Secure Management of IoT Devices Based on Blockchain Non-fungible Tokens and Physical Unclonable Functions

Javier Arcenegui(✉), Rosario Arjona(✉), and Iluminada Baturone(✉)

Instituto de Microelectrónica de Sevilla (IMSE-CNM), Universidad de Sevilla, CSIC, C/Américo Vespucio 28, 41092 Seville, Spain
{arcenegui,arjona,lumi}@imse-cnm.csic.es

Abstract. One of the most extended applications of blockchain technologies for the IoT ecosystem is the traceability of the data and operations generated and performed, respectively, by IoT devices. In this work, we propose a solution for secure management of IoT devices that participate in the blockchain with their own blockchain accounts (BCAs) so that the IoT devices themselves can sign transactions. Any blockchain participant (including IoT devices) can obtain and verify information not only about the actions or data they are taking but also about their manufacturers, managers (owners and approved), and users. Non Fungible Tokens (NFTs) based on the ERC-721 standard are proposed to manage IoT devices as unique and indivisible. The BCA of an IoT device, which is defined as an NFT attribute, is associated with the physical device since the secret seed from which the BCA is generated is not stored anywhere but a Physical Unclonable Function (PUF) inside the hardware of the device reconstructs it. The proposed solution is demonstrated and evaluated with a low-cost IoT device based on a Pycom Wipy 3.0 board, which uses the internal SRAM of the microcontroller ESP-32 as PUF. The operations it performs to reconstruct its BCA in Ethereum and to carry out transactions take a few tens of milliseconds. The smart contract programmed in Solidity and simulated in Remix requires low gas consumption.

Keywords: IoT devices · Blockchain technology · Non fungible tokens · Physical Unclonable Functions

1 Introduction

The Internet of Things (IoT) and blockchain are nowadays two technologies that are attracting a great interest. In general, IoT is a set of interconnected devices that exchange data and offer services to citizens, industries, businesses, and governments. IoT devices make smart the area where they are deployed (factories, hospitals, cities, etc.). Among the features that IoT devices must provide, security is one of the most important since they are the link between the physical world and Internet. An attacker may control either the actuators or sensors of an IoT device to carry out malicious actions. For example, a device with an insulin pump can be attacked to inject a lethal dose to the

J. Zhou et al. (Eds.): ACNS 2020 Workshops, LNCS 12418, pp. 24–40, 2020.
https://doi.org/10.1007/978-3-030-61638-0_2

patient or a blood pressure meter can be attacked to provide false readings. Secure IoT devices must prove that their hardware and software are trusted and that they behave in a trustworthy way. Similarly, they must trust the other devices or users with whom they interact [1, 2].

In the other side, a blockchain is a network of participants that take part in a distributed, synchronized and cryptographically secure data structure (ledger) composed of chained blocks that can be tracked by any participant. A block contains information about transactions (typically the exchange of digital currency or generic assets, date, time, etc.), participants involved in the transactions, data identifying the block univocally, and how the block is linked to the previous one. A new block is added if participants with the role of miners demonstrate (by a proof of work, stake, authority, etc.) that the new block is secure and most of the miners (typically 51% at least) agree to link the block (applying a consensus algorithm). Since not only the inclusion of a new block is based on a consensual agreement but also many of the participants (nodes) have an updated copy of the blockchain, it is very costly for hackers to manipulate any block [3].

Combining IoT and blockchain is very interesting because many transactions in smart areas involve IoT devices [4–6]. While it is not convenient for IoT devices to participate with the role of nodes or miners since they do not have enough memory and computing resources, it is practical they can participate with their own blockchain accounts (BCAs) associated with their cryptographic public keys so that they can take part in transactions and can sign them. This way, traceability of both devices and their data/actions is provided to the rest of blockchain participants, greatly increasing their security. The well-known blockchain Scalability Trilemma, which is to offer security, decentralization and scalability simultaneously, appears when many blocks and participants (being IoT devices or not) have to be handled [7]. In this work, we assume that there are other participants (apart from IoT devices) acting as nodes and miners that guarantee security and decentralization and that only the summary of many transactions carried out off-chain are stored in the blockchain to guarantee scalability.

The new generation of blockchain technologies allows smart contracts as a way to formalize agreements between participants. Typical agreements are to represent a cryptocurrency by a fungible token with a set of specifications (like its owner) and functions (like the way to change of owner). Fungible tokens of the same type are identical (like coins are identical) and are divisible into smaller units (like coins of different values). More recently, non-fungible tokens (NFTs) have been employed to represent unique assets (like collectables, certificates of any kind, any type of access rights, objects, etc.). An NFT is unique, indivisible, and different from another token of the same type. In particular, the ERC-721 standard describes how to build non-fungible tokens in the Ethereum blockchain [8]. Standard attributes of ERC-721 NFTs are: (a) the token identifier (tokenId), (b) the BCA of the NFT owner, and (c) the approved BCA by the owner to transfer the token to another owner. The digital and unique identifier, tokenId, allows recording and tracking a NFT in the blockchain. However, the token identifier does not have to be associated with a physical property of the device. In fact, it is generated automatically when the ERC-721 NFT is created.

In this work, we propose the use of NFTs to represent IoT devices. In particular, we base our development on the ERC-721 standard of Ethereum. The novelty of our

proposed NFTs is that they represent IoT devices that participate in the blockchain and, hence, have a unique BCA. Then, we incorporate the BCA of the IoT device as an NFT attribute. Another novelty of our proposal is that, since the BCA of an IoT device is naturally associated with the physical device, the IoT device generates its BCA from a PUF response. PUFs allow generating unique, intrinsic, unpredictable and distinctive identifiers for each device by exploiting the random variations of the device manufacturing process [9]. The BCA is associated with a cryptographic public key, which in turn is associated with a cryptographic secret key. Our proposal is that IoT devices prove their authenticity if, firstly, any content stored in its memory is removed, and, secondly, they are programmed with a trustworthy firmware that does not contain its secret key. If the IoT device is able to reconstruct its secret key and, hence, its BCA, is because its PUF response is authentic.

In addition, we incorporate the BCA of the user of the IoT device as another NFT attribute in order to distinguish between users, who employ the IoT device for an application, and owners, who assign IoT devices to users and can transfer the token to new owners. Owners can also approve others (approved BCAs) to transfer tokens to other owners.

Our proposal allows a secure management of IoT devices since any participant in the blockchain (including the IoT device itself or another) can verify their manufacturers, managers (owners and approved), users, and the actions or data they are taking. Besides, the IoT device and its physical owner, approved, and user can be subscribed to the events of its associated NFT so that they can receive notifications about the situation of the NFT and behave accordingly.

In summary, the contributions of this work are the following:

- The proposal of an NFT based on the ERC-721 standard of Ethereum that includes as new attributes the BCA of the IoT device and the BCA of the user of the IoT device. Since the IoT device has a BCA, it can take part in blockchain transactions and can sign them. Since the BCA of the user of the IoT device is included, user and owner roles are distinguished.
- The use of PUFs to guarantee that only the IoT device able to provide the required PUF response is the only one able to generate its BCA.
- A solution that merges the IoT and blockchain paradigms to allow the secure traceability of the data generated and the operations performed by IoT devices in scenarios of remote management.
- A proof of concept of the proposed solution by using a Pycom Wipy 3.0 as IoT device that generates its BCA in the blockchain Ethereum from the response of its SRAM PUF.

The paper is structured as follows. Related work is included in Sect. 2. Section 3 presents the proposal of NFTs for secure devices. The extension of the ERC-721 standard for NFTs that not only considers the BCA of the owners (managers) but also the BCAs of the device and its user is described. The process to generate the device BCA from a secret seed obfuscated by the response of a SRAM PUF as well as the device management are explained. Section 4 includes a proof of concept based on the Pycom Wipy 3.0 board. In the one side, the feasibility of obfuscating and recovering 256-bit secret seeds from the

use of internal SRAM PUFs is proven. In the other side, the implementation of the NFT with the SRAM PUF-based BCA by considering the Ethereum blockchain is shown. Section 4 also provides the execution times of the operations required to complete a transaction and the gas consumption of the smart contract functions programmed in Solidity and simulated in Remix [10]. Results are compared to other proposals in the literature. Finally, Sect. 5 concludes the work and adds future research directions.

2 Related Work

In the literature, ERC-721 NFTs are employed for several applications. The works [11] and [12] describe how the traceability of manufactured products can be performed using ERC-721 NFTs. The use of ERC-721 NFTs is also mentioned in [13] and in [14] for car sharing and event reselling applications, respectively. The management of IoT devices through the blockchain is extensively based on the use of smart contracts. However, the tokens used in many applications are not standard. In [15], tokenization is not directly related to the IoT devices. Instead, tokenization is employed through a task manager to ensure that all participants have something to lose if they misbehave. In [16], the IoT devices are grouped into IoT systems (like smart homes, smart hospitals, etc.) and each system is associated with the nearest blockchain-enabled fog node. A smart contract is defined on top of the blockchain-enabled fog nodes to support authentication and authorization of the IoT devices in a distributed fashion. In this proposal, IoT devices can communicate among them if they are registered and authenticated by blockchain-enabled fog nodes. The tokens are considered as certificates that include the device identifier, the device public address and the IoT system identifier. Therefore, the token involves two devices: the fog node and the IoT device. The solution proposed in [17] uses the principle of ERC-721 NFTs to implement a capability-based access control model in a decentralized IoT architecture. In this proposal, the tokens store the access rights for the resources/services available. A device in possession of one of these access tokens can access the resource/service according to the access control rules defined within the token. This solution is tested on a private Ethereum blockchain node.

None of the above commented solutions stablishes a physical link between a device and an NFT logical identifier. In [11], the digital and unique identifier, tokenId, which is a standard attribute of ERC-721 NFTs that allows recording and tracking an NFT in the blockchain, is a randomly selected string assigned to the device and stored in its RFID or QR code. In [15], the tokenId is the hash of a concatenation of the serial number embedded in the device chipset and a randomly generated salt. This can be also replaced by any random string that is not already in use when the device is registered. In [16], IoT devices are identified by certificates generated from a private key. In [17], tokenId is obtained by hashing three logical identifiers (the identifier of the device in possession of the token, the identifier of the resource/service, and the identifier of the resource as per the communication protocol).

Other works that employ the blockchain framework to provide supply chain integrity use PUFs to stablish a physical link between the devices and their logical identifiers [18–21]. The PUFs embedded in the products introduce a higher security level that reduces the risk of counterfeit and tampered electronic devices. However, these works do not employ explicitly the concept of NFTs.

To the best of our knowledge, there are no works in the literature using PUFs in ERC-721 NFTs as presented in the following.

3 Proposed NFTs for Secure Devices

The application scenarios of our proposal are smart areas with IoT devices that must be secure. The combination of IoT and Blockchain technologies enhances security. The main agents in these scenarios are: (a) the IoT devices (referred to as SDs, secure devices); (b) the users of the IoT devices; and (c) the application managers (referred to as owners), who assign devices to users and can transfer the devices to other managers. These three agents take part in the blockchain transactions through their BCAs (BCA_SD, BCA_user, and BCA_owner, respectively). Hence, they can authenticate each other and their messages in scenarios of remote management. A relevant amount of messages can be interchanged off-chain, to improve scalability, but the important transactions are registered to allow traceability in the blockchain. The owner (manager) and user can reset the device to ensure their firmware is trustworthy, avoiding the execution of malware. Conversely, the device allows reset if the request is from the owner or the user. In a smart hospital, for example, the owner can be the technical supervisor that assigns devices to doctors. In a smart infrastructure, the owner can be the manager of the technical workers who, depending on the scheduled tasks, assign the devices to one technician or another.

3.1 Main Features of the Proposed NFT

An IoT device becomes SD after being bound to our proposed NFT. The structure of the proposed token has the attributes shown in Table 1. The variables tokenId and BCA_owner are defined by the standard ERC-721. The standard also defines other variables (approved and operator) to help the owner to transfer NFTs to other owners, but this is not in the scope of this work, so that we omit them. The important variables added in this work are BCA_SD, which binds a device to the NFT, and BCA_user, which binds the device of the NFT to a user.

Table 1. Structure of the non fungible token

Type	Name of variable	Defined by the standard
TokenId_Type	tokenId	Yes
Address	BCA_owner	Yes
Address	BCA_SD	No
Address	BCA_user	No

The standard ERC-721 only declares functions related to the ownership of the token. A summary of them are included in the upper part of Table 2. They return which are the tokenIds of an owner (function “balanceOf”), who is the owner of a tokenId (function “ownerOf”), and how to transfer the tokenId to another address (function “transferFrom” detailed in Table 3).

Table 2. Functions employed in the proposed NFT

Defined by the standard
function balanceOf(address _owner) external view returns (uint256);
function ownerOf(uint256 _tokenId) external view returns (address);
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
Defined for this work
function createToken(address _owner, address _BCA_SD) external returns (uint256)
function userTransfer(uint256 tokenId, address _BCA_user) external;
function completeTransfer(uint256 _tokenId) external;
function tokenFromBCA(address _BCA_SD) external view returns (uint256);
function ownerOfFromBCA(address _BCA_SD) external view returns (address);
function userOf(uint256 _tokenId) external view returns (address);
function userOfFromBCA(address _BCA_SD) external view returns (address);
function userBalanceOf(address _BCA_user) external view returns (uint256);
function userBalanceOfAnOwner(address _BCA_user, address _owner) external view returns (uint256);

Table 3. Pseudo-code of the standard function "transferFrom"

Transfers a token from an owner to a new owner
Input: old_Owner, new_Owner, tokenId
Require (owner, operator, approved) = msg.sender
Require owner of tokenId = old_Owner
Change owner of tokenId to new_Owner
Send event Transfer

The functions needed in our case are shown in the bottom of Table 2. Given the BCA_SD, the functions "tokenFromBCA", "ownerOfFromBCA" and "userOfFromBCA" return, respectively, the tokenId, the BCA_owner and the BCA_user. Given the tokenId, the function "userOf" returns the BCA_user. The tokenIds of any owner assigned to a user are returned by the function "userBalanceOf" and the tokenIds of a particular owner assigned to a user are returned by the function "userBalanceOfAnOwner".

The pseudo-codes of the functions added to the proposed token are shown in Table 4. A token is created by the manufacturer of the IoT device with the function "createToken". It is assumed that the manufacturer creates the token when an "owner" buys the IoT device. The owner of the token can assign a user to the token with the function "userTransfer". If the owner of the token assigns it to the address "0", the token cannot be used by anyone, since this address is reserved in Ethereum. This is the way how an owner sets a device to a non-operative state.

Table 4. Pseudo-codes of the added functions

"createToken": Creates a new token linking BCA_SD to a tokenId
Input: _owner, _BCA_SD Output: tokenId Require (manufacturer) = msg.sender Generate new tokenId Set tokenId to token Set owner of tokenId = _owner Set BCA_SD of tokenId = _BCA_SD Return tokenId
"userTransfer": The owner assigns a user to the token
Input: tokenId, _BCA_user Require (owner) of _tokenId = msg.sender Set BCA_user from _tokenId = _ BCA_user Send event UserTransfer
"completeTransfer": Notifies that the token is already operative
Input: _tokenId Require (user) of _tokenId = msg.sender Send event TransferCompleted

3.2 Binding the IoT Device to Its Associated NFT

The manufacturer challenges the PUF inside the IoT device and receives from the IoT device the public key generated and the BCA_SD associated, as well as the helper data and masks that the device PUF needs to reconstruct its public key and BCA_SD. The steps of this process are detailed in Fig. 1 for the case of SRAM PUFs that use Static Random Access Memories (SRAMs). The manufacturer creates the token for the first owner, and includes the tokenId, PUF challenge, masks and helper data in the firmware associated to the device. Hence, only that device will be able to reconstruct its public key from that firmware because only its PUF will be able to provide the adequate response to the challenge received. Any other device will be unable to reconstruct BCA_SD from that firmware.

Among the electronic circuits that can be employed as PUFs, in this work, we select SRAM PUFs because most of IoT devices include SRAM in its hardware. SRAM PUFs are based on the start-up values obtained by powering up the memory [9]. Each SRAM bit cell is a bistable circuit whose logic memory functionality comes from two cross-coupled inverters. A write operation forces the SRAM cell to transition towards one of the two stable states ('0' or '1'). If the cell is powered-up and no write operation is carried out, the positive feedback between the two inverters leads the cell to the start-up value imposed by the inverter that begins to conduct. Ideally, the two inverters are identical, but the random variations in the manufacturing process make them different so that one of them is the first to conduct in each cell. Flipping bits can appear in the PUF response since the inverters of some bit cells are so similar that their start-up values change due

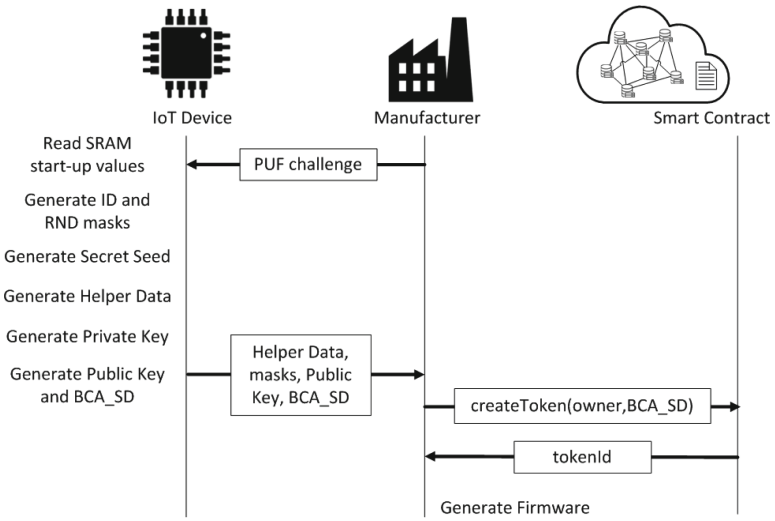


Fig. 1. The manufacturer binds the device to the NFT.

to noise. Particularly, those cells that change their value in half of the measurements, named herein as RND cells, are not adequate to identify the SD but are good to generate true random seeds. In the other side, the cells that provide generally the same start-up value, named herein as ID cells, are good to generate the PUF response. The use of the SRAM PUF inside the IoT device to generate the BCA_SD is illustrated in Fig. 2.

The first step of the token creation is to classify the SRAM cells addressed by the PUF challenge into ID and RND cells. For that purpose, the simple cell classification proposed in [22] is carried out. It consists in obtaining several measurements of start-up values by powering up and down the SRAM several times. For each measurement, the start-up values of all cells are compared. If the cell values do not change for all the measurements, the cells are registered as ID cells by an ID mask. If the cells change in half of the measurements, the cells are registered as RND cells in an RND mask. The second step of the token creation is to generate a true random Secret Seed from the start-up values of a set of RND cells selected by the RND mask. Since the Secret Seed are quite sensitive data because they identify cryptographically the SD, and the SRAM PUF response are also quite sensitive data because they identify physically the SD, the third step of the token creation is to generate non-sensitive data, known as Helper Data, from the Secret Seed and PUF response. The PUF response is obtained from the start-up values of a set of ID cells selected by the ID mask. Then, the Code Offset-based Helper Data algorithm described in [23] is used. It employs an Error Correcting Code to cope with flipping bits in the PUF response. Since the PUF response will show small bit flipping, a simple repetition Error Correcting Coder is employed. The steps of this process are detailed in Fig. 2a. The Secret Seed is not stored anywhere but is recovered from the response of the ID cells and the Helper Data, as illustrated in Fig. 2b. The Private and Public Keys of the device are obtained from the Secret Seed. Finally, the BCA_SD is computed from the Public Key. This is illustrated in Fig. 2c.

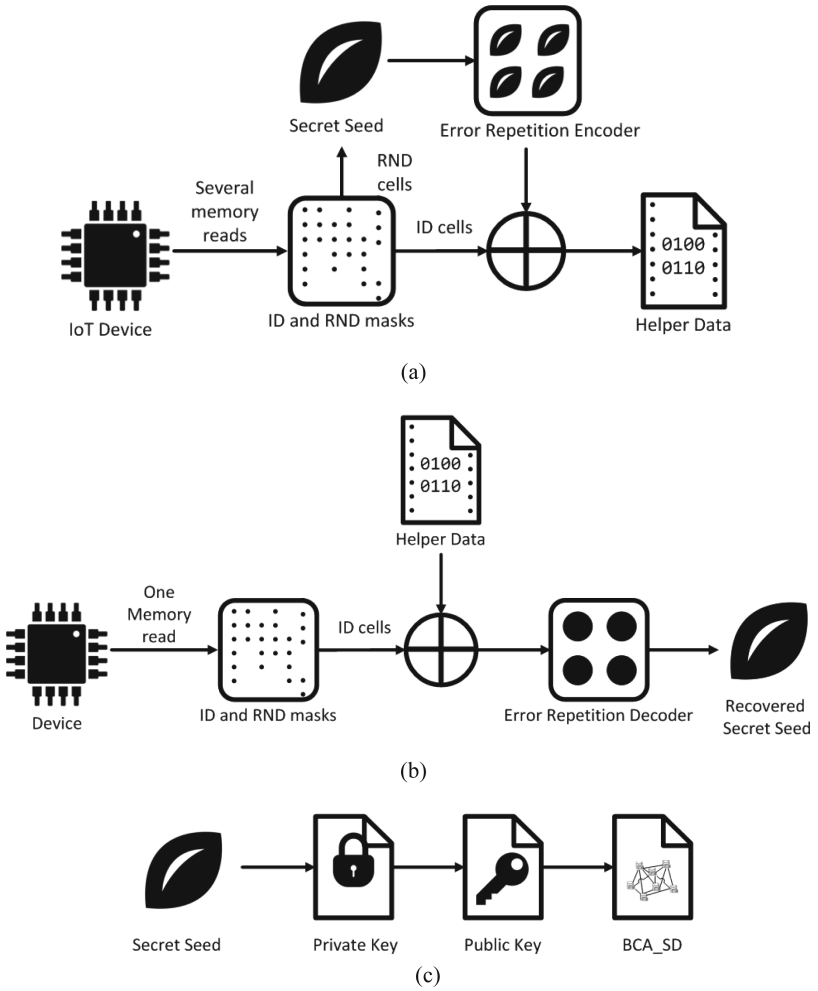


Fig. 2. Using the PUF inside the SD to generate and reconstruct the BCA_SD. (a) Generation of Secret Seed, masks, and Helper Data. (b) Secret Seed reconstruction. (c) Generation of Private Key, Public Key and BCA_SD from Secret Seed.

The manufacturer also programs in the device firmware that the device is subscribed to events "Transfer" (see Table 3), "UserTransfer", and "TransferCompleted" (see Table 4). With the two first events, the IoT device changes its state to "blocked" (non-operative) and can know its owner and user. This is important because the device will verify the BCAs and signatures of owner and user through their public keys if they request the device to update its firmware. The device does not need to store anything so the content of their memories can be deleted and a trustworthy firmware can be updated by its owner or user to ensure that the hardware and software of the device are trusted. Besides, the device will verify also the BCAs and signatures of the owner and user through their public keys when it is activated by them. The event TransferCompleted

notifies that the user and device have authenticated each other successfully so that the device becomes operative for the application. Although being "activated", the IoT device is not ready to work ("operative") until this notification is received. Details of these steps related to user and device are shown in Fig. 3.

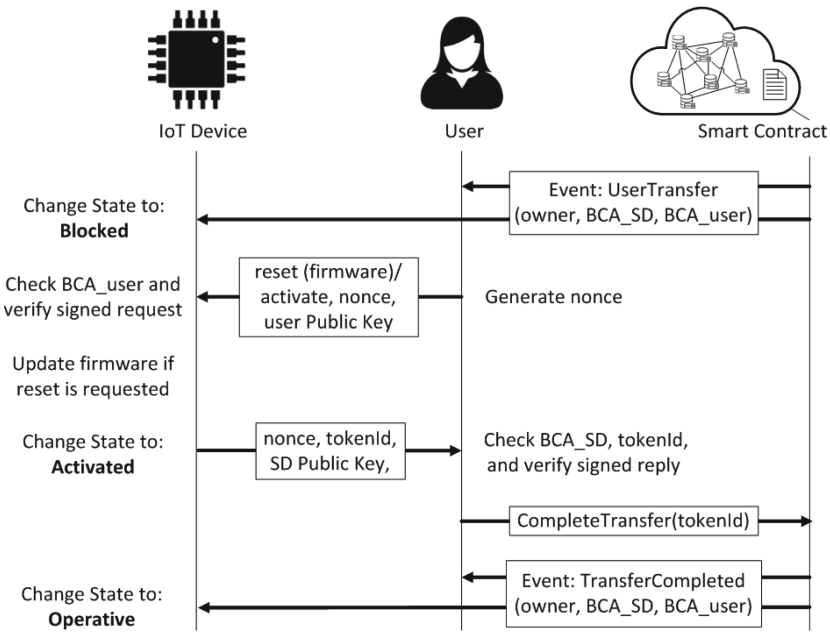


Fig. 3. States of the IoT device depending on events and user messages.

4 Implementation of the Proof of Concept

In this work, we employ a Pycom Wipy 3.0 board composed of an Espressif ESP32 chipset as IoT Secure Device (SD). This is a tiny development platform that allows ultra-low power usage and is very suitable to create IoT devices. The microcontroller ESP-32, which is the hardware core of the SD, contains an internal SRAM of 520 MB. This internal SRAM can be powered down and up without powering off the board completely so that it can be used as SRAM PUF.

4.1 SRAM PUFs from the IoT Device for Secret Obfuscation

One of the contributions of this work is the use of PUFs to obfuscate secret seeds employed to generate BCAs. In order to characterize the SRAM PUF, a specific firmware was developed to carry out the measurements by powering down and up the internal SRAM so as to extract automatically the start-up values. The internal SRAM is divided into three memories. In this work, the first 29,665 bytes (237,320 bits) of the last 100 KB

of the second memory were evaluated since they are enough for a statistical characterization. These bytes were not written or employed by the compiler to store execution variables. Three different boards and 120 measurements were considered. The ID and RND masks for each board were created with the first 20 measurements. The resting 100 measurements were employed for evaluation.

The minimum percentage of ID cells found was 84.07% (which means a minimum of 199,514 ID cells for the cells evaluated in each SRAM). Usually, most of the SRAM cells are ID cells. The PUF responses considered have a size of 2048 bits, so that 97 different responses per each board (291 responses in total) were evaluated. The similarity between PUF responses from the same cells is evaluated by the average intra fractional Hamming distance. The distribution of fractional Hamming distances calculated for responses from the same SRAM cells is known as intra fractional Hamming distance distribution in the PUF literature. For the measurements performed, the average intra fractional Hamming distance calculated was 0.25% (a value close to the ideal value of 0, which means that the PUF responses are equal). The number of intra Hamming distances calculated was 1,440,450 ($100 \cdot 99 \cdot 291/2$).

The decoder of the Error Correcting Code should cope with the noise of PUF responses to reconstruct, with no errors, the secret from the Helper Data. The bit flipping of a start-up value can be modeled essentially as a Bernoulli trial, which takes value '1' (if the bit changes) with probability p and a value of '0' (if the bit does not change) with probability $1 - p$. If the n bits obtained from the start-up values of n cells are assumed to be independent, the probability of finding t flipping bits (or errors) in them is given by a binomial distribution.

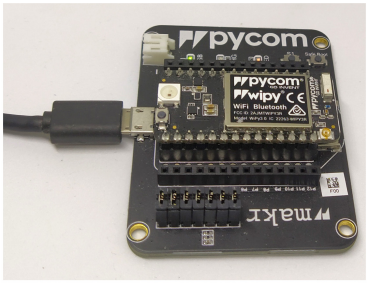
The *binocdf(t, n, p)* Matlab function was employed to compute the failure probability in reconstructing a bit of the secret when using an Error Correcting Code with n-bit codewords and capacity to correct up to t errors, with p estimated as the average intra fractional Hamming distance. An 8-bit repetition Error Correcting Code (with $n = 8$ and $t = 3$) gives a probability of failure in reconstructing a bit of the secret of 2.71e-9 (according to the operation 1-binocdf(3, 8, 0.0025)). The 8-bit repetition Error Correcting Code is selected since an error rate of 10^{-6} is considered by many authors as a conservative value that fulfills the requirements of most of typical security applications [23, 24]. The results shown herein have been obtained for nominal operation conditions (that is, nominal power supply voltage and ambient temperature). Of course, repetition Error Correction Codes with bigger words can be employed to ensure the adequate reconstruction of the secrets in any operation condition.

4.2 Development of an NFT with an SRAM PUF-Based BCA

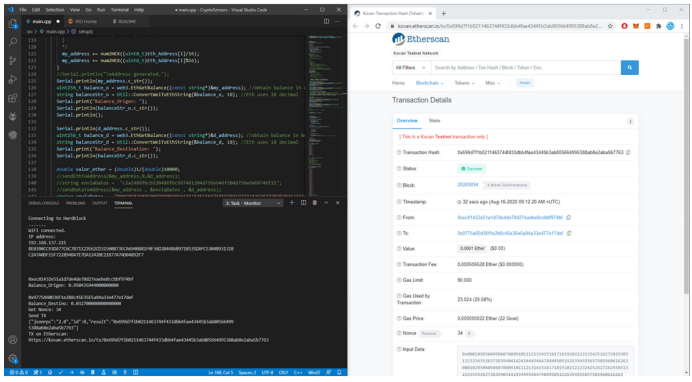
In this work, we used Kovan, which is an Ethereum public testnet, as Ethereum Virtual Machine (EVM) network. Ethereum is one of the most extended public blockchain and is part of the third generation of blockchains (which employs smart contracts). In Ethereum, secure transactions are based on the Elliptic Curve Cryptography (ECC). The Elliptic Curve Digital Signature Algorithm (ECDSA) represents a robust and lightweight signature scheme for constrained devices (such as IoT devices).

Several environments were employed to create the NFT. In the one side, the ESP-IDF (Espressif IoT Development Framework), which is the official development framework

for ESP32 microcontrollers, was employed to use the Pycom Wipy as the core of an IoT secure device based on SRAM PUFs. In the other side, the blockchain functionalities were performed by using the Web3E-alphawallet library to create a BCA and carry out transactions in PlatformIO; Remix to program in Solidity language and deploy smart contracts; and Etherscan to check transactions. Figure 4(a) shows the Wipy board which is connected to a laptop. Figure 4(b) shows a screenshot of a transaction which is executed and checked by using the development environments.



(a)



(b)

Fig. 4. (a) Real picture of the Wipy board. (b) Screenshot of a transaction performed through PlatformIO and checked through Etherscan.

The BCA of the SD is generated from a 256-bit secret seed obfuscated by the SRAM PUF response as explained above. Previously to the obfuscation, SRAM cells are classified to obtain the ID mask. Then, the seed obfuscation leads the ID mask application to obtain 2048 ID cells. The start-values of these cells are XOR-ed with the encoded Secret Seed (using the 8-bit repetition error correction encoder). The result is the 2048-bit Helper Data. The secret seed reconstruction needs the ID mask application to obtain the 2048 ID cells, the XOR operation with the Helper Data to obtain the 2048-bit coded seed and the 8-bit repetition error correction decoding to obtain the 256-bit decoded seed.

In order to create a BCA, private and public keys should be generated. A 256-bit private key is generated by applying a hash operation to the seed. A 64-byte public key results from applying the corresponding Elliptic Curve operation to the private key. In this work, we employ the *secp256k1* curve, which is the elliptic curve used in Ethereum. The BCA is obtained by applying the *Keccak256* operation to the public key and taking the most significant 20 bytes.

The different functions considered are performed by using the libraries provided by the development environments. The *Keccak256* function is obtained by using the *sha3.h* library from Trezor cryptographic library set within the Web3E-alphawallet library. A SHA3 context should be initiated with the *Keccak256* algorithm through the *keccak_256_Init* function. Subsequently, the context is updated with the data to be hashed and, finally, the results are obtained with the *keccak_final* function.

For the BCA creation, the *secp256k1* curve is obtained by using *contract.h* library from the Web3E-alphawallet library. The public key is obtained with the *PrivateKeyToPublic* function. The BCA is obtained with the *PublicKeyToAddress* function. ECDSA is obtained by using *ecdsa.h* library within the Trezor cryptographic library.

The transactions are performed as JSON structures under the Web3E framework. A contract context from the Web3E-alphawallet library is defined by indicating the private key, a nonce with the *SetPrivateKey* function, and the gas price and limit by the *EthGetTransactionCount* function. The transaction is launched by the *SendTransaction* function. This operation generates the JSON structure associated, the transaction signed by using ECDSA with the private key, and a transaction hash as output. The transaction message is sent to the smart contract through a blockchain transaction. The realization of a smart contract is a similar operation but employing the *SetupContractData* function.

The execution times of these operations are included in Table 5. The transaction completion time is the total time to generate a transaction and its transfer to the blockchain or smart contract. In our proposal, the transaction completion time is composed of the seed reconstruction, BCA generation and blockchain transaction times. This value is compared to the resulting transaction completion time obtained in [16] and [20]. The solution proposed in [16] also employs Ethereum blockchain. However, IoT device identifiers are based on certificates generated from a private key that is not obfuscated by PUFs. [20] considers an IoT device that creates hashes of data together with a key generated by a PUF for mining purposes. In contrast to our proposal, this solution does not employ a public and standard blockchain.

Through the simulation of the smart contract functions “createToken”, “transferFrom”, “userTransfer” and “completeTransfer”, the transaction gas consumption was evaluated. Table 6 illustrates the results obtained and shows a comparison with other similar functions proposed in [15] and [21]. The solution proposed in [15] does not employ PUFs and the device identifier is stored in the device-manager smart contract. The manager smart contracts verify information and decide whether a process can continue or not. Tokens are not directly related to the devices but to the tasks. Tokenization (which is not performed under the ERC-721 standard) is implemented by using a token-manager smart contract which is included in the task-manager smart contract. The task-manager smart contract provides a public register of available tasks related to the user, device, and tokens. Instantiations of users and devices are performed through the corresponding

Table 5. Execution times of the operations of a Secure Device based on PUFs

Operation		Execution time (ms)
Seed Obfuscation	SRAM cells classification	$3.8 \cdot 10^5$
	ID mask application, repetition error correction code and XOR operation	2.02
Seed Reconstruction	ID mask application, XOR operation and repetition error correction code	1.60
BCA Generation	256-bit private key generation (Keccak256 operation)	0.45
	64-byte public key generation (secp256k1 operation)	21.15
	20-byte BCA creation (Keccak256 operation)	0.45
Blockchain Transaction	Message preparation (configuration, ECDSA operation)	26.10
	Transfer to blockchain or smart contract	2.90
Transaction Completion in our proposal		52.65
Transaction Completion in [16]		69.0
Transaction Completion in [20]		192.30

manager smart contracts. In this way, manager smart contracts are only instantiated once, while the child (user and device smart contracts) are instantiated for each use. The gas values associated to the smart contract instantiation are included in Table 6. The solution in [21] provides a method for device traceability by using device authentication and ownership via blockchain smart contracts that do not employ NFTs explicitly. Device authentication is performed by PUF identifiers. However, the device has not capability to interact to the blockchain by a BCA associated to the PUF. The "createToken" function of our proposal, which can be compared to the *registerDevice* function of [21], consumes less gas. The standard "transferFrom" function of the ERC-721 NFT can be compared to the *transferOwnership* function of [21].

5 Conclusions

A solution for the secure management of IoT devices has been proposed. IoT devices are considered as Non Fungible Tokens (NFTs) based on the ERC-721 standard, which additionally include the BCA of the device user (not only the BCA of the device owner or manager) and the BCA of the IoT device. Device BCAs are generated from secret seeds obfuscated by Physical Unclonable Functions (PUFs). Any participant in the blockchain (including the IoT devices themselves because of their BCAs) can verify their manufacturers, owners, users, and the actions or data they are taking. Besides, the IoT devices, owners, and users are subscribed to the events of their associated NFTs so that they receive notifications about the situation of the NFT and behave accordingly.

Table 6. Gas consumption of smart contract functions

Proposal	Function	Gas consumption
Our	createToken	112,510
	transferFrom	34,272
	userTransfer	47,683
	completeTransfer	23,770
[15]	User	273,931
	User manager	530,579
	Device	446,652
	Device manager	1,097,206
	Token manager	413,560
	Task	554,883
	Task manager	3,052,709
[21]	registerDevice	121,478
	transferOwnership	30,365

The proposed device was implemented in a Pycom Wipy 3.0 board, proved with Kovan Ethereum testnet interacting with a smart contract programmed in Solidity, and verified with Remix. The SRAM PUF response employed has a size of 2048 bits to reconstruct secret seeds of 256 bits. The operations carried out by the Wipy board to generate the IoT device BCA and its employment in a transaction are carried out in a few tens of milliseconds. Smart contract functions are very simple. In fact, the gas consumption of the functions employed is low. A comparison is performed to other proposals in the literature in terms of execution times and gas consumption.

As future work, we plan to extend the proposal to provide security to the data generated by the IoT device, in terms of integrity, confidentiality, privacy, authentication, and provenance. In this way, not only the management of the IoT devices will be secure but also the storage and transmission of the data generated by these devices.

Acknowledgements. This work was supported in part by the Spanish Agencia Estatal de Investigación and Fondo Europeo de Desarrollo Regional (FEDER) under Projects TEC2017-83557-R and RTC-2017-6595-7, and Consejería de Economía, Conocimiento, Empresas y Universidad de la Junta de Andalucía under Projects AT17_5926_USE and US-1265146. The work of Rosario Arjona was supported by a Post-Doc Fellowship from the Spanish National Cybersecurity Institute (INCIBE).

References

1. Khan, M.A., Salah, K.: IoT security: review, blockchain solutions, and open issues. Future Gener. Comput. Syst. **82**, 395–411 (2018)

2. Stoyanova, M., Nikoloudakis, Y., Panagiotakis, S., Pallis, E., Markakis, K.: A survey on the Internet of Things (IoT) forensics: challenges, approaches, and open issues. IEEE Commun. Surv. Tutor. **22**, 1191–1221 (2020)
3. Buterin, V.: Ethereum whitepaper (2013). https://ethereum.org/whitepaper/. Accessed 19 Aug 2020
4. Christidis, K., Devetsikiotis, M.: Blockchains and smart contracts for the Internet of Things. IEEE Access **4**, 2292–2303 (2016)
5. Popov, S.: IOTA: feeless and free. IEEE Blockchain Technical Briefs (2019)
6. Prada-Delgado, M.A., Baturone, I., Dittmann, G., Jelitto, J., Kind, A.: PUF-derived IoT identities in a zero-knowledge protocol for blockchain. Internet Things **9** (2020)
7. Raiden Network. https://raiden.network/. Accessed 19 Aug 2020
8. ERC-721. http://www.erc721.org. Accessed 19 Aug 2020
9. Maes, R.: PUF-based entity identification and authentication. In: Maes, R. (ed.) Physically Unclonable Functions, pp. 117–141. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41395-7_5
10. Remix. https://remix.ethereum.org. Accessed 19 Aug 2020
11. Westerkamp, M., Victor, F., Küpper, A.: Tracing manufacturing processes using blockchain-based token compositions. Digit. Commun. Netw. **6**, 167–176 (2020)
12. Hasan, M., Binil, S.: Decentralized cloud manufacturing-as-a-service (CMaaS) platform architecture with configurable digital assets. J. Manuf. Syst. **56**, 157–174 (2020)
13. Valaštín, V., et al.: Blockchain based car-sharing platform. In: IEEE International Symposium ELMAR (2019)
14. Le, T., Yoohwan, K., Ju-Yeon, J.: Implementation of a blockchain-based event reselling system. In: 6th IEEE International Conference on Computational Science/Intelligence and Applied Informatics (CSII) (2019)
15. Wickström, J., Magnus, W., Göran, P.: Rethinking IoT security: a protocol based on blockchain smart contracts for secure and automated IoT deployments. arXiv preprint arXiv:2007.02652 (2020)
16. Khalid, U., Asim, M., Baker, T., Hung, P.C.K., Tariq, M.A., Rafferty, L.: A decentralized lightweight blockchain-based authentication mechanism for IoT systems. Cluster Comput. **23**, 2067–2087 (2020). https://doi.org/10.1007/s10586-020-03058-6
17. Sghaier Omar, A., Basir, O.: Capability-based non-fungible tokens approach for a decentralized AAA framework in IoT. In: Choo, K.-K.R., Dehghantanha, A., Parizi, R.M. (eds.) Blockchain Cybersecurity, Trust and Privacy. AIS, vol. 79, pp. 7–31. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38181-3_2
18. Cui, P., Dixon, J., Guin, U., Dimase, D.: A blockchain-based framework for supply chain provenance. IEEE Access **7**, 157113–157125 (2019)
19. Xu, X., Rahman, F., Shakya, B., Vassilev, A., Forte, D., Tehranipoor, M.: Electronics supply chain integrity enabled by blockchain. ACM Trans. Des. Autom. Electron. Syst. **24,** 1–25 (2019). Article 31
20. Mohanty, S.P., Yanambaka, V.P., Kougianos, E., Puthal, D.: PUFchain: A hardware-assisted blockchain for sustainable simultaneous device and data security in the Internet of Everything (IoE). IEEE Consum. Electron. Mag. **9**, 8–16 (2020)
21. Islam, M.N., Kundu, S.: Enabling IC traceability via blockchain pegged to embedded PUF. ACM Trans. Des. Autom. Electron. Syst. **24**, 1–23 (2019). Article 36
22. Baturone, I., Prada-Delgado, M.A., Eiroa, S.: Improved generation of identifiers, secret keys, and random numbers From SRAMs. IEEE Trans. Inf. Forensics Secur. **10**, 2653–2668 (2015)
23. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_5

24. Bösch, C., Guajardo, J., Sadeghi, A.-R., Shokrollahi, J., Tuyls, P.: Efficient helper data key extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 181–197. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_12