International Journal of Information Security (2021) 20:445–460
https://doi.org/10.1007/s10207-020-00513-8

REGULAR CONTRIBUTION

Anti-BlUFf: towards counterfeit mitigation in IC supply chains using blockchain and PUF

Leonardo Aniello[1] · Basel Halak[1] · Peter Chai[1] · Riddhi Dhall[1] · Mircea Mihalea[1] · Adrian Wilczynski[1]

Published online: 25 June 2020

Abstract
The complexity of today's integrated circuit (IC) supply chain, organised in several tiers and including many companies located in different countries, makes it challenging to assess the history and integrity of procured ICs. This enables malicious practices like counterfeiting and insertion of back doors, which are extremely dangerous, especially in supply chains of ICs for industrial control systems used in critical infrastructures, where a country and human lives can be put at risk. This paper aims at mitigating these issues by introducing Anti-BlUFf (Anti-counterfeiting Blockchain- and PUF-based infrastructure), an approach where ICs are uniquely identified and tracked along the chain, across multiple sites, to detect tampering. Our solution is based on consortium blockchain and smart contract technologies; hence, it is decentralised, highly available and provides strong guarantees on the integrity of stored data and executed business logic. The unique identification of ICs along the chain is implemented by using physically unclonable functions (PUFs) as tamper-resistant IDs. We first define the threat model of an adversary interested in tampering with ICs along the supply chain and then provide the design of the tracking system that implements the proposed anti-counterfeiting approach. We present a security analysis of the tracking system against the designated threat model and a prototype evaluation to show its technical feasibility and assess its effectiveness in counterfeit mitigation. Finally, we discuss several key practical aspects concerning our solution ad its integration with real IC supply chains.

Keywords Supply chain · Physically unclonable function · Blockchain · Smart contract · Counterfeit detection · Tracking

1 Introduction

Counterfeited ICs can lead to catastrophic consequences, in particular when they are used in critical infrastructure, military applications or in food and medicine industries. These include significant economic losses (e.g. in the order of billion USD per year in the UK [24]), serious security risks from malfunctioning military weapons and vehicles due to counterfeited parts [16], and potentially loss for human lives (e.g. deaths due to contaminated food, such as 2018 E. coli infection[1]). It is therefore of paramount importance to develop and deploy effective strategies for IC counterfeit mitigation to ensure a trustworthy and secure supply chain. One of main factors magnifying the scale of the counterfeit problem is the trend towards globalisation. The latter is driven by the need to cut costs to gain a competitive advantage and resulted in a remarkable growth of outsourcing levels, which in turn led to a significant increase of supply chains complexity because more firms are involved and the chain must be spread over further tiers [30]. Such an evolution of the supply chain structure has brought about a number of serious challenges linked to the problem of counterfeiting:

✉ Leonardo Aniello
l.aniello@soton.ac.uk

Basel Halak
basel.halak@soton.ac.uk

Peter Chai
xc5g15@soton.ac.uk

Riddhi Dhall
rd12g15@soton.ac.uk

Mircea Mihalea
mm8g15@soton.ac.uk

Adrian Wilczynski
aw11g15@soton.ac.uk

[1] Electronics and Computer Science, University of Southampton, Southampton, UK

[1] Multistate Outbreak of E. coli O157:H7 Infections Linked to Romaine Lettuce (Final Update), available online https://www.cdc.gov/ecoli/2018/o157h7-04-18/index.html.

– *Visibility* [15]. The network of buyer–supplier relationships has become more intricate and participants have little to no visibility and control on upstream stages, which makes it harder to assess the integrity of procured ICs.
– *Traceability* [21]. Tracking data is fragmented and spread among involved companies, which makes it very challenging to uniquely identify each procured IC and trace its history back to its origin and, in case of incidents, there is a shortage of data that can be used for forensics investigations.
– *Accountability* [14]. In such a scenario afflicted by obscurity and lack of information, fraudulent conduct of companies is noticeably facilitated. There is a lack of means to keep organisations accountable for the portion of processing they handle within the supply chain.

Coping with counterfeiting in these IC supply chains calls for a platform integrated throughout the whole chain to reliably record every transition of products between involved companies. The availability of such a ledger would be an effective means to provide any legitimate actor with precise information on what organisations are operating at upstream stages of the chain (*visibility*) and on the history of each procured IC (*traceability*). Moreover, ensuring recorded transactions are truthful and not tampered with is crucial to enable legally binding liability policies (*accountability*). The implementation of such a platform for counterfeit mitigation requires an infrastructure deployed over the considered supply chain, to enable fine-grained monitoring of ICs sold and bought by involved companies. It would be infeasible to identify a single specific authority or enterprise eligible for controlling and operating an infrastructure like this, possibly spanning different countries and diverse regulatory frameworks. Furthermore, such an authority should be trusted globally and have the resources to effectively setup and maintain such a world-wide, complex interconnected network, ensuring at the same time top levels of security, availability and performance.

A decentralised approach is more suitable, where the infrastructure itself is a peer-to-peer network distributed across all the supply chain partners, devoid of any centralised control that may become a single point of failure or a performance bottleneck. An emerging technology that lends itself well to implement a platform like that is the *blockchain*, because of its full decentralisation, high availability and strong guarantees on the immutability of stored data. In brief, a blockchain is a distributed system consisting of a network of peer nodes sharing a ledger of transactions, where each peer keeps a replica of that ledger. The consistency among replicas is ensured by a distributed consensus algorithm run by all the nodes, which also guarantees that transactions cannot be censored or redacted unless an attacker succeeded in controlling a certain percentage of nodes or of computational power. In addition to storing data, blockchain can be used to execute application logic through the *smart contract* technology. A smart contract is an application whose code and execution traces are stored immutably in the blockchain, which provides strong guarantees on execution integrity.

Since such infrastructure has to be run across a predefined set of parties, and considering that part of managed data is not meant to be disclosed publicly, it is reasonable to not rely on existing public permissionless blockchains like Ethereum. Rather, it is more sensible to build on a *consortium blockchain* where nodes are authenticated, membership is predetermined and data cannot be accessed from the outside.

In this paper, *we introduce Anti-BlUFf (Anti-counterfeiting Blockchain- and PUF-based infrastructure), an approach based on consortium blockchain and smart contract technologies for item tracking and counterfeit detection in IC supply chains* . Items, i.e. ICs, are uniquely identified to enable tracking by using tamper-proof tags. We choose to use *physically unclonable functions* (PUF) to implement those tags. PUFs are circuits that provide unique signatures deriving from manufacturing process variations of the circuits themselves. Each alteration of those tags leads to changes of the function computed by the PUF; hence, this technology is well suited to enable counterfeit detection. We provide the design of a supply chain management system based on the proposed approach and carry out a preliminary analysis on its effectiveness and feasibility. We define the adversary model to characterise what types of threats can arise in the context of supply chain counterfeit. We then analyse how the proposed design can address those threats to deliver improved counterfeit detection. Finally, to show the technical feasibility of this solution, we describe its prototype implementation and preliminary experimental evaluation, where we measure the effectiveness of using PUFs for counterfeit detection. Finally, we provide an ample discussion on some key pragmatic aspects of integrating the proposed platform with real supply chains.

Although some other blockchain-based IC supply chain management systems have been proposed in literature and industry, a few of them rely on PUFs for item tracking. The main novelty of this work lies in presenting a more complete solution that encompasses (i) the integration of PUF and consortium blockchain, (ii) the detailed description of smart contract implementation and how PUF data is stored in the blockchain and (iii) a security analysis against a threat model.

Our Contribution In this paper, we rely on blockchain, smart contract and PUF technologies to design a tracking system of ICs for supply chain management, aimed at mitigating the problem of counterfeiting. With respect to the state of the art on this topic, our main research contributions are

- the explicit modelling of the overall system, including IC supply chain, blockchain, smart contracts, PUFs and adversary behaviour, i.e. the *threat model*;
- the detailed design of the proposed tracking system for detecting counterfeits in IC supply chains;
- based on the designated threat model, the identification of the possible attacks to the tracking system aimed to bypass counterfeit detection;
- the analysis of how the proposed tracking system reacts against each of the identified attacks;
- a prototype implementation and preliminary experimental evaluation of the proposed tracking system, where PUF-based counterfeit detection accuracy is assessed;
- a discussion on most relevant points concerning the integration of our solution in real scenarios.

Paper Organisation The remainder of this paper is organised as follows. Section 2 describes related work. Section 3 introduces background information on PUF, blockchain and smart contract technologies. The system model is presented in Sect. 4, as well as the threat model. Our tracking system is detailed in Sect. 5, and its security properties are analysed in Sect. 6. Section 7 describes the prototype implementation and evaluation. Section 8 discusses security analysis results and the limitations of our solution. Finally, Sect. 9 outlines conclusion and future work.

2 Related work

The use of blockchain and smart contracts for supply chain management is currently being investigated in some recent industrial projects,[2] [3] and led to the launch of a number of new businesses and companies, which supports the perceived potentialities of this application. Some of these projects use a blockchain-as-a-service solution provided by a third party, such as TradeLends,[4] which employs the platform delivered by IBM Cloud. The limitation of such an approach is the need to totally trust an external organisation, which brings about the same issues mentioned before regarding centralisation.

Different companies use diverse technologies to tag products and reliably link physical assets to the blockchain. Waltonchain[5] uses RFID (Radio-frequency identification) as tags to identify and track items along the chain. Others make use of proprietary solutions. For example, BlockVerify[6] uses their own Block Verify tags, Chronicled[7] employs trusted IoT chips, Skuchain [8] applies Proof of Provenance codes called Popcodes. The problem of existing approaches that rely on the use of RFID-based tags is that these tags are vulnerable to cloning attacks [17,19], this makes it less effective in protecting against counterfeit attempts.

RFID are also proposed by Toyoda et al. [27]. They introduce a blockchain-based solution for product ownership management system, to be used to prevent counterfeits in the post supply chain. They explain how their system allows to detect counterfeits, and discuss the provided security guarantees only in terms of the possible vulnerabilities of the underlying technology they use, i.e. Ethereum.[9]

Alzahrani and Bulusu [2] propose a solution based on Near Field Communication (NFC). They present Block-Supply Chain, a design for a consortium blockchain-based supply chain where products are tracked using NFC technology to detect counterfeits. Their security analysis is limited to the novel consensus protocol they propose and does not take into account any other aspect of the overall supply chain ecosystem, which includes, but is not restricted to, the blockchain. Furthermore, they do not define a threat model to specify what attacks they want to defend from.

We propose to produce tamper-proof tags by using physically unclonable functions (PUF), i.e. circuits that can generate a unique identifier for each chip due to the intrinsic variability of the IC fabrication process. Previously reported works on using PUF technology in the context of IC supply chain management are limited in both scope and depth. Guardtime [10] proposes the use of PUF for IoT device authentication, based on a consortium blockchain (i.e. KSI Blockchain). However, they provide no clear information on the integration with supply chain, they do not explain how PUF data is stored and do not provide any security analysis.

Islam et al. [18] propose the use of PUF and consortium blockchain for tracing ICs. Their work does not investigate in depth what security guarantees are provided and gives no description of the way PUF data is stored in the blockchain.

Similarly, Negka et al. [23] describe a method to detect counterfeit IoT devices by tracking each single device component along the supply chain. They rely on PUFs to authenticate components and implement their detection logic in Ethereum. Although they provide some figures on the fees

[2] How Blockchain Will Transform The Supply Chain And Logistics Industry (https://www.forbes.com/sites/bernardmarr/2018/03/23/how-blockchain-will-transform-the-supply-chain-and-logistics-industry).

[3] Using blockchain to drive supply chain transparency (https://www2.deloitte.com/us/en/pages/operations/articles/blockchain-supply-chain-innovation.html).

[4] TradeLends, available online https://www.tradelens.com/.

[5] Waltonchain https://www.waltonchain.org/doc/Waltonchain-whitepaper_en_20180208.pdf.

[6] BlockVerify: Blockchain Based Anti-Counterfeit Solution, Introducing transparency to supply chains http://www.blockverify.io/.

[7] Chronicled: Trusted Internet of Things and Smart Supply Chain Solutions, Secure identities, trusted IoT data, and automated business logic https://www.chronicled.com/.

[8] Skuchain: Turn Information Into Capital http://www.skuchain.com/

[9] Ethereum Project (https://www.ethereum.org/).

Table 1 Comparison of Anti-BlUFf with state of the art in blockchain-based anti counterfeit approaches for supply chains

Proposed solution	Tag type	Blockchain type	Counterfeit detection approach	Security analysis
Toyoda et al. [27]	RFID	Ethereum	Smart contracts pseudo-code provided	Yes
Block-Supply Chain [2]	NFC	Consortium	No details are provided on how the smart contract is implemented	No
Guardtime [10]	PUF	Consortium	No details on integration with blockchain, no info on how PUF data is stored in the blockchain, no details are provided on how the smart contract is implemented	No
Islam et al. [18]	PUF	Consortium	No info on how PUF data is stored in the blockchain, no details are provided on how the smart contract is implemented	No
Negka et al. [23]	PUF	Ethereum	No info on how PUF data is stored in the blockchain, no details are provided on how the smart contract is implemented	No
Anti-BlUFf	PUF	Consortium	Smart contract pseudo-code provided, as well as details on how PUF data is stored in the blockchain	Yes

to pay to use Ethereum smart contracts, they do not detail how PUFs and smart contracts are integrated, nor what specific mechanism is actually employed to implement the detection. Obtained detection accuracy and provided security guarantees are not discussed.

To the best of our knowledge, the lack of appropriate security analysis of proposed solutions is currently a gap in the state of the art on the application of blockchain and PUF technologies for counterfeiting mitigation in IC supply chains. Table 1 details how our solution, Anti-BlUFf, compares with respect to the related work considered in this section. Anti-BlUFf is the only proposed approach that at the same time (i) relies on PUF and consortium blockchain, (ii) gives details on smart contract implementation and how PUF data is stored in the blockchain and (iii) includes a security analysis against a threat model.

3 Preliminaries

In this section we introduce some preliminary background on physically unclonable functions (Sect. 3.1) and blockchain and smart contract technology (Sect. 3.2).

3.1 Physically unclonable function

Physically unclonable functions (PUF) are security primitive capable of generating a hardware-based digital signature unique for each device [13]. PUFs are commonly implemented as circuits and ensure that responses are different for each hardware by exploiting the inherent randomness of the internal structure introduced by the manufacturing process. This technology has many attractive advantages, including its relatively low cost (a typical PUF can be built using few thousands transistors), and its inherent security deriving from the extreme difficulty of forging its design. Indeed, it is almost impossible to create a physical clone of a PUF, which means that this technology can be used reliably to identify those physical objects where a PUF can be integrated, and therefore to detect possible forgery. From a mathematical point of view, a PUF is a function that generates an output (also called *response*) starting from an input (also called *challenge*). The challenge–response data (CRD) must be unique for a single device. The use of PUF for building entity-authentication protocols has been extensively explored in the literature [6,31,32]. In general, each entity is provided with a PUF and the authentication scheme consists of two stages [11]:

1. *Enrolment phase* when a new entity has to be enrolled, a verifier collects the required CRD from entity's PUF and stores it in a database, together with the ID of the entity itself.
2. *Verification phase* when an enrolled entity has to be authenticated, the verifier receives the entity ID and retrieves the corresponding CRD from the database. A random challenge–response pair is selected from the CRD and the challenge is sent in clear to the entity, which computes the response by using its PUF and sends it back in clear to the verifier. If the response corresponds to that stored in the database, then the authentication is successful and the challenge–response pair is removed

from stored CRD to prevent replay attacks. Otherwise, the authentication fails.

Ideally, a PUF should always generate the same response for a given challenge. Unfortunately, conditions such as temperature or voltage variations could lead to different responses [12].

A PUF can be implemented in different ways and with different technologies, leading to varying security guarantees. For example, PUFs based on SRAM have been proved to be clonable [12], which questions their suitability to be used to implement authentication protocols. It has been also shown that a PUF can be vulnerable to machine learning (ML)-based modelling attacks [12], where an adversary builds an accurate mathematical model of the PUF by collecting a sufficient number of challenge–response pairs, and uses that model to clone the PUF itself. There are a number of techniques that can be used to mitigate the risks of ML-based attacks, such as using cryptographic blocks to obfuscate the output of the PUF [22], increasing the circuit complexity of the design [26], or solving this issue at the protocol level [32,33].

3.2 Blockchain and smart contract

A blockchain is a ledger of transactions, replicated among a number of nodes organised in a peer-to-peer network. Transactions are submitted to the blockchain network and stored in the ledger. A consensus algorithm is run among blockchain nodes to guarantee the consistency of the ledger, in terms of what transactions are included in which order. A blockchain provides strong guarantees in terms of availability, because a peer-to-peer network with several nodes and no single-point-of-failure is used. Furthermore, as the ledger is replicated and several nodes participate in the consensus algorithm, an adversary should take control of a relevant fraction of nodes to take over the blockchain and tamper with the ledger. That fraction of nodes depends on the chosen consensus algorithm.

In open, *permissionless blockchains* like Bitcoin[10] and Ethereum, any node can join the network without any form of authentication; hence, additional mechanisms are required to cope with the potential presence of malicious nodes. Proof-of-Work (PoW) is commonly employed, which, although effective in countering cyber threats stemming from malicious blockchain nodes, is time-consuming and greatly restricts performance [28]. In *consortium blockchains* like Hyperledger Fabric,[11] blockchain membership is restricted to the nodes owned by interested organisations, so that each involved firm can take part to the overall process and no external actor can interfere with any operation or read any exchanged data. In this way, blockchain nodes are known and can be reliably authenticated, which allows to replace PoW with other, more efficient techniques that ensure high level performance in terms of latency and throughput, such as byzantine fault tolerance algorithms [5].

On top of a blockchain, a smart contract execution environment can be built, to extend the functionalities of the blockchain beyond storing data and allow the execution of any application logic. A smart contract is the code implementing the required application logic and it can be installed in a blockchain likewise a normal transaction, which ensures consequently its integrity. A smart contract defines an interface with methods that can be called externally. Each invocation of a smart contract method is stored as a blockchain transaction; hence, the execution trace can be considered as immutable. In general, computations executed through smart contracts are fully transparent and tamper-proof.

4 System model

This section defines the system model representing supply chain (Sect. 4.1), PUF-equipped items (Sect. 4.2), blockchain and smart contracts (Sects. 4.3 and 4.4, respectively). Finally, thread model is introduced in Sect. 4.5.

4.1 Supply chain model

An IC supply chain $\mathcal{SC}$ includes N parties $\mathcal{P} = \{p_i\}$, i.e. organisations involved in the chain with different roles, and that engage among themselves by supplying and buying items, i.e. ICs. A *supplier* is a party that provides items, while a *buyer* is a party that receives items. Each party can act at the same time as supplier for a number of buyers and as buyer for diverse suppliers. There can be parties that are neither suppliers nor buyers for any other party but operate anyway in the supply chain, such as auditors or regulators. This kind of parties usually needs to access tracking data to assess compliance and solve disputes.

We model $\mathcal{SC}$ as a directed acyclic graph $(\mathcal{P}, \mathcal{R})$, where $\mathcal{R}$ is the set of binary supplier–buyer relationships holding within $\mathcal{SC}$. Figure 1 shows an instance of the supply chain model. Each element of $\mathcal{R}$ is in the form (p_i, p_j), with $p_i, p_j \in \mathcal{P} \wedge p_i \neq p_j$, and represents a supplier–buyer relationship where p_i is the supplier and p_j the buyer. According to these relationships, parties can be organised in stages, i.e. the stages of the supply chain. Let S be the number of stages

[10] Bitcoin (https://bitcoin.org/en/).

[11] Hyperledger Fabric (https://www.hyperledger.org/projects/fabric).

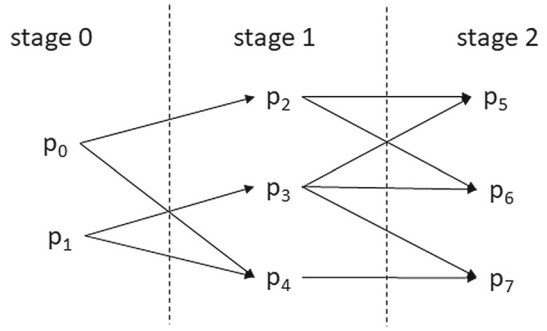


Fig. 1 Example of supply chain with 8 parties $p_0, \ldots, p_7$ spread across 3 stages. The arrows represent the supplier–buyer relationships, e.g. (p_2, p_5) models the fact that p_2 is a supplier of p_5

of $\mathcal{SC}$. Without loss of generality,[12]. we define the function $stage\colon \mathcal{P} \rightarrow \mathbb{N}$ as follows

$$stage(p) = \begin{cases} 0 & \text{iff } \nexists q \in \mathcal{P} \mid (q, p) \in \mathcal{R} \\ i + 1 & otherwise \end{cases} \tag{1}$$

where $i = \max_{q \in \mathcal{P} \mid (q,p) \in \mathcal{R}} stage(q)$.

Equation 1 computes the stage of a party p in the supply chain by recursively identifying the supplier of p operating at the highest stage, i.e. $\max_{q \in \mathcal{P} \mid (q,p) \in \mathcal{R}} stage(q)$. Trivially, the stage of p is one unit higher than the stage of that supplier. If instead p has no supplier (i.e. $\nexists q \in \mathcal{P} \mid (q, p) \in \mathcal{R}$), this means that p operates at stage 0. Although equation 1 covers the cases where a buyer has suppliers in different stages, this is not likely to happen in real supply chains. Indeed, buyers commonly purchase items from parties in the previous stage only. Therefore we introduce the following constraint

$$\forall (p, q) \in \mathcal{R} \quad stage(q) - stage(p) = 1 \tag{2}$$

We assume the existence of a reliable public key infrastructure (PKI) for the parties in $\mathcal{P}$. Each party p_i has a key pair (pk_i, sk_i), where pk_i is the public key known to all the other parties and sk_i is the private key known to p_i only. We discuss in Sect. 8 how such a PKI can be realised and the related issues. Given a key k and a plaintext message m, we indicate with $|m|_k$ the ciphertext derived from encrypting m with k. We use $\langle m \rangle_{\sigma_i}$ to indicate that the message m has been signed by p_i, i.e. that it includes a digest of m encrypted with sk_i.

4.2 PUF-equipped item model

A number of items are moved along the supply chain $\mathcal{SC}$, from parties at stage 0 to downstream parties. We refer to the generic i^{th} PUF-equipped item produced at stage 0 of the supply chain as x_i, and to the party that produced it as its *producer*. Furthermore, as items can be forged along the chain, we define x_i^s as the item x_i *after its processing* at stage s, where $s = 0 \ldots S-1$. That is, x_i^s is the item x_i when it is delivered from the supplier at stage s to the buyer at stage $s+1$ (see Fig. 2).

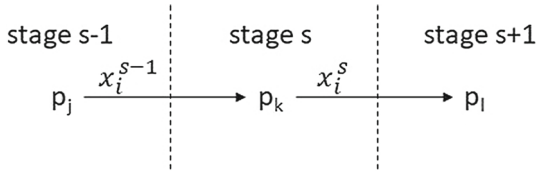


Fig. 2 Example of item x_i moving from p_j in stage $s-1$ to p_k in stage s (x_i^{s-1}), and from there to p_l in stage $s+1$ (x_i^s)

We refer to the function computed by the PUF integrated with item x_i^s as $puf_i^s : \mathbb{N} \rightarrow \mathbb{N}$. When an item x_i is produced at stage 0 and equipped with a PUF, it is considered intact.

If x_i is never tampered with along the chain, then the following property holds with high probability.[13]

$$\forall c \in \mathbb{N} \quad \forall s \in [1 \ldots S-1] \quad puf_i^0(c) = puf_i^s(c) \tag{3}$$

If instead x_i is forged at stage $s > 0$, then $puf_i^0 \neq puf_i^s$ and the following property holds with high probability[14]

$$\forall c \in \mathbb{N} \quad puf_i^0(c) \neq puf_i^s(c) \tag{4}$$

The fact that Eqs. 3 and 4 do not hold with 100% probability can be accounted for by querying the PUF more times, in order to increase that probability exponentially. We consider the case where PUFs are built by using techniques that mitigate the risk of ML-based attacks 3.1; hence, we assume that an adversary cannot clone a PUF by collecting a sufficient number of challenge–response pairs.

4.3 Blockchain model

We consider a *consortium blockchain* $\mathcal{B}$ with N nodes $\mathcal{N} = \{n_i\}$, deployed over the supply chain parties' premises (see Sect. 3.2). More precisely, node n_i is located at party p_i. Nodes can communicate among each other over the network by sending messages. The network is asynchronous, there is no known bound on message latencies but messages are eventually delivered to their destination. $\mathcal{B}$ uses a *byzantine fault tolerant consensus* protocol, such as PBFT [5], which

[12] It would be possible for an organisation to operate at different stages of a supply chain. In these cases, we model such an organisation as multiple parties, one for each stage where it operates.

[13] As explained in Sect. 3.1, the function computed by a PUF is not 100% stable. An in-depth discussion about PUF stability can be found in [12].

[14] Even if the two functions are different, they might return the same response for some challenge.

ensures *safety* if up to $f = \lfloor \frac{N-1}{3} \rfloor$ nodes are byzantine. Section 4.5 will explain how byzantine nodes behave.

Interactions between nodes take place by sending digitally signed messages. When a node n_i wants to send a message m to another node n_j, n_i sends a message $\langle i, j, ts, m \rangle_{\sigma_i}$ to n_j. The parameter ts is a timestamp set by n_i, used to avoid replay attacks.

Clients running within supply chain parties' premises can submit transactions to $\mathcal{B}$ by broadcasting them to all $\mathcal{B}$'s nodes. Submitted transactions are eventually confirmed by $\mathcal{B}$ and persistently stored, with strong guarantees on their immutability, i.e. persisted transactions cannot be tampered with or removed unless more than $f = \lfloor \frac{N-1}{3} \rfloor$ nodes are byzantine.

4.4 Smart contract execution environment model

Consortium blockchains like those described in Sect. 4.3 can support the execution of smart contracts (see Sect. 3.2), i.e. a smart contract execution environment $\mathcal{SCEE}$ can be built on top of a consortium blockchain $\mathcal{B}$. $\mathcal{SCEE}$ is deployed over the same nodes $\mathcal{N}$ of $\mathcal{B}$.

Smart contracts can be installed in $\mathcal{SCEE}$. A smart contract $\mathcal{C}$ includes a number of methods, which can be invoked externally, and a key-value store kvs, which can be accessed internally only, inside those methods. The installation of a smart contract $\mathcal{C}$ in $\mathcal{SCEE}$ and every invocation of $\mathcal{C}$'s methods are persisted as transactions submitted to the underlying blockchain $\mathcal{B}$. This implies that the application logic encoded by a smart contract cannot be tampered with as long as the underlying blockchain $\mathcal{B}$ guarantees immutability, i.e. unless more than $f = \lfloor \frac{N-1}{3} \rfloor$ nodes are byzantine.

The key-value store of each smart contract provides an interface $set(k, v)$ and $get(k)$ to set and get values for given keys, respectively. Any internal key-value storage kvs relies on the underlying blockchain $\mathcal{B}$ to ensure consistency and immutability of its state. In the specific, each set operation invoked through the $set(k, v)$ method is saved as a transaction in $\mathcal{B}$; hence, the whole redo log of the storage is persisted immutably [9]. Furthermore, we assume that a single set operation is allowed for each key, i.e. the value stored for a key cannot be overwritten. In case of overwriting attempt, the set operation returns an error. External applications can also register themselves to receive notifications when specific types of transactions are committed, in order to implement callback-based application logic.

In the considered scenario, there is also the need to verify the identity of the entity that invokes a smart contract method, in order to make sure that the invoker is actually authorised to call the method. We assume that each method invocation includes an additional input parameter that proves the identity of the invoker. In particular, this parameter is the invoker's digital signature of the concatenation of all the other input parameters, plus a timestamp to avoid replay attacks. In the following, we do not explicitly include this additional parameter in the pseudo-code of smart contracts in order to keep them as light as possible. However, we specify what actors are expected to invoke each method, and the corresponding verification is assumed to be carried out by relying on this additional parameter.

4.5 Threat model

The final goal of the adversary is to tamper with items to introduce counterfeit ICs in the supply chain. Hence, it aims at avoiding that counterfeit items are detected to prevent raising suspicion. We assume the existence of a single adversary in the supply chain, Sect. 8 encompasses a brief discussion on considering the presence of more independent adversaries.

At *supply chain level* (see Sect. 4.1), the adversary can operate at one of the parties, say p_A at stage $stage(p_A)$, with $A \in [0 \ldots N-1]$. We assume that the adversary cannot control more than one party and cannot alter any supplier–buyer relationship.

At *item level* (see Sect. 4.2), the adversary can tamper with items during the manufacturing processes of the party p_A where it operates. For each bought item $x_i^{stage(p_A)-1}$, the adversary can decide whether or not to forge it before supplying it in turn to some other party. However, any tampering with $x_i^{stage(p_A)-1}$ affects the internal structure of the integrated PUF, hence $puf_i^0 \neq puf_i^{s_A}$ (see equations 3 and 4). Furthermore, if the adversary succeeds to collect at least N_{PUF} challenge–response pairs, it can build a clone of the PUF and attach it to a different item, i.e. it can replace an original product with a counterfeit.

At *blockchain and smart contract execution environment levels* (see Sects. 4.3 and 4.4), the adversary can control the local node n_A of $\mathcal{B}$ and $\mathcal{SCEE}$, i.e. such node is byzantine. The behaviour of a byzantine node can deviate arbitrarily from the expected conduct; hence, it can, for example, drop messages and send not expected or wrong messages. Anyway, the adversary cannot break used cryptographic protocols; hence, it cannot decrypt messages encrypted without knowing the corresponding keys and cannot forge message signatures.

5 Tracking system

Items are tracked as they move along the supply chain, first when they are produced at stage 0 and then each time they are supplied to a buyer operating at the next stage. When delivered at buyer side, the integrity of each item is verified by using its integrated PUF. Tracking information are stored as blockchain transactions to ensure they are immutable and available to any party in $\mathcal{P}$.

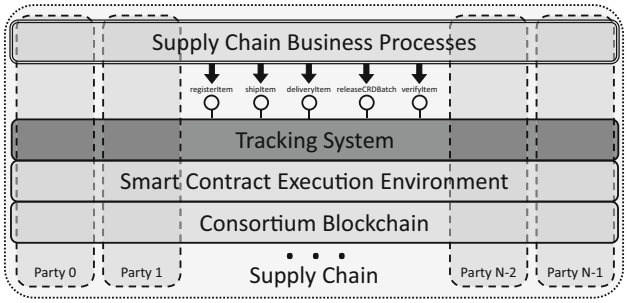


Fig. 3 High-level architecture of the tracking system and its integration within supply chain business processes

The tracking system is built as a smart contract $\mathcal{TS}$ on top of a blockchain-based smart contract execution environment $\mathcal{SCEE}$ (see Sect. 4.4). We consider a consortium blockchain $\mathcal{B}$ like the one presented in Sect. 4.3, and leverage on the PUFs integrated with the items to assess whether they have been tampered with (see Sect. 4.2). The high-level architecture is shown in Fig. 3, where basic building blocks and interfaces with supply chain business processes are highlighted. Consortium blockchain $\mathcal{B}$, smart contract execution environment $\mathcal{SCEE}$ and tracking system $\mathcal{TS}$ are distributed and deployed over the IT infrastructures of all the parties.

Module 1 shows the pseudo-code of the tracking system, which defines the five methods shown in Fig. 3. These methods are used to integrate the proposed tracking mechanism with the business processes of the supply chain. In particular, this integration occurs on three specific events: when an item is first introduced in the supply chain at stage 0 (*event 1*, see Sect. 5.1), when a supplier ships an item to a buyer (*event 2*, see Sect. 5.2) and when an item is verified by a buyer (*event 3*, see Sect. 5.3). After an item has been processed by a party in the last stage, no further tracking is enforced. However, consumers can still verify items they buy asking the corresponding producers to release additional batches of CRDs.

All tracking data is kept in the blockchain-based key-value storage via set operations, where any relevant information is digitally signed (see Sect. 4.1) by the party executing the method where the set operation itself is invoked. This, together with the constraint that keys cannot be overwritten and method invocations are authenticated (see Sect. 4.4), ensures that an adversary cannot execute any tracking system method on behalf of another party.

In order to integrate the business processes of supply chain $\mathcal{SC}$ with the tracking system $\mathcal{TS}$, an additional layer is required to interface the existing legacy business process management software of $\mathcal{SC}$ with the $\mathcal{TS}$ smart contract. This integration can be achieved through standard software engineering approaches and does not entail any element of novelty or challenge, so it is not described here. However, this integration layer needs to be accounted for as another potential attack surface that the adversary may exploit; hence, in Sect. 6 we also address the corresponding security implications (attack 4).

5.1 Event 1: New item

When a new item x_i is produced by a party p_j at stage 0, a PUF is integrated with x_i and $B \times C$ challenge–response pairs $\langle c_k, r_k \rangle$ are collected. C challenges will be used for each item verification, which makes it more robust against possible variations in the responses generated by a PUF (see Sect. 3.1). Hence, up to B parties can verify the integrity of an item at delivery time. B has to be set sufficiently large to accommodate for verifications requested by supply chain parties, end users and external auditors.

The set of pairs is partitioned in B disjoint batches b_w, with $w = 0...B-1$, each containing C pairs. Each challenge–response pair for batch w is produced by generating a unique random challenge $c_{w,k} \in \mathbb{N}$, giving it as input to the PUF of x_i and recording the corresponding output $r_{w,k} = puf_i^0(c_{w,k})$.

We refer to the vector of batches of challenge–response pairs as the *challenge–response data* CRD_i of x_i, i.e. $CRD_i = [b_0, ..., b_{B-1}]$.

CRDs are not disclosed forthwith to all the other parties, otherwise an adversary could develop an ad-hoc circuit to provide correct responses to expected challenges, which could then be used to introduce counterfeits. Rather, at this stage the producer discloses a hashed version of CRD_i, referred to as $hashedCRD_i$, which is a vector of B pairs $\langle hashedC_w, hashedR_w \rangle$, where $hashedC_w = hash(c_{w,0}, ..., c_{w,C-1})$ and $hashedR_w = hash(r_{w,0}, ..., r_{w,C-1})$, i.e. each pair contains (i) the hash of the concatenation of all the challenges of the batch and (ii) the hash of the concatenation of all the responses of the batch. The method $registerItem()$ is invoked after the generation of the CRD. This method simply stores in the key-value storage the information that $hashedCRD_i$ is available and has been produced by party p_j (line 5). Furthermore, p_j registers itself to be notified (see Sect. 4.4) whenever a delivery transaction for x_i is stored into the blockchain (see Sect. 5.3).

In order to prevent that any two items in the whole supply chain could clash in the key-value store, the key used to store hashed CRDs also includes the producer party's identifier. The latter has to ensure that no two items are assigned the same identifier among those it registers .

5.2 Event 2: Item shipping

When a party p_s finishes the manufacturing processes of an item $x_i^{stage(p_s)}$ and supplies it to a buyer p_b operating at the next stage, the procedure $shipItem()$ is invoked. Likewise $registerItem()$, this method simply tracks in the blockchain

Module 1 Tracking Mechanism (*continue on next page*)

global variables:
1: C ▷ number of challenges to send for each verification
2: R ▷ number of responses (out of C) that need to be correct for the verification to succeed
3: kvs ▷ local key-value storage instance

▷ *This method is called by the item producer; p is the item producer identifier, i is the item identifier, $hashedCRD_i$ its hashed challenge-response data*
4: **method** REGISTERITEM($p, i, hashedCRD_i$)
5: $kvs.set(\langle registered, p, i\rangle, crd_i)$
6: **end method**

▷ *This method is called by the seller; p is the item producer identifier, i is the item identifier, s is the seller identifier and b the buyer identifier*
7: **method** SHIPITEM(p, i, s, b)
8: **if** $kvs.get(\langle registered, p, i\rangle) == null$ **then**
9: $kvs.set(\langle notRegistered, p, i\rangle, \langle p, i\rangle)$
10: **else if** $s \neq p \wedge kvs.get(\langle deliver, p, i, s\rangle) == null$ **then**
11: $kvs.set(\langle notDelivered, p, i, s\rangle, \langle p, i\rangle)$
12: **else**
13: $kvs.set(\langle shipped, p, i, s, b\rangle, \langle p, i\rangle)$
14: **end if**
15: **end method**

▷ *This method is called by the buyer; p is the item producer identifier, i is the item identifier, s is the seller identifier and b the buyer identifier*
16: **method** DELIVERYITEM(p, i, s, b)
17: **if** $kvs.get(\langle registered, p, i\rangle) == null$ **then**
18: $kvs.set(\langle notRegistered, p, i\rangle, \langle p, i\rangle)$
19: **else if** $kvs.get(\langle ship, p, i, s, b\rangle) == null$ **then**
20: $kvs.set(\langle notShipped, p, i, s, b\rangle, \langle p, i\rangle)$
21: **else**
22: $kvs.set(\langle delivered, p, b, i\rangle, s)$
23: **end if**
24: **end method**

▷ *This method is called by the item producer; p is the item producer identifier, i is the item identifier, w is the batch index and $crdBatch$ the w^{th} batch of challenge-response pairs for item i, where each response is hashed*
25: **method** RELEASECRDBATCH($p, i, w, crdBatch_{i,w}$)
26: **if** $w > 0 \wedge kvs.get(\langle crdBatchReleased, p, i, w-1\rangle) == null$ **then**
27: $kvs.set(\langle invalidBatchID, p, i, w\rangle, crdBatch_{i,w})$
28: **else**
29: $hashedCRD_i = kvs.get(\langle register, p, i\rangle)$
30: **if** $hashedCRD_i[w].hashedC_w \neq hash(crdBatch_{i,w}.challenges)$ **then**
31: $kvs.set(\langle invalidBatch, p, i, w\rangle, crdBatch_{i,w})$
32: **else**
33: $kvs.set(\langle crdBatchReleased, p, i, w\rangle, crdBatch_{i,w})$
34: **end if**
35: **end if**
36: **end method**

the fact that item x_i, produced by party p_p, has been shipped from party p_s to party p_b. At line 13 of module 1, all the relevant shipping information are included in the key to make it easier to retrieve shipping data. The value, i.e. the second parameter of the set operation, is not significant and is set to $\langle p, i\rangle$ by convention. Indeed, when querying the blockchain on whether the shipping of item x_i, produced by p_p, from party p_s to party p_b took place, it suffices to check that the value stored for the key $\langle shipped, p, i, s, b\rangle$ is not null.

The method $shipItem()$ also queries the key-value store to perform checks regarding the registration of x_i by p_p and, in case $p_p \neq p_s$, whether x_i has been previously delivered to p_s.

5.3 Event 3: Item delivery and verification

When an item is delivered to a party p_b from a supplier p_s operating at the previous stage, an integrity verification is carried out. This process includes three steps, each corresponding to a different method: (i) the buyer first notifies that the item has been delivered, then (ii) the item producer releases a batch with C challenge–response pairs, where chal-

```
▷ This method is called by the buyer; p is the item producer identifier, i is the item identifier, s is the seller identifier and b the buyer identifier,
w is the batch index and crdResponses_{i,w} is the vector with the C PUF responses
37: method VERIFYITEM(p, i, s, b, w, crdResponses_{i,w})
38:     if kvs.get(⟨verify, p, i, s, b, w⟩) ≠ null then
39:         kvs.set(⟨batchAlreadyVerified, p, i, w⟩, ⟨p, i⟩)
40:     end if
41:     crdBatch_{i,w} = kvs.get(⟨crdBatchReleased, p, i, w⟩)
42:     if crdBatch_{i,w} == null then
43:         kvs.set(⟨noBatchReleased, p, i, w⟩, ⟨p, i⟩)
44:     else
45:         correctResponses = 0
46:         for y = 0 to C − 1 do
47:             if crdBatch_{i,w}.hashedResponses[y] == hash(crdResponses_{i,w}[y]) then
48:                 correctResponses ++
49:             end if
50:         end for
51:         if correctResponses < R then
52:             kvs.set(⟨verify, p, i, s, b, w⟩, FAIL)
53:         else
54:             kvs.set(⟨verify, p, i, s, b, w⟩, SUCCESS)
55:         end if
56:     end if
57: end method
```

lenges are in clear and responses are hashed and, finally, (iii) the buyer queries the item PUF with those challenges and publishes obtained responses to enable item verification by any party in the supply chain. The following three subsections describe each step in detail.

5.3.1 Item delivery

The buyer p_b acknowledges the reception of x_i by invoking the method $deliveryItem()$, which stores in the blockchain the fact that x_i, produced by p_p and shipped by p_s, has been delivered to p_b. This method also carries out sanity checks to verify the existence of blockchain records proving that x_i was actually produced by p_p and shipped by p_s to p_b. Party p_b also registers itself to be notified (see Sect. 4.4) whenever a new batch release transaction for x_i is committed (see next Sect. 5.3.2).

5.3.2 Challenge–response batch release

The producer p_p of x_i is notified of the delivery and releases a new batch of challenge–response pairs. Party p_p keeps track of how many batches have been already released for x_i and makes sure to select from CRD_i a batch that has not been disclosed before. Let w be the index in CRD_i of the new batch to release. The challenges need to be published in clear to enable the buyer to feed them to the item PUF. The responses need to be hashed instead, to allow to verify whether obtained responses are valid without disclosing the correct responses in clear.

In the specific, p_p prepares a vector $crdBatch_{i,w}$ with C entries, built as follows. Let b_w be the w^{th} batch of CRD_i, i.e. $b_w = [\langle c_0, r_0\rangle, ..., \langle c_{C-1}, r_{C-1}\rangle]$. The k^{th} entry of $crdBatch_{i,w}$ is the pair $\langle c_k, hash(r_k)\rangle$. The method $releaseCRDBatch()$ is invoked by p_p to store $crdBatch_{i,w}$ in the key-value store (line 33).

The sanity checks performed by this method aim to ensure that $w - 1$ batches have been already released (line 26) and that the challenges in this batch are consistent with the $hashedCRD_i$ disclosed at item registration time (line 30). To simplify the notation, we introduce the following two convenient fields of $crdBatch_{i,w}$:

- $crdBatch_{i,w}.challenges$ is the concatenation of all the challenges in the batch, i.e. c_0, ..., c_{C-1}
- $crdBatch_{i,w}.hashedResponses[k]$ is the k^{th} hashed response of the batch, i.e. $hash(r_k)$

5.3.3 Item verification

Party p_b is informed when the batch $crdBatch_{i,w}$ is released. The PUF of item x_i is then queried with the challenges $crdBatch_{i,w}.challenges$ and responses are collected in a vector $crdResponses_{i,w}$. Finally, p_b calls the procedure $verifyItem()$ to disclose obtained responses to the other parties and let them verify whether these responses are valid.

In the specific, this method first verifies that the same batch has not been already verified, in order to avoid replay attacks where an adversary tries to reuse correct responses learned previously (line 38). Then, the $crdBatch_{i,w}$ data is retrieved and the responses provided by the buyer are checked against the hashed responses included in $crdBatch_{i,w}$. If at least R responses out of C are valid, then the verification is considered as succeeded.

6 Security analysis

In this section we discuss what a malicious party p_A operating at stage $stage(p_A)$ can do and how our proposed tracking mechanism would react. We first define the relevant attacks an adversary may launch in Sect. 6.1, based on the threat model introduced in Sect. 4.5 and the tracking system proposed in Sect. 5. Then, in Sect. 6.2 we analyse the response of our tracking system to each of the identified attacks and whether it succeeds in coping with them.

6.1 Attacks definition

According to the threat model introduced in Sect. 4.5, the adversary p_A can operate at different levels. As it cannot collude with any other party nor control their resources, attacks at *supply chain level* are not relevant. At *item level*, p_A has several options. The basic one is to just forge an intact item before supplying it to another buyer (attack 1):

Attack 1 *The adversary p_A tampers with an item received from an honest supplier and delivers it to an honest buyer at the next stage.*

If party p_A works at stage 0, it can tamper with an item before its PUF is fed with the required number of challenges to compute the corresponding CRD. In this way, the CRD stored in the blockchain matches the forged item (attack 2):

Attack 2 *The adversary p_A tampers with an item at stage* 0 *before its CRD is generated and delivers it to an honest buyer at the next stage.*

At *blockchain and smart contract execution environment* levels, the adversary can try to compromise the application logic of the smart contract or the data stored in the blockchain by properly instructing the local node n_A, i.e. node n_A becomes byzantine (attack 3).

Attack 3 *The adversary p_A alters the behaviour of the local node n_A, i.e. node n_A becomes byzantine.*

The layer between supply chain business processes and tracking system is an additional attack surface to consider (see Sect. 5). At this level, the adversary can compromise the way smart contract methods are invoked, e.g. by using maliciously modified parameters or by not calling a method at all (Attack 4):

Attack 4 *The adversary p_A alters how methods of the tracking system smart contract are called.*

6.2 Attacks analysis

For each of the four attacks identified in the previous subsection, we provide an analysis of how the proposed tracking system reacts.

Analysis of Attack 1. In this scenario, party p_A tampers with an item $x_i^{stage(p_A)-1}$ received by an honest supplier p_s. Since the supplier is honest, we assume that $x_i^{stage(p_A)-1}$ has not been forged yet. We also assume that party p_p, producer of x_i, is honest; we will cover the case where the producer is malicious in the analysis of attack 2. The tampered item is supplied to another honest party p_j at stage $stage(p_A)+1$. As p_p and p_j are honest, they comply with the tracking mechanism described in Sect. 5; hence, p_j declares it received x_i by invoking the method $deliveryItem(p, i, A, j)$ and p_p releases a new batch of challenge–response pairs for x_i by calling the method $releaseCRDBatch()$. Afterwards, p_j retrieves this batch and uses the included C challenges in clear to query the PUF $puf_i^{stage(p_A)}$ and collect the corresponding responses, which will be used to invoke the method $verifyItem()$ of the tracking system.

We can assume that p_A stored the correct tracking information regarding the shipping of x_i, otherwise an alert discrediting p_A would be raised (module 1, line 20). We can also assume that the correct CRD of x_i has been stored in the storage, indeed in this scenario we assume the producer of x_i is honest. With reference to module 1, this means that the check at line 42 is positive and the C PUF responses in $crdResponses_{i,w}$ can be compared against those in $crdBatch_{i,w}.hashedResponses$. From the properties expressed by equations 3 and 4, and by the fact that $x_i^{stage(p_A)}$ has been tampered with, it follows that, with high probability, less than R out C responses match; hence, an alert is raised (line 52) to notify the detection of a counterfeit item supplied by p_A. The accuracy of this forgery detection mechanism clearly depends on the choice of R. In Sect. 7 we show an experimental evaluation where R is tuned to maximise the probability that counterfeits are recognised and minimise the chances that intact items are mistaken for forged.

Note that the challenge–response pairs that will be used for the verification are known by the producer party only; hence, an adversary could not discover them in advance and build a model to implement a clone.

Analysis of Attack 2. If p_A operates at stage 0 and tampers with an item x_i^0, then there are two cases. If the counterfeiting occurs after the invocation of method $registerItem()$, then this attack is equivalent to attack 1 and the forgery is detected by the buyer of x_i at stage 1. Otherwise, if the tampering is made before and the stored CRD $hashedCRD_i$ accurately corresponds to puf_i^0, then this attack cannot be detected by the proposed tracking mechanism.

Analysis of Attack 3. The attacker can make the local blockchain node n_A behave arbitrarily, i.e. n_A becomes a byzantine node, with the aim of compromising data stored in the blockchain or the application logic encoded in the smart contract of the tracking system. By design, according to the model presented in Sect. 4.3, in a blockchain with N nodes

the adversary should control at least $\lfloor \frac{N-1}{3} \rfloor + 1$ nodes to compromise the consensus; hence, if there are at least 4 parties in the supply chain, each with its own local blockchain node, then this attack cannot succeed.

Analysis of Attack 4. The adversary can interact with the methods provided by the tracking system differently from what expected. In the specific, p_A can either invoke a method when it should not, or avoid to call a method at all, or purposely specify wrong values for methods parameters. As explained in Sect. 5, an adversary cannot call any method on behalf of another party; hence, p_A can only operate on the methods it is expected to invoke.

If p_A operates at stage 0, it can intentionally avoid to store the CRD for item x_i, i.e. it can skip calling $registerItem()$ method. The motivation could be to prevent forgery checks from taking place and indeed such a goal can be partially achieved by the attacker. Anyway, the honest party p_j receiving x_i from p_A easily discovers that the required CRD crd_i is missing (line 17) and raises an alert (at line 18). Although no forgery can be actually detected in this way, that alert marks x_i as a suspicious item and p_A as a disreputable party because it did not store the expected CRD.

If p_A does not call method $shipItem()$ when expected, then the next party receiving the corresponding item x_i detects this anomaly at line 19 and consequently raises an alert at line 20, which again explicitly points at p_A as the party responsible for this misbehaviour.

Avoiding the execution of methods $releaseCRDBatch()$ and $verifyItem()$ would bring no advantage to the adversary, with respect to its goal (see Sect. 4.5) of introducing counterfeited products without being detected.

Altering the parameters used for either $registerItem()$ or $shipItem()$ method has the same effect of not calling them at all. Altering the parameters of $releaseCRDBatch()$ or $verifyItem()$ methods would not be beneficial for the adversary to introduce counterfeits.

7 Experimental evaluation

We implemented a prototype of the proposed solution to verify the technical feasibility of the integration of blockchain and PUF, and to assess the reliability of PUF technology to accurately detect counterfeit. We used HyperLedger Fabric[15] to implement the consortium blockchain and the smart contract execution environment (see Sects. 4.3 and 4.4). We chose this platform because it is one of the most stable and well documented platforms for consortium blockchains. The tracking system $\mathcal{TS}$ defined in module 1 has been coded as a Fabric chaincode. A 4 bit sequential ring oscillator architecture [29] PUF has been synthesised and implemented on 17 separate Zynq Zybo 7000 FPGA boards [7].

The interface between the tracking system and the PUFs has been implemented as a Java application. The communication with PUF has been done using RXTXComm,[16] a library which makes use of Java Native Interface (JNI[17]) to provide a fast and reliable method of communication over serial ports. The communication at PUF side has been encapsulated in a dedicated module which used General Purpose Input Output (GPIO) as Tx and Rx pins for Universal Asynchronous Receiver/Transmitter (UART) serial communication.

We first describe how we tuned the PUF (Sect. 7.1), then we describe the use case we tested and what results we obtained (Sect. 7.2).

7.1 PUF tuning

The tuning of PUFs consisted in choosing the right value of parameter R, i.e. how many responses out of C need to be correct for the validation to succeed, where C is the number of unique challenges sent to the PUF. We set C to 10.

We first generated the CRD for all the 17 PUFs by collecting a large number of challenge–response pairs for each PUF (more than 21,000 pairs). We then randomly selected 3 out of the available 17 PUFs for tuning, while the others were used for the prototype test (Sect. 7.2). We refer to those 3 PUFs as the tuning PUFs. Challenges drawn from CRD data of all the PUF have been sent to the tuning PUFs to collect the correspondent responses. The resulting dataset has been used to find a value of R that guarantees that each tuning PUF (i) passes the validation when stimulated with its own CRD and (ii) fails the validation when stimulated with CRD of any of the other 16 PUF.

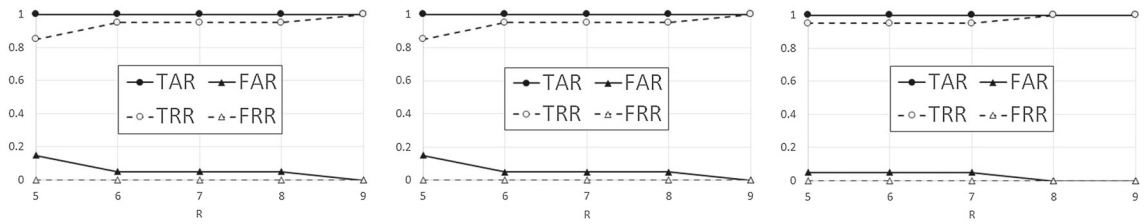
Each tuning PUF has been stimulated with $C = 10$ unique challenges from each of the 17 PUFs (hence including itself) for 15 times. For each batch of C challenge–response pairs, different values of R has been tested, ranging from 5 to 9, and the corresponding validation outcome has been recorded. The metrics of interest for the tuning are

- True Admission Rate (TAR): rate of successful validations when the tuning PUF is validated against its own CRD;
- False Admission Rate (FAR): rate of successful validations when the tuning PUF is validated against the CRD of another PUF;
- True Rejection Rate (TRR): rate of failed validations when the tuning PUF is validated against the CRD of another PUF;

[15] Hyperledger Fabric (https://www.hyperledger.org/projects/fabric).

[16] RXTXComm (https://seiscode.iris.washington.edu/projects/rxtxcomm).

[17] JNI (https://docs.oracle.com/javase/8/docs/technotes/guides/jni/).



(a) Tuning PUF 1 **(b)** Tuning PUF 2 **(c)** Tuning PUF 3

Fig. 4 Tuning of three PUFs, where R is varied between 5 and 9 out of 10, and the corresponding values TAR, FAR, TRR, FRR are shown

– False Rejection Rate (FRR): rate of failed validations when the tuning PUF is validated against its own CRD;

The ideal situation is when TAR and TRR are 1 while FAR and FRR are 0.

Figure 4 shows the values of those metrics for R varying from 5 to 9 (out of 10) for the three tuning PUFs. It can be noted that TAR is always 1 and FRR always 0, which means that the tuning PUFs are successfully validated all the times their own CRD is used. When the validation is based instead on CRD of a different PUF, sometimes tuning PUFs still pass the validation. This happens because the functions computed by different PUFs can overlap for certain challenges. Figure 4 shows that the probability that this occurs (i.e. FAR) decreases as R grows, and that with $R = 9$ FAR is 0 (and TRR is 1) for all the 3 tuning PUFs. Hence, for the prototype test, the validation of a PUF is considered successful if at least 9 out of 10 responses match those stored in the corresponding CRD.

7.2 Prototype test

We developed a prototype with three organisations: manufacturer, logistic and distribution. The corresponding supplier–buyer relationships are depicted in Fig. 5. We considered two cases: when no adversary is present and when the logistic organisation is malicious and tampers with the items supplied by the manufacturer before delivering them to the distribution organisation.

We used the other 14 PUFs for the prototype test, 8 for the case where no party is malicious and 6 for the case where the logistic organisation is the adversary. In the latter case, the manufacturer delivers 3 PUFs to the logistic organisation, which replaces each of them using the other 3 PUFs and deliver them to the distribution organisation.

When there is no adversary, all the 8 PUFs pass the validation both at the logistic and at the distribution organisation; hence, the TAR is 1 and FRR is 0. When instead the logistic organisation replaces the three PUFs, all of them fail the validation at the distribution organisation, therefore the FAR is 0 and TRR is 1.

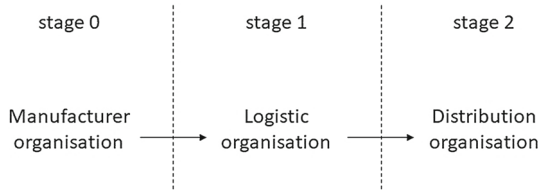


Fig. 5 Graphical representation of the supplier–buyer relationships in the prototype

These preliminary results are promising to prove both the technical feasibility and the effectiveness in counterfeit mitigation of the proposed tracking system.

8 Discussion

This section discusses several key aspects of the proposed solution, pointing out key limitations and main research directions to investigate as future work: the results of the security analysis (Sect. 8.1), the issues of implementing a PKI infrastructure for a consortium blockchain (Sect. 8.2), the limitations of the chosen threat model (Sect. 8.3), the feasibility of embedding PUFs within the items to track (Sect. 8.4), possible privacy issues when sharing data among parties through the blockchain (Sect. 8.5), observations on consortium blockchain performance and scalability (see Sect. 8.6) and, finally, considerations on the costs associated with adopting the proposed solution in real supply chains (Sect. 8.7).

8.1 Security analysis results

The results of security analysis presented in Sect. 6 show the capability of the proposed tracking system to be effective against the identified attacks. Any attempt to counterfeit items (attack 1) is correctly detected and attributed to the right malicious party.

If the adversary operates at stage 0 and tampers with the item before the corresponding CRD is built and stored in the blockchain (attack 2), then the tracking system fails to detect the forgery. This derives trivially from relying on the

CRD itself to be the trust root of the whole counterfeit detection mechanism. Enhancing the proposed approach to cover threats happening before CRD generation is one of our main future work.

The other attacks at software level, to make a blockchain node byzantine (attack 3), or at the interface between supply chain business processes and tracking system (attack 4), have been shown to be not effective. On the one hand, this derives from by-design security properties provided by blockchain-based systems, indeed using PBFT-like consensus algorithms allows to tolerate a single byzantine node when the blockchain includes at least four nodes (attack 3). On the other hand, the tracking system prevents an adversary from invoking smart contract methods on behalf of a different party, so attacks based on altering how methods are called (attack 4) are not relevant.

8.2 PKI infrastructure for consortium blockchains

The proposed tracking system relies on a consortium blockchain (see Sect. 4.3), which in turn requires a reliable PKI to obtain the relationships between parties' identities and public keys. These certificates are issued when the platform is setup at the beginning and when the supply chain membership changes. From a security perspective, the PKI is a single-point-of-failure, i.e. an adversary may target the PKI to take over the whole blockchain, and thus the tracking system.

This problem has been already addressed in literature. For example, there exist proposed solutions based on blockchain to decentralise the PKI so as to make it much more resistant to cyber attacks [1,8], and provide attack tolerance guarantees comparable to those already provided by the tracking system. These solutions are based on public blockchains, which may introduce privacy issues. Other approaches have been proposed for privacy-preserving blockchain-based PKI, such as PB-PKI [3]. The integration of the tracking system with this type of PKI is out of the scope of this paper and is left as future work.

8.3 Threat model limitations

The list of attacks identified in Sect. 6.1 depends tightly on the threat model introduced in Sect. 4.5, which in turn derives from three main assumptions: (i) there is a single adversary, (ii) it controls exactly one party and (iii) only aims at introducing counterfeits in the supply chain. It can be reasonable to consider the implications of relaxing those assumptions and identify what additional attack scenarios may arise when an adversary can control more parties, when more adversaries are active, either independently or by colluding among themselves, and when the adversary has a different goal.

We can expect that a security analysis of the proposed tracking system against such a stronger attack model would point out further vulnerabilities. For example, an adversary could aim at blaming another party by tampering with an item just after the delivery and before it gets verified by the tracking system. To avoid any attribution, the adversary can blame the corresponding supplier for the shipping of a counterfeit item. However, this analysis should be integrated with a risk assessment to measure the likelihood of more advanced attacks, and should estimate to what extent they can be considered reasonable. Taking into account wider threat models is an additional potential future work.

8.4 Embedding PUFs within items to track

The effectiveness of tracking items by using PUFs strictly depends on how easily an adversary can forge items without affecting the PUFs themselves. If a PUF can be removed from an item and embedded within a different one, then the whole counterfeit detection mechanism is flawed. In the end, this boils down to preliminarily check whether it is technically feasible to embed PUFs within items in such a way that all the properties of the PUF-equipped item model hold true (see Sect. 4.2).

Electronic components are items where PUFs can be easily and cheaply implanted by integrating PUF circuitry inside the component circuitry, ensuring that PUFs cannot be removed and replaced. Hence, the approach we propose fits well with integrated circuits and IoT devices supply chains. However, an aspect to be taken into account is that a failure of the PUF circuit is likely to lead to inaccuracies in the counterfeit detection process. Although this problem is intrinsic of any tag-based tracking mechanism, it would be interesting to explore the feasibility and challenges of devising methodologies to distinguish between a counterfeited PUF and a damaged PUF.

8.5 Privacy issues

Although the network of companies involved in the supply chain should be made as transparent as possible to enhance visibility, organisations can be legitimately reluctant to disclose their own supplier network and procurement history to other, possibly competitor firms. What information should be shared needs to be adjusted according to this kind of confidentiality requirements, on a case by case basis. An important applied research direction to investigate, for each target supply chain market, concerns this trade-off between privacy and visibility, with the aim to find the sweet spot where information on supplier network and procurement history can be shared smoothly.

A general approach to address those privacy issues is to make each transaction only visible to a specific subset of par-

ties. In the specific, only those parties having some stakes on the item referenced in the transaction should be able to read it, so that visibility can be preserved and limited to interested actors only. With reference to our prototype implementation based on Hyperledger Fabric, we could implement this general approach by leveraging on the concept of *channels* to establish between subsets of nodes. A transaction can be associated to a specific channel to ensure only the nodes in that channel can see its content. Our prototype can be enhanced with privacy-preserving techniques by relying on Fabric channels.

8.6 Performance and scalability

While public permissionless blockchains like Ethereum's are known to provide limited performances in terms of transaction latency and throughput, consortium blockchain can commit thousands of transactions per seconds with subsecond latency [4], also in WAN settings [25]. In terms of scalability, BFT-tolerant algorithms have been proposed in literature that can scale to tens of nodes with minor performance penalties [20,28], which matches realistic supply chain setting including tens of different organisations.

8.7 Platform integration costs

Each supply chain works according to specific business processes which may differ significantly from market to market. On the one hand, pinpointing the right abstraction level for the interface provided by the tracking system is crucial to increase the cases where it can be integrated. On the other hand, the integration with those business processes deserves a deeper analysis in terms of security, to figure out whether additional cyber threats can be identified at those integration points (see attack 4 in Sect. 6.1), and cost-effectiveness, to quantify whether and to what extent the benefits of counterfeiting mitigation outweigh the costs to accomplish such a large-scale integration.

In terms of cost-effectiveness, it is to note that relying on consortium blockchains rather public permissionless blockchains allows to cut any cost due to the fees to pay when submitting transactions. Indeed, while supply chain tracking solutions based on Ethereum have a per-transaction cost (e.g. see Negka et al. [23]), submitting transactions in Hyperledger Fabric is totally free.

9 Conclusion

In this paper we design a tracking system to mitigate counterfeits in IC supply chains. The solution we propose is based on blockchain and smart contract technologies to provide high availability and strong tolerance against integrity attacks to stored data and application logic. We rely on physically unclonable functions to uniquely identify and accurately track ICs along the supply chain. We validate our solution against a specific threat model and find out that it is effective to counter the identified attacks, but an adversary operating at the first stage of the supply chain can bypass the anti-counterfeit mechanism. Finally, we implemented and tested a prototype of the proposed tracking system to prove it is technically feasible and accurate in correctly validating both intact and forged items.

In addition to investigate possible solutions to the limitations discovered in the security analysis, other future work include the integration of a reliable PKI infrastructure within the tracking system and the implications of considering a stronger threat model.

Acknowledgements We thank the anonymous reviewers for the helpful comments they provided on earlier drafts of the manuscript. This research is partly funded by the Royal Academy of Engineering Grant (IF\192055).

Compliance with ethical standards

Conflict of interest Author Leonardo Aniello declares that he has no conflict of interest. Author Basel Halak declares that he has no conflict of interest. Author Peter Chai declares that he has no conflict of interest. Author Riddhi Dhall declares that she has no conflict of interest. Author Mircea Mihalea declares that he has no conflict of interest. Author Adrian Wilczynski declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Al-Bassam, M.: SCPKI: a smart contract-based pki and identity system. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, ACM, New York, BCC '17, pp 35–40 (2017). https://doi.org/10.1145/3055518.3055530
2. Alzahrani, N., Bulusu, N.: Block-supply chain: a new anti-counterfeiting supply chain using nfc and blockchain. In: Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems. ACM, New York. CryBlock'18, pp 30–35 (2018). https://doi.org/10.1145/3211933.3211939

3. Axon L, Goldsmith M (2016) PB-PKI: a privacy-aware blockchain-based PKI. In: Proceedings of the 14th International Joint Conference on e-Business and Telecommunications, SCITEPRESS
4. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with BFT-smart. In: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (2014)
5. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, OSDI '99, pp. 173–186 (1999). http://dl.acm.org/citation.cfm?id=296806.296824
6. Chatterjee, U., Govindan, V., Sadhukhan, R., Mukhopadhyay, D., Chakraborty, R.S., Mahata, D., Prabhu, M.M.: Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database. IEEE Trans. Dependable Secure Comput. (2018)
7. Crockett, L.H., Elliot, R.A., Enderwitz, M.A., Stewart, R.W.: The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc. Strathclyde Academic Media (2014)
8. Fromknecht, C., Velicanu, D., Yakoubov, S.: A decentralized public key infrastructure with identity retention. In: IACR Cryptology ePrint Archive **2014**, 803 (2014)
9. Gaetani, E., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., Sassone, V.: Blockchain-based database to ensure data integrity in cloud computing environments. In: Proceedings of the First Italian Conference on Cybersecurity (ITASEC17), Venice, Italy, January 17–20. pp. 146–155 (2017). http://ceur-ws.org/Vol-1816/paper-15.pdf
10. Guardtime (2017) Internet of Things Authentication: A Blockchain Solution Using SRAM Physical Unclonable Functions. Available online: https://www.intrinsic-id.com/wp-content/uploads/2017/05/gt_KSI-PUF-web-1611.pdf
11. Halak, B.: Hardware-based security applications of physically unclonable functions. In: Physically Unclonable Functions, Springer, Berlin pp 183–227 (2018)
12. Halak, B.: Security attacks on physically unclonable functions and possible countermeasures. In: Physically Unclonable Functions. Springer, Berlin, pp 131–182 (2018)
13. Halak, B., Zwolinski, M., Mispan, M.S.: Overview of puf-based hardware security solutions for the internet of things. In: Circuits and Systems (MWSCAS), 2016 IEEE 59th International Midwest Symposium on, IEEE, pp. 1–4 (2016)
14. Hartmann, J., Moeller, S.: Chain liability in multitier supply chains? Responsibility attributions for unsustainable supplier behavior. J. Oper. Manag. **32**(5), 281–294 (2014). https://doi.org/10.1016/j.jom.2014.01.005
15. Hohenstein, N.O., Feisel, E., Hartmann, E., Giunipero, L.: Research on the phenomenon of supply chain resilience: a systematic review and paths for further investigation. Int. J. Phys. Distrib. Logist. Manag. **45**(1/2), 90–117 (2015)
16. Horvath, B.T.: Not all parts are created equal: the impact of counterfeit parts in the air force supply chain. Air War College, Air University Maxwell AFB United States, Tech. rep. (2017)
17. Huang, J., Li, X., Xing, C., Wang, W., Hua, K., Guo, S.: DTD: a novel double-track approach to clone detection for rfid-enabled supply chains. IEEE Trans. Emerg. Top. Comput. **5**(1), 134–140 (2017). https://doi.org/10.1109/TETC.2015.2389532
18. Islam, M.N., Patii, V.C., Kundu, S.: On IC traceability via blockchain. In: VLSI Design, Automation and Test (VLSI-DAT), 2018 International Symposium on, IEEE, pp 1–4 (2018)
19. Jain, R., Chaudhary, D.K., Kumar, S.: Analysis of vulnerabilities in radio frequency identification (RFID) systems. In: 2018 8th International Conference on Cloud Computing, pp. 453–457. Data Science and Engineering (Confluence), IEEE (2018)
20. Junqueira, F.P., Reed, B.C., Serafini, M.: Zab: high-performance broadcast for primary-backup systems. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN), IEEE, pp 245–256 (2011)
21. Khojasteh-Ghamari, Z., Irohara, T.: Supply chain risk management: a comprehensive review. In: Supply Chain Risk Management, Springer, pp 3–22 (2018)
22. Mispan, M.S., Su, H., Zwolinski, M., Halak, B.: Cost-efficient design for modeling attacks resistant pufs. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), IEEE, pp. 467–472 (2018)
23. Negka, L., Gketsios, G., Anagnostopoulos, N.A., Spathoulas, G., Kakarountas, A., Katzenbeisser, S.: Employing blockchain and physical unclonable functions for counterfeit iot devices detection. In: Proceedings of the International Conference on Omni-Layer Intelligent Systems, ACM, pp 172–178
24. OECD: Trade in Counterfeit Products and the UK Economy. OECD Publishing, Paris. (2017) https://doi.org/10.1787/9789264279063-en
25. Sousa, J., Bessani, A.: Separating the wheat from the chaff: an empirical design for geo-replicated state machines. In: 2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS), IEEE, pp. 146–155 (2015)
26. Su, H., Zwolinski, M., Halak, B.: A machine learning attacks resistant two stage physical unclonable functions design. In: 2018 IEEE 3rd International Verification and Security Workshop (IVSW), IEEE, pp. 52–55 (2018)
27. Toyoda, K., Mathiopoulos, P.T., Sasase, I., Ohtsuki, T.: A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain. IEEE Access **5**, 17465–17477 (2017). https://doi.org/10.1109/ACCESS.2017.2720760
28. Vukolić, M.: The quest for scalable blockchain fabric: proof-of-work vs. BFT replication. In: International Workshop on Open Problems in Network Security, Springer, Berlin, pp. 112–125 (2015)
29. Wachsmann, C., Sadeghi, A.R.: Physically unclonable functions (PUFs): applications, models, and future directions. Synth. Lect. Inf. Secur. Privacy Trust **5**(3), 1–91 (2014)
30. Wiengarten, F., Humphreys, P., Gimenez, C., McIvor, R.: Risk, risk management practices, and the success of supply chain integration. Int. J. Prod. Econ. **171**, 361–370 (2016)
31. Wild A, Becker GT, Güneysu T: A fair and comprehensive large-scale analysis of oscillation-based PUFs for FPGAs. In: Field Programmable Logic and Applications (FPL), 2017 27th International Conference on, IEEE, pp 1–7 (2017)
32. Yilmaz, Y., Gunn, S.R., Halak, B.: Lightweight PUF-based authentication protocol for IoT devices. In: 2018 IEEE 3rd International Verification and Security Workshop (IVSW), IEEE, pp 38–43 (2018)
33. Yu, M., Hiller, M., Delvaux, J., Sowell, R., Devadas, S., Verbauwhede, I.: A lockdown technique to prevent machine learning on pufs for lightweight authentication. IEEE Trans. Multi-Scale Comput. Syst. **2**(3), 146–159 (2016). https://doi.org/10.1109/TMSCS.2016.2553027