Remote Configuration of Integrated Circuit Features and Firmware Management via Smart Contract

Md Nazmul Islam, Sandip Kundu
University of Massachusetts Amherst, MA, USA
{mislam, kundu}@ecs.umass.edu

Abstract—**The cost to develop a new integrated circuit (IC), its fabrication, debug and volume production has been escalating with scaling of transistor feature size. According to an IBS report, the cost of developing a System on Chip (SoC) at $14nm$ may be as high as $300 million [1]. The economics of semiconductor IC development favors high volume production, while high volume cannot be attained without developing an IC that serves a large number of applications. Some of these applications are in low-margin Internet of Things (IoT) devices, where an SoC cannot command a high price. Consequently, without the ability to customize IC features after production, the price of an IC will be determined by its lowest priced application. This motivates the manufacturers to develop capabilities for post-production IC customization. The commodity microprocessor business offers an example of post-production customization, where a manufacturer can tailor cache size, number of cores and frequency of operation for a target market segment after a chip has been manufactured. Today, such customization is limited to one-time programming (OTP) for predetermined IC bins. In this paper, we explore how an IC can be programmed repeatedly and securely using a blockchain-based smart contract. This will enable users to upgrade IC features, or rent upgraded IC features for a fixed period after it has been purchased. The availability of such a system could, for example, allow a buyer to upgrade her processor from i3 to i5 after it has been purchased to scale to her computing needs in exchange of a payment made to the manufacturer. IC feature configuration is implemented by firmware updates from the manufacturer. The smart contract takes the feature configuration request from the IC as input and outputs the source of corresponding firmware. To support remote and authorized update by manufacturer, we propose an on-die hardware module that communicates with the smart contract and enforces its functionalities. Availability of this module also facilitates secure firmware update. The blockchain makes the update protocol secure and prevents users from obtaining unauthorized update.**

Index Terms—**Blockchain, Smart Contract, Firmware Update, Chip Features, Internet of Things.**

I. Introduction

With aggressive scaling, CMOS technology has been a driver for growth in semiconductor industry for more than three decades. With scaling of transistor feature size, the cost to develop a new integrated circuit, its fabrication, debug and volume production has been escalating. As the industry transitions to $7nm$ technology and beyond, the cost of development of a new SoC, the cost of design debug and design iterations becomes ever larger. Traditionally the economic model semiconductor industry is built upon economies of scale and this works well for high-volume markets, such as the smartphone. However for smaller, more fragmented markets, the volume for any one chip is insufficient to justify the high levels of investment required – particularly for advanced process nodes such as FinFET, for example [2].

Consequently, without the ability to customize IC features after production, the price of an IC is determined by its lowest priced application. This motivates the manufacturers to develop capabilities for post-production IC customization. The commodity microprocessor business offers an example of post-production customization, where a manufacturer can tailor cache size, number of cores and frequency of operation for a target market segment after a chip has been manufactured [1]. Based on market demand, chip manufacturers sell the highest-end chips as lesser chip parts. Today, such customization is limited to one-time programming (OTP) for predetermined IC bins [3].

In this paper, we explore how an IC can be programmed repeatedly and securely using a blockchain-based smart contract. This will enable users to upgrade IC features, or rent upgraded IC features for a fixed period after it has been purchased. The availability of such a system could, for example, allow a buyer to upgrade her processor from i3 to i5 after it has been purchased to scale to her computing needs in exchange of a payment made to the manufacturer. In such a platform concept, multiple IoT markets may be addressed by a single chip. Chip features may be enabled or disabled during manufacture or in the field. While chip cost is fixed, the chip selling price may be set according to the features enabled and markets being addressed.

Chip feature configuration is implemented by firmware updates from the manufacturer. Traditional firmware update protocols for IoT systems are dependent on a centralized architecture [4]. Information is sent from the device to the cloud where the data is processed using analytics and then sent back to the IoT devices. For example, the Linux Vendor Firmware Service (LVFS) provides resources and support for helping vendors package their firmware updates, and serves as an online repository for obtaining these updates [5].

However, with billions of devices set to join IoT networks in the coming years, this type of centralized system for firmware updates has very limited scalability, exposes billions of weak points that compromise network security. In this paper, for enabling chip features on-field, we resort to blockchain-based smart contract. The smart contract takes the feature configuration request from the IC as input and outputs the source of corresponding firmware. To support remote and authorized update by manufacturer, we propose an on-die hardware module that enforces a smart contract. Availability of this module also facilitates secure firmware update. The blockchain makes the update protocol secure and prevents users from obtaining unauthorized update.

Smart contracts are blockchain powered automated computer programs that, once started, automatically execute the conditions pre-programmed. These contracts allow devices to function securely and autonomously by creating agreements that are only executed upon completion of specific requirements. The main advantage of deploying smart contracts in a blockchain is the assurance provided by the blockchain that the contract terms cannot be modified. In our proposed smart contract, it enables automated execution of enrollment, authentication, firmware update of a chip. The contractual clauses are translated into embedded hardware and software. The embedded hardware and software can self-verify that conditions have been met to execute the contract.

For a secure firmware management system via blockchain, a physical device must be authenticated securely and uniquely. For this purpose, we utilize physical unclonable functions

(PUFs) as the hardware root-of-trust. PUFs are innovative, lightweight circuit primitives that harness the intrinsic disorder in an IC introduced during the fabrication process. In this article, we propose a method of securely and uniquely authenticating an IC using SRAM-based Weak PUF [6].

Our proposed protocol can enable manufacturers to develop devices that are blockchain-powered. Adoption of the proposed protocol in mainstream electronic device manufacturing could help transition the GDP driven economy to a service-based economic model. Consumers would no longer be charged for product features they do not use. Our proposed hardware and software solution can bring the benefits of blockchain to the IoT device. The major contributions of this paper are:

- Proposing a protocol for remote, secure, and repeated configuration of IC features via smart contract.
- Proposing a smart contract which automates self-execution of the enrollment, authentication, and firmware management for chip feature upgrades.
- Demonstrating the proposed solution for the example case of Arm Project Trillium in Ethereum blockchain. The code has been made publicly accessible in Github [7].

The paper is organized as follows: Section II describes the related works on chip firmware upgrade by blockchain. Section III briefly presents backgrounds related to our proposed methodology for establishing feature upgrade by blockchain. Section IV introduces the system requirements and implemented smart contracts for our proposed protocol. In Section V, we present our proposed methodology to chip feature upgrade using blockchain. Section VI discusses security analysis of the proposed protocol. Section VII outlines a demonstration of the protocol and discusses future directions. Finally, Section VIII concludes the paper.

II. Related Works

Several blockchain-based firmware update protocols have been proposed in the literature [8]–[10]. Boudguiga *et al.* proposed a peer-to-peer mechanism, to spread firmware updates between IoT devices that have limited access to the Internet [8]. They investigated how the use of a blockchain infrastructure can 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 [9]. 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 [10]. 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.

III. Background

In this section, we introduce some relevant concepts used to build our proposed protocol.

A. 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) [11]. By applying the challenge and observing the unique, unpredictable response, each IC can be authenticated.

B. Configurability of IC Features

Several semiconductor industries apply IC configuration after manufacturing. For example, Intel Skylake microarchitecture features a highly-configurable design. Intel can meet the different market segment requirements using the same macro cells [12]. The Skylake family consists out of 5 different actual dies, which can be further segmented by disabling different features, e.g. GT1 graphics are based on GT2 graphics with half the execution units disabled [13].

C. Remote Firmware Update

Secure firmware updates of deployed IoT devices over networks is one of essential features nowadays. Vendors use the remote firmware update to provide new functionalities and also to patch vulnerabilities on the IoT devices. The current automatic firmware update process utilizes client-server architecture, where the vendor's repository is the server side and the IoT device is the client side. A vendor maintains its own firmware repository that contains pre-compiled binaries of the firmware for embedded devices.

D. Why Decentralized Firmware Management

The current model for firmware distribution from the vendor's repository to the IoT device adopts the client–server model. In this model, excessive network traffic may occur when devices request the firmware update files simultaneously. In the IoT environment where tens of millions of devices are possibly required to be updated simultaneously, this client–server model is inappropriate [14], [15].

IV. System Requirements and Smart Contract Implementation

In this section, we outline the key system requirements and implemented smart contracts for our proposed IC feature configuration and firmware management protocol. First, we describe the approach and operational requirements of our protocol. Next, we explain the implementation of smart contract.

A. Approach

1) Device Authentication: For a secure feature configuration, each chip must be authenticated uniquely. The manufacturer/Original Equipment Manufacturer (OEM) claims the initial ownership of an IC and writes the relevant PUF authentication data in the blockchain. At any point of supply chain over the device's lifetime, relevant IC ownership transfer information is also recorded in the blockchain. This enables the *proof-of-ownership* without an explicit need for trusted intermediary and thus prevents unauthorized parties, like counterfeiters, from falsely claiming ownership of an IC.

Furthermore, authorized parties can use the PUF data to authenticate, track, analyze, and provision chips. Fig. 1 presents our proposed approach using blockchain. For automating the process of registering an IC with PUF authentication data, transferring ownership, verifying the ownership information, the manufacturer implements a smart contract in the blockchain. In the next section, we detail the implemented smart contract for device authentication.

2) Feature Configuration and Firmware Management: In our proposed protocol, users can upgrade IC features, or rent upgraded IC features for a fixed period after it has been purchased. This is implemented by repeated, remote and secure programming of the IC using a smart contract. The owner of a device requests for remote configuration by sending the list of upgrade-able IPs, and intended time period of usage to the smart contract. The smart contract first verifies the

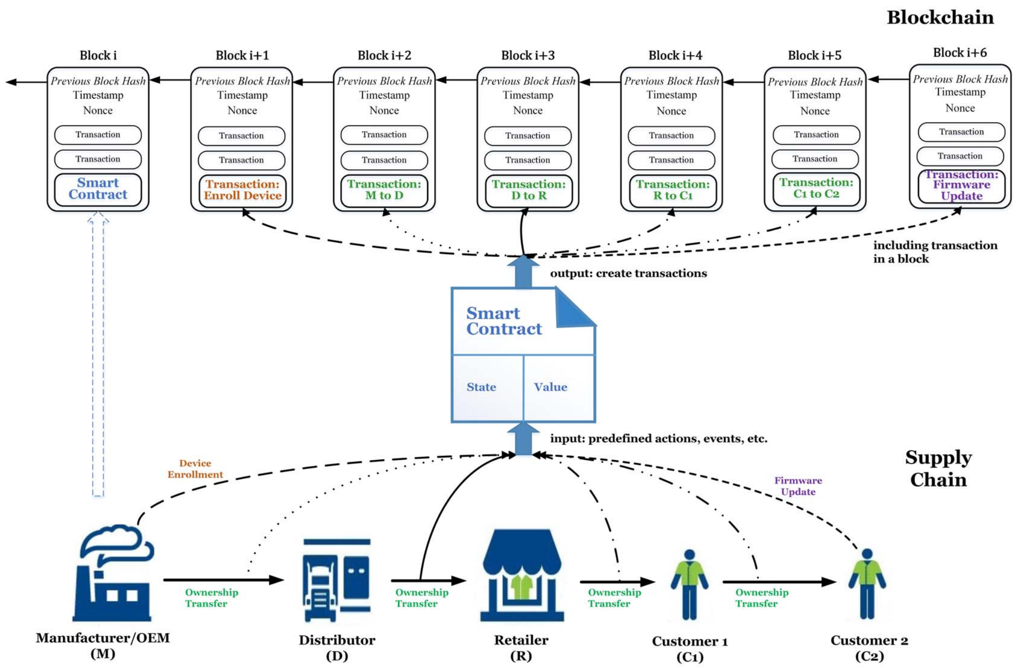


Fig. 1. Proposed approach for IC registration, authentication, and firmware update at various supply chain stages from manufacturer to the end-user. OEM: Original Equipment Manufacturer.

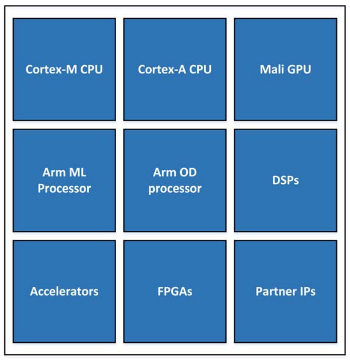


Fig. 2. Arm's AI-enabled devices consisting of ML and OD (Object Detection) processors in project Trillium.

authenticity of the device and the ownership. According to the requested list of configurable features, the smart contract sends the URI (Uniform Resource Identifier) of the corresponding firmware to the device.

For the purpose of demonstration, here we present the upgrade-able/downgrade-able features of Arm project Trillium Machine Learning (ML) platform [16]. The Trillium is a complete, heterogeneous compute platform for ML which includes a new, highly scalable processor line that is compatible with all programmable Arm IP. It consists of a suite of Arm IP including new highly scalable processors that delivers enhanced ML and Neural Network functionalities (Fig. 2).

B. Implementation of Smart Contracts

In our proposed protocol, a smart contract defines the condition for enrolling a new device by a manufacturer, transferring the ownership of a device to the next owner (*buyer*) by the current owner (*seller*), verifying the current ownership information, authenticating the device, and feature upgrades. In this section, we describe all the elements needed for implementing the smart contract.

1) Ownership Keys and Addresses: For uniquely identifying a supply chain participant and preventing any forgery, all the supply chain participants are assigned private-public key pairs. Device owners' private-public keys are generated using standard ECDSA algorithm. Device specific private-public key is generated using the protocol described in Section V-D2.

A user can manage personal keys and addresses using a digital wallet. The digital wallet can be used to perform any transaction. A digital wallet is hardware and software, or specifically designed hardware, that holds the public-private keys and the address.

2) Smart Contract Implementation for Feature Upgrade: The smart contract for feature upgrade is created by the manufacturer to maintain uniform applicability and usability for all of the owners of all devices. In our proposed work, smart contract (SC) provides the services for device registration, checking the ownership information, device authentication, ownership transfer, and firmware updates. Next, we explain these services and how these are implemented by our proposed smart contract.

a) Device Registration: To introduce a device into supply chain, it must be registered first. This is implemented by the smart contract function `registerDevice()`. Alg. 1 presents the pseudo-code of function `registerDevice()`. This function enrolls a device if the message sender is the manufacturer. The *deviceInfo* in the function includes data for *identification* and *authentication*. The identification data is used to look up the targeted device being queried among a collection of devices. It can be a serial number of the device (e.g. Electronic Product code, or EPC). The device identifier is used for the purpose of *identification*, and not as the primary means of *authentication*. For *authentication*, the manufacturer includes PUF data as input to the function `registerDevice()`.

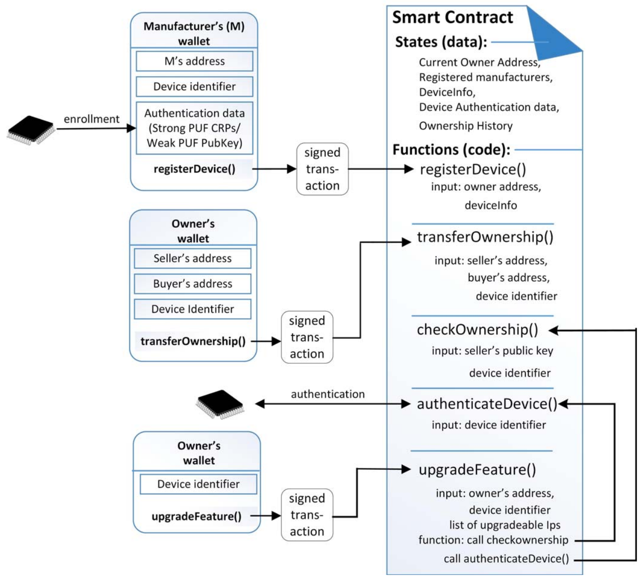

Manufacturer's (M) wallet
M's address
Device identifier
Authentication data (Strong PUF CRPs/ Weak PUF PubKey)
registerDevice()
enrollment
signed trans- action
Smart Contract
States (data):
Current Owner Address, Registered manufacturers, DeviceInfo, Device Authentication data, Ownership History
Functions (code):
registerDevice()
input: owner address, deviceInfo
Owner's wallet
Seller's address
Buyer's address
Device Identifier
transferOwnership()
signed trans- action
transferOwnership()
input: seller's address, buyer's address, device identifier
checkOwnership()
input: seller's public key device identifier
authentication
authenticateDevice()
input: device identifier
Owner's wallet
Device identifier
upgradeFeature()
signed trans- action
upgradeFeature()
input: owner's address, device identifier list of upgradeable Ips
function: call checkownership
call authenticateDevice()

Fig. 3. Smart contract implementation for IC feature upgrade.

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

```
Inputs: Manufacturer's address (addrManufacturer),
        and device information (deviceInfo)
1 if Message sender is in manufacturer's list then
2 |   Specify owner of the device as addrManufacturer
3 |   Register deviceInfo on the blockchain
4 else
5 |   Do nothing
6 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
1 if ($hash(sellerPubKey)$ ==
    $blockchain[deviceIdentifier].owner$) then
2 |   return True
3 else
4 |   return False
5 end
```

b) Ownership Verification: Device ownership can be verified by our proposed smart contract function `checkOwnership()`. Alg. 2 presents the pseudo-code of function `checkOwnership()`. This function verifies the ownership of the device against the owner's address. If the device with the provided identifier is owned by the message sender, it returns `True`. The function is invoked when any device owner wants to upgrade features.

c) Device Authentication: Any entity trying to communicate with a device must confirm that the device is authentic; not a counterfeit one. This is implemented by smart contract function `authenticateDevice()` Alg. 3 presents the pseudo-code of `authenticateDevice()`. This function starts device authentication process for a given device identifier. The process includes getting the challenge-response data

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

```
Inputs : Identifier of the device to be transferred
         ($deviceIdentifier$)
Output: A boolean True or False
1 set deviceChallenge =
    blockchain[$deviceIdentifier$].$Challenge$
2 get deviceResponse from the device after applying
    deviceChallenge
3 if (deviceResponse ==
    $blockchain[deviceIdentifier].Response$) then
4 |   return True
5 else
6 |   return False
7 end
```

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

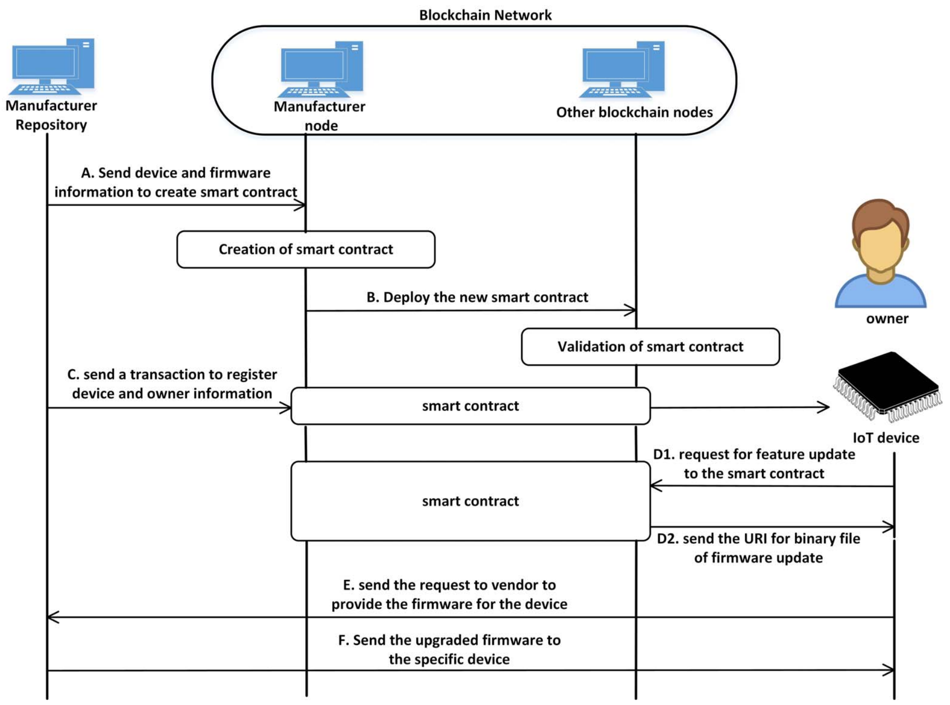
```
Inputs: Buyer's address ($addrBuyer$), and device
        identifier ($deviceIdentifier$)
1 if ($addrMessageSender$ ==
    $blockchain[deviceIdentifier].owner$) then
2 |   set blockchain[$deviceIdentifier$].owner =
    |     $addrBuyer$
3 else
4 |   Do nothing
5 end
```

for particular *deviceIdentifier*, applying the challenge to the device, calculates the response and matching the challenge-response pairs.

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

e) Feature Upgrade: The feature upgrade policy is defined on the smart contract that enforces its execution without an intermediary, so only authorized devices can request and receive the update. The policy includes manufacturer's predefined Service Level Agreement (SLA) with the customer. The required cost for getting a particular feature upgrade can be set using various different policies which are out of the scope of this paper.

The smart contract function `upgradeFeature()` implements the feature upgrade policy. This function first verifies if the request sender for the feature upgrade is the owner of the device by calling smart contract function `checkOwnership()`. It then checks the authenticity of the device by calling the smart contract function `authenticateDevice`. If both the conditions are true, the function then calculates the required transaction fee for that specific feature update and initiates the transfer of transaction fee from the owner's wallet to manufacturer's wallet. Finally, it sends the URI (Uniform Resource Identifier) of the firmware binary to the requesting device.



Blockchain Network
Manufacturer Repository
Manufacturer node
Other blockchain nodes
owner
A. Send device and firmware information to create smart contract
Creation of smart contract
B. Deploy the new smart contract
Validation of smart contract
C. send a transaction to register device and owner information
smart contract
IoT device
D1. request for feature update to the smart contract
smart contract
D2. send the URI for binary file of firmware update
E. send the request to vendor to provide the firmware for the device
F. Send the upgraded firmware to the specific device

Fig. 4. Detailed diagram for remote configuration of IC features and firmware management protocol.

Algorithm 5: Pseudo-code of `upgradeFeature()` for updating the firmware of the device.

```
Input: Owner's address ($addrOwner$), device
       identifier ($deviceIdentifier$), a list of
       updateable IPs, and requesting time period
1 if (addrMessageSender ==
    blockchain[$deviceIdentifier$].owner) and
    (authenticateDevice($deviceIdentifier$) ==
    True) then
2     Calculate required fee for feature update
3     Transfer the transaction fee from the owners's
       wallet to manufacturer's wallet
4     Get firmware update for $deviceIdentifier$
5 else
6     Do nothing
7 end
```

V. Proposed IC Feature Configuration and Firmware Update Protocol

In this section, we describe our proposed protocol for IC feature configuration and firmware update as shown in Fig. 4. The protocol consists of several phases which we describe as follows:

A. Creation of Smart Contract for Feature Configuration and Firmware Update

To automate the process of feature upgrade, the manufacturer first creates a smart contract. The manufacturer sends its device and firmware source information from the repository to its blockchain node to create the contract. The device information includes the feature upgrade policy for that specific type of device.

B. Deployment of Smart Contract for Feature Configuration and Firmware Update

The manufacturer node in blockchain creates the smart contract consisting of five functions as described in IV-B2. After creating the smart contract, the vendor node deploys it in the blockchain. Once validated by the blockchain nodes, the smart contract is added to the blockchain as a transaction and an address is assigned to it. After this initial transaction, the contract becomes a part of the blockchain forever and its address never changes. The manufacturer embeds this address in the security module of the device during manufacturing (Fig. 5).

C. Enrollment of a Device by the Manufacturer

When a manufacturer wants to register a device, it sends a transaction `registerDevice()` to the smart contract SC (Fig. 3). The manufacturer also sends necessary device information, such as, identifier of the device, 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 corresponding information of all the devices. This facilitates scalability of the protocol. As all the blockchain transactions are digitally signed by the transactor, any counterfeiter cannot illegally claim to be a non-authorized manufacturer.

D. Request for Feature Upgrade to the Smart Contract

Using the embedded address in the security module, the owner of a device sends a request for feature upgrade to the smart contract (Fig. 5). Blockchain node receives the feature upgrade request and checks the requirements on the firmware upgrade contract that matches with the received firmware upgrade request. If the requirements are satisfied, the passive

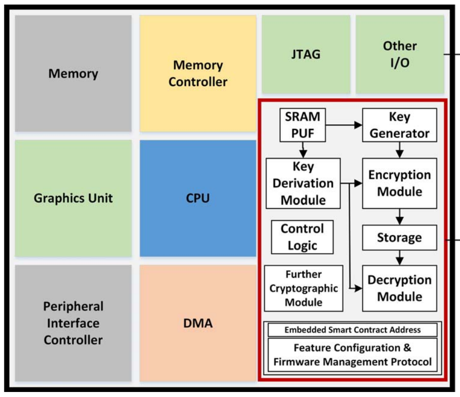


Fig. 5. Hardware authentication module for generating cryptographic key from PUF embedded in an SoC [17].

node sends the URI of the corresponding firmware binary for feature upgrade to the requesting device. Before sending the URI, the smart contract verifies the ownership and authenticity of the requesting device as described following:

1) Verifying the ownership information: To verify the current ownership information, the `upgradeFeature()` invokes `checkOwnership()` function of the smart contract. If the return is `True`, the ownership of the device is verified.

2) Authenticating the IC: Verifying the ownership of a device is not sufficient enough. A malicious attacker can replace the original device with a counterfeit one and request for feature upgrade for the counterfeit one. Therefore, the device also needs to be authenticated with the stored blockchain information. We propose Weak PUF based public key cryptography for authentication [18]. Fig. 5 presents a hardware authentication module to generate cryptographic key from SRAM-based Weak PUF [19]. The manufacturers first records Weak PUF data in the blockchain by issuing a transaction `registerDevice()`. At any stage of the supply chain, the device can be authenticated by invoking `authenticateDevice()`. The components of the authentication module and their functions are described as follows:

a) Key Generation Module: During enrollment phase, the key generator in the authentication module generates a private-public key pair. Using the private key, a device can create signature of a message protecting the message's integrity and proving its authenticity. At a receiving end of message, the digital signature can be verified for authenticity using the public key corresponding to the private key. The private-public key pair may be keys for RSA, DSA, Elliptic Curve based public key cryptosystems etc. For the purpose of illustration, we describe RSA public key cryptography here. First, the key generator finds two prime numbers from a seed which is 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 key (K_d^-), public key (K_d^+) pair is constructed. The RSA key pair generation is computationally intensive process and done only during enrollment phase.

b) 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 symmetric encryption algorithm and the encrypting key is obtained from any random 128-bit PUF. The manufacturer records the public key in the blockchain by issuing a transaction `registerDevice()` for the device (Fig. 6(a)).

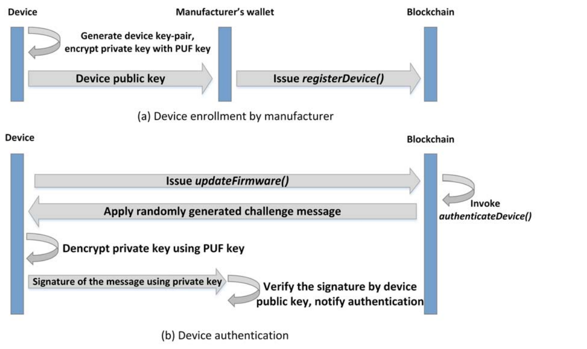

Device
Manufacturer's wallet
Blockchain
Generate device key-pair, encrypt private key with PUF key
Device public key
Issue registerDevice()
(a) Device enrollment by manufacturer
Device
Blockchain
Issue updateFirmware()
Invoke authenticateDevice()
Apply randomly generated challenge message
Dencrypt private key using PUF key
Signature of the message using private key
Verify the signature by device public key, notify authentication
(b) Device authentication

Fig. 6. (a) Device enrollment and (b) authentication process via Weak PUFs.

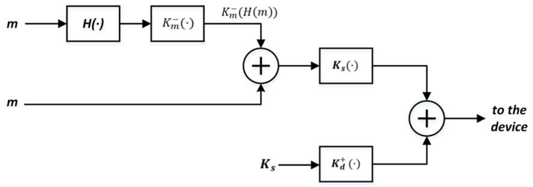


Fig. 7. Manufacturer uses symmetric key cryptography ($K_s(\cdot)$), public key cryptography, a hash function ($H(\cdot)$), and a digital signature ($K_m^-(H(\cdot))$) to provide secrecy, sender authentication, and message integrity.

c) Decryption Module: During 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.

The device can be authenticated by invoking `authenticateDevice()` with the device identifier. The smart contract 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 smart contract can verify the digital signature for the authenticity of the device as shown in Fig. 6(b).

E. Sending request to vendor for firmware update

After the device receives the URI of the firmware for the requested feature upgrades, it sends a request to the manufacturer repository to download the firmware binary. As a proof of feature upgrade request by the device, it sends the request in the form of transaction generated from executing the smart contract function `updateFirmware()`. This request submission is managed by the Firmware Configuration and Firmware Management Protocol (FCFMP) of the security module.

F. Sending Firmware Update to the device

Once the manufacturer repository receives the request from the device, it prepares the corresponding firmware binary to implement the requested feature upgrades. The manufacturer gets the contracted time period from the requested transaction and prepares the firmware by time-bounding it with that contracted period. The firmware will be invalid after that specific time period. Afterwards, the vendor repository sends the requested binary file to the requesting device.

The firmware update infrastructure should implement the CIA triad properties, i.e., confidentiality, integrity and availability. Fig. 7 presents such an implementation. For firmware integrity and sender authentication, the manufacturer creates a signature on the hash of the firmware using its private

TABLE I
COST OF VARIOUS OPERATIONS.

Operation	Gas limit	Gas price	Cost (in ETH)	Cost (in USD)
`registerDevice()`	121478	10.89×10^{-9}	0.0013229	0.11
`transferOwnership()`	30365	10.89×10^{-9}	0.00033067	0.03
`updateFirmware()`	20423	10.89×10^{-9}	0.0002224	0.018

key (K_m^-). Using the embedded manufacturer public key (K_m^+), the device can verify the firmware integrity and sender authentication.

For maintaining confidentiality, the manufacturer encrypts the firmware with a randomly generated symmetric key (K_s) and further encrypts the symmetric key with the devices public key (K_d^+). Only the device with the corresponding private key (K_d^-) can decrypt the symmetric key. Next, the device decrypts the firmware using the symmetric key and installs it.

VI. Security Analysis of the Protocol

In this section, we analyze various attacks and how our proposed solution can prevent them.

a) A malicious attacker saves the firmware for upgraded features and keeps using it even after the contracted time period without paying: The manufacturer gets the contracted time period from the requested transaction and prepares the firmware by time-bounding it with that contracted period. The firmware will be invalid after that specific time period. Feature Configuration & Firmware Management Protocol (FCFMP) enforces the time period validity of the installed firmware. After the contracted time period, the FCFMP installs the latest firmware present in the storage.

b) A malicious attacker gets a firmware update for one device and tries to install it in other devices: The manufacturer encrypts the firmware with a randomly generated symmetric key (K_s) and further encrypts the symmetric key with the devices public key (K_d^+). Only the device with the corresponding private key (K_d^-) can decrypt the symmetric key. This private key is unique to each device and any other device cannot decrypt the firmware and install it.

VII. Protocol Demonstration and Discussion

The proposed protocol has been implemented in `Solidity` scripting language and evaluated in terms of its operational cost. The code has been made publicly accessible in Github [7]. In particular, the cost of each operation has been estimated by measuring the gas amount (execution fee for the operation made on Ethereum) for all of the functions involved in the process, that is, (i) registering device (`registerDevice()`), and (ii) transferring ownership (`transferOwnership()`), (iii) upgrading features `updateFirmware()` and then converting it into a real currency. As the amount of gas is fixed for each operation in Ethereum, e.g. an 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*, to measure the gas amount since this tool has the ability of automatically counting the gas amount. With current value of 1 ETH = 83.15USD and 1 Gas = 10.89×10^{-9} ETH (10.89 Gwei), the operation cost for a chip in supply chain is calculated in Table I.

VIII. Conclusion

In this paper we presented a smart contract based integrated circuit configuration management system that allows individual integrated circuits to be custom configured to a specification mutually agreed by manufacturer and buyer. In the proposed solution, a buyer can upgrade or *rent* integrated circuit features after the integrated circuit has been sold. Traditionally, integrated circuits are tailored for specific market segment via one-time programming that is irreversible and does not allow a path for upgrade. In the solution presented here, an integrated circuit can be upgraded as well as downgraded via conditions pre-programmed in smart contract. A security analysis of the proposed protocol has also been presented in the paper. As the industry transitions to $7nm$ technology and beyond, the cost of development of a new SoC, the cost of design debug and design iterations becomes ever larger. The non-recurrent design cost is typically recovered by amortizing over large volume of parts. In a segmented market, the volume of parts in any given segment may not be high. Such markets are served by high volume parts via customization or tailoring. This paper presents a viable approach for such tailoring that allows post-production *secure* customization.

References

[1] E. Sperling, "How much will that chip cost?" *URL: https://semiengineering.com/how-much-will-that-chip-cost/*, March, 2014.

[2] A. Faulkner, C. Clark, J. Lipman, and C. Downing, "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.

[3] D. Dingee, "Customized pmics with otp in automotive and iot," *URL: https://www.semiwiki.com/forum/content/6122-customized-pmics-otp-automotive-iot.html*, August, 2016.

[4] S. Downum, W. Phillips, and B. Samuel, "Firmware update control mechanism using organizational groups," Nov. 6 2018, uS Patent App. 15/352,356.

[5] "Linux users will soon be able to update dell firmware from inside the os," *URL: https://news.softpedia.com/news/linux-users-will-soon-be-able-to-update-dell-firmware-from-inside-the-os-497479.shtml*, December, 2015.

[6] M. N. Islam and S. Kundu, "Enabling ic traceability via blockchain pegged to embedded puf," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 3, p. 36, 2019.

[7] "Firmware update via blockchain," *URL: https://github.com/jesnur/Firmware-Update-via-Blockchain*.

[8] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "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)*. IEEE, 2017, pp. 50–58.

[9] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an internet of things environment," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 1152–1167, 2017.

[10] M. Baza, M. Nabil, N. Lasla, K. Fidan, M. Mahmoud, and M. Abdallah, "Blockchain-based firmware update scheme tailored for autonomous vehicles," *arXiv preprint arXiv:1811.05905*, 2018.

[11] M. N. Islam, V. C. Patii, and S. Kundu, "On ic traceability via blockchain," in *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 2018, pp. 1–4.

[12] J. Mandelblat, "Technology insight: Intel's next generation microarchitecture code name skylake," in *Intel Developer Forum, San Francisco*, 2015.

[13] "Skylake (client) - microarchitectures - intel," *URL: https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)*.

[14] M. N. Islam and S. Kundu, "Preserving iot privacy in sharing economy via smart contract," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2018, pp. 296–297.

[15] M. N. Islam, V. C. Patil, and S. Kundu, "Determining proximal geolocation of iot edge devices via covert channel," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2017, pp. 196–202.

[16] "Project trillium machine learning platform," *URL: https://www.arm.com/products/silicon-ip-cpu/machine-learning/project-trillium*.

[17] H. Handschuh and P. T. Tuyls, "Device and method for obtaining a cryptographic key," Jan. 19 2011, uS Patent App. 13/574,311.

[18] M. N. Islam, V. C. Patil, and S. Kundu, "A guide to graceful aging: How not to overindulge in post-silicon burn-in for enhancing reliability of weak puf," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[19] M. N. Islam, V. C. Patil, and S. Kundu, "On enhancing reliability of weak pufs via intelligent post-silicon accelerated aging," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 3, pp. 960–969, March 2018.