IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 10, 2021, accepted September 29, 2021, date of publication October 4, 2021, date of current version October 13, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3117938*

# D2Gen: A Decentralized Device Genome Based Integrity Verification Mechanism for Collaborative Intrusion Detection Systems

**IMRAN MAKHDOOM[1,2], (Member, IEEE), KADHIM HAYAWI[3], (Member, IEEE), MOHAMMED KAOSAR[4], (Senior Member, IEEE), SUJITH SAMUEL MATHEW[3], (Member, IEEE), AND PIN-HAN HO[5], (Fellow, IEEE)**

[1]Department of Information Security, National University of Sciences and Technology, Islamabad 44000, Pakistan
[2]Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, NSW 2007, Australia
[3]College of Technological Innovation, Zayed University, Abu Dhabi, United Arab Emirates
[4]Discipline of Information Technology, Murdoch University, Murdoch, WA 6150, Australia
[5]Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada

Corresponding author: Kadhim Hayawi (abdul.hayawi@zu.ac.ae)

This work was supported by Zayed University under the Cluster Research Grant R20140.

**ABSTRACT** Collaborative Intrusion Detection Systems are considered an effective defense mechanism for large, intricate, and multilayered Industrial Internet of Things against many cyberattacks. However, while a Collaborative Intrusion Detection System successfully detects and prevents various attacks, it is possible that an inside attacker performs a malicious act and compromises an Intrusion Detection System node. A compromised node can inflict considerable damage on the whole collaborative network. For instance, when a malicious node gives a false alert of an attack, the other nodes will unnecessarily increase their security and close all of their services, thus, degrading the system's performance. On the contrary, if the spurious node approves malicious traffic into the system, the other nodes would also be compromised. Therefore, to detect a compromised node in the network, this article introduces a device integrity check mechanism based on "Digital Genome." In medical science, a genome refers to a set that contains all of the information needed to build and maintain an organism. Based on the same concept, the digital genome is computed over a device's vital hardware, software, and other components. Hence, if an attacker makes any change in a node's hardware and software components, the digital genome will change, and the compromised node will be easily detected. It is envisaged that the proposed integrity attestation protocol can be used in diverse Internet of Things and other information technology applications to ensure the legitimate operation of end devices. This study also proffers a comprehensive security and performance analysis of the proposed framework.

**INDEX TERMS** Insider attacks, integrity check, collaborative intrusion detection system, device genome, device security, blockchain, Internet of Things.

## I. INTRODUCTION

The pervasiveness of the internet has widened the spectrum of connected real-world things and has also provided promising opportunities to build robust industrial systems and related applications. Correspondingly, the Industry 4.0 paradigm has witnessed significant changes by leveraging the growing ubiquity of the Industrial Internet of Things (IIoT) [1]–[3]. This integration has undoubtedly improved the efficiency of processes, reduced spatio-temporal investments, and bestowed faster return on investments. While this industrial evolution has been extensively reviewed and studied [3]–[5], there is a growing concern that the increased connectivity of critical infrastructures such as thermal powerhouses, electricity grids, hospitals, hotels, banking, and defense systems makes them vulnerable to numerous cyber-attacks [6]–[8]. Resultantly, a hacker can hack into the end devices and install malware or modify the software components. Moreover,

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Quan.

suppose an end device is physically compromised. In that case, the attacker can also change the hardware components, i.e., extend device memory, increase RAM, alter processor speed, change network configuration, activate or deactivate unauthorized ports or interfaces (e.g., JTAG, UART), change I/O (input/output) pin configuration, etc. Correspondingly, these software and hardware modifications will affect the legitimate operation of the devices and the security and privacy of user data. Besides, an unintentional or unprovoked technical fault, hardware or software failure, or human error can also cause an end device malfunction.

Furthermore, sophisticated cyber-attacks, including Black-Energy Crimeware [9], Dragonfly-Group/Energetic Bear [10], Mirai [11], and Stuxnet [12] are some testimonies to the vulnerabilities of the critical infrastructure. These attacks disrupted not only vital operations but also infringed sensitive business and personal information. Moreover, in some cases, there had been physical damage, and substantial financial loss [12]. Also, there is no restriction on the type of device that can be fully compromised; it may be an IoT sensor/actuator, a Programmable Logic Controller (PLC), or a node of Collaborative Intrusion Detection System (CIDS). Correspondingly, some untrustworthy members of the CIDS may perform a malicious act, thus introducing the possibility of an insider attack [13]–[15]. Consequently, a compromised CIDS node can inflict considerable damage on the whole collaborative network. Therefore, when this malicious node gives false alerts of an attack, the other nodes may unnecessarily increase their security and may close all of their services [16]. Similarly, if it approves malicious traffic into the system, the other nodes would be compromised too.

Therefore, there is a need to develop a framework to verify devices' integrity, especially IDS nodes. So that if an attacker makes a change in the hardware, software, or configuration of an IDS node, it can be detected and rectified.

## II. RELATED WORK

As shown in Table.1 significant work has been done in the past to verify the integrity of embedded devices based on code, firmware, or memory attestation. For instance, [17] introduced a Software-based Memory Attestation scheme (SWATT) for embedded devices. The proposed solution detects an embedded device with malicious code in its memory. However, due to a challenge-response protocol, the verification procedure is vulnerable to Rainbow [18] and Interference attacks [19]. Moreover, the Time of Check to Time of Use (TOCTTOU) [20] is also different. In addition, SWATT is vulnerable to the rootkit-based attack that involves Return Oriented Programming (ROP). The attackers use ROP as a security exploit to control the call stack and manipulate the flow of the trusted software running on the target machine. Resultantly, the attackers can execute malicious code on the target system [21], [22]. Similarly, authors in [23] presented a distributed, secret sharing, and majority voting-based attestation scheme for IoT sensors. Being distributed, the proposed solution prevents trust issues involving a single trusted verifier. However, it is believed to be computation-intensive and vulnerable to "Good Mouth" and "Bad Mouth" attacks. Also, the TOCTTOU is different. Consequently, it is susceptible to rootkit-based ROP attack [24]. Moreover, the initialization of the attestation procedure is also speculative.

In another work [25], the researchers proposed a One-way Memory Attestation Protocol (OMAP) for smart meters. The scheme detects a malicious device without a challenge-response protocol. However, in this scheme as well, the initialization of the code attestation procedure is not clearly defined, and the TOCTTOU is also different. Hence, it is vulnerable to rootkit-based attacks [24]. Correspondingly, [24] introduced One-way Code Attestation Protocol (OWCAP), an improved version of OMAP, for Wireless Sensor Networks (WSN). Nonetheless, this work is limited to a star network topology in which only a cluster head has the responsibility of attesting and verifying a sensor node. In the same way, researchers in [26] presented a blockchain-based secure firmware update mechanism for embedded devices in IoT. The framework protects against message authentication, confidentiality, replay, and integrity attacks. It also replaces the client-server based firmware update protocol with a blockchain-based decentralized scheme. However, it also has certain limitations; The IoT node initiates a firmware update message, but it is unclear when it should initiate the process. Similarly, two processes of firmware verification run in parallel, which will create unnecessary network traffic, node operations, and increased energy consumption of end devices. Moreover, it does not protect against node compromise attacks, in which an attacker, instead of tampering with firmware, installs an executable malicious code in the memory of the node to launch further attacks.

In a more recent work [27], the researchers introduced a remote attestation strategy for control systems that combines software attestation and control process validation. It is a challenge-response based protocol that protects against threats to the integrity of PLCs' logic code. It also prevents replay attacks involving sensor readings. However, being a challenge-response protocol, it is also assumed to be vulnerable to the Rainbow and Interference attacks. Similarly, [28] proposed a technique to seek behavior transparency and control for smart home IoT devices. The scheme detects misbehaving devices by analyzing application layer network flows between clients and servers. It enables transparency by reporting which device exhibits what behaviors, when, and how often. The authors claim that the proposed framework effectively classifies behaviors of IoT devices. And these behaviors may include heartbeat, firmware check, reporting, and uploading audio or video recordings. However, there is a possibility that a compromised device may input false data to the network by exhibiting normal behavior like Stuxnet [29]. Correspondingly, [30] presented "HEALED," a software-based remote attestation and disinfection scheme for embedded devices. However, this solution protects only against malware attacks.

**TABLE 1.** Comparison of device attestation techniques.

| Existing Solution | Reference | Methodology | Protections Offered | Limitations |
|---|---|---|---|---|
| SWATT | [17] | Software-based memory attestation scheme | Detects an embedded device with a malicious code in its memory | Due to challenge-response protocol, verification procedure is vulnerable to Rainbow and Interference Attacks |
| | | | | TOCTTOU (Time of Check to Time of Use) is different |
| | | | | Vulnerable to rootkit-based attack (Return oriented programming) |
| Distributed software-based attestation | [23] | Secret sharing and majority voting-based memory attestation scheme | Distributed attestation scheme that protects against single trusted verifier | Computation intensive |
| | | | | Vulnerable to “Good Mouth,” and “Bad Mouth” attacks |
| | | | | TOCTTOU is different |
| | | | | Initiation of attestation procedure is speculative |
| | | | | Vulnerable to rootkit-based attack |
| OMAP | [25] | One way memory attestation protocol | Detects a malicious node | Initialization of code attestation is speculative |
| | | | | TOCTTOU is different |
| | | | | Vulnerable to rootkit-based attack |
| OWCAP | [24] | One way code attestation protocol for WSN | Detects a malicious node based on memory attestation | Limited to star network topology |
| Blockchain-based secure firmware update | [26] | Blockchain-based firmware update mechanism | Replaces client-server based firmware update with a distributed technology | The time of initiation of firmware update procedure is not clearly defined |
| | | | Protects against message authentication, confidentiality, replay, and integrity attacks | Prone to increased network traffic and energy consumption |
| | | | | No protection against node compromise attacks |
| PAtt | [27] | Performs attestation of control systems based on software attestation and control process validation | Detects threats to the integrity of PLCs' logic code | A challenge-response based protocol |
| HomeSnitch | [28] | Detects misbehaving smart home devices by analyzing application-layer network flows between clients and servers | Provides transparency and control over smart home devices by classifying their behaviors | Vulnerable to a node compromise attack, where the victim may exhibit normal behavior but input false data to the network |

**TABLE 1.** *(Continued.)* **Comparison of device attestation techniques.**

| Existing Solution | Reference | Methodology | Protections Offered | Limitations |
|---|---|---|---|---|
| HEALED | [30] | A software-based remote attestation and disinfection scheme for embedded devices | Detects malicious software of a device | Provides security against malware only |
| CIDNs and insider attacks | [31] | Presents a survey on CIDNs and their robustness against insider attacks | Introduces various insider attacks and some generalized protections to prevent them | Does not propose any solution to detect a compromised IDS node |
| HBCIDS | [32] | Relies on simple trust management system to distinguish between honest and dishonest nodes | Prevents collusion attacks | Vulnerable to attacks against reputation systems |
| ABDIAS | [33] | Supports majority voting-based system to detect compromised nodes | Provides early warnings for pre-attack activities | Vulnerable to collusion attacks |
| Worminator | [34] | IDS nodes share alert information to detect worms | Achieves better detection accuracy by using alert correlation | Lack of trust management among nodes<br>Vulnerable to insider attacks |
| IDS for detecting compromised gateways | [35] | Detects compromised gateways in an IoT network based on packet drop probability | Detects a malicious gateway that may intentionally drop or corrupt the packets | Not effective against compromised CIDS nodes |
| Enhanced CIDN protection against insider attacks | [36] | Uses blockchain to protect CIDN against advanced insider attacks such as PMFA | Detects betrayal and PMFA faster than other methods | Seems more communication intensive<br>May not be effective, if an attacker compromises many nodes in the same neighborhood |

Looking at some CIDS specific literature, [31] carried out a survey of some Collaborative Intrusion Detection Networks (CIDNs) and analyzed their robustness against insider attacks. However, this work just listed down various insider attacks and respective generalized solutions to contain them. It does not propose a strategy to detect compromised nodes as a result of insider attacks. Moreover, it classifies various CIDNs based on their vulnerability to insider attacks. Similarly, some of the Collaborative Intrusion Detection Systems (CIDS) use a simple trust management mechanism to distinguish honest nodes and dishonest nodes, such as Host-based Collaborative Intrusion Detection System (HBCIDS) [32]. HBCIDS prevents collusion attacks but being based on trust estimation, it is assumed to be vulnerable to attacks against reputation systems [37]. Correspondingly, in another CIDS [33], a majority voting-based system is used to detect compromised nodes. However, it is vulnerable to collusion attacks. In addition, some CIDS like Worminator [34] rely on alert correlation to achieve better detection accuracy. Nonetheless, due to a lack of trust management among CIDS nodes, it is vulnerable to numerous insider attacks. Furthermore, authors in [35] proposed an IDS for detecting compromised gateways in an IoT network. The suggested scheme detects malicious gateways based on the probability of dropped or corrupt packets either in the uplink or downlink direction. However, it is assumed that such a solution cannot protect against a malicious gateway that may input messages with false data in the network. Moreover, this technique is not expected to be effective against compromised CIDS nodes that may inject false alerts into the network.

Finally, researchers in [36] presented a blockchain-based solution to protect a CIDN against advanced insider attacks such as Passive Message Fingerprint Attacks (PMFA).

It effectively detects betrayal and PMFA faster than other challenge-based trust models. However, due to the propagation of challenge-request pairs (in respect of a tested node) to other network nodes for verification by the tester, this scheme appears to be communication intensive. Moreover, there is a question about the effectiveness of this scheme once the attacker compromises many nodes in the same neighborhood and modifies their signature/alarm library/responses.

Although some of the node attestation methods mentioned above protect against firmware, code, or software (application) modification attacks, none of these methods protects against a physical compromise of the device and any modification of the hardware components, device configuration, or network settings. Moreover, most of the subject methods are based on challenge-response protocols known to have glaring weaknesses, including vulnerability to network attacks, TOCTTOU gaps, increased energy consumption, time synchronization problems, low detection rate, and restricted to star/cluster tree network topology. Besides, a few solutions focus on the software components and fail to verify the overall state of a device at run-time. Correspondingly, devices equipped with trusted computing hardware have security against data breaches and software integrity but not against general device state alterations, including changes in network configuration/settings and modification in device hardware to affect its legitimate operation. Hence, there is a need for an absolute scheme that should verify the integrity of devices in an IoT network by achieving the following objectives:

a. A compromised or a malfunctioned device should be detected at the earliest.
b. The device attestation process should be secure yet transparent.
c. No single party should be able to influence or forge the attestation process.
d. The solution should be computationally economical.

Therefore, in this article, we introduce a new method to verify the integrity of digital devices inspired by human DNA (Deoxyribonucleic Acid) identification [38]. DNA typing technologies and associated bioinformatic tools have been used for more than two decades to identify humans, especially the victims of Mass Fatality Incidents (MFI) [39]. These incidents range from air crashes to tragedies like the World Trade Centre. In this context, "Genome" is an organism's complete set of DNA, including all of its genes. It contains all of the information needed to build and maintain the organism [40]. Therefore, taking motivation from the concept of genome, we conceived the idea of verifying the integrity of digital devices by computing a "Digital Device Genome (D2Gen)" over the device's software and hardware components, network configuration, and vital metadata. In this context, any unauthorized modification/malfunction in the device's hardware or software components, network configuration, or ambient conditions will result in a different D2Gen as compared to the one referring to default settings. Hence, in such a way, a compromised device can be easily detected.

Moreover, to ensure the integrity of the D2Gen protocol, we have leveraged blockchain technology and its inherent benefits such as decentralized control, data immutability, trustless operation, and the ability to run distributed applications (DApps). Once developed and tested, the device integrity check concept can be extended to more specific Cyber Physical Systems (CPS), such as PLCs' security in the Supervisory Control and Data Acquisition (SCADA) networks. Accordingly, a gist of vital contributions of this research is proffered in the following section.

### A. CONTRIBUTIONS OF THIS RESEARCH

1) Introduces a new method of end device integrity verification.
2) Facilitates detection of an illegitimately compromised or malfunctioned end device.
3) Prevents a varsity of insider attacks involving modifications/alterations of end device hardware, software, or network configuration and ambient operating conditions.
4) Avoids trust issues during verification process by inheriting decentralized control, network consensus, data immutability and transparency features of the blockchain technology.
5) Protects against application forgery/modification attacks.
6) Prevents initiation of forged/fake transactions by unauthorized CIDS nodes in the network.
7) Resilient against replay attacks.
8) Proffers a comprehensive analysis on security and performance efficiency of the proposed framework.
9) Expected to augment existing device attestation protocols.

### B. ORGANIZATION OF THE PAPER

The rest of the paper is organized as follows: Section III introduces D2Gen. Then, Section IV illustrates the integrity check protocol, security guarantees, and associated challenges and limitations. Similarly, Section V presents a comprehensive security and performance analysis of the Proof of Concept (POC) of proposed framework. Whereas Section VI concludes the article with a hint of future work.

## III. D2Gen

This section introduces the concept of computation of D2Gen for an IDS node over its hardware and software components and network configuration to ensure its legitimate operation. A productive design and development of D2Gen can help detect a malicious node even if a single byte of data is changed in its software components, including firmware, malware definitions, and important system files. Similarly, D2Gen aspires to detect a compromised node even if a minute change is made in its hardware and software components, or network configuration. We also envisage that irrespective of the type of application, e.g., may it be the blockchain or cloud-supported services, D2Gen will augment and give a new direction to the existing IoT/IIoT/IT device attestation protocols.

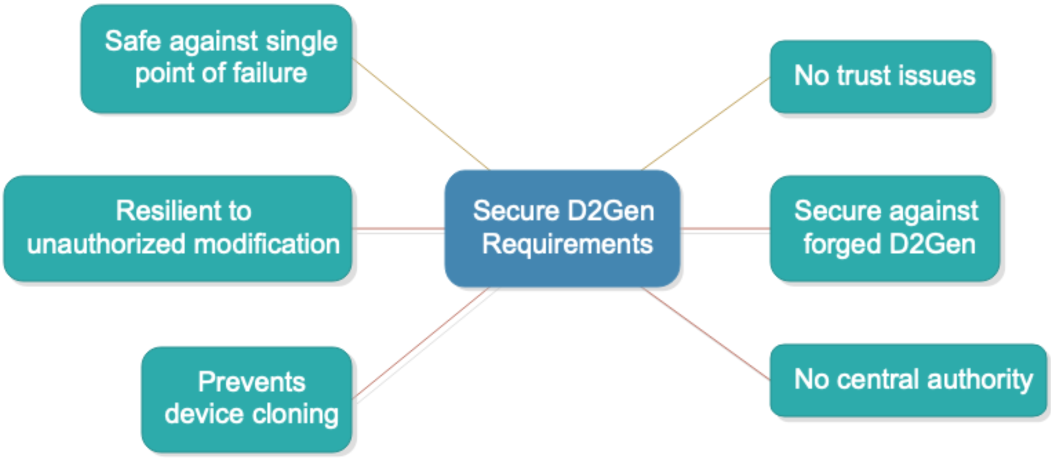

Safe against single point of failure
No trust issues
Resilient to unauthorized modification
Secure D2Gen Requirements
Secure against forged D2Gen
Prevents device cloning
No central authority

**FIGURE 1.** Requirements of a secure D2Gen protocol.

### A. COMPUTATION OF D2Gen

D2Gen is computed by measuring particular hardware and software characteristics, network configuration/settings, and associated metadata for respective devices. The hardware and software configurations may differ as per the type of IoT/IIoT or other network devices such as IDS. However, in this paper, we propose the most generalized parameters for the computation of D2Gen. Whereas, more specific parameters for the integrity check of an IDS node will be highlighted in Section V during our discussion about the POC.

#### 1) MEASUREMENT OF HARDWARE CHARACTERISTICS

We believe that some of the hardware properties that can contribute to verifying and assessing a digital device's integrity include.

a) Measuring voltage and currents on the specific ports to ascertain if any additional hardware is connected to the device.
b) Measuring memory size and processor speed to verify that the attacker has not altered the device's computational and storage capacity.
c) Checking boot sequence to identify any misappropriation during device boot up.
d) Checking the status of various hardware components installed to detect any change in device hardware configuration, e.g., the battery capacity and drainage, may help identify illegal device operation other than the approved/default functionality.

#### 2) MEASUREMENT OF SOFTWARE CHARACTERISTICS

In case of any successful security breach, that may be a cyber-attack or a physical compromise; the attacker may modify the firmware, system/application software, or the state of data stored on the device. Moreover, in the case of an IDS, the data or device state may include malware signature library and the order of the signatures within the signature bank. Hence, numerous aspects concerning device's software components and related metadata must be monitored and measured.

a) Checking device identity may help detect and prevent a Sybil attack [41]. In which an attacker creates and presents multiple identities using a single device.
b) Monitoring the number of applications running on a device will help detect unauthorized applications installed. It is most suitable for corporate environments with Bring Your Own Device BYOD) policies.
c) Checking types and versions of applications installed. It will also help in the identification of unauthorized applications installed on the device.
d) Monitoring access rights of the applications. This aspect is critical because an application with escalated privileges may spy on the user and steal private/sensitive data.
e) Probing access permissions of various files. This factor is also essential in disclosing any change in the default permissions of the system files installed on a device.
f) System files' read and write permissions. This is an important factor in preventing device malfunction and unauthorized operation.
g) Checking the state of IDS node's attack/malware signatures library is also essential.
h) Checking the order of malware signatures in the IDS node's attack signatures library.
i) Computing cryptographic hash of vital system files and firmware/application code. This is another crucial requirement to detect any unauthorized changes to the system files, firmware, or application code.

#### 3) VERIFICATION OF NETWORK CONFIGURATION AND SETTINGS

The integrity of a device cannot be effectively assessed without evaluating its network settings and other metadata. These parameters may include:

a) Configuration of active and deactivated ports.
b) Network credentials and approved settings.

c) Checking connected devices to detect any unauthorized hardware.
d) Pin configuration/settings (for embedded /IoT devices).
e) Mode of operation, i.e., client or server mode.
f) Checking device location is always helpful in providing location-based services and the successful implementation of geo-fencing. Based on the same concept, a device's location is included in the computation of D2Gen so that a device communicating from an unwarranted or unusual place can easily be detected.
g) Verifying device position (flat, raised, vertically hanging, upside down, etc.).
h) Analyzing the ambient conditions of the device, including temperature, humidity, light intensity, etc. This is critical for the integrity of sensing and actuating devices, i.e., whether these devices are operating and communicating from their designated location/environment.

## IV. THE INTEGRITY CHECK PROTOCOL

To ensure that the device integrity verification protocol performs legitimately, we inferred some essential security requirements as shown in Fig.1. Therefore, blockchain technology has been used to satisfy most of the security issues. Detailed reasoning in this regard is later proffered in Section IV-A.

Elaborating on D2Gen protocol, a device is first configured and then installed in the operational environment, e.g., in a CIDS, smart grid, or a remote health monitoring system. The overview of the entire device integrity check process (D2Gen) is shown in Fig.2. Before its deployment in the network, firstly, the base genome map of the device is computed over its selected hardware and software characteristics, network configuration, and other metadata as illustrated in Section III. The Base_D2Gen is then stored for the respective device on the blockchain through a smart contract. The end device can be a lite blockchain client or a full node based on its storage and computation resources. In both cases, the device can interact with the blockchain using smart contracts. Later, whenever the device is scheduled to send data such as IDS log message or a sensor update, it will compute and append D2Gen to the routine data and push the data into the blockchain through the DApp (smart contract). At this moment, before updating the state of the blockchain, the block proposer/miner compares the device's genome forwarded with the message or sensor data with the Base_D2Gen stored on the blockchain.

If the genome matches, only then the IDS log or sensor data will be published in the blockchain. Otherwise, in case of a conflict, the message (transaction) will be discarded, and an alert message will be sent to the network administrator and device owner, notifying the possible malicious behavior of the respective device. The infected or invalid device's identity (ID) will be appended to the list of blacklisted nodes stored on the blockchain for easy access by all the stakeholders. Later, when the device is investigated and rectified, the base genome is recomputed and stored on the blockchain after approval of all the stakeholders maintaining miner nodes in the blockchain network. It is followed by the removal of the device from the blacklist after the first successful D2Gen verification. In addition, if a node's state is legitimately modified, i.e., its software is updated, or hardware components are upgraded, then its Base_D2Gen will be recomputed and stored on the blockchain.

### A. SECURITY GUARANTEES

Based on the initial analysis, it is implied that our proposed blockchain-based device integrity check protocol protects against numerous security threats. As D2Gen comprises a device's software and hardware components and related information, the protocol protects against software/code modifications, execution of unauthorized applications, hardware alterations, changes in device configuration/settings, and modification of device data. Moreover, some information concerning the device's network configuration, location, and authorized/default connected hardware is also part of D2Gen. Hence, it is perceived that there are some security assurances against change in device location/position, modifications in network configuration, and attachment of unauthorized devices to different ports.

Here, a question may arise, what benefits do we get by using blockchain. The main objective is to leverage some inherent advantages of the distributed ledger technology [42]–[47] such as decentralized control, distributed architecture and applications, data immutability, trustless operation, and network consensus to approve any transaction (TX). In this context, our proposed framework uses smart contracts (DApps) to extract a varsity of device data to compute D2Gen through methods/functions built-in the blockchain like web3.js methods in Ethereum blockchain [48]. Therefore, it is nearly impossible for an attacker to emulate a fake response to the D2Gen attestation or change the Base_D2Gen stored on the blockchain without detection. Moreover, there is no need for a trusted verifier or an authentication server. The integrity check is done through smart contracts in a decentralized way. It is also believed that such a plenary solution is likely to protect against software license spoofing attacks.

### B. CHALLENGES

There is a need to overcome specific challenges to employ this innovative idea in real-world IoT/IIoT networks like any other technology. The critical issue is the development of blockchain technology with integral methods/functions (like web3.js library for Ethereum blockchain) to extract data/information about a device's hardware and software components. The second prominent requirement is the normalization of the variations in the current and voltage measurements required for the computation of D2Gen. Thirdly, the decision on what information about a device's software and hardware components, network configuration/settings, and device metadata is to be used to verify the device integrity can only be made after numerous tests and trials. Nonetheless, the proposed solution is likely to bridge the gaps in

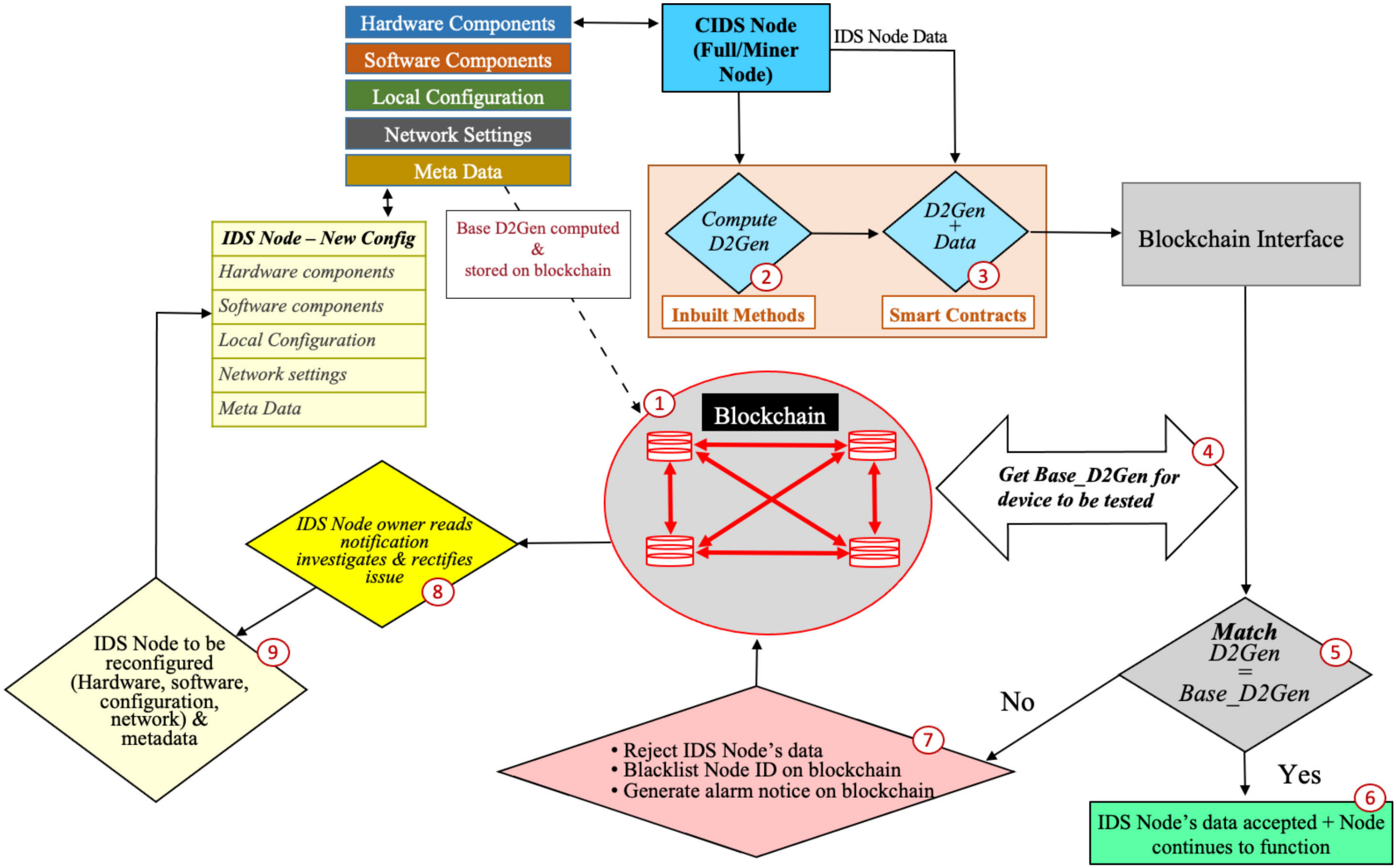

Hardware Components
Software Components
Local Configuration
Network Settings
Meta Data
CIDS Node (Full/Miner Node)
IDS Node Data
Compute D2Gen
2
D2Gen + Data
3
Inbuilt Methods
Smart Contracts
Blockchain Interface
IDS Node – New Config
Hardware components
Software components
Local Configuration
Network settings
Meta Data
Base D2Gen computed & stored on blockchain
1
Blockchain
4
Get Base_D2Gen for device to be tested
Match D2Gen = Base_D2Gen
5
No
Yes
6
IDS Node's data accepted + Node continues to function
7
• Reject IDS Node's data
• Blacklist Node ID on blockchain
• Generate alarm notice on blockchain
8
IDS Node owner reads notification investigates & rectifies issue
9
IDS Node to be reconfigured (Hardware, software, configuration, network) & metadata

FIGURE 2. Conceptual diagram of the digital device integrity check based on digital genome.

existing security vulnerabilities and the requirements of an effective device integrity check mechanism. However, at the same time, there is a need for multiple tests and trials to determine when to compute the device genome. The possible options include: at the device restart, at various system checkpoints, after major installations, whenever there is an unusual resource utilization, or at runtime with every sensor/device data update.

Correspondingly, a high frequency of computation and verification of D2Gen may also affect the system's usability and performance. Therefore, there needs to be a balance between security, usability, and performance. It is presumed that such a relationship heartily varies with the security and safety requirements of the subject systems. A security-critical system may sacrifice usability and utilization of resources over enhanced protection against device compromise and malicious operation.

### C. LIMITATIONS

During the elementary evaluation of the proposed device integrity check framework, certain limitations have also been deduced. These limitations constitute:

1) A specialized hardware/Integrated Circuit (IC) is required to detect variations in current and voltage at specific points of the device hardware.
2) Currently, the proposed framework is suitable for blockchain-based IoT/IIoT systems only.
3) Overall, D2Gen is to be computed over many hardware, software, network configuration, and other device parameters. Whereas some of these parameters are dynamic and others are static. E.g., voltage and current levels at specific ports may vary at every measurement. While the static parameters such as OS CPU architecture, system file permissions, device hostname, and total memory size may remain unchanged for a long time. Accordingly, unless we segregate the dynamic and static parameters and also set/normalize the threshold values for the dynamic parameters, there is always a chance that the D2Gen computed at runtime will differ from the Base_D2Gen. This mismatch will result in false negatives, i.e., sometimes legitimate nodes may not pass the integrity check, and their TXs will be rejected. Nonetheless, false negatives are better than false positives in a critical IoT infrastructure scenario, e.g., smart grid, smart cars, intelligent traffic control systems, etc., where safety and security are of paramount importance.

## V. PROOF OF CONCEPT AND PERFORMANCE ANALYSIS OF D2Gen

To carry out an authentic security and performance analysis of the D2Gen protocol, we developed a POC as per the experimental setup shown in Fig.3. A list of software and hardware used in the experiment is shown in Table.2. The testbed (Fig.3) comprises two CIDS server nodes that are also the blockchain miner nodes, two Raspberry Pi (Rpi) 3 model B running snort IDS with a light blockchain application, and two network devices including a smartphone and a personal computer. Once the hardware was available, we first developed the smart contract on Remix-IDE in solidity language. The pseudocode for the smart contract ( illustrated later in Section V-A) is shown in Algorithm.1. In this testbed, only the CIDS server, which is the trusted owner/manager of the CIDS network, is authorized to deploy the smart contract on the blockchain and store Base_D2Gen values for the Rpi-based CIDS nodes. However, these operations are confirmed based on network consensus. It is imperative to highlight that Base_D2Gen is computed over sixteen different hardware, software, and network configuration parameters. These parameters include:

1) Network interfaces
2) Hostname
3) OS architecture
4) Total memory size
5) Permissions for /etc/security/access.conf file
6) Contents of /etc/networks file
7) Ambient temperature of the node (measured using DS18B20 temperature sensor)
8) Information about each logical cpu core
9) OS platform
10) OS release details
11) OS type
12) OS version
13) Default directory for temporary files
14) Information about the user including username, uid, gid, shell, and homedir
15) OS specific end-of-line marker
16) Details of all the connected (authorized) devices

After the contract is deployed, the CIDS servers start listening for specified event emissions on the blockchain network. In the POC, currently, the Rpi-based CIDS nodes emit two events once they initiate the TXs containing the new IDS alert message and the current D2Gen. As shown in Fig.4, an event named newTx_D2Gen emits information about the current D2Gen of the respective Rpi-based CIDS node. Moreover, the account belonging to the Rpi node, which was used to pay for the TX fee, is also shown against the "from" field just above the red box. This event notification also displays the smart contract address, block number in which this TX is included, and the TX hash.

Similarly, in Fig.5, the event named "newIDSLogEv" provides details of the latest IDS log message (enclosed in red box) that includes date and time, source and destination IP address, IP protocol, TTL, and Type of Service (ToS). These details may vary depending upon the rules configured in Snort IDS. In addition, this event notification also shows the date and time when the IDS log message TX was initiated.

For experiment purposes, while installing and configuring snort on the Rpis, we defined a custom rule in local.rules file to generate an alert if an ICMP message is sent from any source to any destination on the home network. Accordingly, some changes were also made in the snort.conf file. Hence,

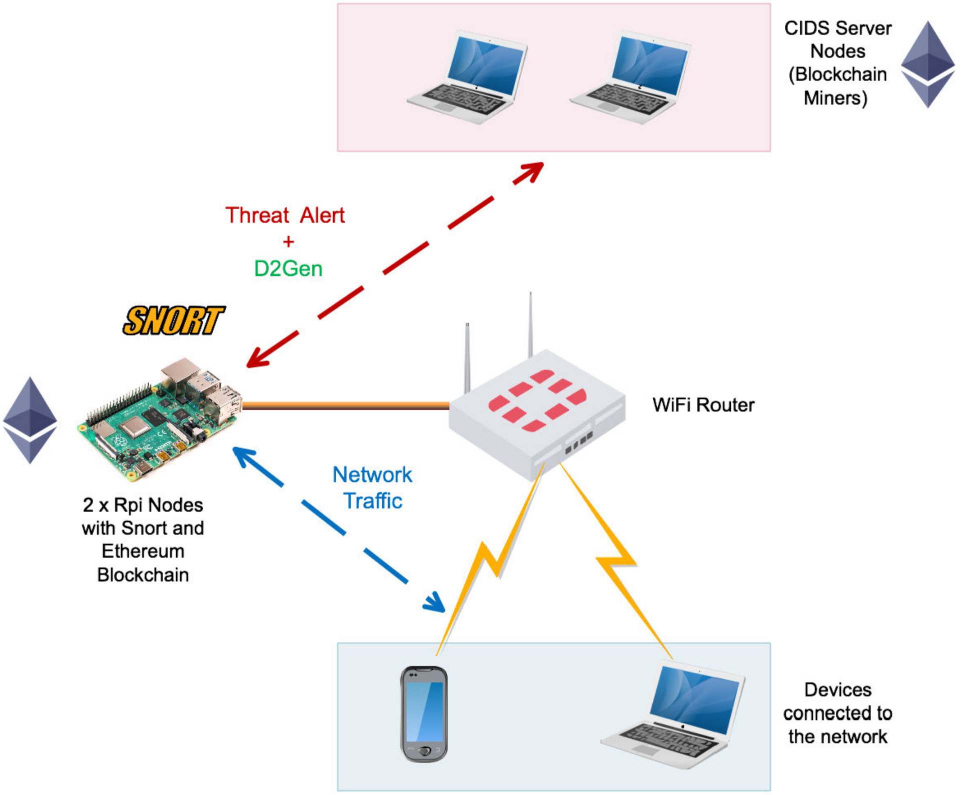
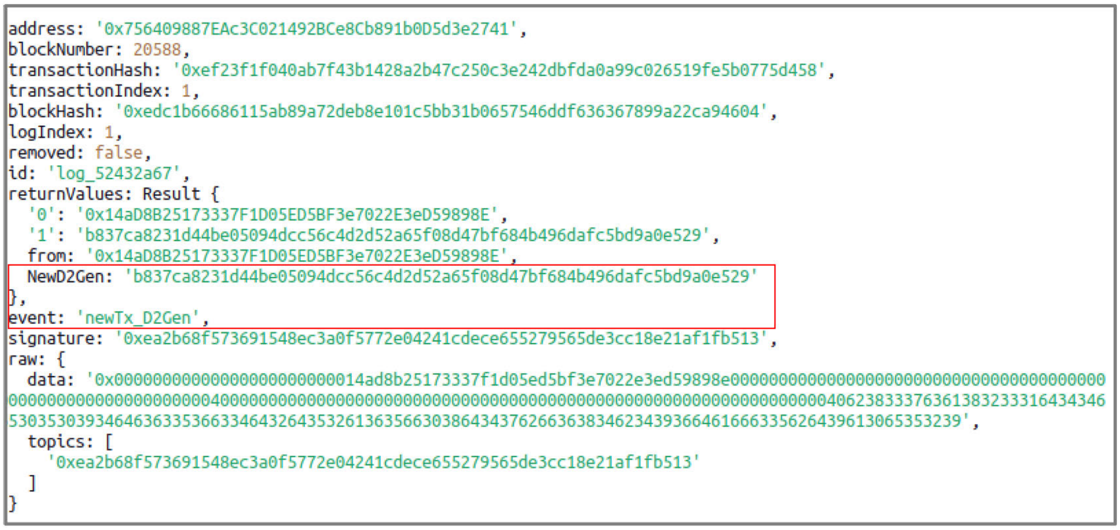

CIDS Server Nodes (Blockchain Miners)
Threat Alert + D2Gen
SNORT
WiFi Router
Network Traffic
2 x Rpi Nodes with Snort and Ethereum Blockchain
Devices connected to the network

**FIGURE 3.** Experimental setup for the POC.

**TABLE 2.** Software and hardware requirements for the POC.

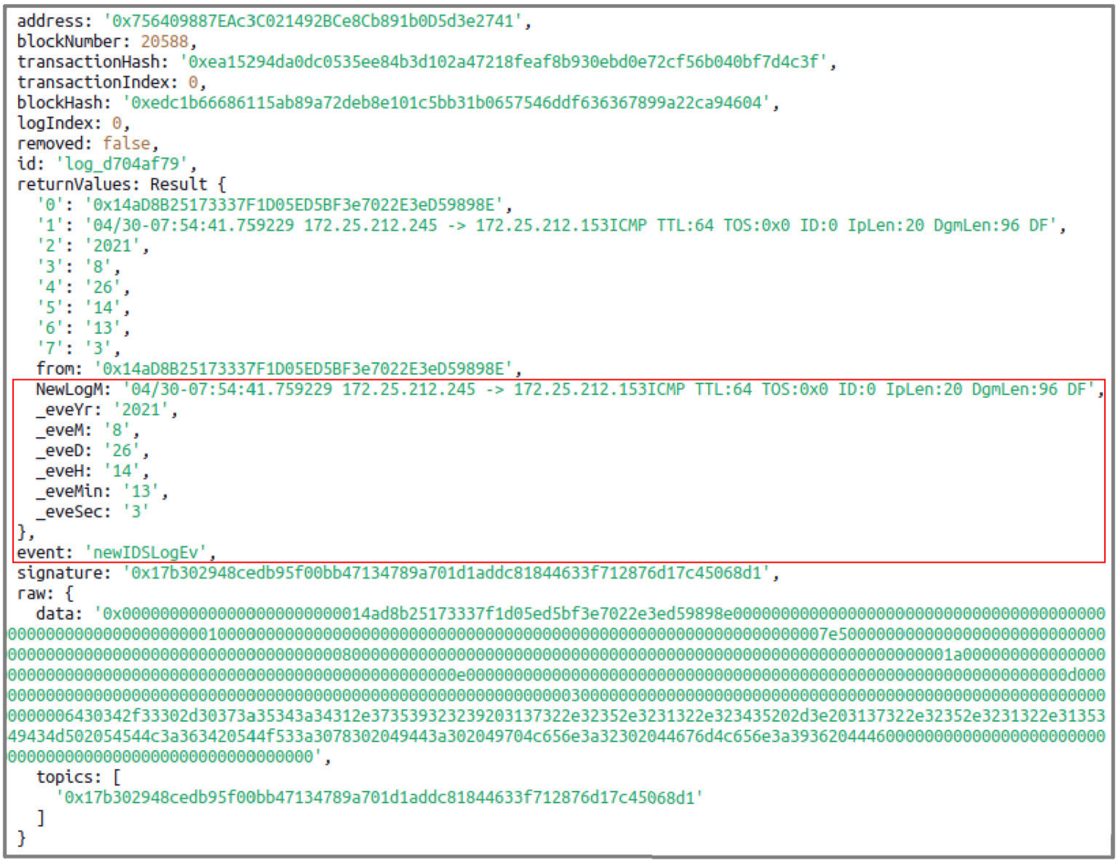
| Component | Specifications |
|---|---|
| **Software** | |
| Ethereum blockchain for Rpi | geth-linux-arm7 |
| IDS for Rpi | snort-2.9.17.1 |
| Rpi Imager | v1.6.1 |
| Rpi OS | Raspbian GNU/Linux 10 (buster) |
| Data acquisition library (DAQ) for Snort (On Rpi) | daq-2.0.7 |
| golang (for CIDS server/miner nodes) | go1.15.linux-amd64 |
| Ethereum (for CIDS server/miner nodes) | Geth 1.8.27 64-bit or higher ver |
| Ethereum IDE to create/test smart contracts | Remix-Ethereum IDE |
| Node js | v14.x |
| NPM | v6.14.x |
| OS (for CIDS server/miner nodes) | Ubuntu 64-bit 21.044 |
| **Hardware** | |
| Raspberry Pi 3 Model B | 64 bit ARM Cortex-A53 Quad Core Processor, 1 GB RAM, 64 GB SD Card (Class 10 or higher), |
| MacBook Pro | 2.6 GHz 6-Core Intel Core i7, 16 GB RAM |

when the smartphone initiates a command to ping a personal computer, the Rpi-based CIDS node immediately detects the ICMP packet and generates an alert. The alert message is output to the snort log file. As soon as a new snort log file is created, a JavaScript on the respective Rpi node executes, and it reads the vital information from the log file (snort.log.x), including protocol type, source, and destination IP address, and ToS. This alert message is then fed into the blockchain by the Rpi CIDS node as a TX. Once the TX is mined in a block by the CIDS server/miner nodes, the rest of the



```
address: '0x756409887EAc3C021492BCe8Cb891b0D5d3e2741',
blockNumber: 20588,
transactionHash: '0xef23f1f040ab7f43b1428a2b47c250c3e242dbfda0a99c026519fe5b0775d458',
transactionIndex: 1,
blockHash: '0xedc1b66686115ab89a72deb8e101c5bb31b0657546ddf636367899a22ca94604',
logIndex: 1,
removed: false,
id: 'log_52432a67',
returnValues: Result {
  '0': '0x14aD8B25173337F1D05ED5BF3e7022E3eD59898E',
  '1': 'b837ca8231d44be05094dcc56c4d2d52a65f08d47bf684b496dafc5bd9a0e529',
  from: '0x14aD8B25173337F1D05ED5BF3e7022E3eD59898E',
  NewD2Gen: 'b837ca8231d44be05094dcc56c4d2d52a65f08d47bf684b496dafc5bd9a0e529'
},
event: 'newTx_D2Gen',
signature: '0xea2b68f573691548ec3a0f5772e04241cdece655279565de3cc18e21af1fb513',
raw: {
  data: '0x00000000000000000000000014ad8b25173337f1d05ed5bf3e7022e3ed59898e00000000000000000000000000000000000000
0000000000000000000000400000000000000000000000000000000000000000000000000000000000000040623833376361383233316434346
5303530393464636335366334643264353261363566303864343762663638346234393664616663356264396130653532 39',
  topics: [
    '0xea2b68f573691548ec3a0f5772e04241cdece655279565de3cc18e21af1fb513'
  ]
}
```

FIGURE 4. New D2Gen event emission.



```
 address: '0x756409887EAc3C021492BCe8Cb891b0D5d3e2741',
 blockNumber: 20588,
 transactionHash: '0xea15294da0dc0535ee84b3d102a47218feaf8b930ebd0e72cf56b040bf7d4c3f',
 transactionIndex: 0,
 blockHash: '0xedc1b66686115ab89a72deb8e101c5bb31b0657546ddf636367899a22ca94604',
 logIndex: 0,
 removed: false,
 id: 'log_d704af79',
 returnValues: Result {
   '0': '0x14aD8B25173337F1D05ED5BF3e7022E3eD59898E',
   '1': '04/30-07:54:41.759229 172.25.212.245 -> 172.25.212.153ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:96 DF',
   '2': '2021',
   '3': '8',
   '4': '26',
   '5': '14',
   '6': '13',
   '7': '3',
   from: '0x14aD8B25173337F1D05ED5BF3e7022E3eD59898E',
   NewLogM: '04/30-07:54:41.759229 172.25.212.245 -> 172.25.212.153ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:96 DF',
   _eveYr: '2021',
   _eveM: '8',
   _eveD: '26',
   _eveH: '14',
   _eveMin: '13',
   _eveSec: '3'
 },
 event: 'newIDSLogEv',
 signature: '0x17b302948cedb95f00bb47134789a701d1addc81844633f712876d17c45068d1',
 raw: {
   data: '0x00000000000000000000000014ad8b25173337f1d05ed5bf3e7022e3ed59898e000000000000000000000000000000000000000
0000000000000000000001000000000000000000000000000000000000000000000000000000000000007e5000000000000000000000000000
0000000000000000000000000000000000080000000000000000000000000000000000000000000000000000000000001a000000000000000
00000000000000000000000000000000000000000000000e000000000000000000000000000000000000000000000000000000000000000d000
00000000000000000000000000000000000000000000000000000000000300000000000000000000000000000000000000000000000000000000
0000006430342f33302d30373a35343a34312e373539323239203137322e32352e3231322e323435202d3e203137322e32352e3231322e31353
49434d502054544c3a363420544f533a3078302049443a302049704c656e3a32302044676d4c656e3a393620444600000000000000000000000
00000000000000000000000000000000',
   topics: [
     '0x17b302948cedb95f00bb47134789a701d1addc81844633f712876d17c45068d1'
   ]
 }
```

FIGURE 5. IDS alert message event emission.

CIDS nodes can update their IDS rules based on this threat alert. Hence, blockchain provides an immutable and transparent view of the threat environment to all the CIDS nodes simultaneously.

### *A. PSEUDOCODES OF THE SMART CONTRACT AND THE JavaScript*

The pseudocodes of the smart contract and the JavaScript code to compute D2Gen, extract IDS log message, and initiate

**Algorithm 1** - CIDS Genome Computation

```
procedure CIDS_Gen(sender_address)   ▷ The address of the contract owner is determined
end procedure
Owner initiates the smart contract Deploy TX
procedure pubBase_D2Gen(Base_D2Gen)
    if $sender == owner$ then
        Set Base_D2Gen
    else
        Return: Operation not permitted
    end if
end procedure
procedure pubLog(IDS_Alert)   ▷ All the CIDS nodes can publish alert messages
    compute current Date and Time, and sender ID
    read IDS log message
    emit new IDS log event
end procedure
procedure pubD2Gen(Current_D2Gen)   ▷ All the CIDS nodes can submit D2Gen with every IDS alert
    compute current D2Gen of Rpi node that is sending IDS log message
    emit new D2Gen event
end procedure
procedure getD2Gen()
    if $sender == owner$ then
        show D2Gen
    else
        return: operation not permitted
    end if
end procedure
procedure getIDS_Log()
    return IDS log message
end procedure
```

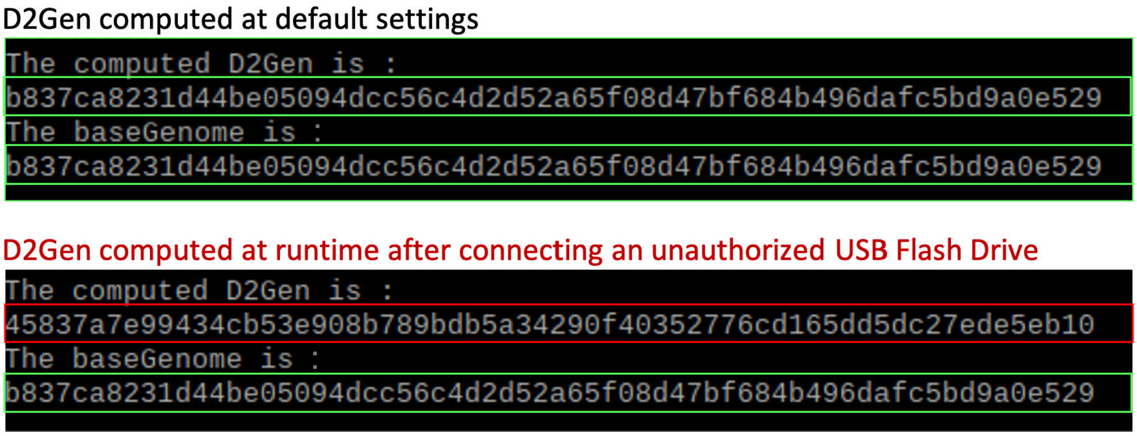
respective TXs are exhibited as Algorithm.1 and Algorithm.2 respectively.

As shown in Algorithm.1, the first function/procedure *CIDS_Gen* sets the contract owner's address on blockchain. Then the contract owner initiates a TX to publish the smart contract on the blockchain. Similarly, only the smart contract owner can save/publish the Base_D2Gen in respect of an end device on the blockchain using a predefined method *PubBase_D2Gen*. The smart contract will check whether the account address that initiated the TX is of the contract owner or not. If the address is not of the smart contract owner, then the operation will not be permitted. Correspondingly, all the CIDS nodes in the network can initiate IDS alert TXs using function *PubLog* that takes IDS_Alert message as a parameter. Once this function is called, it firsts compute the current date and time and sender ID as a part of the time stamp. Then the alert message from the IDS log is read. Later, when the TX to publish IDS log/alert is approved based on network consensus, an event concerning the new IDS log is emitted. Any CIDS server monitoring the network will see the requisite event notification.

Accordingly, a method named *PubD2Gen* can be called by all the CIDS nodes to publish their current D2Gen (at runtime) along with the IDS log/alert TX. In this case, as well, an event is emitted once the TX is approved/mined. The next method in the smart contract is *GetD2Gen*, which the smart contract owner calls to check the Base_D2Gen of a node. Finally, the *GetIDS_Log* function returns the last published IDS alert message.

**Algorithm 2** - Rpi Log Alert and D2Gen TX Submission

```
Require: crypto, os, fs, web3
    procedure os.networkInterfaces()
        return: Rpi network interfaces
    end procedure
    procedure os.hostname()
        return: Rpi hostname
    end procedure
    procedure os.arch()
        return: Rpi CPU architecture
    end procedure
    procedure os.totalmem()
        return: total memory size of Rpi
    end procedure
    procedure check_File_Permissions(access.conf)
        return: status of read and write access to the file
    end procedure
    procedure fs.readFile(networks)
        if Error in reading file then
            print error message
        else
            read: Network_Config
            return: Network_Config
        end if
    end procedure
    procedure getTemp()
        return: Current temperature around the node
    end procedure
    procedure os.cpus()
        return: info about each logical cpu core
    end procedure
    procedure os.platform()
        return: OS platform details
    end procedure
    procedure os.release()
        return: OS platform release details
    end procedure
    procedure os.type()
        return: Type of OS
    end procedure
    procedure os.tempdir()
        return: default dir for temp files
    end procedure
    procedure os.userInfo()
        return: info about user (username, uid, gid, shell, and homedir)
    end procedure
    procedure usb.getDeviceList()
        return: all the connected devices
    end procedure
    procedure D2Gen_Calc(Rpi_Parameters)
        return: D2Gen of Rpi (a hash string)
    end procedure
    procedure fs.readFile(snort_logs)
        if Error in reading file then
            print error message
        else
            read: IDS log message
            return: IDS log data
        end if
    end procedure
    connect to the RPC provider at localhost:8043
    set web3.eth.defaultAccount
    init an instance of smart contract
    procedure pubLog(IDS_Log_data)
        if account is unlock then
            init TX
            display: hash of TX
        else
            display: error message
        end if
    end procedure
    procedure pubD2Gen(new_D2Gen)
        if account is unlock then
            init TX
            display: hash of TX
        else
            display: error message
        end if
    end procedure
```



D2Gen computed at default settings

```
The computed D2Gen is :
b837ca8231d44be05094dcc56c4d2d52a65f08d47bf684b496dafc5bd9a0e529
The baseGenome is :
b837ca8231d44be05094dcc56c4d2d52a65f08d47bf684b496dafc5bd9a0e529
```

D2Gen computed at runtime after connecting an unauthorized USB Flash Drive

```
The computed D2Gen is :
45837a7e99434cb53e908b789bdb5a34290f40352776cd165dd5dc27ede5eb10
The baseGenome is :
b837ca8231d44be05094dcc56c4d2d52a65f08d47bf684b496dafc5bd9a0e529
```

**FIGURE 6.** **Detection of unauthorized USB device.**

Similarly, as shown in Algorithm.2, the JavaScript code on the Rpis (CIDS nodes) require crypto, os (operating system), fs (file system), and web3.js modules for different operations. Moreover, as already enumerated at the end of Section-V para 1, various procedures are used to extract information about specific hardware, software, and network parameters of the respective CIDS node for the computation of Base/runtime D2Gen. Due to the paucity of space, fourteen out of sixteen methods are shown in the algorithm. For instance, *OS.NetworkInterfaces* function returns an array of objects that each contains/describes a network interface that has been assigned a network address. In the same way, *OS.Arch* method returns the operating system CPU architecture of the respective Rpi/CIDS node and *OS.TotalMem* provides the total memory size. Similarly, the *Check_File_Permissions* method retrieves the status of read and write access to the access.conf file.

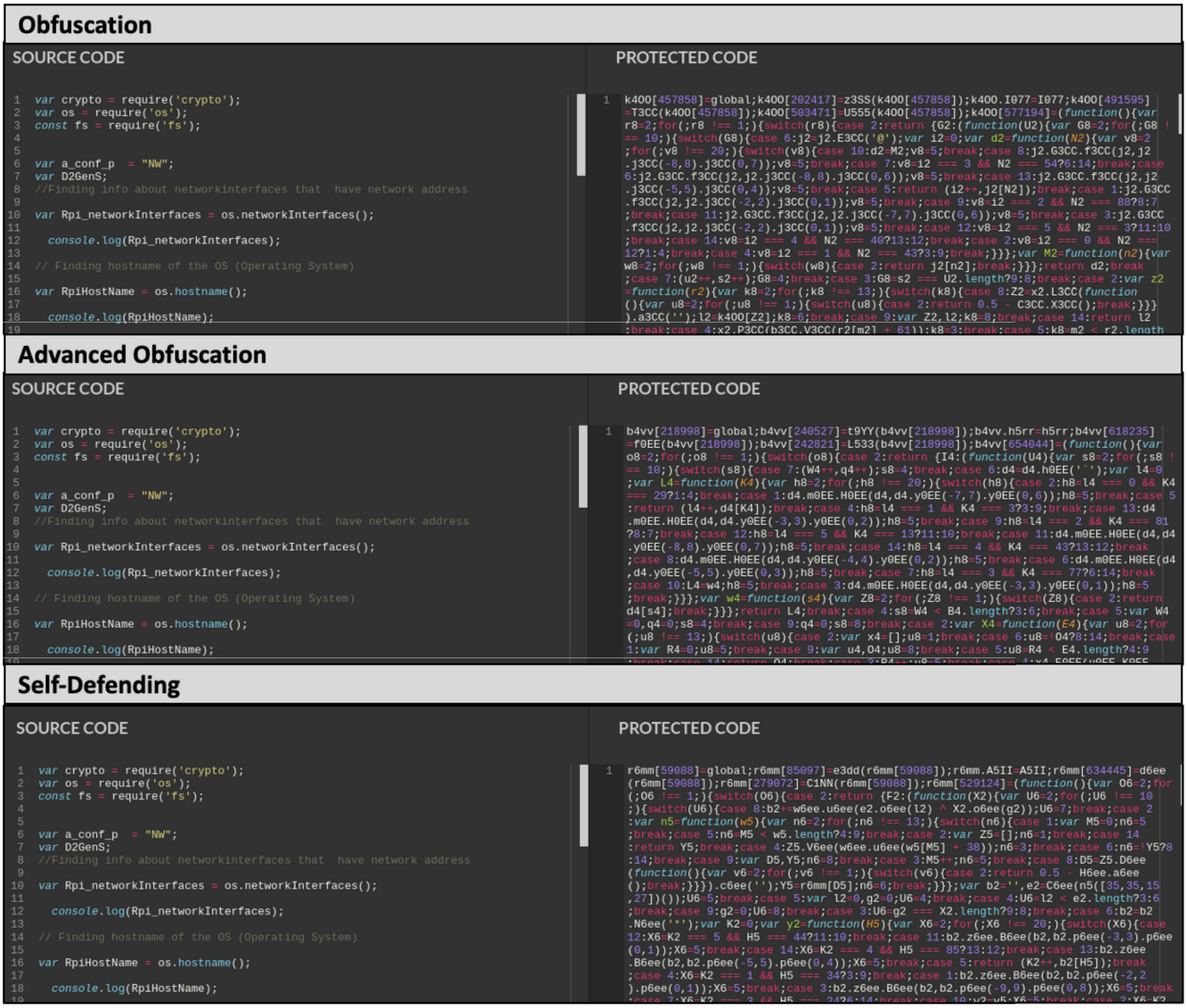
Correspondingly, *FS.ReadFile* function extracts the contents of /etc/networks file, and *GetTemp* returns the ambient temperature of the node. As the temperature is a dynamic parameter, we defined a range of ±5 of the default temperature at different times of the day. Likewise, *OS.CPUs* method provides information about each logical CPU core and *OS.Release* function returns OS platform release details. Similarly, information about the default directory (dir) to store temporary (temp) files is obtained through *OS.TempDir* method. In addition, some important information about the user, including username, user ID (uid), group ID (gid), shell, and home directory (homedir) is extracted by using *OS.UserInfo* function. Another vital parameter that plays a significant role in identifying a compromised node is information about all the connected devices. This is a piece of very useful information that is obtained through *USB.getDeviceList* method.

Furthermore, *D2Gen_Calc* and *FS.ReadFile* methods are called to compute the D2Gen and read IDS log data, respectively. It is followed by the connection of the respective Rpi (CIDS node) on a specific RPC port with the blockchain network. Also, it is essential to set the default Ethereum account/address on Rpi that will be used to pay for the TX costs (in terms of Ethers or Weis). After this, the procedures named *PubLog* and *PubD2Gen* are called to publish IDS alert messages along with respective Rpi node's D2Gen. For both these TXs, the respective Rpi node's default Ethereum account needs to be unlocked using a secure password. Otherwise, the TXs will not be permitted, and error messages will be displayed.

Accordingly, if an attacker alters even a single parameter out of sixteen concerning an end node, the D2Gen computed at runtime will not match the Base_D2Gen already stored on the blockchain. Resultantly, the TX will be rejected, and the respective node will be marked as compromised. E.g., Fig.6 shows the change in D2Gen computed at the run time once an unauthorized (other than default devices) USB device is connected to the respective Rpi node.

### B. SECURITY ANALYSIS

The blockchain-based CIDS framework has been analyzed from the security point of view. In this context, Table.3 highlights the anticipated threats and requisite security measures. Correspondingly, in a private/consortium network setting, every CIDS node's default account is initialized with some ethers or weis by the CIDS server that initializes the blockchain network through genesis file. Hence, no unauthorized CIDS node can input or forge a fake TX in the network. Moreover, being a private/consortium network, the parameters including network ID, RPC, and web socket port numbers required to start and synchronize a node with the blockchain can be hidden from external parties. In addition, whenever a CIDS node initiates a threat alert TX, it has to unlock its default Ethereum account with a preset password known only to him. Hence, it is nearly impossible to spoof a blockchain-based CIDS node. Another security feature is that every node in the blockchain network has a unique enode ID. Unless the miner nodes add that enode ID



Obfuscation

SOURCE CODE

```
var crypto = require('crypto');
var os = require('os');
const fs = require('fs');


var a_conf_p  = "NW";
var D2GenS;
//Finding info about networkinterfaces that  have network address

var Rpi_networkInterfaces = os.networkInterfaces();

  console.log(Rpi_networkInterfaces);

// Finding hostname of the OS (Operating System)

var RpiHostName = os.hostname();

  console.log(RpiHostName);
```

PROTECTED CODE

```
k400[457858]=global;k400[202417]=z3SS(k400[457858]);k400.I077=I077;k400[491595]
=T3CC(k400[457858]);k400[503471]=U555(k400[457858]);k400[577194]=(function(){var
r8=2;for(;r8 !== 1;){switch(r8){case 2:return {G2:(function(U2){var G8=2;for(;G8 !
== 10;){switch(G8){case 6:j2=j2.E3CC('@');var i2=0;var d2=function(N2){var v8=2
;for(;v8 !== 20;){switch(v8){case 10:d2=M2;v8=5;break;case 8:j2.G3CC.f3CC(j2,j2
.j3CC(-8,8).j3CC(0,7));v8=5;break;case 7:v8=i2 === 3 && N2 === 54?6:14;break;case
6:j2.G3CC.f3CC(j2,j2.j3CC(-8,8).j3CC(0,6));v8=5;break;case 13:j2.G3CC.f3CC(j2,j2
.j3CC(-5,5).j3CC(0,4));v8=5;break;case 5:return (i2++,j2[N2]);break;case 1:j2.G3CC
.f3CC(j2,j2.j3CC(-2,2).j3CC(0,1));v8=5;break;case 9:v8=i2 === 2 && N2 === 88?8:7
;break;case 11:j2.G3CC.f3CC(j2,j2.j3CC(-7,7).j3CC(0,6));v8=5;break;case 3:j2.G3CC
.f3CC(j2,j2.j3CC(-2,2).j3CC(0,1));v8=5;break;case 12:v8=i2 === 5 && N2 === 3?11:10
;break;case 14:v8=i2 === 4 && N2 === 40?13:12;break;case 2:v8=i2 === 0 && N2 ===
12?1:4;break;case 4:v8=i2 === 1 && N2 === 43?3:9;break;}}};var M2=function(n2){var
w8=2;for(;w8 !== 1;){switch(w8){case 2:return j2[n2];break;}}};return d2;break
;case 7:(u2++,s2++);G8=4;break;case 3:G8=s2 === U2.length?9:8;break;case 2:var z2
=function(r2){var k8=2;for(;k8 !== 13;){switch(k8){case 8:Z2=x2.L3CC(function
(){var u8=2;for(;u8 !== 1;){switch(u8){case 2:return 0.5 - C3CC.X3CC();break;}}}
).a3CC('');l2=k400[Z2];k8=6;break;case 9:var Z2,l2;k8=8;break;case 14:return l2
;break;case 4:x2.P3CC(b3CC.V3CC(r2[m2] + 61));k8=3;break;case 5:k8=m2 < r2.length
```

Advanced Obfuscation

SOURCE CODE

```
var crypto = require('crypto');
var os = require('os');
const fs = require('fs');


var a_conf_p  = "NW";
var D2GenS;
//Finding info about networkinterfaces that  have network address

var Rpi_networkInterfaces = os.networkInterfaces();

  console.log(Rpi_networkInterfaces);

// Finding hostname of the OS (Operating System)

var RpiHostName = os.hostname();

  console.log(RpiHostName);
```

PROTECTED CODE

```
b4vv[218998]=global;b4vv[240527]=t9YY(b4vv[218998]);b4vv.h5rr=h5rr;b4vv[618235]
=f0EE(b4vv[218998]);b4vv[242821]=L533(b4vv[218998]);b4vv[654044]=(function(){var
o8=2;for(;o8 !== 1;){switch(o8){case 2:return {I4:(function(U4){var s8=2;for(;s8 !
== 10;){switch(s8){case 7:(W4++,q4++);s8=4;break;case 6:d4=d4.h0EE('`');var l4=0
;var L4=function(K4){var h8=2;for(;h8 !== 20;){switch(h8){case 2:h8=l4 === 0 && K4
=== 29?1:4;break;case 1:d4.m0EE.H0EE(d4,d4.y0EE(-7,7).y0EE(0,6));h8=5;break;case 5
:return (l4++,d4[K4]);break;case 4:h8=l4 === 1 && K4 === 3?3:9;break;case 13:d4
.m0EE.H0EE(d4,d4.y0EE(-3,3).y0EE(0,2));h8=5;break;case 9:h8=l4 === 2 && K4 === 81
?8:7;break;case 12:h8=l4 === 5 && K4 === 13?11:10;break;case 11:d4.m0EE.H0EE(d4,d4
.y0EE(-8,8).y0EE(0,7));h8=5;break;case 14:h8=l4 === 4 && K4 === 43?13:12;break
;case 8:d4.m0EE.H0EE(d4,d4.y0EE(-4,4).y0EE(0,2));h8=5;break;case 6:d4.m0EE.H0EE(d4
,d4.y0EE(-5,5).y0EE(0,3));h8=5;break;case 7:h8=l4 === 3 && K4 === 77?6:14;break
;case 10:L4=w4;h8=5;break;case 3:d4.m0EE.H0EE(d4,d4.y0EE(-3,3).y0EE(0,1));h8=5
;break;}}};var w4=function(s4){var Z8=2;for(;Z8 !== 1;){switch(Z8){case 2:return
d4[s4];break;}}};return L4;break;case 4:s8=W4 < B4.length?3:6;break;case 5:var W4
=0,q4=0;s8=4;break;case 9:q4=0;s8=8;break;case 2:var X4=function(E4){var u8=2;for
(;u8 !== 13;){switch(u8){case 2:var x4=[];u8=1;break;case 6:u8=!O4?8:14;break;case
1:var R4=0;u8=5;break;case 9:var u4,O4;u8=8;break;case 5:u8=R4 < E4.length?4:9
;break;case 14:return O4;break;case 2:R4++;u8=5;break;case 4:x4.E0EE(u0EE.K0EE
```

Self-Defending

SOURCE CODE

```
var crypto = require('crypto');
var os = require('os');
const fs = require('fs');


var a_conf_p  = "NW";
var D2GenS;
//Finding info about networkinterfaces that  have network address

var Rpi_networkInterfaces = os.networkInterfaces();

  console.log(Rpi_networkInterfaces);

// Finding hostname of the OS (Operating System)

var RpiHostName = os.hostname();

  console.log(RpiHostName);
```

PROTECTED CODE

```
r6mm[59088]=global;r6mm[85097]=e3dd(r6mm[59088]);r6mm.A5II=A5II;r6mm[634445]=d6ee
(r6mm[59088]);r6mm[279072]=C1NN(r6mm[59088]);r6mm[529124]=(function(){var O6=2;for
(;O6 !== 1;){switch(O6){case 2:return {F2:(function(X2){var U6=2;for(;U6 !== 10
;){switch(U6){case 8:b2+=w6ee.u6ee(e2.o6ee(l2) ^ X2.o6ee(g2));U6=7;break;case 2
:var n5=function(w5){var n6=2;for(;n6 !== 13;){switch(n6){case 1:var M5=0;n6=5
;break;case 5:n6=M5 < w5.length?4:9;break;case 2:var Z5=[];n6=1;break;case 14
:return Y5;break;case 4:Z5.V6ee(w6ee.u6ee(w5[M5] + 38));n6=3;break;case 6:n6=!Y5?8
:14;break;case 9:var D5,Y5;n6=8;break;case 3:M5++;n6=5;break;case 8:D5=Z5.D6ee
(function(){var v6=2;for(;v6 !== 1;){switch(v6){case 2:return 0.5 - H6ee.a6ee
();break;}}}).c6ee('');Y5=r6mm[D5];n6=6;break;}}};var b2='',e2=C6ee(n5([35,35,15
,27])());U6=5;break;case 5:var l2=0,g2=0;U6=4;break;case 4:U6=l2 < e2.length?3:6
;break;case 9:g2=0;U6=8;break;case 3:U6=g2 === X2.length?9:8;break;case 6:b2=b2
.N6ee('*');var K2=0;var y2=function(H5){var X6=2;for(;X6 !== 20;){switch(X6){case
12:X6=K2 === 5 && H5 === 44?11:10;break;case 11:b2.z6ee.B6ee(b2,b2.p6ee(-3,3).p6ee
(0,1));X6=5;break;case 14:X6=K2 === 4 && H5 === 85?13:12;break;case 13:b2.z6ee
.B6ee(b2,b2.p6ee(-5,5).p6ee(0,4));X6=5;break;case 5:return (K2++,b2[H5]);break
;case 4:X6=K2 === 1 && H5 === 34?3:9;break;case 1:b2.z6ee.B6ee(b2,b2.p6ee(-2,2
).p6ee(0,1));X6=5;break;case 3:b2.z6ee.B6ee(b2,b2.p6ee(-9,9).p6ee(0,8));X6=5;break
;case 7:X6=K2 === 3 && H5 === 24?6:14;break;case 10:y2=y5;X6=5;break;case 2:X6=K2
```

FIGURE 7. Code obfuscation methods.

as a peer, the respective node cannot synchronize to the main blockchain.

Concerning the security of smart contracts, as per our testbed settings, only one of the CIDS server nodes is allowed to deploy the smart contract to the blockchain as an owner. Later, all other nodes can access the smart contract methods using the smart contract application binary interface (Abi) and the contract address. Except for the smart contract owner, no node can discretely make any change to the smart contract. Moreover, even if the owner modifies or upgrades the smart contract, it needs to be re-deployed on the blockchain. Hence, this adds to the system's overall security, as no change can be made to the smart contacts stealthily. Besides, the access to the smart contract mutator methods that modify the state of the threat alerts in respect of a CIDS node can be restricted to the authorized addresses/accounts of the nodes. In general, the blockchain-based CIDS is secure against data integrity, forgery, and application/policy violation threats. Accordingly, the IDS log message and the D2Gen TXs initiated from the respective Rpi-based CIDS node cannot be replayed once mined in a block. It is because the IDS log TX emits an event bearing a timestamp showing date and time when the TX was submitted. Hence, a replayed TX with an old timestamp can easily be detected by the CIDS server that is listening to the event emissions.

Another critical aspect is the security of the JavaScript code running on Rpi-based CIDS nodes. This script computes D2Gen at runtime, extracts snort IDS log messages, and then pushes these parameters into the blockchain. Though blockchain is immutable and prevents data forging and modification attacks, data integrity is threatened before it is pushed into the blockchain. Moreover, there is always a credible threat of physical compromise of CIDS nodes. An attacker may read and modify the code to prevent computation

**TABLE 3.** Threats and countermeasures.

| Threats | Security Measures |
|---|---|
| Participation of unauthorized nodes in the network | • Only authorized nodes' default accounts are initialized with ethers or weis through genesis file<br>• Parameters required to initialize a node such as a network ID, RPC, and web socket port numbers, are not made public<br>• To initialize a TX, a CIDS node has to unlock its default account with a password known only to the node owner<br>• Only authorized nodes are added as peers to the miner or validator nodes. Hence, an unauthorized node cannot synchronize and download blockchain data |
| Modification or forging of smart contracts | • Only the network owner can deploy a smart contract on the blockchain (after network consensus)<br>• Except for network owner, no other CIDS node can make any change to the smart contracts<br>• Even network owner cannot modify the smart contracts stealthily due to the requirement of network consensus to deploy an updated version of the smart contracts |
| Fake TXs affecting the performance of the network | Access to smart contract mutator methods is limited to selected/authorized CIDS nodes only |
| Replay attacks | A timestamp is appended to every TX |
| Security of JavaScript code against tampering, unauthorized analysis, fake emulated response, and replaying of old D2Gen | • Use of JavaScript obfuscator<br>• Use of various methods/functions builtin the blockchain to extract device information |
| Man-in-the-Middle (MITM) attacks, where an attacker may monitor and eavesdrop on CIDS nodes' communication links | • Every CIDS node is a blockchain client. So, these nodes input data directly to the blockchain at their end. Hence, no MITM attacker without access to the blockchain network can see TX data<br>• Only authorized CIDS servers can see the Base_D2Gen of the end nodes for attestation |

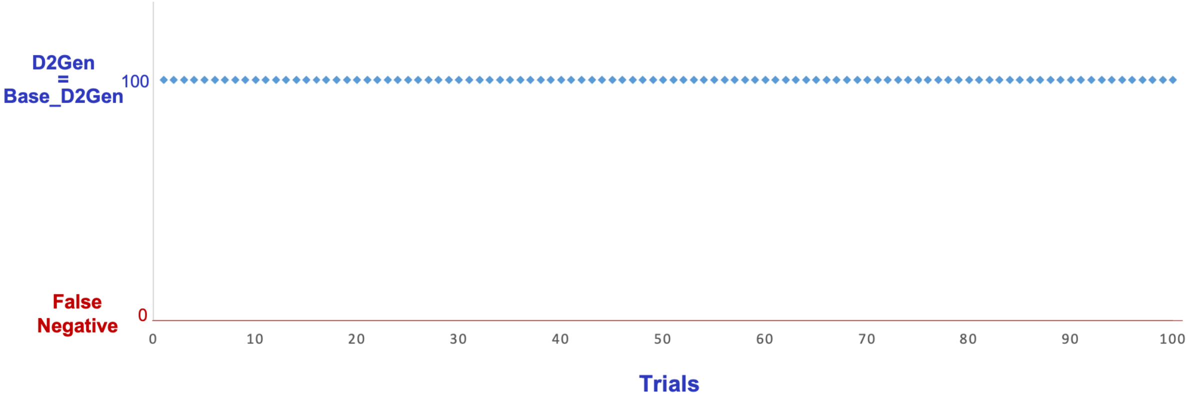


**FIGURE 8.** Number of false negatives.

of D2Gen at runtime and replay an old response. Hence, to ensure code integrity and avoid replay attacks, we used JavaScript Obfuscator [49] to convert the JavaScript source code running on Rpi-based CIDS nodes into an unreadable form thus preventing unauthorized analysis and tampering. Currently, "Obfuscation" is considered as one of the cogent methods for protecting JavaScript codes from reverse engineering. The obfuscated code becomes unintelligible for

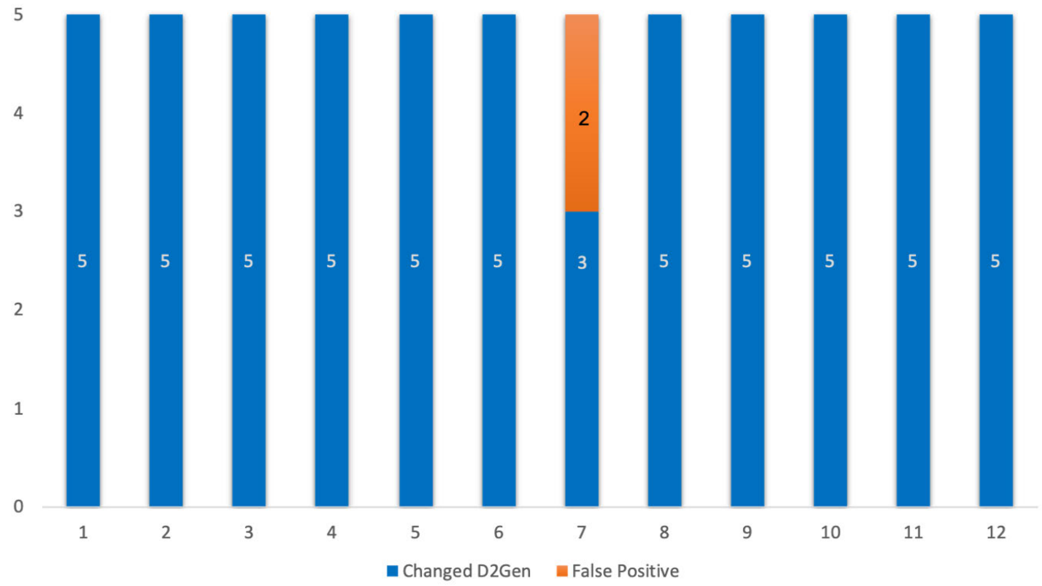


**FIGURE 9.** Number of false positives.

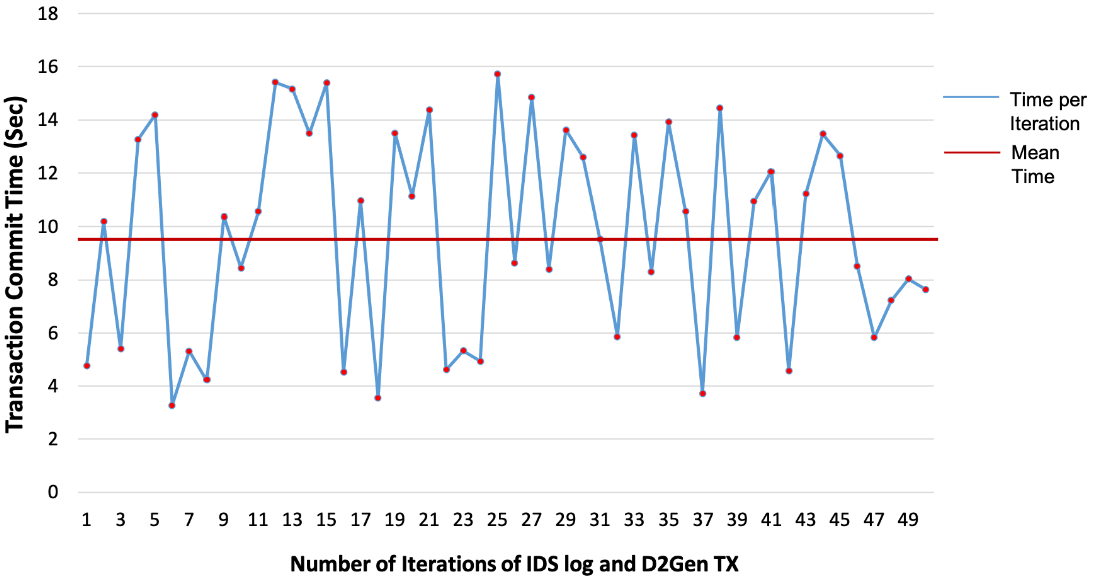


**FIGURE 10.** Transaction commit time.

viewers, but its functionality remains the same as the original code.

Our experiments tested three different code obfuscation methods: simple obfuscation, advanced obfuscation, and self-defending. As shown in Fig.7, the protected code looks different in the case of all three methods, i.e., obfuscation, advanced obfuscation, and self-defending. However, once executed, all the protected codes displayed the same functionality. Moreover, it is also quite evident that the protected code is nearly impossible for an unauthorized or malicious person to understand and modify.

Nonetheless, there may be a question about the security and performance efficiency of the JavaScript obfuscators. In this regard, researchers in [50] carried out a thorough comparison of online JavaScript obfuscators based on obfuscation techniques, potency, resilience to deobfuscation, and costs associated with the process. The authors, identified "obfuscators.io" [51] to be the most potent, and difficult to

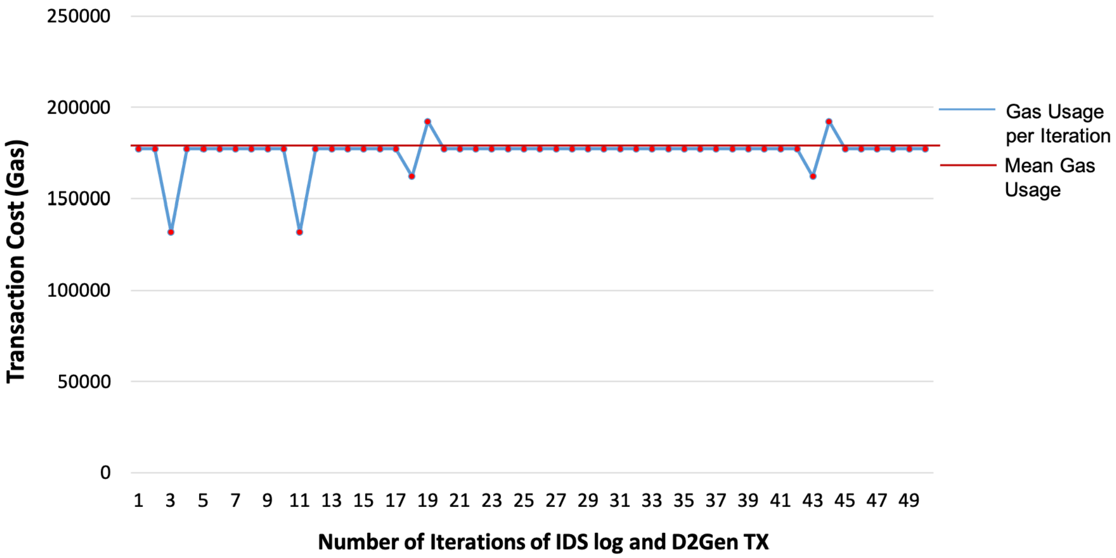


**FIGURE 11.** Transaction cost (gas usage).

be reversed obfuscator. However, with increased resilience and potent obfuscation, the size of the code also increases significantly. Accordingly, obfuscator.io infers an increase of 36.23% as compared to an increase of just 1.06% in the case of JavaScript obfuscator [49]. Hence, the selection of an obfuscator may depend on the security and the performance preferences of a system/application. Thus, for critical systems, the security of code and resilience to adversary attacks may lead to the acceptance of low performance or increased code size. On the contrary, for some real-time systems, a low level of resilience may be enough in relation to increased performance output. Correspondingly, considering the security requirements of the D2Gen protocol, we finally used obfuscator.io to scramble JavaScript codes.

Furthermore, to evaluate the accuracy of the proposed framework concerning false negatives, i.e., a legitimate node fails the integrity verification, we ran the D2Gen protocol for hundred times. As shown in Fig.8, for all hundred trials, D2Gen computed at runtime was exactly the same as Base_D2Gen. Likewise, to determine the probability of false positives, i.e., a compromised node qualifies the integrity check, we changed twelve out of sixteen parameters five times each. The altered parameters include configuration of network interfaces, attached devices, the hostname of the node, file permissions, contents of networks file, ambient temperature, OS release, type and version, the default directory for temporary files, and user information. The rest of the four parameters required changes in the device hardware. Accordingly, as shown in Fig.9, all parameter modifications except serial 7 (ambient temperature) resulted in a change in D2Gen, thus dictating that the end node has been tampered with. However, we obtained two false positives concerning ambient temperature, i.e., twice the change in ambient temperature resulted in unchanged D2Gen. This behavior substantiates the need to carefully set/normalize the acceptable range of values for the dynamic parameters.

### C. PERFORMANCE ANALYSIS

To assess the performance efficiency of the D2Gen framework, we measured TX commit time which includes time taken to compute D2Gen of a node at run-time, extraction of IDS log message, submission of TXs, and finally mining of the TXs in a block. We also measured TX cost in terms of gas usage and the block difficulty for fifty iterations of the D2Gen protocol. Where gas is a unit of measuring computational effort that is required to execute specific operations on the Ethereum network [52]. Since each Ethereum TX requires computational resources to execute, every TX requires a fee. Hence, gas refers to the cost needed to conduct a TX on Ethereum successfully. Therefore, a TX with low gas usage is considered to be economical. Nonetheless, in each iteration, the Rpi-based IDS node computes its D2Gen over sixteen different parameters (already mentioned in Section V) and then submits two separate TXs comprising D2Gen and IDS alert message.

As shown in Fig.10, the time taken to run the D2Gen protocol and mine respective TXs in a block varies from 3.28 secs to 15.72 secs with a mean TX commit time of 9.67 secs. Whereas the expected block time in the public Ethereum blockchain is between 10 to 19 secs, and in Bitcoin blockchain, it is 10 mins. Similarly, Fig.11 shows the TX cost in terms of gas usage for the fifty iterations of the D2Gen protocol. It is evident that except few iterations, the cost (gas usage) for the mining of respective blocks is almost static at 177190, i.e., 0.177 Million as compared to the Ethereum block gas limit of 15000000 (Fifteen Million) [53]. Therefore, it can be inferred that the computation requirements to compute D2Gen over maximum software and hardware

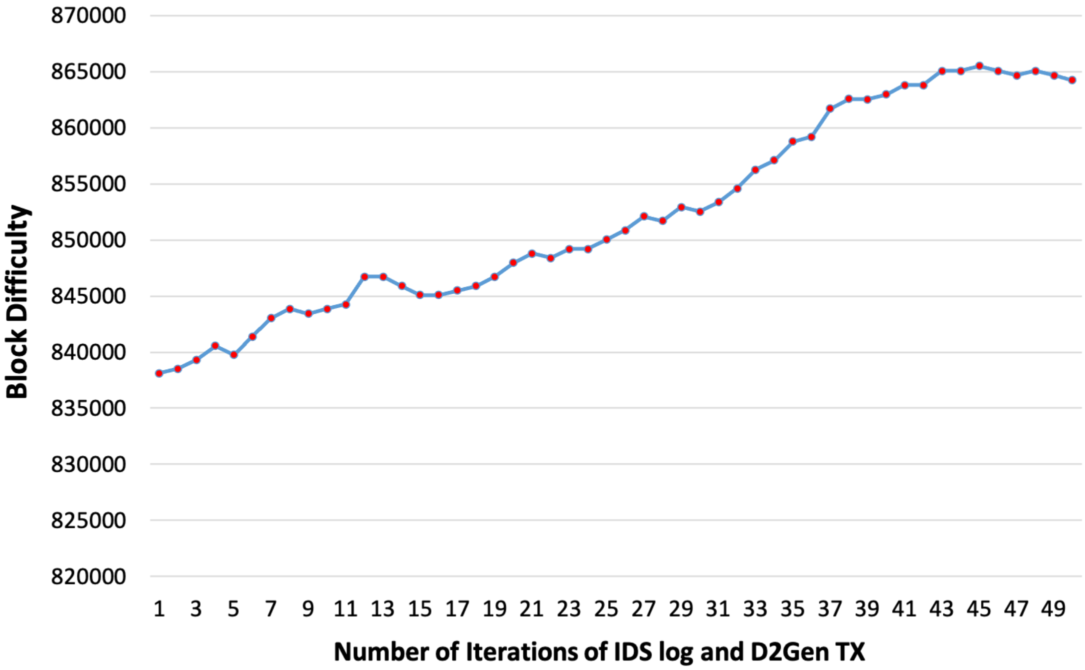


**FIGURE 12.** Block difficulty.

**TABLE 4.** Experimental results.

| Test | Measurement (Avg of 50 iterations) | Comparative Value |
|---|---|---|
| TX commit time | 9.67 sec | Ethereum - 10 to 19 sec and Bitcoin - 10 min |
| TX cost (gas) | 178918 (0.178 million) | Ethereum block gas limit - 15 million |
| Block difficulty | 8.52 x $10^5$ | Ethereum blockchain block difficulty - 8.54 x $10^{15}$ |

components and mine respective TXs in a block would be well under the allowable Ethereum block gas limit of fifteen million. In addition, Fig.12 presents the trend of change in block difficulty for fifty iterations. A summary of the experimental results is shown in Table.4.

Though in this experiment, the block limit gradually increases, it drops as well at a few instances. Similarly, in a real-world Ethereum network, the block difficulty experiences both increases and decrease as an adjustment to keep the block time between 10-19 secs. E.g., if the current block is mined in less than 10 secs, then the block difficulty will be increased accordingly to bring the block time for the next block within the desired limit of 10-19 secs [54]. Similarly, on $4^{th}$ September 2021, the Ethereum block difficulty was $8.60 \times 10^{15}$ [55] as compared to D2Gen protocol's difficulty of $8.52 \times 10^5$.

The phenomenon of varying difficulty is called as "Difficulty Bomb." Correspondingly, an adjustment of 10-20 secs is deemed essential after every 100,000 blocks in the Ethereum blockchain. It serves two purposes, i.e., security and scalability. Concerning security, the increase in block difficulty caters to the rise in computation power by the block miners. Correspondingly, a limit to minimum block creation time prevents the creation of blocks more quickly, thus avoiding a rapid increase in blockchain size.

Besides, there may be a concern that the Ethereum 1.0 network with a TX throughput of only 30 TXs per sec may not be feasible for a CIDS network. In this regard, it is highlighted that Phase-0 of transition to Ethereum 2.0 network had been initiated in Dec 2020 [56]. The notable features of the Ethereum 2.0 upgrade include; Proof of Stake (PoS) consensus protocol [57], shard chains, and an entirely new blockchain named as "Beacon Chain" [58]. By implementing blockchain shards, nodes in Ethereum 2.0 network will be able to manage slices of the network. As Ethereum 2.0 is expected to be launched with 64 shards hence, it will give 64 times more TX throughput than its predecessor Ethereum 1.0. Correspondingly, it is foreseen that the TX throughput of Ethereum 2.0 will be around 100,000 TXs per second [58] as compared to the maximum achievable TX throughput of 20,000 TXs per sec in case of an optimized version of Hyperledger Fabric [59].

## VI. CONCLUSION AND FUTURE WORK

This article introduced a unique scheme of validating IoT device integrity by computing the device's digital genome. This concept is derived from the identification of humans based on their genome profiling. The very idea of genome profiling motivated us to apply the same approach to digital devices and develop an initial design of the proposed integrity check framework for CIDSs. The computation of the device's digital genome is based on numerous attributes measured from the device hardware and software components, network configuration/settings, and device metadata. To ensure secure computation of the base genome and its immutable storage, blockchain technology is a pedestal of the proposed framework design. Although the proffered security and performance analysis of the POC compliments the feasibility of the D2Gen-based device integrity check mechanism, some distinct challenges and limitations highlighted in the article need to be addressed to develop a fully functional product.

Correspondingly, in the future, we intend to extend the number of hardware, software, and device configuration parameters that contribute to the computation of D2Gen. Sequel to this, segregation of dynamic and static parameters is also very vital. Accordingly, the dynamic parameters, such as voltages and current levels at specific device interfaces, contents of system files, device configuration settings, etc., need to be evaluated to set/normalize the threshold so that malicious behavior can be easily detected. Moreover, to select the most suitable blockchain technology, we aim to test the extended version of D2Gen on Ethereum 2.0, Hyperledger Fabric, and IOTA Smart Contracts Protocol (ISCP). Then based on the security and performance analysis, the best performing blockchain platform will be used in the production version of D2Gen.

It is believed that a successful realization of this scheme will help detect a malicious IoT/IIoT/digital device even if a single byte of data is changed in its software components. Also, if a minute change is made in device's hardware/network configuration or metadata, the forged device will be easily identified. Resultantly, it will not only provide security guarantees for a large number of IoT applications but will also boost the financial impact of the already established multi-billion-dollar IoT industry.

## REFERENCES

[1] Y. Lu, "The blockchain: State-of-the-art and research challenges," *J. Ind. Inf. Integr.*, vol. 15, pp. 80–90, Sep. 2019.

[2] Y. Li, M. Hou, H. Liu, and Y. Liu, "Towards a theoretical framework of strategic decision, supporting capability and information sharing under the context of Internet of Things," *Inf. Technol. Manage.*, vol. 13, no. 4, pp. 205–216, Dec. 2012.

[3] L. Da Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[4] M. Zarei, A. Mohammadian, and R. Ghasemi, "Internet of Things in industries: A survey for sustainable development," *Int. J. Innov. Sustain. Develop.*, vol. 10, no. 4, pp. 419–442, 2016.

[5] S. Vitturi, C. Zunino, and T. Sauter, "Industrial communication systems and their future challenges: Next-generation Ethernet, IIoT, and 5G," *Proc. IEEE*, vol. 107, no. 6, pp. 944–961, Jun. 2019.

[6] I. Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni, "Anatomy of threats to the Internet of Things," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1636–1675, 2nd Quart., 2018.

[7] Z. Trabelsi, K. Hayawi, A. Braiki, and S. Mathew, *Network Attacks and Defenses: A Hands-on Approach*. Boca Raton, FL, USA: CRC Press, 2012. [Online]. Available: https://books.google.com.au/books?id=lUPSBQAAQBAJ

[8] K. Hayawi, Z. Trabelsi, S. Zeidan, and M. M. Masud, "Thwarting ICMP low-rate attacks against firewalls while minimizing legitimate traffic loss," *IEEE Access*, vol. 8, pp. 78029–78043, 2020.

[9] M. Mesbah and M. Azer, "Cyber threats and policies for industrial control systems," in *Proc. Int. Conf. Smart Appl., Commun. Netw. (SmartNets)*, Dec. 2019, pp. 1–6.

[10] S. K. Venkatachary, J. Prasad, and R. Samikannu, "Cybersecurity and cyber terrorism–in energy sector—A review," *J. Cyber Secur. Technol.*, vol. 2, nos. 3–4, pp. 111–130, Oct. 2018.

[11] A. Wang, R. Liang, X. Liu, Y. Zhang, K. Chen, and J. Li, "An inside look at IoT malware," in *Proc. Int. Conf. Ind. IoT Technol. Appl.*, F. Chen and Y. Luo, Eds. Wuhu, China: Springerg, 2017, pp. 176–186.

[12] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, Apr. 2011.

[13] E. E. Schultz, "A framework for understanding and predicting insider attacks," *Comput. Secur.*, vol. 21, no. 6, pp. 526–531, Oct. 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740480201009X

[14] N. Kolokotronis, S. Brotsis, G. Germanos, C. Vassilakis, and S. Shiaeles, "On blockchain architectures for trust-based collaborative intrusion detection," in *Proc. IEEE World Congr. Services (SERVICES)*, Jul. 2019, pp. 21–28.

[15] W. Meng, W. Li, and L. Zhu, "Enhancing medical smartphone networks via blockchain-based trust management against insider attacks," *IEEE Trans. Eng. Manag.*, vol. 67, no. 4, pp. 1377–1386, Nov. 2020.

[16] C. Duma, M. Karresand, N. Shahmehri, and G. Caronni, "A trust-aware, P2P-based overlay for intrusion detection," in *Proc. 17th Int. Conf. Database Expert Syst. Appl. (DEXA)*, Sep. 2006, pp. 692–697.

[17] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: Software-based attestation for embedded devices," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 272–282.

[18] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in *Advances in Cryptology—CRYPTO*, D. Boneh, Ed. Berlin, Germany: Springer, 2003, pp. 617–630.

[19] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil attack in sensor networks: Analysis & defenses," in *Proc. 3rd Int. Symp. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2004, pp. 259–268.

[20] L. J. Craig, "A tour of tocttous," SANS Inst., Rockville, MD, USA, White Paper v1.4b, Aug. 2002, pp. 1–11. [Online]. Available: https://www.sans.org/reading-room/whitepapers/securecode/tour-tocttous-%1049

[21] M. Prandini and M. Ramilli, "Return-oriented programming," *IEEE Secur. Privacy*, vol. 10, no. 6, pp. 84–87, Nov. 2012.

[22] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, Oct. 2007, pp. 552–561, doi: 10.1145/1315245.1315313.

[23] Y. Yang, X. Wang, S. Zhu, and G. Cao, "Distributed software-based attestation for node compromise detection in sensor networks," in *Proc. 26th IEEE Int. Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2007, pp. 219–230.

[24] I. Makhdoom, M. Afzal, and I. Rashid, "A novel code attestation scheme against Sybil attack in wireless sensor networks," in *Proc. Nat. Softw. Eng. Conf.*, Nov. 2014, pp. 1–6.

[25] K. Song, D. Seo, H. Park, H. Lee, and A. Perrig, "OMAP: One-way memory attestation protocol for smart meters," in *Proc. IEEE 9th Int. Symp. Parallel Distrib. Process. With Appl. Workshops*, May 2011, pp. 111–118.

[26] B. Lee and H.-J. Lee, "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment," *J. Supercomput.*, vol. 73, no. 3, pp. 1152–1167, Sep. 2016.

[27] H. R. Ghaeini, M. Chan, R. Bahmani, F. Brasser, L. Garcia, J. Zhou, A.-R. Sadeghi, N. O. Tippenhauer, and S. Zonouz, "PAtt: Physics-based attestation of control systems," in *Proc. 22nd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, Beijing, China: USENIX Association, Sep. 2019, pp. 165–180. [Online]. Available: https://www.usenix.org/conference/raid2019/presentation/ghaeini

[28] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "HomeSnitch: Behavior transparency and control for smart home IoT devices," in *Proc. 12th Conf. Secur. Privacy Wireless Mobile Netw.*, May 2019, pp. 128–138, doi: 10.1145/3317549.3323409.

[29] N. Falliere, L. O. Murchu, and E. Chien, "W32. Stuxnet dossier," *White Paper, Symantec Corp., Secur. Response*, vol. 5, no. 6, p. 29, 2011.

[30] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "Healed: Healing & attestation for low-end embedded devices," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, I. Goldberg and T. Moore, Eds. St. Kitts, Saint Kitts and Nevis: Springer, 2019, pp. 627–645.

[31] C. J. Fung, "Collaborative intrusion detection networks and insider attacks," *J. Wireless Mobile Netw., Ubiquitous Comput., Dependable Appl.*, vol. 2, no. 1, pp. 63–74, 2011.

[32] C. J. Fung, O. Baysal, J. Zhang, I. Aib, and R. Boutaba, "Trust management for host-based collaborative intrusion detection," in *Proc. Int. Workshop Distrib. Syst., Oper. Manage.*, F. De Turck, W. Kellerer, and G. Kormentzas, Eds. Berlin, Germany: Springer, 2008, pp. 109–122.

[33] A. Ghosh and S. Sen, "Agent-based distributed intrusion alert system," in *Proc. 6th Int. Workshop Distrib. Comput. (IWDC)*, A. Sen, N. Das, S. K. Das, and B. P. Sinha, Eds. Hiroshima, Japan: Springer, 2005, pp. 240–251.

[34] M. E. Locasto, J. J. Parekh, A. D. Keromytis, and S. J. Stolfo, "Towards collaborative security and P2P intrusion detection," in *Proc. 6th Annu. IEEE Syst., Man Cybern. (SMC) Inf. Assurance Workshop*, Jun. 2005, pp. 333–339.

[35] N. V. Abhishek, T. J. Lim, B. Sikdar, and A. Tandon, "An intrusion detection system for detecting compromised gateways in clustered IoT networks," in *Proc. IEEE Int. Workshop Tech. Committee Commun. Qual. Rel. (CQR)*, May 2018, pp. 1–6.

[36] W. Meng, W. Li, L. T. Yang, and P. Li, "Enhancing challenge-based collaborative intrusion detection networks against insider attacks using blockchain," *Int. J. Inf. Secur.*, vol. 19, no. 3, pp. 279–290, 2019.

[37] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Syst.*, vol. 43, no. 2, pp. 618–644, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923605000849

[38] I. Makhdoom, F. Tofigh, and M. Abolhasan, "A method of electronic device integrity check based on device digital genome (d2igen)," Australia Patent 2 018 204 784, Jan. 16, 2020. [Online]. Available: http://pericles.ipaustralia.gov.au/ols/auspat/applicationDetails.do?app%licationNo=2018204784

[39] B. Leclair, R. Shaler, G. R. Carmody, K. Eliason, B. C. Hendrickson, T. Judkins, M. J. Norton, C. Sears, and T. Scholl, "Bioinformatics and human identification in mass fatality incidents: The world trade center disaster," *J. Forensic Sci.*, vol. 52, no. 4, pp. 806–819, Jul. 2007.

[40] NIH. (2020). *A Brief Guide to Genomics*. Accessed: Mar. 31, 2021. [Online]. Available: https://www.genome.gov/about-genomics/fact-sheets/A-Brief-Guide-to-Geno%mics

[41] J. R. Douceur, "The Sybil attack," in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Germany: Springer, 2002, pp. 251–260.

[42] V. Buterin, "A next-generation smart contract and decentralized application platform," Ethereum, Zug, Switzerland, White Paper, 2014, vol. 3, no. 37.

[43] C. Christian, "Architecture of the hyperledger blockchain fabric," in *Proc. Workshop Distrib. Cryptocurrencies Consensus Ledgers*, Chicago, IL, USA, vol. 310, no. 4, 2016, pp. 1–4.

[44] (2020). *Introduction to Hyperledger Fabric*. Accessed: Jun. 28, 2021. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/blockchain.htm%l

[45] G. Gideon. (2015). *Multichain Private Blockchain White Paper*. Accessed: Jun. 28, 2021. [Online]. Available: https://www.multichain.com/download/MultiChain-White-Paper.pdf

[46] (2018). *Quorum White Paper*. Accessed: Jun. 28, 2021. [Online]. Available: https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepap%er%20v0.2.pdf

[47] N. A. Dawit, S. S. Mathew, and K. Hayawi, "Suitability of blockchain for collaborative intrusion detection systems," in *Proc. 12th Annu. Undergraduate Res. Conf. Appl. Comput. (URC)*, Apr. 2020, pp. 1–6.

[48] (2016). *Web3.Js Ethereum JavaScript Api*. Accessed: Jun. 29, 2021. [Online]. Available: https://web3js.readthedocs.io/en/v1.3.4/

[49] (2021). *The Most Secure Way to Protect JavaScript Code*. Accessed: Jun. 16, 2021. [Online]. Available: https://javascriptobfuscator.com

[50] S. Rauti and V. Leppanen, "A comparison of online JavaScript obfuscators," in *Proc. Int. Conf. Softw. Secur. Assurance (ICSSA)*, Jul. 2018, pp. 7–12.

[51] (2021). *JavaScript Obfuscator Tool*. Accessed: Jun. 17, 2021. [Online]. Available: https://obfuscator.io

[52] W. Paul. (2021). *Gas and Fee*. Accessed: Jun. 29, 2021. [Online]. Available: https://ethereum.org/en/developers/docs/gas/

[53] (2021). *Ethereum Gas Limit Hits 15M as ETH Price Soars*. Accessed: Jun. 14, 2021. [Online]. Available: https://www.coindesk.com/ethereum-gas-limit-eth-price-soars

[54] S. Prabath. (2017). *The Mystery Behind Block Time*. Accessed: Jun. 15, 2021. [Online]. Available: https://medium.facilelogin.com/the-mystery-behind-block-time-63351e3560%3a

[55] (2021). *Ethereum Difficulty Chart*. Accessed: Jun. 14, 2021. [Online]. Available: https://www.coinwarz.com/mining/ethereum/difficulty-chart

[56] M. Julia. (2020). *When Will Ethereum 2.0 Fully Launch*. Accessed: Jun. 15, 2021. [Online]. Available: https://cointelegraph.com/news/when-will-ethereum-2-0-fully-launch-road%map-promises-speed-but-history-says-otherwise

[57] (2021). *Proof-of-Stake (PoS)*. Accessed: Jun. 15, 2021. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[58] (2020). *What is Ethereum 2.0? Overview, Features and Price Implications*. Accessed: Jun. 15, 2021. [Online]. Available: https://www.diginex.com/insights/what-is-ethereum-2-0-overview-features%-and-price-implications/

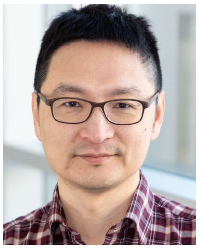
[59] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "FastFabric: Scaling hyperledger fabric to 20 000 transactions per second," *Int. J. Netw. Manage.*, vol. 30, no. 5, Sep. 2020, Art. no. e2099.

**IMRAN MAKHDOOM** (Member, IEEE) received the master's degree in information security from the National University of Sciences and Technology, Pakistan, in 2015, and the Ph.D. degree from the University of Technology Sydney, in 2020. He is currently a Postdoctoral Researcher at the University of Technology Sydney. He was a Food Agility Scholar, from 2019 to 2020, and has made a valuable contribution to data security and privacy in the Food Tech/Agri Tech. He has published numerous articles in some of the prestigious journals and conferences. His research interests include autonomous systems, science and technology parks, blockchain, the Internet of Things, distributed consensus, networks, and computer security.

**KADHIM HAYAWI** (Member, IEEE) received the M.Sc. degree in computer science from Dalhousie University, Canada, in 2004, and the Ph.D. degree from the University of Waterloo, Canada, in 2018. He is currently an Assistant Professor at the College of Technological Innovation, Zayed University. He is a member of the Cybersecurity and Digital Forensics Research Group, where he teaches a wide variety of courses and pursues his research endeavors. He earned several prestigious industry certifications and has over 20 years of experience in academia and industry. His research interests include tackling information security and privacy challenges of emerging technologies, such as the IoT, Industrial IoT, fog, cloud, and social networks, using real-time and distributed deep learning, GAN, and blockchain technologies.

**MOHAMMED KAOSAR** (Senior Member, IEEE) is currently working as a Senior Lecturer with the Discipline of IT, Media and Communications, Murdoch University, Australia. Prior to that, he has worked in several universities, including RMIT University, Victoria University, Effat University, and Charles Sturt University. He has worked in number of national and international research projects and grants. He has published number of research papers in reputable journals and conferences, including IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING (TKDE), *Data and Knowledge Engineering* (DKE), IEEE International Conference on Data Engineering (ICDE), *Computer Communications* (ComCom), and IEEE SIGCOMM. He has been supervising many postgraduate students, mentoring junior colleagues, and collaborating with many national and international researchers. He is an Active Member of various professional organizations, including Australian Computer Society (ACS), European Alliance for Innovation (EAI), The Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (ICST), South Pacific Competitive Programming Association, and Industrial Engineering and Operation Management (IEOM).

**SUJITH SAMUEL MATHEW** (Member, IEEE) received the Ph.D. degree in computer science from The University of Adelaide, SA. He has 20 years of experience working both in the IT industry and in IT academia. He has held positions as the Group Leader, a Technical Evangelist, and a Software Engineer within the IT industry. In academia, he has been teaching various IT related topics and pursuing his research interests in parallel. He is currently an Assistant Professor with the College of Technological Innovations (CTI), Zayed University, Abu Dhabi, United Arab Emirates. His research interests include ubiquitous and distributed computing, with focus on the Internet of Things and smart city applications.



**PIN-HAN HO** (Fellow, IEEE) is currently a Full Professor with the Department of Electrical and Computer Engineering, University of Waterloo. He is the author/coauthor of over 400 refereed technical papers and several book chapters. He is the coauthor of two books on Internet and optical network survivability. His current research interests include broadband wired and wireless communication networks, including wireless transmission techniques, mobile system design and optimization, and network dimensioning and resource allocation. He is a Professional Engineer Ontario (PEO).