EasyChair Preprint

№ 7098

Auto-PUFChain: an Automated Interaction Tool for PUFs and Blockchain in Electronic Supply Chain

Chandan Chaudhary, Urbi Chatterjee and
Debdeep Mukhopadhayay

November 28, 2021

Auto-PUFChain: An Automated Interaction Tool for PUFs and Blockchain in Electronic Supply Chain

Chandan Kumar Chaudhary
IIT Kharagpur, India
cchaudhary278@kgpian.iitkgp.ac.in

Urbi Chatterjee
IIT kanpur, India.
urbic@cse.iitk.ac.in

Debdeep Mukhopadhayay
IIT Kharagpur, India.
debdeep@cse.iitkgp.ac.in

***Abstract*—Physically Unclonable Functions (PUFs) primitives and blockchain technologies, also known as PUFchain, are recently taking huge attention to integrate the security parameters in Supply Chain Management (SCM) system. In this work, we devise a technique to interface an electronic chip and a blockchain platform by providing access to a unique hardware fingerprint and the stored information of chips on blockchain at one single point. To the best of our knowledge, this is the only scalable and automated technique that has been successfully implemented to authenticate the hardware device at any stage of the supply chain management system. The technique realises the concept of embedded PUF instance and the smart contract deployment on the ethereum blockchain. In the proposed technique, we have highly reduced the cost of implementing the interaction by storing the CRP on a distributed file system called as InterPlanetary File System (IPFS). Further, we implement and analyse the interfacing for 'IC traceability' to ensure the current ownership of a product, its point of origin, and ownership history in the SCM system. We employ Nexys 4 DDR Artix-7 FPGA board and go-ethereum blockchain library for implementing "IC traceability" to demonstrate the workflow and overhead involved in our interfacing technique.**

***Index Terms*—Electronic Supply Chain, Ethereum, Physically Unclonable Functions, Blockchain**

I. Introduction

Since the development of blockchain technology, a lot of attention has been shifted towards blockchain to utilise it for the purpose of authentication and integrity checking of devices in the supply chain management system. Although the blockchain and PUF integrated techniques fit suitable for the electronic supply chain to prevent counterfeiting, a lot of human interventions like collecting CRPs from embedded PUF and manually feeding it into a smart contract to get stored on the blockchain in [1] and [2] might create some insecure channel for an adversary to alter the CRPs. To avoid such human intervention, we propose an automated interfacing technique to interconnect the hardware chip and the smart contract communication at one place. According to the state-of-the-art literature, [3], [1] and [2] have implemented the smart contract for their proposed methodology but no discussion on the overhead of storing the CRPs on the blockchain can be found. Therefore, we discuss the overhead of storing CRPs on the blockchain, and thereby in our proposed automated interfacing techniques, we reduce the storage on blockchain by storing the CRP hashes generated after uploading the CRP on InterPlanetary File System (IPFS) [4] [5] for the first time. An IPFS is a peer-to-peer, distributed, versioned file system that generates a fixed-length string that is unique to the file and its data. Now, we summarise our contribution in this article as follows:

- We propose an automated interfacing technique for hardware and blockchain interaction powered by IPFS to facilitate the verifiability of transactions in supply chain network.
- We develop the supply chain business logic in the form of smart contract written in solidity to automate the authentication and ownership transfer protocol.

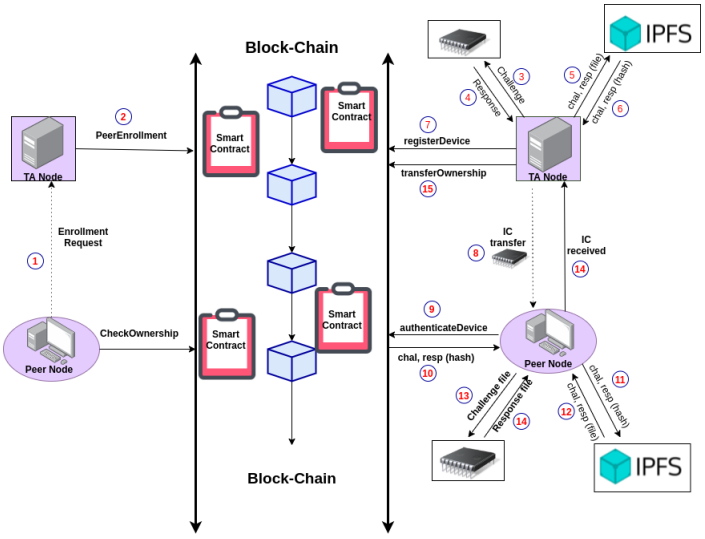

Fig. 1: The schematic diagram for proposed electronic supply chain

- We implement a prototype for 'IC traceability' on Nexys 4 DDR Artix-7 FPGA board and ethereum blockchain to show the workflow of interfacing technique and its corresponding overhead. The code for the same can be found at github [6].

The remainder of the paper is organised as follows: Section II discusses our proposed automated technique for dealing with counterfeits in SCM system and its possible implementation methodology. In Section III, we present our experimental setup and comparison results with previously proposed methodologies and finally section IV concludes our work.

II. The Automated Electronic Supply Chain Framework

In this section, we give an overview of the proposed automated interfacing technique and then we discuss the methodology for implementing the proposed scheme via smart contract algorithms and finally we brief the benefits of using IPFS in supply chain management system.

A. Overview

We divide the entities involved in our electronic supply chain into two broad categories: a) trusted authority (TA node) and b) peer node. A TA node is a privileged authority such as IP owner and original component manufacturers (OCM) that can do PUF characterization of a chip. The peer nodes have restricted privilege meaning they are not allowed to write data on the blockchain, and hence no change of state of ethereum is possible by peer nodes. Authorities like IP owners, OCM, distributors, PCB assemblers, integrators in the SCM system are linked with blockchain as a node and act as buyers or sellers in the chip transfer protocol. In our framework, as chip registration

needs to be completed by a TA node, so IP owners or OCM are given the privilege of a TA node. Other authorities like distributors, PCB assemblers and, integrators are peer nodes. Every buyer, in our framework, has the freedom to authenticate the device before purchasing it. A chip once registered to the blockchain will again be de-registered only by the TA node. So, if any node detects any counterfeit then it should inform a TA node to de-register the chip or update the chip by removing the defects from it.

Throughout the paper, we use the term "automated" in the following three contexts:

- Every end-user and intermediaries who have joined the system are permitted to authenticate an electronic device without getting assistance from any other authority.
- The use of permission-less blockchain allows all entities to access any chip-related information at any point without going to a particular authority for that information.
- The manual collection of CRP, the chip information from blockchain, and IPFS files have been avoided now with introducing a standard interface for them.

System Assumptions: Here we assume that the communication between trusted authority and IPFS is secure and confidential. An adversary can only register himself as a peer node. A TA node can register into the system only through a contract deploying authority. So, a contract deploying authority should have the information about all TAs prior to the deployment of the contract.

B. Proposed methodology

In an electronic supply chain, the tracking of chips while transferring from one stage to another is one of the most crucial steps to check the integrity and the authenticity of chips to mitigate counterfeiting. During authentication, an authenticating body needs to interact with several independent systems such as blockchain, PUF-embedded hardware, and some storage systems. However, there remains a possibility of counterfeiting due to the manual intervention during such interaction with those systems. This motivates us to propose an automated tool for providing an interface to interact among different independent systems. Secondly, an efficient methodology to store the challenge-response pairs for the embedded PUF instances of every chip on distributed database throughout the supply chain is one of the major factors that decide the cost efficiency of any authentication protocol. Here we use IPFS as cost-effective solution to such problem. In Figure 1, the small enumerated circles show the steps for a chip transfer protocol. We discuss these steps one by one taking the system setup as the beginning of blockchain based electronic supply chain.

1) System Setup: In the setup phase, we deploy our smart contract on Ethereum blockchain after including certain TA node addresses to the contract. Our smart contract is privately administered contract in the sense that some externally owned accounts on Ethereum blockchains are given higher privilege than others, for instance TA nodes possess more privilege than peer nodes. The permission-less blockchain here allows any peer node to access/call the smart contract anywhere. The smart contract is equipped with the following two structures to store informations for chip transfer protocol.

1) *authority:* This structure stores the details of nodes registered into the system. It has member variables as, *Entity* to store a 160-bit unique address associated with each node, and *Permission* to store the permission of an entity in an integer format. Based on the properties of a node, its value can be 0 for peer node, 1 for a TA node and, -1 for a revoked node. A revoked node is a node who has been marked as a part of some counterfeit.
2) *deviceInfo:* This structure stores the information related to chips (devices) registered in the system. It has member variables as, *deviceID* to store the unique identity for every chip, *owner* to store the current owner of the chip, *addedby* to store the address of the TA node who added this device, *chalHash* to store the hash of challenge set uploaded on IPFS, *respHash* to store the hash of response set (corresponding to the challenge set) uploaded on the IPFS, *lastupdatedby* to store the address of the TA node who has recently updated the chip's CRP hash on the blockchain, and *deviceOwnershipTrace* to store a list of all previous owners of this device so that if any counterfeit is detected at any stage then it becomes easy to trace the point of counterfeit and further possible legal action can be taken against such authority.

The system begins with deployment of smart contract by a contract deploying authority. A contract deploying authority has same power as a TA node except this ability to deploy smart contract. Once the contract is deployed on Ethereum, even a contract deploying authority cannot modify the contract at any cost. After the deployment of the smart contract, the address of smart contract is made available to all nodes publicly.

2) Adding peer node: To add a peer node an address associated with the peer node is given as input. Then, it checks the permission of the message sender, if the message sender is a TA node then it adds the node in the system with permission value 0. In Figure 1, Step 1 and 2 show a peer node requesting for enrolment and a TA node completing the enrolment of the peer node on blockchain.

3) Device Registration: The device registration is the most crucial stage of the electronic supply chain. Algorithm 1 shows the logic for registering a chip. It takes a unique *deviceID*, the address of the owner of the device, a challenge hash, and a response hash as inputs and register the chip. To collect a set of responses, the TA node generates a set of challenges and feeds it to the embedded PUF instance of the chip. Then, it connects with the IPFS and stores the set of challenges and responses to get back a corresponding constant hash value of 92 bytes. These constant size challenges and response hashes are used as a parameter for registering a device on the blockchain. In Figure 1, Steps 3-7 altogether add-up to register a device successfully. The algorithm first checks the permission of the message sender, and if the message sender is a TA node, then it registers the chip successfully otherwise aborts. Once a device is registered, any node can access the information related to the chip, like the owner of the device, who added this device, etc.

Algorithm 1: registerDevice

```
Data: deviceID, owner, chal_hash, resp_hash
if authority[msg.sender].permission == 1 then
    deviceInfo[deviceID].addedby = msg.sender;
    deviceInfo[deviceID].lastupdatedby = msg.sender;
    deviceInfo[deviceID].chalHash = chal_hash;
    deviceInfo[deviceID].respHash = resp_hash;
    deviceInfo[deviceID].owner = owner;
    deviceInfo[deviceID].deviceOwnershipTrace.push(owner)
end
```

4) Authenticating Device: Before buying a chip, a buyer authenticates the chip by calling smart contract with function *authenticateDevice*. Algorithm 2 shows the logic behind *authenticateDevice* function. The algorithm intakes the *deviceID* and the hash of the challenge and response available on the blockchain. Then, the hash of challenge is given to IPFS to retrieve the plaintext of the challenge

set, then the algorithm feeds this challenge to the device and receives the corresponding responses from the embedded PUF. The response received from the device is again posted on the IPFS; if the hash received is same as the response hash available on the blockchain, its authentication returns true otherwise false. In Figure 1, Steps 9-14 altogether represent authentication of a chip.

Algorithm 2: authenticateDevice

Data: deviceID, respHashFromBlockchain, chalHashFromBlockchain
Result: True or False
chal = IPFSGetFile(chalHashFromBlockchain);
respFromDevice = CRPGeneratorOnDevice(chal);
hashedResp = IPFSUploadFile(respFromDevice);
if *respHashFromBlockchain* == *hashedResp* **then**
  return True;
else
  return False;
end

5) Tracing Ownership: Every node connected to the system can trace all the previous and current owners of a particular chip. This helps in catching a node involved in counterfeit. The node involved in counterfeit is also blacklisted, and the permission for that node is assigned -1 with the help of a TA node.

6) Device Information: Any valid node in the system can call *deviceInfo* function to get informations like the address of current device owner, address of TA node who added this particular device into the system, and the relevant challenge and response hashes stored on the blockchain.

7) Ownership Transfer: The ownership of a chip is transferred from a seller to a buyer in three steps:

- *Ownership verification:* A buyer calls the smart contract to retrieve the information stored for a particular chip, It then verifies the seller by checking the permission of the seller. If the permission is found to be -1, it aborts the process otherwise the seller is authentic.
- *Chip authentication:* It calls the smart contract with authenticate device function and checks if the chip is authentic or not.
- *Payment & ownership transfer:* The buyer makes a payment on condition that the seller transfers the ownership of the chip to him on the blockchain. A payment made in terms of Letter of Credit (LoC) allows to settle the payment only if seller makes a transaction for transferring the ownership. The *Ownership Transfer* algorithm first checks the address of the message sender; if the message sender is the owner of the chip then the function executes otherwise aborts.

8) Update Device: This is special feature available to all TA, whenever a doubt about counterfeit in a chip is detected. The previous recorded challenge-response pair (CRP) hash is deleted from blockchain and updated with newly CRPs hash values. This update feature can help to bring the chip back into supply chain by mitigating the counterfeits.

C. Automation through interface

The interfacing is defined to allow a node to connect all the independent and unrelated systems to meet at one place to achieve interaction and communication among themselves without human intervention. Our interfacing technique consolidates the interaction and operations related to the blockchain, the PUF CRPs and, the IPFS with the help of python code. Our python interface interaction

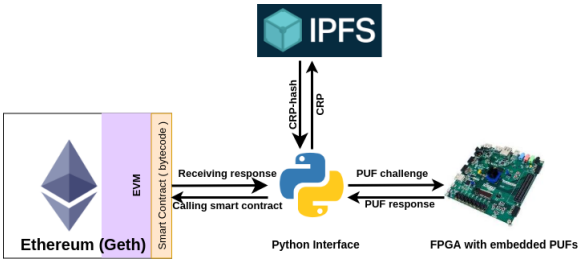

Fig. 2: Flow diagram for interfacing technique

is shown in Figure 2. The interface creates a dependency among other independent systems to synchronise all the involved operations at a particular stage of electronic supply chain. We discuss the peer node and the TA node interface for our framework as follows:

- *Peer node interface:* The interface for a peer node first connects to a chip through *digilent adept* software and programs the chip with a bit file (to configure the chip for serial port communication from the interface). Next, it develops a program to send the challenge to the chip and receive the response from it. Secondly, it creates connection to ethereum blockchain via a web3 python library. The web3 helps the interface to query the blockchain about any transaction and also about the blocks that are getting added. Thirdly, it connects to the IPFS system through another python library called *ipfsAPI*. Once the connections are established, it can provide the trace of the ownership by calling smart contract with function *traceOwnership*, which in turn returns all the owners from the beginning till now of that particular device. A significant feature available to all peer nodes is *authenticateDevice*; using this function it receives the CRP hash from blockchain, which in turn is used as input to IPFS to get back the original stored CRP. Now the hardware instance is called from this interface to collect the response from a particular chip currently attached to this peer node. The hash of responses received from the IPFS and hashed responses received from the blockchain are matched, and the authenticity of the chip is decided based on a comparison of results. *transferOwnership* is another feature available to update the ownership in the blockchain.
- *TA node interface:* The TA nodes also connect to the chip, blockchain and, IPFS in similar way as peer node interface does. Besides all the functionalities available to peer node's interface, TA nodes have some extra privilege with functions *addPeer*, *registerDevice* and, *updateDevice*. While registering a device, the TA interface triggers a python code and sends a set of challenges to collect responses from the board on which PUF instance is embedded. The TA interface now connects to IPFS and uploads the challenge file to get back the corresponding unique hash. This way, even if the IPFS hash for CRP is getting leaked to an adversary (peer) in the later stages of the supply chain, he would never get any chance to modify the CRPs stored in the blockchain as blockchain is immutable in nature and, also IPFS generates hash according to the content of the file so any modification in challenge-response will lead to the generation of a new hash and will never overwrite the previous hashed value. *updateDevice* is very much similar to *registerDevice*, and it needs to update challenge-response hashes stored in blockchain.

Advantages of using IPFS: IPFS, being version-controlled and distributed in nature, provides the availability of the hashes of challenge and response to all TA and peer nodes globally. Secondly, the challenge-response stored on the IPFS remain immutable forever. It increases the trust of all the intermediate seller-buyer on the

responses available on IPFS. Thirdly, unlike blockchain storage, it requires investing only one-time money to store a file on the IPFS [4]. We only keep the hashes of challenge and response file on blockchain in our proposed technique to minimize the cost of storage.

TABLE I: The Gas Costs for various operations in proposed electronic supply chain (Gasprice= 1 GWEI, 1 ETH = 2285 USD)

Operations	GAS LIMIT	ETH	USD ($)
DeployContract	1402381	0.028048	64.09
addPeer	44502	0.000891	2.03
registerDevice	252821	0.00506	11.56
updateDevice	34456	0.00069	1.59
transferOwnership	55646	0.00112	2.56

TABLE II: Storage overhead per device registration on blockchain

Number of CRPs	[7]	Proposed work
128	2176 bytes	92 bytes
512	8704 bytes	92 bytes
1024	17408 bytes	92 bytes

III. Experimental Setup and Results

In our experimental setup, we chose go-ethereum (*geth*) [8] version *1.9.25-stable-e7872729* for creating a local ethereum blockchain network running on Intel(R) core i7-4790 CPU @3.60GHz with 12GB RAM. The proposed smart contract is written in a solidity scripting language and is compiled through *solc 0.4.0* compiler. We generate bytecode and application binary interface (abi) using the commands *solc –bin eletronicSupplyChain.sol* and *solc –abi electronicSupplyChain.sol* respectively. Next, the *web3* python library utilises the bytecode and *abi* to help smart contracts getting deployed. It has been also used to develop the interface for TA and peer node. Python-based web3 library assists interfaces to connect to a blockchain node and the IPFS both at one place through two different ports. Before connecting to IPFS, we initialise the IPFS with command *'ipfs init'* to perform all internal settings. This command also creates a repository where all internal data can be stored. Next, running command *'ipfs daemon'* makes the IPFS and gateway server start listening for a client. For PUF related setup, we use *Digilent Adept* software to program Nexys 4 DDR Artix-7 FPGA board. Later, we use 5-4 Double Arbiter PUF [9] and take a set of multiple of 32 challenges to generate 128-bit response. We have applied the majority voting technique where a PUF instance is characterised for nine times for every challenge at the time of enrolment. The response obtained after the majority voting is considered as golden response to be used for authentication. The number of measurements for the majority voting is chosen so that the probability that the correct response bit gets the majority vote is higher than some threshold.

The cost of using ethereum needs to be paid in terms of ether in the blockchain. The amount of gas used during a particular operation is directly linked with cost in US dollars. So, with the current value, 1 ETH = 2285 USD and 1 gas = 20 Gwei where 1 Gwei = 0.000000001 ETH, we calculate the cost of executing different operation for our smart contract in Table I. The transactions that cause the alteration in the ethereum virtual machine's state are only responsible for calculating the cost overhead. Our smart contract has *addPeer*, *registerDevice*, *updateDevice* and *transferDevice* that modify the state of EVM and hence responsible for pricing. The gas used during the smart contract deployment is directly proportional to the lines of codes written. In our case, the deployment cost is coming out to be 64.09 USD, but this would be a one-time investment. The logic is that the more functionality you achieve through the smart contract, the more operation you need to do and hence more ethers you have to spend during the deployment. Given the cost of ownership transfer operation, we can use the following formula to calculate the total cost incurred until n^{th} stage of a chip, *Total cost = registration cost + (n - 1) * (ownership transfer cost)* . For a typical 5 stage electronic supply chain, it would cost approximately 21.56 USD.

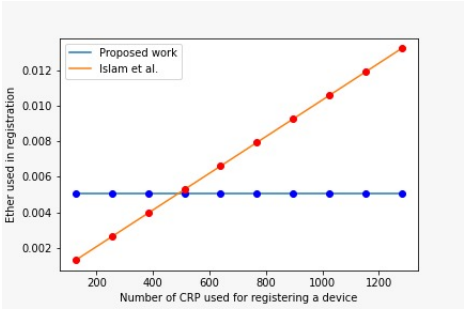

Fig. 3: Comparison of ether used during device registration with [7]

Registering a device on the blockchain is a one time process. But, if it increases linearly with the number of CRPs that is getting stored for the authentication purpose, it can cost high in terms of ether. We made a similar comparison of our proposed technique and [7] in Figure 3 to highlight the expense of registering a device when it requires storing a large number of CRPs for authentication purposes. The overhead of storing CRPs on the blockchain has been calculated and compared with [7] in Table II. In this comparison, we would like to highlight that we store only IPFS hash on the blockchain, which is quite small in size and remains constant even after increasing the number of challenges.

IV. Conclusion

This article outlines a methodology which allows any stack holder to automatically authenticate the chip without depending upon any other authority in the system. The permission-less blockchain ethereum helps deploy the proposed smart contract and execute it through the ethereum virtual machine (EVM). The use of IPFS for storing challenge and response have profusely reduced the storage overhead on blockchain and makes our proposed automatic technique of chip authentication more practical. Our interfacing tool is adaptable to all blockchain-based PUF authentication scheme developed so far in the literature for electronic supply chain management system.

References

[1] L. Aniello, B. Halak, P. Chai, R. Dhall, M. Mihalea, and A. Wilczynski, "Towards a supply chain management system for counterfeit mitigation using blockchain and puf," 2019.

[2] X. Xu, F. Rahman, B. Shakya, A. Vassilev, D. Forte, and M. Tehranipoor, "Electronics supply chain integrity enabled by blockchain," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 3, May 2019.

[3] 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)*, 2018, pp. 1–4.

[4] J. Omaar, forever Isn't Free: The Cost of Storage on a Blockchain Database, https://medium.com/ipdb-blog/forever-isnt-free-the-cost-of-storage-on-a-blockchain-database-59003f63e01, accessed on 30 August 2021.

[5] J. Benet, "Ipfs - content addressed, versioned, p2p file system," 2014.

[6] C. K. Chaudhary, "Auto-pufchain," implementation of proposed technique, https://github.com/ck-chaudhary/Auto-PUFChain.

[7] M. N. Islam and S. Kundu, "Enabling ic traceability via blockchain pegged to embedded puf," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 3, Apr. 2019.

[8] Go-ethereum official, https://github.com/ethereum/go-ethereum, accessed on 30 August 2021.

[9] U. Chatterjee, S. Chatterjee, D. Mukhopadhyay, and R. S. Chakraborty, "Machine learning assisted puf calibration for trustworthy proof of sensor data in iot," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 4, Jun. 2020.