NoSneaky: A Blockchain-Based Execution Integrity Protection Scheme in Industry 4.0

Chiu, Wei-Yang; Meng, Weizhi; Ge, Chunpeng

Published in:
IEEE Transactions on Industrial Informatics

Link to article, DOI:
10.1109/TII.2022.3215606

Publication date:
2022

Document Version
Peer reviewed version

Citation (APA):
Chiu, W.-Y., Meng, W., & Ge, C. (Accepted/In press). NoSneaky: A Blockchain-Based Execution Integrity Protection Scheme in Industry 4.0. IEEE Transactions on Industrial Informatics, PP(99), 1-9. [9924304]. https://doi.org/10.1109/TII.2022.3215606
NoSneaky: A Blockchain-based Execution Integrity Protection Scheme in Industry 4.0

Wei-Yang Chiu†, Weizhi Meng∗ and Chunpeng Ge†

∗Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark
†College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China

Corresponding Email: {weme@dtu.dk}

Abstract—The advancement of information technology allows the creation of smart devices that not only are programable, but also can perform machine-to-machine communication in order to reach a flexible large-scale manufacturing strategy in Industry 4.0. However, as more components are connected to the Internet, cyber-criminals can perform malicious actions remotely. As one lasting threat, sabotaging smart devices’ execution integrity can cause a large financial loss, i.e., causing malfunctioning. Hence, it is important to secure the execution integrity of smart devices in Industry 4.0. Motivated by the emerging blockchain technology, in this paper, we focus on how blockchain can help Industry 4.0 application protect execution integrity and propose a blockchain-based execution protection scheme named NoSneaky, which is low-cost and can be easily integrated into the current production systems. In the evaluation, we demonstrate its performance and effectiveness in securing the execution integrity.

Index Terms—Blockhain technology, Industry 4.0, Execution Integrity, Smart Device, Production System

I. INTRODUCTION

The idea of standard operation procedures, assembly-line technique, and machinery created the first industrial revolution. When electricity gradually accepted and popularized, its combination with electric motors then created the second industrial resolution. Later, the digitization brought by computers allowed factories having more flexibility during productions [1]. Then network access technologies allowed the production to be more adaptive and resource-efficient. The increase of interconnectivity and data-exchange supported automation enabled the industry to be smarter and more precise, which is considered as the Fourth Industrial Revolution, or Industry 4.0 [2].

However, as Industry 4.0 is pushing productions into the full scenario of digital twin, the reflection correctness and preciseness between physical and virtual world has become an important topic [3]. [4]. The issue of imprecise reflection can cause difficulties and potential monetary loss during production and simulation. As an attacker, the ability of breaking and tearing the reflection either virtual or physical in an IoT-embedded and information-driven assembly line is good enough to become an ideal hostage to ransom the production organization [5].

Further, there are other common but critical situations that can affect the reflection, e.g., a malfunctioned device. Both can result in unqualified productions, or procedures not correctly done according to the pre-set standard.

As machines are capable of networking, these inaccurately reflected digital data can cause severe issues, as untruthful information may be broadcasted around the whole network. For example, a standard canning operation requires proper sealing and vacuuming before canned product can be sterilized at hot temperatures. If the vacuuming is not processed correctly due to malfunction or wrong configuration, it not only can result in a problematic product, but also can damage the assembly line due to high air pressure in the can.

Ensuring the reflection between virtual and physical trust and confidence is important for factories when moving them from Digital Model to Digital Twin, which can fully utilize the advantages of Industry 4.0 with security. When any malicious events being detected, a prevention step should be involved as soon as possible.

To ensure that all outputs from production lines are close to what people like and what they expected. A second pair of eye is involved, with the advent of blockchain technology, it can be deployed as a log service and guidance reference [6].

Hence we propose NoSneaky, a blockchain-based solution to enhance the connection security between the digital object to be realized expectedly and the physical object. To implement a simple and easy to understand scenario, our system is based on the following assumption:

Given a digital command, then there is an expected outcome being observed, if the reflected physical action or actions being performed correctly.

Based on the given assumption, what we can secure is the bonding between the given command and the expected outcome. If the result breaks this bonding, then the alert or respected actions will be raised and performed. The bonding information should have the highest integrity, which means the audit-ability and transparency of these records should be highly regarded. Further, this information should be highly available under different circumstances.

With all these requirements, blockchain technology is considered as a promising solution for storing up such information [7]. Each storage unit, called block, is secured with hash and proof generated by consensus algorithms. Furthermore, each block is hash-connected toward the preceding blocks, creating a chain-like structure with enhanced security due to the computational costs to tamper [8]. [9].

Motivated by the above challenge and requirements, in this work, we propose NoSneaky, a blockchain-based execution integrity protection scheme, which does not require, or only require a little adaption of the current production system to
integrate. That is, NoSneaky can effectively lower the deployment cost. The contributions of our work can be summarized as below.

- We design NoSneaky to help check the execution integrity of smart devices in Industry 4.0. We particularly analyze how to select a suitable underlying blockchain platform, and develop some critical components such as executor, monitor, validator, trust / responsible list, and action report / integrity datastore.
- In the evaluation, we implement NoSneaky (with detailed software and hardware configuration) and examine its performance under malicious entities. Our results demonstrate that our proposed system can protect a production system from execution integrity attack.

The remaining parts are scheduled as follows. Section II introduces the background of blockchain technology and discusses related work on blockchain-based solutions in Industry 4.0. Section III describes our proposed NoSneaky in detail. Section IV presents the environmental configuration and the experimental results. Section V discusses the limitation and open challenges, and Section VI concludes this work.

II. BACKGROUND AND RELATED WORK

The application of blockchain technologies into Industry 4.0 domain is countless; however, not many of them are applied into the realm of execution integrity. In this section, we briefly introduce the background on blockchain and related work on blockchain-based applications for Industry 4.0.

A. Background on Blockchain

Blockchain got its name due to the data structure of its basic storage unit (see Figure 1), and the mechanism that keeps data on-chain immutable with high integrity.

Since consensus algorithms are the mechanism used to ensure only one unified blockchain version, it provides a protection nonce for each block, which enables the immutability to blockchain. It can be regarded as the core of blockchain [12]. Different algorithms can provide various advantages alongside with disadvantages [13]. For instance, some of them are suitable for a large public network, but may be inefficient in the ratio of transaction throughput and consumed resources. Some may be suitable for a small private network, whereas the network is costly when scaling.

B. Related Work on Blockchain in Industry 4.0

In recent years, blockchain technology has become popular, and has been studied in helping design suitable Industry 4.0 applications. For example, Lin et al. [14] introduced BSeln, a blockchain-based secure mutual authentication system that can ensure the fine-grained access control polices in Industry 4.0. It could provide privacy and security protection by integrating attribute signature, multi-receivers encryption and message authentication code, as well as anonymous authentication and auditability.

Fernandez-Carames et al. [15] focused on Unmanned Aerial Vehicles (UAVs), and utilized blockchain technology to keep certain inventory data gathered through UAVs. The blockchain is used to ensure the data trustworthiness and the data availability. Demertzis et al. [16] considered how to safeguard the network communication between traded Industrial Internet of Things (IoT) devices and then designed a Deep Autoencoder Neural Network to identify malicious events in the associated blockchain transactions within a critical infrastructure network. Faridi et al. [17] introduced a blockchain-based product traceability system for textile manufacturers. It can facilitate all stakeholders to track and monitor the product quality and ensure the efficiency of the supply chain.

Shukla et al. [18] focused on digital twin in Industry 4.0 and introduced a six-layered architecture for digital twin to help estimate the detection via an ensemble technique and a public Ethereum blockchain. Qu et al. [19] aimed to enhance the cognitive computing in the Industry 4.0 automation, and developed a blockchain solution for big data-driven cognitive computing based on federated learning. The blockchain-enabled federated learning can help reach a quick convergence with the advanced verifications and member selections. More relevant studies of blockchain-based Industry 4.0 can refer to several surveys [20]–[23].

By analyzing the existing work, most of them are focusing on the protection of data storage in Industry 4.0 via blockchain, or how to build a secure, trusted communication channel. Different from the current state-of-the-art, our work focuses on how to ensure the integrity of execution result, and develops NoSneaky, a blockchain-based execution integrity protection scheme. Our work aims to stimulate more research on building a secure Industry 4.0 application via blockchain.

C. Digital Twin and Blockchain in Manufacturing

Utilizing blockchain to secure Digital Twin in manufacturing is very common. Huang et al. [24] proposed a blockchain-
secured data management system for Digital Twin of Product. It utilized blockchain’s characteristic of decentralization to provide storage, access and sharing in a large number of parties with access control, authenticity and version control. The smart contract functionality in blockchain provides automation with trust, and automation with data sharing procedure. Mandolla et al. [25] proposed a blockchain-integrated additive manufacturing (AM) process that can securely interconnect between AM Infrastructures in aviation industry. This is because aviation components require incredible precision during manufacturing, securing digital model and parameters till manufacturing process for aviation safety. Putz et al. [26] proposed a system that utilized blockchain decentralization and auditability to provide an audit-able and secure management system of Industry 4.0 digital twin assets. Further, it utilized smart contracts to develop DApps and ensure fine-grained access control. This created a system with integrity and availability. Finally, Leng et al. [27] developed a blockchain-based system that resolved the inconsistency between overview planning and local execution during manufacturing.

There are many existing frameworks that utilize blockchain to protect Digital Twins; however, few of them are focusing on ensuring the given objects and parameters that could be correctly reflected to the physical objects. Further, many of them required further integration (e.g., updating software or adding support firmware) with the system that runs currently. Though, our NoSneaky performs similarly on functionality, it provides a system that is non-invasive to the current system. As it can be run on its own network with observation from the side, the only component that requires integration is the Validator. Although it can be hooked up to shut off power directly, it requires integration with the current system to allow performing further actions.

III. OUR APPROACH - NoSNEAKY

This section first briefly introduces and discusses how we select the underlying blockchain platform and then describes the system design and workflow of NoSneaky.

A. Underlying Blockchain Platform

Though there are lots of blockchain platforms loading with different advantages, not all of them are suitable for our demands. As we look into a typical assembly line, we immediately notice the following characteristics:

- **Fixed count of smart devices:**
  The smart devices on each different part of the assembly line are usually fixed and known. Variations are possible from time-to-time, but not as frequent as a public blockchain network. Further, the join and the leave within the network must be done by authorized personnel.

- **Real-time:**
  Each step on the assembly line should be finished on time. As the success of NoSneaky’s operations is heavily relied on the push-and-pull of data to blockchain, the blockchain transactions must be completed before the target reaches the checkpoint. Otherwise, there will be no data for the checkpoint to observe. Transaction speed must be stable and fast, blockchain system that has unstable transaction speed is not favorable.

- **Smart Contract supported:**
  Smart contracts not only provide a simple way of store data, but also provide partial automation with trust and transparency as well as multiple parties involved.

Firstly, maintaining a fast and steady transaction speed is critical; however, the blockchain type should also be carefully considered. For example, a federated blockchain can resist better on 51% attack compared to a pure-privately controlled permissioned blockchain, which can provide a cost-effective solution for some platforms that do not have enough IT to hold a private blockchain. However, such implementation cannot be done in a closed-network, whereas some production lines may require, due to security practices.

Table I summarizes the features among some typical private and federated blockchains. We finally decided our platform as GoQuorum, due to its speed and the following features:

- **Ethereum Compatible:** Ethereum can be considered as one of the most widely used DApps and smart-contract platform. Since GoQuorum is compatible with Ethereum on both DApps and smart contracts, it makes our system portable. If there exists a platform that outperforms GoQuorum and, at the same time, is Ethereum compatible, then switching platforms can be easily done.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Consensus Algorithm</th>
<th>Smart Contracts</th>
<th>SC Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authority</td>
<td>Public/Private</td>
<td>PoW(etHash) / BFT-like(Clique/IBFT)</td>
<td>Solidity</td>
<td>EVM</td>
</tr>
<tr>
<td>Ethereum</td>
<td>Permissioned</td>
<td>Pluggable Consensus (Default Raft)</td>
<td>Go/Java/JavaScript</td>
<td>Container</td>
</tr>
<tr>
<td>FISCO BCOS</td>
<td>Permissioned</td>
<td>Round Robin Validation</td>
<td>JavaKotlin</td>
<td>Runtime</td>
</tr>
<tr>
<td>Hyperledger Besu</td>
<td>Permissioned</td>
<td>PoW(etHash) / BFT-like(Clique/IBFT)</td>
<td>Solidity</td>
<td>EVM</td>
</tr>
<tr>
<td>Hyperledger Fabric</td>
<td>Permissioned/Federated</td>
<td>BFT-like(Clique/IBFT)</td>
<td>Solidity</td>
<td>EVM</td>
</tr>
<tr>
<td>MultiChain</td>
<td>Permissioned</td>
<td>Raft</td>
<td>Javascript</td>
<td>Runtime</td>
</tr>
<tr>
<td>GoQuorum</td>
<td>Permissioned/Federated</td>
<td>Raft / BFT-like(IBFT/Clique)</td>
<td>EVM</td>
<td>EVM</td>
</tr>
<tr>
<td>R3 Corda</td>
<td>Federated</td>
<td>Raft</td>
<td>Javascript</td>
<td>Runtime</td>
</tr>
</tbody>
</table>
output for debugging, hence we adopt such a way to monitor intrusive way to monitor the system. Many devices have data NoSneaky can be costly. This is the reason why we use non-
the requirement of re-implementing the firmware to integrate board firmware may provide the best system performance, 
its processing time remains stable. Though deploying smart contracts may display a huge variation between platforms, while each of 
completing one single transaction, such as pushing a new contract with 100KB of binaries or more).

We noticed that the speed of deploying a new contract can be severely different between platforms, while each of its processing time remains stable. Though deploying smart contracts may display a huge variation between platforms, completing one single transaction, such as pushing a new record into the smart contract, can remain fast and efficient.

Hence, we decided to select GoQuorum as our platform, not only because of its speed, but also due to its similarities with Ethereum.

**B. System Design of NoSneaky**

Figure 2 depicts the system design of NoSneaky. We assume a production line with many smart devices that are controlled machinery – Actors, where each actor is externally embedded with a monitor. Several critical check points with validators should observe the outcome made by the actor. To prevent possible security issues, monitors should be connected with their own proprietary network, which is different and separate from the typical control network. Executors can remain their communication through their own network.

There are four major roles in NoSneaky:

- **Actor**: The actor is the smart device that can command the machinery on the assembly line in order to perform real actions on the product. We assume that such device is embedded with an interface where we can read its status, and that it will constantly report the undertaken actions. The Actor is not part of the NoSneaky system, but is the machinery controlling device that NoSneaky is constantly monitoring.
- **Monitor**: The monitor is an additional smart device that only listens to the output information from the Actor, as we assume that most of the available commercial devices may not allow any modification toward its firmware. The Monitor will interact with the Actor (i.e., sending commands through Interface). It will report the Actor’s actions, such as completing an expected task, and upload the **Action Report** onto the blockchain.
- **Validator**: The validator is the additional smart device that observes the physical outcome of the product through sensors. It will refer to reports given by the Monitors, and the information on the **Action Integrity Data** on the blockchain. If the reports look odd according to **Action Integrity Data**, then the pre-set action will be taken, and an alarm will be raised.
- **System Administrator**: This is the role that defines rules for **Action Integrity Data** on the blockchain, managing both trust list and responsible list.

In a close view of the relation between the actor and the monitor, we implemented an example diagram by utilizing Audrino as Actor, while a Raspberry Pi as monitor. Both are connected to UART serial interface with each other.

Arduino – the Actor will periodically report its actions to Raspberry Pi – the Monitor. While the Raspberry Pi and the monitor can constantly receive reports on when and what have happened on its monitoring device, as shown in Figure 3.

**There are four important data entities in NoSneaky:**

- **Monitors**: This is a list of Monitors’ addresses. It is used to ensure that only devices in the list can make **Action Reports** onto the blockchain. The report sent by others will be rejected. Figure 4 shows the structure of Monitor.
  - **m_Address**: The address of Monitor’s account.
- **friendly_name**: A column to store a meaningful human name for the Monitor, e.g., Sterilizer.
- **dType**: A column to store the device type on which the Monitor is monitoring.

- **Responsible List**: This is a list of Validators’ addresses, with a hashmap that stores their responsible Monitor(s). The structure can refer to Figure 4:
  - **v_Address**: The address of Validator’s account.
  - **friendly_name**: A column to store a meaningful name for the Monitor, e.g., Assembly Line 2.
  - **mp_Addresses**: A column that tells the responsible area of this Validator.

- **Action Report**: This is a data entity stored on the blockchain that allows Monitors to store the reports. The Validator will retrieve the latest report from the same entity. The structure can refer to Figure 4:
  - **m_Address**: The address of Monitor that makes this report.
  - **timeStamp**: The timestamp of when such report has been made.
  - **outputHash**: The hash value of `hash(Output Information from Actors)`
  - **confirmed**: The boolean value that shows whether such report has been successfully processed by the Validator or not.
  - **alert**: The boolean value that shows whether the Validator detects anomalies.
  - **m_Sign**: The signature of the report, made by the Monitor.
  - **v_Sign**: The signature of the report, made by the Validator after the report has been processed.

- **Action Integrity Data**: This is a set of policies stored on the blockchain that are defined by System Administrators. It regulates what given command is bonded with what expected outcome. The structure can refer to Figure 4:
  - **dType**: The device type we are going to define the sets of expected actions and outcomes.
  - **resultBond**: The `resultBond` mapping provides a map that bonds the `hash(Output Information from Actors)` to the value of `hash(Expected Result)`. For example, if we expect a Bronze outlook of the product when the Actor states to the Monitor: `Temperature Ok!`, then we can define the bond as: `hash(Temperature Ok!) = hash(Bronze)`

To simplify the relationship between the roles and entities, Figure 5 shows the access permission of different data entities and roles.

From the Administrator's perspective of view, they have the utmost power of managing everything, except for the **Action Report**. From the Monitor's perspective of view, the only action that it can perform is to send out data to the **Action Report**. The **Action Report** will reference whether the report sender is on the Trust List, if not, the report will be rejected. Finally, from the Validator’s perspective of view, it can reference all entities except for the Trust List. Validators require querying the Responsible List to know which Monitor is under its responsibility. Accessing both the **Action Report** and **Action Integrity Data** enables Validators to verify the report and the physical outcome of the product.
C. The Workflow of NoSneaky

Here we introduce the workflow regarding how to configure the system in the first time, how the report can be made, and how the validator prevents misbehavior from malfunctioning or tampered executors.

a) Defining and Initializing: System administrator must predefine the following data entities to activate the system:

- Trust List: Administrators must deploy and give a set of Monitors to the system, so that the system can initiate with an access list of who can make the Action Report.
- Responsible List: Administrators must deploy and give a set of Validators to the system, so that the Validators can know who to look and what to expect.
- Action Integrity Data: Administrators must pro-\provide a set of policies for Validators to follow and compare.

b) Monitoring and Reporting: If the Monitor is in the listening mode, then it will not respond to any output made by the Actor. Suppose that when each action is taken by the Actor, there should be an output that can tell such action is performed.

In our assumption, an Actor-controlled machinery only has limited operations, especially when the scenario is on an assembly line, where most machineries have only one expected operation. If any output has been made, the Monitor will create a data record with the data structure Report and deliver the report into the Action Report on the blockchain.

c) Validating and Preventing: The Validators will first grab two sets of reports:

- Reports listed as "unconfirmed": Any report that is noted as confirm = false.
- Reports that have timeStamp around a reasonable time: Any report that is sent in the reasonable time duration according to timeStamp. Suppose the assembly line has to take 10 seconds from the Actor to the Validator, then, the reasonable time duration should be around 10 seconds.

Afterwards, the Validators will trigger the workflow to check a report’s original state, as shown in Figure 6.

In the first step, the Validator will check whether the timeStamp provided by the report looks reasonable. For example, it considers a tampered value if a timeStamp is far away from now in the future, or is far away from the past. After checking the timeStamp, the Validator will check other properties that should only be proposed by Validators. If any of these properties do not appear in its predefined status, the Validator will raise an alert and taking actions.

If the report looks reasonable, then the Validator will start checking the observed result, and compare it with the output from the Action Integrity Data on the blockchain, as shown in Figure 6.

In our NoSneaky, the Action Integrity Data is difficult to tamper, as blockchain requires all nodes to hold a full or partial copy of the chain. For a successful attempt, an attacker has to not only swap the locally stored blockchain of all involved nodes at once, but also calculate all hashes that are connected after the transaction.

IV. EVALUATION

In the evaluation, our testbed was running on a virtualized environment constructed by vmware ESXi 7.0, and each virtual machine (VM) was given a 2-core Intel Xeon W-2133 CPU, 4GB RAM, and 48GB of HDD. Within each VM, we installed mxLinux 21 as the operating system and GoQuorum v21.7 as the blockchain platform.

As shown in Figure 7, there are three actor-monitor combinations and one validator. On the validator machine, a shared folder was used to receive the output from actors. We adopted the following reaction rule: if any output result raises an alert, then the validator will shut down the virtual machine of the corresponding generator.

A. Experimental Flow

This section presents an evaluation to investigate the performance of our system, with the design as follows:

- The Actor: A program was constantly generating files every 5 seconds, and in each file, a character represents as
the executed result. Meanwhile, an IPC pipeline is created so that the Monitor can receive the output information from the Actor. If the Actor successfully performs an operation, it will print out “Generated Successfully” through IPC socket to the Monitor, and at the same time, print out the character into the file, i.e., the expected outcome is a. However, we deliberately crippled some generators to have a chance of randomly printing out other characters.

- The Monitor: A program was constantly listening to the output information from the Actor through the IPC, and putting the output result to the blockchain, as shown in Figure 8.

- The Validator: A program reads the executing results from the Actor, and compares them with the information on the blockchain. If the Validator figures out that a particular generator provides an answer that does not match with what the Monitor suggested, it will command the vmware ESXi to shut down the problematic generator (virtual machine).

In our testing scenario, we designed three generators (Actor / Monitor combinations) with one validator machine. The configuration of Trust List (containing Monitors) is summarized in Table III. Note that all Actors were controlling respectively a device type called Generator. While this device type can have an expected behavior (see Table IV), if it reports “Generated Successfully”, the outcome should be a. Finally, as there is only one Validator, the Responsible List only has one record, as shown in Table V.

### B. Implementation Methods of Actor

There are generally two implementation methods of Actor. One is an active method, and the other is a passive method. The active method requires the Actor to report every step that is about to perform, and every step must obtain the Validator’s approval to continue. If the Actors are all honest, such implementation can terminate malicious actions as soon as possible.

We deliberately implemented another version of Actor that requires an execution permit from the Validator, and then compared how much it may affect. In this testing case, all Actors output one character alongside with a string of report information; however, the modified version was required to send the operation and the expected output before the modified version can continue the operation (see Table VI).

The main difference between modified version 1 and modified version 2 is that: during a malicious event, the modified version 1 provides malicious estimated-execution results to the Validator, whereas the modified version 2 provides faked estimated-execution results and continues performing malicious outcome.

The outcome of Table VI is not surprising. The active method has a slight advantage if the Actor is honest, re-

<table>
<thead>
<tr>
<th>dType</th>
<th>outputHash</th>
<th>resultHash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generator</td>
<td>hash(&quot;Generated Successfully&quot;)</td>
<td>hash(&quot;a&quot;)</td>
</tr>
</tbody>
</table>

### TABLE III

<table>
<thead>
<tr>
<th>m_Address</th>
<th>friendly_name</th>
<th>dType</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xf0e...b2b5</td>
<td>Generator 1</td>
<td>Generator</td>
</tr>
<tr>
<td>0xb30...d0ca</td>
<td>Generator 2</td>
<td>Generator</td>
</tr>
<tr>
<td>0x088...2610</td>
<td>Generator 3</td>
<td>Generator</td>
</tr>
</tbody>
</table>

### TABLE IV

<table>
<thead>
<tr>
<th>dType</th>
<th>outputHash</th>
<th>resultHash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generator</td>
<td>hash(&quot;Generated Successfully&quot;)</td>
<td>hash(&quot;a&quot;)</td>
</tr>
</tbody>
</table>

### TABLE V

<table>
<thead>
<tr>
<th>v_Address</th>
<th>friendly_name</th>
<th>mp_Address[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x48...7e45</td>
<td>Validator 1</td>
<td>0xf0e...b2b5, 0xb30...d0ca, 0x088...2610</td>
</tr>
</tbody>
</table>

### TABLE VI

<table>
<thead>
<tr>
<th>Exec/Item</th>
<th>Normal Round</th>
<th>Malicious Shut Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>5.057 ms</td>
<td>413 ms</td>
</tr>
<tr>
<td>Modified1</td>
<td>5.326 ms</td>
<td>379 ms</td>
</tr>
<tr>
<td>Modified2</td>
<td>N/A</td>
<td>706 ms</td>
</tr>
</tbody>
</table>
porting the malicious activities in the execution estimation and sending it to the Validator, which can shutdown the virtual machine at once. However, this implementation does not make sense for an attacker – if an attacker’s goal is to sabotage the production line, with the knowledge of such system, he or she may not report the malicious activity in the phase of requesting the execution permit. This is the reason why we implemented the modified version 2, which is closer to what an attacker would do in practice.

However, it makes the active prevention useless and wasted. This is because the first round of gaining the execution right does not stop anything but take time to complete the step. Once the malicious outcome was performed and the Validator was detected, it would have the same protection performance of passive detection method, while the additional time cost in the first round makes such method more undesirable.

C. Malicious Monitors

In addition to the Actors, the Monitors can also be malicious. In practice, the main goal of attackers is to fool the Validator. However, it can be challenging:

- **Covering for the Actor**: It is obvious that the Monitors can be not truthful, by reporting the good, without minding whatever outcome an Actor has reported. However, the Validator can still detect the malicious outcome, by comparing it with the data on the blockchain.

- **Not Reporting**: The situation of not reporting is intriguing. If the outcome is the right action, whether reporting or not will not make an impact. However, if the outcome is undesirable, it can still be detected when the Validator performs a seek-up search of the action integrity datastore. The disadvantage is that this action takes some time to complete. However, when malicious event is detected, the Validator can take actions (i.e., shutting down the VM) to prevent it from the beginning.

- **Falsifying Report**: This is a similar problem to “covering for the Actor”. The main difference is what to falsify. If the outcome is undesirable, eventually such device will be shut down or offline, according to what actions have been programmed. If the outcome is desirable, the purpose of falsifying a report will not cause any loss for the company.

D. Blockchain Fork Attack

When the network is small, the influence of a fork attack can be tremendous. However, it depends on when the attack can be launched. If the fork attack occurs in a later stage, the Validator can still rely on the integrity datastore, which is usually created when the blockchain is initiated. The fork attack can, indeed, mess up the report system; however, it usually will not happen in the initial setup stage. Further, we encourage the implementation method to hold this audit system in a proprietary network, which should be separated from the control network. A closed network may have some inconveniences, but if considering that the environment of a production line is usually fixed and stable, the closed-network may provide more benefits with security.

V. Discussion

In this work, we introduce how blockchain can provide integrity protection on either data or execution result. However, due to the resource limitation, we cannot examine the system on a large-scale network. Instead, we investigated the system in a small-scale network environment with three actors and one validator. We discuss some open challenges as below.

- **Malicious Validators**: In our proposed NoSneaky, both validators and the system administrators should be well-protected from the attacker. It is relatively easy in a small-scale network, but it may become quite challenging if the network extended. In our system, multiple validators can be applied. As long as not all validators are malicious at the same time, we can still prevent the malicious validator. However, the protocols between different validators should be upgraded, so the malicious validators can be phased out. Such implementation can provide better protection.

- **Scalability of Blockchain**: As assembly cooperation between machines based on data sharing is feasible, we can expect that such cooperative manner will be extended to cross-organizations. When production projects involve with multiple parties, the network of NoSneaky will be extended. Making NoSneaky scalable is the top priority, as we have not tested NoSneaky in a large-scale environment. The voting-based consensus algorithms tend to have performance degradation with the network extending. As of 2025, the expected count of IoT devices can up to 75.44 billion [29], which is almost twice the number as 2022, we can expect the same thing occurred to the assembly line. Hence we need to examine any method that can make blockchain-based applications efficient and speedy, especially in a large and dynamic network.

VI. Conclusion

In Industry 4.0, the interconnectivity provides many benefits such as flexibility and scalability. However, the interconnected online factories may also face various security threats, e.g., production integrity. In this paper, we introduce NoSneaky, a blockchain-based and low-cost solution to protect the production line from getting sabotaged by cyber criminals. We particularly discussed why to choose GoQuorum as the underlying blockchain platform, and presented the main roles and entities in NoSneaky: executor, monitor, validator, trust / responsible list, and action report / integrity datastore. The blockchain is mainly used for checking and ensuring the integrity of reported outcome. In the evaluation, we implemented our system and provided the detailed configurations of software and hardware. The results demonstrated the viability and the effectiveness of NoSneaky in defending against malicious execution reports in a production system.
REFERENCES


