Archive 2012* (2012), 296.
- [193] Skudlarek, Joseph P, Katsioulas, Tom, and Chen, Michael. A platform solution for secure supply-chain and chip life-cycle management. *Computer* 49, 8 (2016), 28–34.
- [194] Sperling, ED. How much will that chip cost? URL: <https://semiengineering.com/how-much-will-that-chip-cost/> (March, 2014).
- [195] Spiridon, Constantin, Popescu-Stanesti, Vlad Mihail, Shyr, You-Yuh, Hartular, Alexandru, and Densham III, William L. Single chip power management unit apparatus and method, Jan. 14 2003. US Patent 6,507,173.
- [196] Srinivasan, Sudarshan, Kurella, Nithesh, Koren, Israel, and Kundu, Sandip. Dynamic reconfiguration vs. dvfs: A comparative study on power efficiency of processors. In *VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), 2016 29th International Conference on* (2016), IEEE, pp. 563–564.
- [197] Staake, Thorsten, Thiesse, Frédéric, and Fleisch, Elgar. Extending the EPC network: the potential of RFID in anti-counterfeiting. In *Proceedings of the 2005 ACM symposium on Applied computing* (2005), ACM, pp. 1607–1612.

- [198] Stolk, P. A., Widdershoven, F. P., and Klaassen, D. B. M. Modeling statistical dopant fluctuations in MOS transistors. *IEEE Transactions on Electron Devices* 45, 9 (Sep 1998), 1960–1971.
- [199] Su, Ying, Holleman, Jeremy, and Otis, Brian P. A digital 1.6 pj/bit chip identification circuit using process variations. *IEEE Journal of Solid-State Circuits* 43, 1 (2008), 69–77.
- [200] Suh, G Edward, and Devadas, Srinivas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual design automation conference* (2007), ACM, pp. 9–14.
- [201] Sun, Daying, Xu, Shen, Sun, Weifeng, Lu, Shengli, and Shi, Longxing. Low power design for soc with power management unit. In *ASIC (ASICON), 2011 IEEE 9th International Conference on* (2011), IEEE, pp. 719–722.
- [202] Swan, Melanie. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.
- [203] Szabo, Nick. Formalizing and securing relationships on public networks. *First Monday* 2, 9 (1997).
- [204] Tapscott, Don, and Tapscott, Alex. *Blockchain Revolution: How the technology behind Bitcoin is changing money, business, and the world*. Penguin, 2016.
- [205] Triantis, Dimitris P, Birbas, Alexios N, and Kondis, D. Thermal noise modeling for short-channel MOSFETs. *IEEE Transactions on Electron Devices* 43, 11 (1996), 1950–1955.
- [206] Tschorsch, Florian, and Scheuermann, Björn. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 2084–2123.
- [207] Tsutsui, Gen, Saitoh, Masumi, Nagumo, Toshiharu, and Hiramoto, Toshiro. Impact of SOI thickness fluctuation on threshold voltage variation in ultra-thin body SOI MOSFETs. *IEEE Transactions on nanotechnology* 4, 3 (2005), 369–373.
- [208] Tuyls, Pim, and Batina, Lejla. RFID-tags for anti-counterfeiting. In *Cryptographers' Track at the RSA Conference* (2006), Springer, pp. 115–131.
- [209] Vassighi, Arman, and Sachdev, Manoj. Thermal runaway in integrated circuits. *IEEE Transactions on Device and Materials Reliability* 6, 2 (2006), 300–305.
- [210] Vatinel, Christophe Pierre, and Loisel, Jerome Henri. Changing an operating performance point, Nov. 11 2014. US Patent 8,885,694.
- [211] Vattikonda, Rakesh, Wang, Wenping, and Cao, Yu. Modeling and minimization of pmos nbti effect for robust nanometer design. In *Proceedings of the 43rd annual Design Automation Conference* (2006), ACM, pp. 1047–1052.
- [212] Vijayakumar, A., Patil, V. C., Prado, C. B., and Kundu, S. Machine learning resistant strong PUF: Possible or a pipe dream? In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (May 2016), pp. 19–24.

- [213] Vijayakumar, Arunkumar, Patil, Vinay, and Kundu, Sandip. On Improving Reliability of SRAM-Based Physically Unclonable Functions. *Journal of Low Power Electronics and Applications* 7, 1 (Jan 2017), 2.
- [214] Waksman, Adam, Suozzo, Matthew, and Sethumadhavan, Simha. Fanci: identification of stealthy malicious logic using boolean functional analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 697–708.
- [215] Walport, MGCSA. Distributed ledger technology: Beyond blockchain. *UK Government Office for Science* (2016).
- [216] Wang, Yao, Cotofana, Sorin D, and Fang, Liang. Statistical reliability analysis of NBTI impact on FinFET SRAMs and mitigation technique using independent-gate devices. In *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on* (2012), IEEE, pp. 109–115.
- [217] Wasicek, Armin. Protection of intellectual property rights in automotive control units. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 7, 2014-01-0338 (2014), 201–212.
- [218] Wolf, Tilman, Zink, Michael, and Nagurney, Anna. The cyber-physical marketplace: A framework for large-scale horizontal integration in distributed cyber-physical systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops* (2013), IEEE, pp. 296–302.
- [219] Wu, Debby. Engineers found guilty of stealing micron secrets for china. URL: <https://news.bloomberglaw.com/ip-law/chip-engineers-found-guilty-of-stealing-micron-secrets-for-china> (2020).
- [220] Xiao, Kan, Rahman, M.T., Forte, D., Huang, Yu, Su, Mei, and Tehranipoor, M. Bit selection algorithm suitable for high-volume production of SRAM-PUF. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on* (May 2014), pp. 101–106.
- [221] Xie, Yang, and Srivastava, Ankur. Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction. In *Proceedings of the 54th Annual Design Automation Conference 2017* (2017), ACM, p. 9.
- [222] Yang, Kun, Forte, Domenic, and Tehranipoor, Mark. An rfid-based technology for electronic component and system counterfeit detection and traceability. In *Technologies for Homeland Security (HST), 2015 IEEE International Symposium on* (2015), IEEE, pp. 1–6.
- [223] Yang, Kun, Forte, Domenic, and Tehranipoor, Mark. Resc: An rfid-enabled solution for defending iot supply chain. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23, 3 (2018), 29.
- [224] Yu, Meng-Day Mandel, and Devadas, Srinivas. Pervasive, dynamic authentication of physical items. *Communications of the ACM* 60, 4 (2017), 32–39.

- [225] Zhang, Fan, Cecchetti, Ethan, Croman, Kyle, Juels, Ari, and Shi, Elaine. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (2016), pp. 270–282.