IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 13 July 2022, accepted 11 August 2022, date of publication 18 August 2022, date of current version 31 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3199878

RESEARCH ARTICLE

SIGNORA: A Blockchain-Based Framework for Dataflow Integrity Provisioning in an Untrusted Data Pipeline

YUSTUS EKO OKTIAN[1], SHINWOOK HEO[2,3], AND HOWON KIM[2], (Member, IEEE)
[1]Blockchain Platform Research Center, Pusan National University, Busan 609735, South Korea
[2]School of Computer Science and Engineering, Pusan National University, Busan 609735, South Korea
[3]SmartM2M, Busan 46300, South Korea

Corresponding author: Howon Kim (howonkim@pusan.ac.kr)

This work was supported in part by the Institute for Information and Communications Technology Planning and Evaluation (IITP) Grant through the Korea Government [Ministry of Science and ICT (MSIT)], Convergence security core talent training business (Pusan National University), under Grant 2022-0-01201; and in part by the Human Resources Development Program of the Korea Institute of Energy Technology Evaluation and Planning (KETEP) Grant through the Korea Government Ministry of Trade, Industry and Energy, under Grant 20184030202060.

ABSTRACT In a typical data pipeline, the dataflow starts from the first node, where the data is initiated, and moves to the last node in the pipeline, where the processed data will be stored. Due to the sheer number of involved participants, it is crucial to protect the dataflow integrity in the pipeline. While previous studies have outlined solutions to this matter, the solution for an untrusted data pipeline is still left unexplored, which motivates us to propose SIGNORA. Our proposal combines the concept of a chain of signatures with blockchain receipt to provide dataflow integrity. The chain of signatures provides a non-repudiation guarantee from participants, while the hash of the data and signatures is anchored in the blockchain for a non-tampering guarantee through blockchain receipt. Aside from that, SIGNORA also satisfies essential requirements of running data pipeline processing in an open and untrusted environment, such as (i) providing reliable identity management, (ii) solving the trust and accountability issues through a reputation system, (iii) supporting various devices through multiple cryptographic algorithms (i.e., ECDSA, EdDSA, RSA, and HMAC), and (iv) off-chain processing. Our experiment results show that SIGNORA can provide dataflow integrity provisioning in multiple scenarios of data payload size with reasonable overhead. Furthermore, the cost of smart contract methods has also been analyzed, and several off-chain solutions have been addressed to reduce transaction costs. Finally, the reputation system can adapt to the history of nodes' activities by increasing their scores when they actively perform honest behavior while reducing their scores when they become inactive. Therefore, SIGNORA can provide a high degree of accountability for participants collaborating in an untrusted environment.

INDEX TERMS Dataflow integrity, blockchain, chain of signatures, blockchain receipt, data pipeline.

I. INTRODUCTION

The combination of digital signature and hashing algorithm can provide non-repudiation and non-tampering guarantee during data transmission from senders to receivers. At the receivers' ends, third-party validators can check the signatures and hash to determine if senders previously transmitted the data and whether receivers have modified the data.

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

Those solutions are easy-to-implement in two actors (i.e., a sender and receiver) scenarios. However, some complex dataflow (e.g., supply-chain manufacturing, internet traffic routing, and commodity trading) requires extension usages of those digital signatures and hash algorithms applicable to multiple senders and receivers. This is where the chain of signatures emerges as a contender for solutions.

The chain of signatures provides accountability to multiple entities in a data pipeline by allowing participants to sign the data they process before transmitting it to others. This signing operation begins sequentially from the start of the pipeline to the end, where the data and signatures will be hashed and stored for future validations. While the chain of signatures alone can support integrity for multiple users, it is still not enough to provide integrity in an open data pipeline, where several untrusted organizations are most likely to exist in such environments. Distrust between parties hinders the collaborative effort to produce a reliable chain of signatures and becomes a significant challenge.

Blockchain [1] and smart contract [2] have been widely used as a tool to preserve digital data by anchoring the hash of the data in the blockchain network [3]. This trend rises as the possibility of blockchain being used as a forensically-sound legal argument is discussed [4]. Many researchers then leveraged blockchain as trusted platforms in an untrusted environment (e.g., in a crowdsourcing platform [5] or in a decentralized marketplace [6]). Similarly, previous studies have constructed blockchain-based preliminary assessments to make the deploying chain of signatures in a data pipeline possible. However, those studies assume a more trusted setting, where each participant is always assumed to act honestly when cooperating in the data pipeline. No accountability metrics are available; thus, malicious behaviors are left unpunished. Furthermore, they also do not consider the heterogeneity of the pipeline nodes, which limits the data pipeline to operate only in a closed environment.

Our proposal targets a more sophisticated data pipeline environment where (i) several competing and untrusted organizations exist as nodes in the pipeline, and (ii) they can deploy their nodes in a BYOD (bring your own device) fashion. Furthermore, we define several requirements (R) that must be addressed to provide dataflow integrity in such a data pipeline.

R1: *The system must deploy trusted identity management.*

A reliable identity service is essential, especially in an open environment, where anyone can create new identities and can perform Sybil attack [7] easily; all participants in the pipeline must be identifiable through a trusted identity service. Therefore, the system can pinpoint the source of problems and hold the entity that causes it accountable if such cases happen.

R2: *The system needs to provide a reliable accountability metric for each participant contributing to the pipeline.*

A regulation must be established in the system to enforce participants to behave honestly such that adversaries cannot gain any benefits from performing malicious actions. When unfortunate events (crashes or byzantine failures) are detected, the system should punish destructive behaviors while rewarding good conduct from participants.

R3: *The system has to support various devices by employing variations of cryptography algorithms.*

Since participants may deploy various device types in a given pipeline, the system must provide several alternatives of cryptography algorithms to match the device specifications. For example, a prior signature signed with *A* algorithm can be extended with *B* algorithm, thus providing flexibility and further boosting usability because it can cover a wide range of possible devices.

R4: *The system must allow participants to process a chain of signatures off-chain.*

As a continuation of the previous requirement, not all devices have enough resources to become a blockchain node, and chain of signatures operations should have an option to be processed off-chain while still preserving a relatively high degree of integrity guarantee.

Contributions: Driven by the motivation to fulfill those requirements, we propose `SIGNORA`, a framework for dataflow integrity provisioning in an untrusted data pipeline using chain-of-signature (chainsig) and blockchain receipt (receipt) ,[1] which can be summarized through the following contributions.

- We build identity management in the smart contract to answer **R1**. Each organization deploys its trusted Certificate Authority (CA) in the network, which can be used to generate accounts for devices in the pipeline.
- A reputation system is deployed in the smart contract to solve **R2**, which monitors the devices' history of behavior throughout the lifetime of a pipeline. Positive behavior results in better scores than malicious ones. Furthermore, the organization's reputation will also be affected by the device's performance they own.
- As our commitment to solving **R3**, we support four signature algorithms: ECDSA, EdDSA, RSA, and HMAC, with four hashing algorithms from the SHA-2 family: SHA-224, SHA-256, SHA-384, and SHA-512.
- **R4** is satisfied by providing two types of mode to create chainsig: `FULL` and `PAIR` mode, which translates into whether to perform chainsig operations off-chain (in the former case) or on-chain (in the latter case).
- Finally, we present a prototype implementation of our proposal and evaluate it to give insights on some limitations and pointers to be taken by developers when deploying the proposal in a production case.

Roadmaps: We organize the rest of this paper as follows. Section II explores previous related works. Section III presents our system models in more detail while their security analysis and feasibility assessments are explained in Section IV and V. We then discuss several limitations of our approach in Section VI. Finally, we conclude in Section VII.

II. RELATED WORK

This section discusses studies that provide dataflow integrity using chainsig and receipt from a diverse type of data pipeline. The brief summary is depicted in Table 1.

A. CHAIN OF SIGNATURES

TrustChain [8] proposes a digital signature preservation system for X.509 certificates using blockchain. Proposers submit new certificates in the blockchain, and TrustChain,

[1]Throughout the rest of this paper, we use the term "chain of signature" and "chainsig," also "blockchain receipt" and "receipt," interchangeably.

TABLE 1. Feature comparisons of previous works whether the given works provide: chain of signature (C), blockchain receipt (B), prototype implementation (P), identity management (R1), accountability through reputation or incentive (R2), diverse set of cryptographic algorithms (R3), and off-chain computation alternative (R4). We give a checkmark (✓) if the work satisfies the requirement and cross (✗) if not. The off-chain approach only applies for chainsig; thus, the dash (-) indicates inapplicable.

Related Studies	Data Pipeline	C	B	P	R1	R2	R3	R4
TrustChain [8]	X.509 Certificate	✓	✗	✗	✓	✗	✗	✗
Abegg et al. [9]	Publish/Subscribe	✓	✗	✗	✓	✗	✗	✓
Brotsis et al. [10]	IoT Gateway	✓	✗	✗	✓	✗	✗	✗
FiF-IoT [11]	IoT Traffics	✓	✗	✓	✓	✗	✗	✓
Zhao et al. [12]	GPS location	✓	✗	✗	✓	✗	✗	✗
Forensic-Chain [13]	Chain of Custody	✓	✗	✓	✓	✗	✗	✗
Huang et al. [14]	RDF Dataset	✓	✗	✓	✓	✗	✗	✓
Li et al. [15]	DNS Management	✓	✗	✓	✓	✗	✗	✗
ProvChain [16]	Cloud Log	✗	✓	✓	✓	✗	✗	-
BlockCloud [17]	Cloud Log	✗	✓	✗	✓	✗	✗	-
Zhang et al. [18]	Cloud Log	✗	✓	✗	✓	✗	✗	-
Kwon et al. [19]	Cloud Storage	✗	✓	✓	✗	✗	✗	-
Liang et al. [20]	IoT Drones	✗	✓	✓	✓	✗	✗	-
Baldi et al. [21]	Academic Certificate	✗	✓	✓	✓	✗	✗	-
Ours	General Purpose	✓	✓	✓	✓	✓	✓	✓

through Certificate Authority (CA), will check if the certificate already exists, is expired, or is revoked. Furthermore, TrustChain also allows nodes to vote on each other's certificates. The more votes that a certificate gets, the more trustable it becomes. The chainsig is used to map interactions between certificate proposers, CAs, and voters.

Abegg *et al.* [9] leverage chains of signature to secure the publish/subscribe protocol. Publishers include their signature when publishing a new message, where it will be chained with the previous messages from the same topic. The blockchain is then used as a fail-safe mechanism; when publishers do not receive acknowledgments from subscribers, they upload their signatures to the blockchain as proof of distribution.

The chainsig can be used to preserve digital evidence data for forensic purposes. Brotsis *et al.* [10] employ a smart gateway agent (SGA) that monitors traffic and performs device profiling, anomaly detection, and evidence gathering. This chainsig evidence is then submitted to the blockchain, where the evidence is stored and preserved. Zhao *et al.* [12] use chainsig to preserve Beidou's GPS location in the smart grid. The GPS measurements from devices will be stored in the server and protected by blockchain for auditing. In FiF-IoT [11], chainsig is used to track sequences of the requests and responses between IoT users and IoT devices to preserve forensic evidence in IoT networks. In particular, all entities involved in an IoT request/response workflow must include their signatures when processing IoT data. The last node in the workflow uploads all signatures, including the hash of request and response bytes, as transactions in the blockchain.

When processing digital forensic evidence, a trusted and chronological Chain of Custody (CoC) must be created to assess the legality of methods used during evidence processing. For this purpose, Forensic-Chain [13] proposes to use blockchain as a platform to build CoC. The chainsig reveals the interaction between the evidence submitter and all investigators who process the evidence, which can later be presented in the court.

Moving aside from forensic domains, Huang *et al.* [14] propose an RDF data quality assessment model in a decentralized environment using blockchain with the chainsig used for logging quality assessment records. First, a quality assessment is broadcast to all nodes. Then, according to the order of joining the system, each node signs and returns its assessments to the system. After obtaining all node signatures, the results can be saved in the blockchain.

Finally, chainsig can also be used in Domain Name System (DNS) management as presented in [15]. The authors leverage chainsig to chain domain trust from Hyperledger to extend domain names and their corresponding data files.

B. BLOCKCHAIN RECEIPT

Blockchain receipt [3] can be used to boost the integrity of data storage in the Cloud environment. ProvChain [16] and BlockCloud [17] build a public time-stamped log of all user operations on cloud data without the presence of a trusted third party. They do so by requiring every provenance entry to be assigned with a blockchain receipt for future validation. This way, all users' actions in the Cloud have a corresponding permanent data log in the blockchain. Zhang *et al.* [18] leverage blockchain receipt to log the submissions of Cloud forensic data from Cloud Service Providers (CSP). The Provenance Auditor will process this log, which will later be relayed to the Investigator for court actions. Finally, Kwon *et al.* [19] create a receipt for every user file submitted to the Cloud to protect its integrity so that the file can be easily identified if modified by adversaries in the future.

Aside from Cloud, blockchain receipt can be used in IoT use cases. For example, Liang *et al.* [20] employ receipt to secure drone communication during data collection and transmission and to preserve the data's integrity. Specifically, the control system anchors the hashed data records collected from drones to the blockchain network and generates a blockchain receipt for each data record. This record is then forwarded to the Cloud for permanent storage. Before accepting the request, the Cloud verifies the submitted data and its receipt. Therefore, all data stored in the Cloud have the corresponding receipt, boosting storage integrity.

Finally, Baldi *et al.* [21] discuss the usage of receipt to provide authenticity assurance of academic certifications. The certificate's hash and metadata are stored in the blockchain using a receipt. From this receipt, others can validate if a given certificate is authentic or counterfeit.

C. COMPARISON WITH RELATED WORKS

Most related studies mentioned identity management (**R1**) in their proposals. Since they utilize blockchain in their system, which requires nodes to create a blockchain address as identity, this requirement can be easily overcome. We also notice that some researchers takes consideration about the cost of blockchain operations by providing an off-chain solutions (**R4**) to perform chainsig as in [9], [11], and [14].

TABLE 2. List of actors available in the given data pipeline.

Actor	Description
n	A node in a data pipeline; it can be IoT devices (e.g., sensor or actuator devices), consumer devices (e.g., laptop, PC), network peripherals (e.g., switch, router), or servers (e.g., edge, fog, cloud servers).
m	A manager; individual or organization responsible for managing the node.
v	A trusted validator that can be summoned to audit the data pipeline.

TABLE 3. List of notations used in this paper.

Notation	Description
SK_y	A private key of an entitiy y.
PK_x	A public key of an entity x.
D	A data protected by SIGNORA.
N	The total number of nodes in the pipeline.
S_n	A chain of signature element, signature value created by node n.
$\mathcal{S}$	The whole array of chain of signatures S_n.
X_{leaf}, X_{root}	A leaf and root hash of the Merkle Tree.
t, t_{now}	The time, the current timestamp.
r	Proof of malicious behavior
$\alpha_v, \alpha_m, \alpha_n$	A blockchain address of validator, manager, and node.
C_v, C_m	A public key certificate of validator and manager.
Y_{commit}	A commitment hash.
η	A random nonce.
β	A bounty score for reputation system.
γ	The name of the used cryptography algorithm.
Z_n, Z_m	The reputation score of node n and manager m.
$\mathcal{V}, \mathcal{M}, \mathcal{N}$	List of stored validators, managers, and nodes
$\mathcal{R}, \mathcal{Z}, \mathcal{P}$	List of stored root hashes, contribution scores, and malicious reports
$SIGN_{SK_y}(J)$	Generates a digital signature for data J using private key SK_y.
$VER_{PK_x}(J, K)$	Verifies whether the holder of public key PK_x signs data J and generates the digital signature K.
$H(J)$	Generates a hash of data J.
$X \parallel Y$	A concatenation of data X with data Y.
$\bigcup_{i=1}^{I} J_i$	A concatenation of data J for all index i.
MERKLEROOT($\cdot$)	Generate a root hash X_{root}.

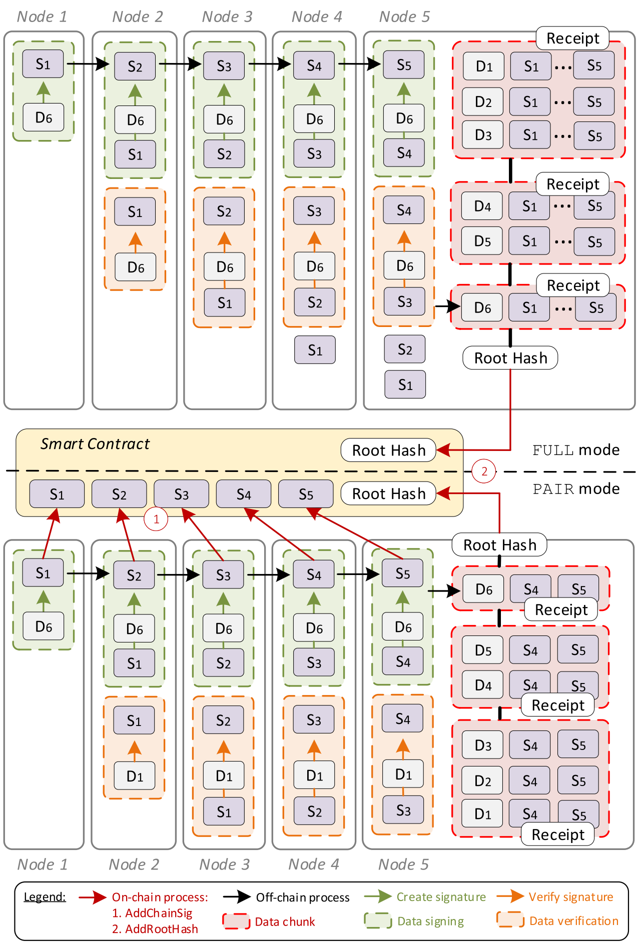

FIGURE 1. An overview of SIGNORA's integrity provisioning on a single dataflow (D_6) over five nodes ($N = 5$) using FULL (top) and PAIR mode (bottom).

This indicates that secure and trusted off-chain methods are always preferable to on-chains.

Unfortunately, other studies have not considered the diversity of the nodes that can participate in their data pipeline. When many different nodes with heterogeneous computing capability join the pipeline, we need to provide diverse sets of cryptography (**R3**) for everyone involved. Furthermore, other researchers also do not consider that nodes can behave maliciously or have different interests. Therefore, they do not provide any accountability (**R2**) for pipeline nodes either through a reputation system or incentive mechanism.

We propose SIGNORA as a general-purpose framework that can be integrated with related studies while also trying to satisfy the requirements (**R1-R4**) that were partly left unanswered by previous studies.

III. SYSTEM MODEL

Our proposed system can be divided into three parts. The first describes how the dataflow provisioning works, while its validation process is explained in the second part. The third section explores how we leverage the smart contract to boost the reliability of the provisioning and validation tasks. Table 2 lists all actors involved in a given dataflow pipeline, and Table 3 summarizes all notations used in the rest of this paper.

A. DATAFLOW INTEGRITY PROVISIONING

Nodes in a data pipeline can have one of the following roles: (i) generator, (ii) relayer, or (iii) keeper. The generator node is the one who generates data and becomes the initiator of chainsig by producing genesis chainsig as a *proof of origin*. The keeper node is located at the other end of the pipeline, storing the received data and chainsig permanently. Meanwhile, the relayer is an intermediary node that routes data from the generator to the keeper while extending chainsig along the way as a *proof of distribution*. When relaying data, nodes can choose whether to retransmit all elements of chainsig (i.e., so-called FULL mode) or only the current and previous element (i.e., PAIR mode) to the next node. Once the data arrives at the keeper nodes, they create blockchain receipts and anchor the root hash in the blockchain as a *proof of storage*. Figure 1 depicts the overview of our dataflow integrity provisioning strategy.

1) GENERATING AND EXTENDING CHAINSIG

Let us assume that $\mathcal{N} = \{1, 2, 3, \ldots, n, \ldots N\}$ is a list of nodes n in a data pipeline. The generator node is indexed with $n = 1$ while the keeper node is $n = N$. Meanwhile, the rest are relayer nodes. Note that N also indicates the total number of nodes in the pipeline. The whole chainsig process can be explained in the following paragraphs.

Step 1 Creating Genesis Chainsig: The generator node produces important data D, which its integrity must be protected by `SIGNORA`. The node signs the data and produces a genesis chainsig S_1 as follows.

$$S_1 = SIGN_{SK_n}(D), \text{ where } n = 1$$

The node then sends D and S_1 to the relayer nodes.

Step 2 Extending Chainsig: The relayer nodes then first get the latest chainsig element (i.e., $(n-1)^{th}$ signature, assuming that this node is the n^{th} node). After that, nodes verify whether the previous nodes signed the signature correctly. If this node is the first relayer node, then the node verifies whether

$$VER_{PK_{n-1}}(D, S_{n-1}) \text{ equals } \texttt{True} \text{ for } n = 2$$

Otherwise, the node makes sure that

$$VER_{PK_{n-1}}(D \parallel S_{n-2}, S_{n-1}) \text{ equals } \texttt{True} \text{ for } n > 2$$

Upon successful verification, the node extends the chainsig by signing the previous signature and the relayed data to produce a new signature S_n, as shown in Equation 1.

$$S_n = SIGN_{SK_n}(D \parallel S_{n-1}) \tag{1}$$

Note that if verification is invalid, the chainsig will not be extended, and the process will stop.

Depending on how the received data and chainsig is retransmitted from one node to another, relayer nodes can use `FULL` or `PAIR` mode. In `FULL` mode, the relayer nodes redistribute all of the received data and elements of chainsig, which is $D, S_1, \ldots, S_n$ to the next nodes. Meanwhile, they only relay D, S_{n-1}, and S_n to the next nodes if using `PAIR` mode.

Each element of chainsig S_n contains additional metadata such as (i) the cryptographic algorithm γ_n used to generate the signature, (ii) the blockchain address α_n of the signature creator, and (iii) the timestamp t_n indicating when the signature is generated. This metadata is submitted through AddChainSig($\cdot$) method in the smart contract when using `PAIR` mode.

Note that each node still needs to send S_n and its metadata to the next node off-chain because the subsequent node must verify the relayed data and chainsig quickly.[2] Therefore, both `PAIR` and `FULL` nodes receive S_n and its metadata. However, they are not uploaded to the smart contract when using `FULL` mode.

[2]The data stored in the blockchain may not be processed fast enough. Depending on the consensus algorithm, the blockchain may produce forks; therefore, transactions cannot be confirmed immediately.

These steps are repeated in all relayer nodes until the data and chainsig reach keeper nodes.

Step 3 Storing Chainsig: The keeper nodes store all of the received data and chainsig to its database permanently. Due to the difference in the delivery process, when using `FULL` mode, keeper nodes stores the whole chainsig (i.e., $\mathcal{S} = \{S_1, S_2, S_3, \ldots, S_n, \ldots, S_N\}$) in their local storage off-chain. Meanwhile, `PAIR` mode utilizes `SIGNORA` smart contract to store $\mathcal{S}$ on-chain.

2) PRODUCING BLOCKCHAIN RECEIPTS

Let us assume that the manager m for keeper nodes $n = N$ provide some values to the following parameters: t_{leaf} and t_{root}. The t_{leaf} indicates the interval when we should create leaf hashes, which are parts of the Merkle Tree for the receipt. Meanwhile, t_{root} tells us the interval when we should form the root hash of the Merkle Tree based on the previously calculated leaf hashes. The whole receipt process can be described as follows.

Step 1 Leaf Hash Generation: When it is time to create leaf hashes, keeper nodes queries all of the previous data D and chainsig $\mathcal{S}$ from $t_{now} - t_{leaf}$ to t_{now} and hash them all together to create a leaf hash X_{leaf}. Formally,

$$X_{leaf} = H\left(\bigcup_{t=t_{now}-t_{leaf}}^{t_{now}} D_t || \mathcal{S}_t\right)$$

This procedure occurs at every interval of t_{leaf}. If the keeper node does not receive any data during that interval, the node can create a dummy payload (e.g., 1 byte of zero data) to fill in leaf hashes. t_{now} is the current timestamp.

Step 2 Root Hash Generation: Given enough time, keeper nodes will produce several leaf hashes. After that, they can create a root hash when t_{root} is triggered. Keeper nodes gather all previous leaf hashes and form root hash X_{root} following algorithm specified in Chainpoint [3] as formally denoted in Equation 2.

$$X_{root} = \text{MerkleRoot}\left(\bigcup_{t=t_{now}-t_{root}}^{t_{now}} X_{leaf,t}\right) \tag{2}$$

This procedure repeats at each interval of t_{root}.

Step 3 Blockchain Anchoring: After the keeper nodes obtain the root hash, they upload it to the smart contract. The `SIGNORA` smart contract has the AddRootHash($\cdot$) method, which allows keeper nodes to anchor their root hash in the blockchain network. Once the root hash is stored, keeper nodes create receipts for each leaf hashes in JSON format, similar to ones in [3]. This receipt can be used later to verify the integrity of the data chunk stored in the keeper nodes.

B. DATAFLOW INTEGRITY VALIDATION

Trusted validators are tasked to verify dataflow integrity on the keeper nodes' storage. In particular, they validate whether

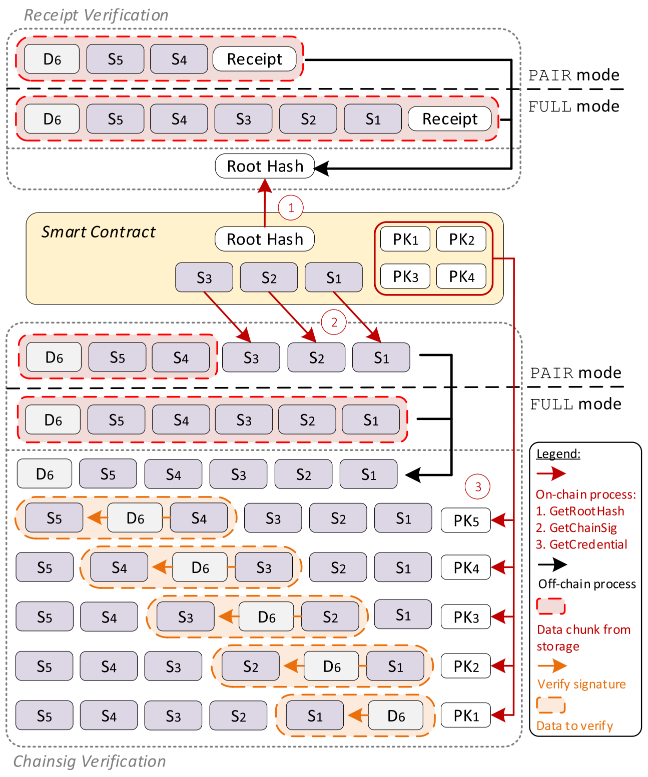

FIGURE 2. **An overview of SIGNORA's integrity validation on a single dataflow (D_6) over five nodes ($N = 5$) using FULL and PAIR mode. The validation in the keeper node includes receipt verification (top) and chainsig validation (bottom).**

the data stored in keepers' storage are not maliciously altered by checking the corresponding blockchain receipts. After that, validators continue verifying each element of chain signatures, starting from the latest to the genesis one. Finally, validators can conclude that the stored data has complete dataflow integrity only if both the receipt and chainsig verification are successful. Figure 2 shows the overview of our dataflow integrity validation strategy.

1) VALIDATING DATA CHUNK OF BLOCKCHAIN RECEIPTS

The first validation step is to ensure that the data stored in the keeper's database has not been maliciously modified. Validators perform these steps for formal verification.

Step 1 Data Splitting: Validators obtain t_{leaf} and t_{root} parameters from the keeper node. They then split the whole database into multiple data chunks according to that information. Each data chunk will be represented by one blockchain receipt.

Step 2 Receipt Verification: Depending on the task assigned, validators can verify one, several, or all data chunks from the database. They can follow detailed steps in the receipt to verify a particular data chunk. Specifically, the receipt instructs how to hash the data chunk and obtain the Merkle Root hash (c.f., [3] on detail verification steps).

Step 3 Anchor Verification: After obtaining the Root hash, validators must check if this Root hash exists in the smart contract or not by invoking GetRootHash(·) method. If validators can find the hash, the data chunk has a strong integrity guarantee because it would be tough to tamper with the smart contract and blockchain security.

2) VERIFYING ELEMENT OF CHAINSIG

The following validation step is to verify the corresponding chainsig for each data chunk.

Step 1 Chainsig Construction: First, the validator must get all of the chainsig-to-be-verified. In FULL mode, all elements of chainsig can be found from the keeper's database directly. Meanwhile, only the latest two elements are found from the keeper's database if using PAIR mode. Therefore, validators need to do extra steps by calling GetChainSig(·) method to get all the remaining chainsig from the smart contract before constructing all elements of chainsig.

Step 2 Credentials Gathering: From the constructed chainsig, we can tell how many nodes are involved in generating and distributing this data. Validators must collect all nodes' credentials for verification. Aside from the signature value, each element of chainsig also has a blockchain address α_n, which indicates who generated this signature element. From this address, validators can query the required keys to verify the data in the smart contract. Specifically, they call the GetCredential(·) method to obtain all $PK_n; \forall n \in \mathcal{N}$ where $S_n \in \mathcal{S}$.

Step 3 Chainsig Validation: After the chainsig is fully constructed and all keys are ready, validators can begin the validation. Assuming that the node under verification is indexed as n, starting from the latest element of chainsig, validators verify if

$$VER_{PK_n}(D \parallel S_{n-1}, S_n) \text{ equals } \mathtt{True} \text{ for } n > 1$$

This action confirms whether this node n signs the data payload from the previous node or not. In this case, the previous payload to be verified is $D \parallel S_{n-1}$. Validators repeat this verification operation for all elements of chainsig. The last element to be verified will be the genesis chainsig, which the validators check differently by making sure if

$$VER_{PK_n}(D, S_n) \text{ equals } \mathtt{True} \text{ where } n = 1$$

Note that, different from the rest of the signatures, genesis chainsig only signs the original data. Therefore, the payload to be verified is only D. If all chainsig elements are successfully verified, the validators can conclude that the data corresponding to this chainsig is valid.

C. SIGNORA SMART CONTRACT

SIGNORA smart contract provides valuable on-chain services for (i) identity management, (ii) HMAC key distribution, and (iii) a reputation system that further augments the security of the proposed chain of signatures and blockchain receipt. Algorithm 1 depicts the list of available methods in the smart contract.

Algorithm 1 SIGNORA Smart Contract Methods

```
1: {List of writable methods}
2: procedure AddChainSig($S_n, t_n, \gamma_n, \alpha_n$)
3:     saves $S_n \parallel t_n \parallel \gamma_n \parallel \alpha_n$ to $\mathcal{S}$
4: procedure AddRootHash($X_{root}, \alpha_n$)
5:     if $X_{root} \in \mathcal{R}$ then abort end if
6:     saves $X_{root} \parallel \alpha_n$ to $\mathcal{R}$
7: procedure AddValidator($\alpha_v, C_v$)
8:     saves $\alpha_v \parallel C_v$ to $\mathcal{V}$
9: procedure AddManager($\alpha_m, C_m$)
10:    saves $\alpha_m \parallel C_m$ to $\mathcal{M}$
11: procedure AddNode($\alpha_n, \alpha_m$)
12:    saves $\alpha_n \parallel \alpha_m$ to $\mathcal{N}$
13: procedure RegisterKey($\alpha_v/\alpha_m/\alpha_n, PK/Y_{commit}, \gamma$ )
14:    if $\gamma$ is HMAC then $k = Y_{commit}$ else $k = PK$
15:    if $\alpha_v$ then saves $\alpha_v \parallel k \parallel \gamma$ to $\mathcal{V}$
16:    if $\alpha_m$ then saves $\alpha_m \parallel k \parallel \gamma$ to $\mathcal{M}$
17:    if $\alpha_n$ then saves $\alpha_n \parallel k \parallel \gamma$ to $\mathcal{N}$
18: procedure RevealHMAC($Y_{commit}, \alpha_n$)
19:    remove $Y_{commit}$ for $\alpha_n$ in $\mathcal{N}$
20: procedure SubmitScore($z_n, \alpha_n$)
21:    $t_{z_n} = t_{now}$
22:    saves $z_n \parallel \alpha_n \parallel t_{z_n}$ to $\mathcal{Z}$
23: procedure SubmitReport($r_n, \alpha_n$)
24:    saves $r_n \parallel \alpha_n \parallel t_{now}$ to $\mathcal{P}$
25: {List of read-only methods}
26: function GetChainSig($S_n$)
27:    return $S_n \parallel t_n \parallel \gamma_n \parallel \alpha_n$ from $\mathcal{S}$
28: function GetRootHash($X_{root}, \alpha_n$)
29:    if $X_{root} \in \mathcal{R}$ for $\alpha_n$ return True else False
30: function GetCredential($\alpha_v/\alpha_m/\alpha_n$)
31:    if $\alpha_v$ return $\alpha_v \parallel PK/Y_{commit} \parallel \gamma$ from $\mathcal{V}$
32:    if $\alpha_m$ return $\alpha_m \parallel PK/Y_{commit} \parallel \gamma$ from $\mathcal{M}$
33:    if $\alpha_n$ return $\alpha_n \parallel PK/Y_{commit} \parallel \gamma$ from $\mathcal{N}$
34: function GetReputation($\alpha_m/\alpha_n$)
35:    if $\alpha_m$ return $Z_m$ from Equation 10
36:    if $\alpha_n$ return $Z_n$ from Equation 9
```

1) IDENTITY MANAGEMENT

All participants must have verifiable identities to provide accountability during data creation, distribution, or storage. Managers m and validators v register at least one blockchain address α to the SIGNORA smart contract by calling the AddManager($\cdot$) and AddValidator($\cdot$) method. Meanwhile, m can use the AddNode($\cdot$) method to register their pipeline nodes n address. When calling this method to store n's identity, the smart contract will create a map that ties m and n identity. Therefore, anyone can be directly informed to which m this n belongs.

Aside from the blockchain address, several parameters must also be included during the registration. First, managers and nodes must disclose the keys used to verify their signatures by uploading them to the smart contract through RegisterKey($\cdot$) method. Those keys can vary depending on the signature algorithm (e.g., ECDSA, EdDSA, RSA, or HMAC). Secondly, because anyone can create new identities and upload them to the smart contract, it becomes effortless to spoof an identity or perform Sybil attack [7] on the identity. To mitigate these issues, managers must have a trusted and reputable CA to represent their organization, similar to how identities are organized in Hyperledger Fabric [22]. This CA can approve any identity created for their organization by signing the corresponding address as a certificate, which is illustrated in Equation 3.

$$C_m = SIGN_{SK_{CA_m}}(\alpha_m) \tag{3}$$

Managers must include this certificate C_m to the smart contract when registering their identities.

Note that we assume the CA information has been recorded in the smart contract prior to the manager registration; hence, anyone can get the CA's public key and verify the published certificates from the smart contract. Furthermore, m will act as CAs for all n that they register. Since the smart contract has mapped the relationship between m and n, certificates are unnecessary for n.

2) HMAC KEY DISTRIBUTION

To facilitate various use cases, especially those in constrained resources such as IoT devices, SIGNORA allows nodes in a data pipeline to sign using a symmetric algorithm such as HMAC, despite its weaker non-repudiation guarantee compared to asymmetric ones.

Ideally, the HMAC secret key must be shared securely between the nodes and the validator at the initiation stage. However, because we never know who will be the future validator for our data, it is not easy to do the sharing early. Suppose if the key is uploaded to the smart contract during registration, anyone can then spoof the identity by generating unintended signatures because the information in the smart contract is open to the public. In contrast, if the key is uploaded later (e.g., during the validation process after the validator has been appointed), the nodes can repudiate the already-generated signature by claiming that they did not make the signature. From the validators' point of view, there is no way to tell if the nodes are telling the truth because the key is not shared at the early stage.

SIGNORA solves this problem by mandating HMAC nodes to reveal their secret keys in the form of commitment hashes.

Step 1 Commitment Hash Submission: Managers m generate a new secret key for nodes SK_n and deliver the key securely to the nodes. The nodes save this key in their local private database. After that, m create a commitment hash Y_{commit} by hashing the concatenation of the secret key with a random nonce η, as shown in Equation 4.

$$Y_{commit} = H(SK_n \parallel \eta) \tag{4}$$

m then upload Y_{commit} and α_n to the smart contract by calling the RegisterKey($\cdot$) method. Note that only m, as the CAs/operators for n, can call this method. The smart contract then ties this hash to n's identity.

Step 2 Perform Chainsig Using HMAC: In daily operation, n sign the data using SK_n that is previously assigned.

Step 3 Commitment Hash Verification: During validation, the validators v may verify an element of chainsig S_n that is previously signed by n using HMAC. First, v must obtain information regarding m in the smart contract (recall that the mapping between m and n is stored in the smart contract). Then, v ask m to share the node n's secret key SK'_n along with its nonce η'. The validator then create a new commitment hash Y'_{commit} and checks if this hash matches Y_{commit} in the smart contract using RevealHMAC(·) method. This step is formally denoted in Equation 5.

$$\begin{aligned} & Y'_{commit} = H(SK'_n \parallel \eta') \\ & \text{with } Y'_{commit}, \begin{cases} \text{valid, if } Y'_{commit} = Y_{commit} \\ \quad \text{invalid, otherwise} \end{cases} \end{aligned} \tag{5}$$

An equal comparison indicates that the secret key that was previously used to sign the data in the earlier stage is the same as the one in the current validation process; therefore, we can trust this HMAC signature.

Additionally, the secret key can be accompanied by an expiry time t_{exp} to improve the security of HMAC further. When enabled, m must include this expiry time when calling RegisterKey(·) method. During validation, aside from the commitment hash comparison, v must also validate if the nodes signed the chainsig with an expired key or not. They can do so by contrasting the timestamp property in the chainsig t_n with t_{exp} (i.e., signature is invalid if $t_n > t_{exp}$).

Finally, if a secret key expires or if a secret key is revealed, m must create a new secret key, re-deliver it to the nodes, and re-upload a new commitment hash to the smart contract.

3) REPUTATION SYSTEM

SIGNORA takes the history of nodes' activity to build their corresponding reputation scores. When nodes perform honest dataflow integrity provisioning more frequently, they receive more scores. Furthermore, SIGNORA also considers recent activities to yield a more significant value than older activities. This policy is to encourage active behavior from nodes.

a: REWARDING DATA GENERATION

The generator nodes receive some scores for each dataflow originating from them, denoted in Equation 6.

$$z_n = \beta, \text{ for all } n \in \mathcal{N}, \text{ where } n = 1 \tag{6}$$

A constant value of bounty score β is previously configured as part of the initiation of the smart contract.

b: REWARDING DELIVERY TIME

The timestamp property in each chainsig element is used to calculate delivery scores for relayer nodes. We define $t_1, t_2, t_3, \ldots, t_n, \ldots, t_N$ as the timestamp of a particular node n from its corresponding chainsig. The delivery scores z_n for each node will be measured as in Equation 7.

$$\begin{aligned} & z_n = \beta - \left(\beta \times \frac{|t_n - t_{n-1}|}{t_{exp}}\right) \\ & \text{for all } n \in \mathcal{N}, \text{ where } n \neq 1 \text{ and } n \neq N \end{aligned} \tag{7}$$

This equation tells us that the delivery scores depend on how fast the node processes the received data and retransmits it to the subsequent nodes. A smaller delay between the current and previous nodes indicates better processing quality.

c: REWARDING STORAGE SPACE

We create a separate storage score metric to assess keeper nodes' performance. We define two parameters: L and E. L corresponds to the total number of leaf hashes in one root hash, and E is the total number of chainsig elements in one leaf hash. Then, the storages scores z_n can be calculated as in Equation 8.

$$z_n = \beta \times \sum_{l=1}^{L} E_l, \text{ for } n \in \mathcal{N}, \text{ where } n = N \tag{8}$$

This equation informs us that the storage scores depend on how many storage spaces are allocated to store chainsig and receipts. More spaces mean the nodes are on a higher burden; thus, they must be rewarded accordingly by giving them better scores.

d: SUBMITTING CONTRIBUTION SCORES

The generation, delivery, and storage scores are measured when the validator conducts the dataflow integrity validation in Section III-B. After obtaining the scores, v must upload them to the smart contract by calling SubmitScore(·) method. The smart contract saves the scores z_n and records the timestamp indicating when the scores are saved t_{z_n} in the blockchain.

Note that validators must submit contribution scores for all nodes involved. However, because the scores for one node may come from multiple sources (e.g., multiple chainsig and receipts), instead of calling SubmitScore(·) method many times, validators can aggregate scores for one node off-chain. Once fully aggregated, they can settle the final score to the smart contract.

e: REPUTATION FOR NODES

At any given time, other participants can determine the total reputation score Z_n that a particular node has by calling GetReputation(·) method, defined as:

$$Z_n = \sum_{z_n \in \mathcal{Z}} z_n \times e^{-\left(\frac{t_{now} - t_{z_n}}{t_{exp}}\right)} \tag{9}$$

Equation 9 sums all of the node's generation, delivery, or storage scores, where $\mathcal{Z}$ contains all scores for node n in the smart contract. Each contribution score will be weighted relative to the current timestamp. Older scores will be weighted less than new ones. Hence, amplifying the impact of inactive nodes as they will produce lower scores.

f: REPUTATION FOR MANAGERS

Organizations have their own reputation scores. If the organizations have many nodes that perform poorly in SIGNORA, it reduces the overall organization's reputation score because the system sums up all the nodes' reputation scores that the organizations manage. We define $\mathcal{Q}_m = \{1, 2, 3, \ldots, q, \ldots Q\}$ as a list of nodes q under management of manager m. Q represents the total number of nodes that m have. The organization reputation scores Z_m can be denoted as in Equation 10.

$$Z_m = \sum_{q \in \mathcal{Q}_m} Z_q, \text{ where } q \in \mathcal{N} \tag{10}$$

Anyone can call the calculation of organization reputation through the same GetReputation(·) method.

IV. SECURITY ANALYSIS

A. SECURITY ASSUMPTIONS

We assume that our on-chain and off-chain implementations have no bugs and work as intended. We also assume that the adopted cryptographic algorithm design and implementation are secure. The smart contract is deployed initially by a trusted law enforcer. The source code for the contract is also publicly available and can be validated before usage. Furthermore, we also assume that the underlying blockchain network remains secure (no 51% or eclipse attack). All participants can trust that the smart contract execution is always deterministic.

B. THREAT MODELS

Each manager only trusts its own nodes and, by default, does not trust the operations taken by nodes owned by other managers. Based on this trust environment, the following threat models may exist in the system.

Managers can create and register as many nodes as they want: Managers may perform Sybil attacks with intentions of (i) flooding the system with bogus accounts and (ii) gaining more reputation scores because the managers' reputation score is the sum of their nodes' score (c.f., Equation 10). Therefore, the more clients the manager has, the better the score may become.

Managers can create fake nodes to impersonate other managers: Managers can create new accounts and claim that other managers own them. Then, they use these fake accounts to perform malicious behavior in the pipeline to blame their competitors.

Managers can lie about their involvement in the pipeline: In case of an accident in the pipeline, managers may claim that their generator, relayer, or keeper nodes do not process suspected data (the one that causes harm) even though they did so.

Malicious nodes may withhold a given data and delay the retransmission to the subsequent node: Upon receiving data from previous nodes, relayer nodes may intentionally postpone the data processing. In this case, they create a long delay before transmitting the processed data to the subsequent nodes. This way, malicious nodes can indirectly prolong the overall pipeline processing.

Malicious nodes may fabricate false information about their works: As a continuation of the previous threat, malicious nodes may provide fake timestamp information attached to their processed data. This way, when delaying the retransmission of data, adversaries may break through other managers' suspicion by claiming that they process the data in a very short duration and gain better delivery scores (c.f., Equation 7).

C. SECURITY GUARANTEES

Based on the previously mentioned security assumptions and threat models, we analyze that `SIGNORA` can provide the following security guarantees.

Theorem: Nodes cannot repudiate their contributions to a dataflow pipeline.

Proof: Each node must sign the delivered data along with the signatures from the previous nodes (c.f., Equation 1), creating a strong verifiable chain-of-execution that provides non-repudiation properties to all involved nodes. This guarantee remains true even if nodes use the arguably weaker non-repudiation property of the HMAC signature. Because the manager uploads the HMAC commitment hash to the blockchain in the early stage (c.f., Equation 4), the associated nodes cannot refute their signature during the validation stage (c.f., Equation 5).

Theorem: Keeper nodes cannot modify the stored data without being detected.

Proof: The keeper nodes anchor their stored data through Root Hash in the blockchain when creating blockchain receipts (c.f., Equation 2), which can be used later to verify if the data is modified or not.

Theorem: Managers cannot impersonate other managers from a different organization as long as the corresponding CAs are secure.

Proof: During the identity registration, managers must include the certificate of their corresponding CAs (c.f., Equation 3). Lets assume that C_m is the certificate created by CA_m. Then, other party can validate if a given manager's address α_m belongs to organization m by make sure that $VER_{PK_{CA_m}}(\alpha_m, C_m)$ returns `True`.

Theorem: Managers cannot steal other party's node identities as long as the organization account is secure.

Proof: Only m can create nodes n by calling AddNode(·) method. The mapping between them is stored in the blockchain. Thus, participants cannot steal nodes' identities without compromising the account used to register the nodes.

Theorem: The reputation system discourages organizations from creating fake nodes and performing Sybil attacks.

Proof: New nodes will not have any delivery or storage score; therefore, the resulting reputation scores will always be zero. Lets assume that a is a new node in the system, $z_a = 0$ and $t_{z_a} = 0$. The reputation scores for this node in Equation 9 will become $Z_a = \sum_{z_a \in \mathcal{Z}} 0 \times e^{-\left(\frac{t_{now}-0}{t_{exp}}\right)} = 0$.

When $\beta \neq 0$, any existing nodes n that have performed a task in the system will have some value of Z_n. The generator node obtains scores for each data it generates based on Equation 6. The divisors in Equation 7 is an absolute value;

therefore, relayer nodes always get some score values for each distribution. Finally, E in Equation 8 is a count value; thus, $E \neq 0$ when the keeper nodes have performed some actions. Based on this reasoning, the reputation of existing nodes will always be bigger than 0. Thus, new nodes do not have any benefit over existing nodes.

Assume that a manager m owns several of new nodes $\mathcal{A}_m = \{1, 2, 3, \ldots, a, \ldots A\}$ on top of existing nodes $\mathcal{Q}_m = \{1, 2, 3, \ldots, q, \ldots Q\}$. A and Q are the total number of new and existing nodes. The manager reputation from Equation 10 can be calculated as follows.

$$\begin{aligned} Z_m &= \sum_{q \in \mathcal{Q}_m} Z_q + \sum_{a \in \mathcal{A}_m} Z_a, \text{ where } q, a \in \mathcal{N} \\ &= \sum_{q \in \mathcal{Q}_m} Z_q + 0 \end{aligned}$$

The scores from new nodes do not impact the overall organization scores. Therefore, managers cannot gain any score advantage simply by creating a lot of new nodes.

Theorem: Nodes are discouraged from withholding data processing in the pipeline.

Proof: Let us assume that node n withholds the data processing to delay the overall pipeline workflow. Because of how the delivery score is calculated in Equation 7, n's action are not beneficial for n's sake because it will create a bigger divisor when compared to the timestamp processed by previous node $(n-1)$, resulting in a lower score.

Theorem: The reputation system remains secure as long as we can trust the timestamp.

Proof: In PAIR mode, each node records the chainsig metadata in the smart contract using AddChainsig($\cdot$) method. The smart contract code will create a timestamp t_n in the blockchain depending on the time this method is executed. Since we trust the smart contract, we can also trust the created timestamp. Furthermore, the timestamp t_{z_n} from SubmitScore($\cdot$) method is also performed similarly. Thus, it has the same trust guarantee as t_n.

Because the chainsig metadata in FULL mode is processed off-chain, each node can generate fake timestamps when relaying chainsig to one another. Assuming that the receiving nodes n are honest, if they found that $t_{n-1} < t_{n-2}$, which means that the sender $(n-1)$ creates invalid out-of-order timestamp, n can report this malicious behaviour to the smart contract using SubmitReport($\cdot$) method. Otherwise, if n found that $t_{n-1} > t_n$, which means that the sender generates ahead-of-time timestamp, n can also report this malicious actions to the smart contract. The contract will mark the timestamp when n execute SubmitReport($\cdot$) method in the blockchain. Thus, the validator can later verify if n's claim against the sender is true or not.

Unfortunately, the system cannot detect a fake timestamp that is not out-of-order or ahead-of-time in FULL mode. Such timestamp integrity is difficult to validate in an off-chain scenario. A possible solution is to get the timestamp from Trusted Execution Environment (TEE). However, this will limit the requirement of joinable nodes to only TEE-enabled devices and reduce system openness.

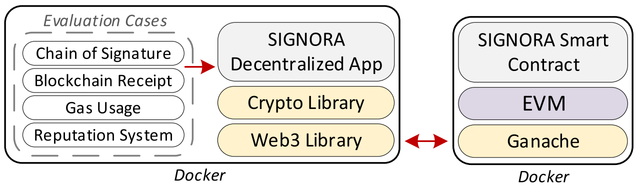

FIGURE 3. The testbed environment that is used in this paper.

V. EXPERIMENTAL RESULTS

This section analyzes and discusses our proposal's feasibility and possible overhead based on our local testbed environment shown in Figure 3.

A. OFF-CHAIN EVALUATION

We first measure the overhead of our chain of signature and blockchain receipt implementation in an off-chain scenario, where we analyze parts of the proposals unrelated to the blockchain.

1) SETUP

The experiment is performed in hardware with the following specification: Intel Core i7-10700K CPU @ 3.80 GHz and Samsung DIMM @ 2667MHz RAM. We build docker containers utilizing 1 core of CPU and 2 GB of RAM to run the benchmark. SIGNORA's dataflow integrity provisioning and validation is implemented in the Go language.

2) PRODUCING CHAINSIG BENCHMARK

a: SCENARIO

We perform multiple creating-chainsig operations for up to 100 chains. The first node signs a given data payload and generates a genesis signature. After that, the node passes the data and signature to the second node. The node first verifies the received signature when receiving signatures from the previous node. Then, the node signs the received data and signatures to generate a new subsequent signature. After that, this node relays the data and signature to the subsequent nodes following a configuration whether the node applies FULL or PAIR mode. These steps repeat until they reach the last node, the 100th node.

It is essential to point out that we perform this scenario 100 times at each chain, using various signature algorithms such as ECDSA, EdDSA, RSA, and HMAC. Meanwhile, the SHA256 algorithm is chosen as a complementary hashing method to generate the signature. We also vary the data payload size using a configuration of 1 KB, 10 KB, 100 KB, and 1 MB data payload to simulate many application use cases (e.g., IoT, Web, Multimedia files). Furthermore, all nodes are simulated on the same docker container; therefore, we ignore the network latency between nodes and focus more on the internal processing at each node. Finally, we record the execution time for each scenario and present the results in Figure 4.

b: RESULTS

First of all, huge drops appear at the early chains (from chain #1 to chain #2) throughout the whole scenario because it is easier (having less overhead) to generate the genesis chainsig than extending a chainsig. The node only needs to perform signing operations when generating, while the node must perform verification and signing operations when extending.

Second, during transmission of chainsig to subsequent nodes, the sender must encode or marshall the chainsig element to JSON format. The receiver decodes or unmarshals the JSON into the Go object and obtains the transmitted chainsig element. After that, the node can verify the last element and extend it by producing a new signature. We notice that this JSON marshaling cause some overheads during our benchmark. The bigger the size of input data to encode or decode, the slower the system processes it. Thus, the total overhead in Figure 4 is for both marshaling and generating/verifying a signature.

Third, the performance of `FULL` worsens as the number of chains grows when processing 1 KB, 10 KB, and 100 KB of data payload (c.f., Figure 4(a), 4(b), 4(c)).

In `FULL` mode, each node must relay data and all previous chainsig it receives (from $(N-N+1)^{th}$ to the N^{th} signature) to the next node. As we can see from Figure 5(a), the chainsig size grows linearly as the number of chains increases. The amount of data to encode, decode, sign, and verify then becomes bigger at later chains, thus, decreasing the throughput. For 1 KB and 10 KB data payload scenarios, the size of chainsig is greater than the data payload itself. This causes a big difference in the amount of data to process between the beginning of the chain and at the later chain, pushing those scenarios to suffer more performance degradation than the 100 KB scenario.

On the other hand, `PAIR` mode allows each node to relay only the data, the $(N-1)^{th}$ signature, and the N^{th} signature to the next node instead of relaying the whole chain of signatures. Therefore, the number of data payloads that each node must encode, decode, sign and verify remains relatively the same throughout the whole 99 chains (c.f., Figure 5(b)). This behaviour results in more consistent overall performance throughout the whole chains, as shown in Figure 4(e), 4(f), 4(g), and 4(h).

Fourth, the throughput of `FULL` and `PAIR` are relatively similar on 1 MB data payload (c.f., Figure 4(d) and 4(h)). At higher data payload, the difference of the total number of data to process between `FULL` and `PAIR` is slightly small because the data payload is way bigger than the chainsig size.

Finally, the HMAC performance is better at 1 KB and 10 KB data payload sizes. However, HMAC losses to ECDSA and EdDSA at 100 KB and 1 MB scenario. This may suggest that HMAC is better used on smaller data payload sizes. Moreover, the performance of RSA is outperformed by ECDSA and EdDSA throughout the whole scenario while also producing larger chainsig sizes compared to ECDSA and EdDSA.

3) VALIDATING CHAINSIG BENCHMARK

a: SCENARIO

We first generate up to 100 chains using the same data payload size, signature, and hashing algorithms as the previous benchmark. Then, we measure the execution time of verifying chainsig from the latest N^{th} signature down to the genesis $(N-N+1)^{th}$ signature. This scenario is used to simulate the overhead of a third-party validator when verifying a chainsig. Finally, we run our experiment 100 times and record the execution time to complete each scenario at the end of each run. The results are summarized in Figure 6.

b: RESULTS

The overall trends show that the throughput of verifications significantly decreased as chains increased. This is because more chains mean that the system must verify more elements of chainsig. For example, during chain #100, the system must verify 100 signatures, while in chain #50, the system validates 50 signatures. Furthermore, the data payload size also impacts the verification speed, with a more extensive data payload yielding slower throughput than a smaller data payload.

Similar to the results in the previous benchmark, HMAC seems to generate better results on the smaller data payload (1 KB and 10 KB), while ECDSA and EdDSA outperform it in larger data payload sizes (100 KB and 1 MB). Interestingly, RSA verification performance is better compared to ECDSA and EdDSA. This result is in line with the findings in the community that suggest RSA verifications are faster than ECC-variant [23].

4) BLOCKCHAIN RECEIPT BENCHMARK

a: SCENARIO

We analyze the blockchain receipt implementation for diverse data payload sizes: 1 KB, 10 KB, 100 KB, and 1 MB using multiple hashing algorithms from the SHA-2 family such as SHA224, SHA256, SHA384, and SHA512. We assume the system will generate a root hash whenever the total data accumulated in the storage reaches 100 MB. Furthermore, the system generates one corresponding leaf hash for each data payload. As a result, for a 1 KB data payload, the system creates 100,000 leaf hashes, while it produces 100 leaf hashes for a 1 MB data payload.

We run simulations of generating and verifying receipts 100 times and record the execution time to complete each scenario. We also measure the size of blockchain receipts produced in each scenario. Figure 7 summarizes our experiment results.

b: RESULTS

The throughput of generating and verifying receipts increases as the data payload size rises (c.f., the result of 1 KB and 1 MB

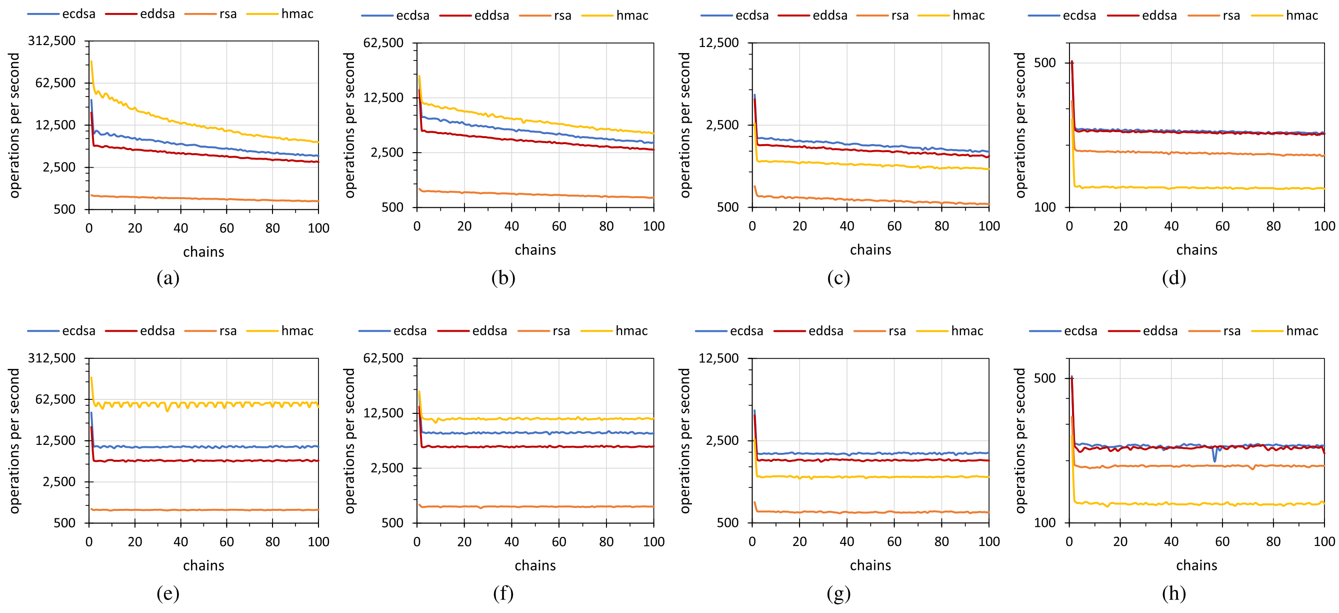

FIGURE 4. The performance evaluation of creating chainsig using `FULL` (a-d) and `PAIR` (e-h) mode on multiple data byte: 1 KB (a, e), 10 KB (b, f), 100 KB (c, g), and 1 MB (d, h). SHA256 algorithm was used to create up to 100 chains of signatures using multiple signature algorithms: ECDSA, EdDSA, RSA, and HMAC.

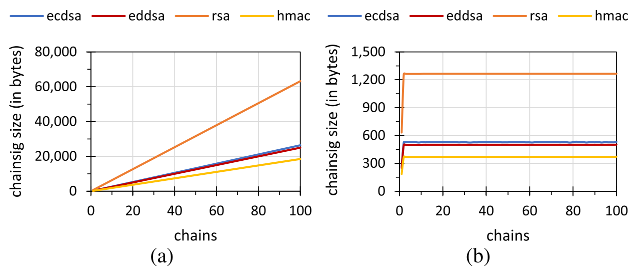

FIGURE 5. The size of chainsig in `FULL` (a) and `PAIR` (b) mode. SHA256 algorithm was used to create up to 100 chains of signatures using multiple signature algorithms: ECDSA, EdDSA, RSA, and HMAC.

in Figure 7(a) and 7(b)). A higher data payload produces fewer leaf hashes, meaning the system performs hashing operations less frequently. However, the throughput increase does not scale linearly as we can see that the performance of 100 KB and 1 MB data payload are nearly similar in Figure 7(a) and 7(b). Finally, the performance of SHA384 and SHA512 is higher than SHA224 and SHA256 algorithms because we run the experiment on the x64 machine. In contrast, the x86 machine will favor the latter over the former [24].

In terms of storage, a smaller data payload size generates a larger total byte size of receipts (c.f., Figure 7(c)) since smaller data payload generates more leaf hashes. Furthermore, the hashing algorithms also determine the receipt byte size. The figure shows that SHA512 produces more receipt bytes than the alternatives.

B. ON-CHAIN EVALUATION

In the next experiment, we evaluate our implemented smart contract methods as one way to analyze blockchain-related performance.

1) SETUP

We build a docker container utilizing 1 core of CPU and 2 GB of RAM to run Ganache [25], a simulated local Ethereum testbed. The `SIGNORA` smart contract is written in Solidity language and is deployed to the Ganache network using Truffle [26]. `SIGNORA` client communicates with Ganache using the Web3 library.

2) SMART CONTRACT COMPLEXITY

Ethereum introduces a "gas" unit to prevent nodes from overloading the network (e.g., storing massive garbage data or performing an infinite loop that depletes other nodes' resources). Most operations done in the Ethereum network is subject to a gas. For example, the amount of transactions inserted into a block is partly controlled by a "gas limit." The execution of smart contract methods also has the same restrictions in which the more complex the method is, the more gas it consumes, resulting in a higher transaction fee. Therefore, it is ideal for contract methods to be as efficient and straightforward as possible.

We can analyze the overhead of on-chain operations by measuring the gas used property for each implemented smart contract method. Table 4 summarizes the list of `SIGNORA` smart contract's methods and their gas consumptions. Only writeable methods are subject to a gas fee; meanwhile, read-only methods, e.g., GetChainSig(·) or GetReputation(·), do not consume gas because they do not modify the blockchain network state.

Results: Adding a new node costs more (about twice) than registering for a new manager or validator. The node registration for generator, relayer, or keeper nodes includes mapping between the manager and nodes. Therefore we store more data in this method, increasing the gas consumption.

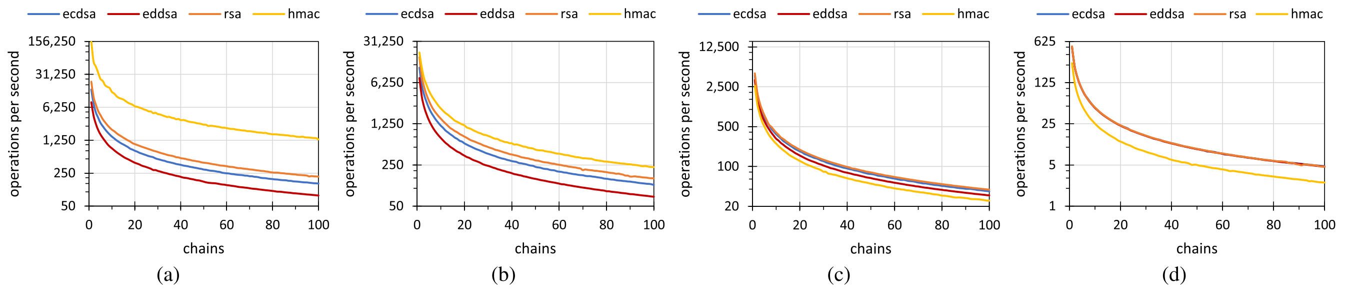

FIGURE 6. The performance evaluation of verifying chainsig from multiple data bytes: 1 KB (a), 10 KB (b), 100 KB (c), and 1 MB (d). SHA256 algorithm was initially used to create up to 100 chains of signatures using multiple signature algorithms: ECDSA, EdDSA, RSA, and HMAC.

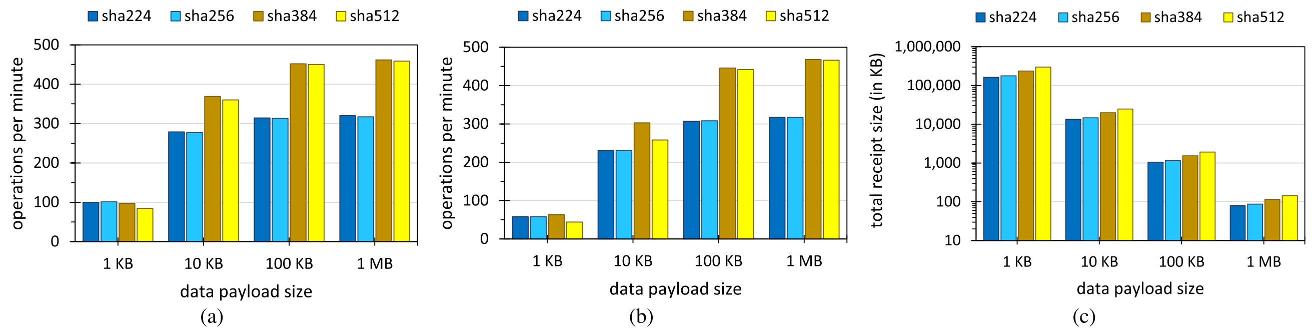

FIGURE 7. The performance evaluation of creating (a) and verifying (b) blockchain receipts on multiple data bytes: 1 KB, 10 KB, 100 KB, and 1 MB. Multiple hashing algorithms such as SHA224, SHA256, SHA384, and SHA512 are used to generate receipts. The total size of the generated receipt is shown in (c).

TABLE 4. List of writable smart contract methods in SIGNORA. The USD price statistic is taken from ETH Gas Station [27] on Feb 14th, 2022, using an average gas price of 48 Gwei.

Methods	Description	Gas Used	Price
ADDMANAGER	Add new manager	46,654	6.47
ADDNODE	Add new node	102,938	14.28
ADDVALIDATOR	Add new trusted validator	66,959	9.29
REGISTERKEY	Store HMAC commit hash	46,262	6.42
REGISTERKEY	Store EdDSA public key	88,279	12.25
REGISTERKEY	Store ECDSA public key	129,927	18.02
REGISTERKEY	Store RSA public key	256,022	35.52
ADDCHAINSIG	For HMAC with SHA256	153,097	21.24
ADDCHAINSIG	For EdDSA with SHA256	173,685	24.09
ADDCHAINSIG	For ECDSA with SHA256	193,961	26.91
ADDCHAINSIG	For RSA with SHA256	297,213	41.23
SUBMITREPORT	Submit timestamp report	89,578	12.43
ADDROOTHASH	Anchor root hash	48,234	6.69
REVEALHMAC	Reveal commitment hash	50,389	6.99
SUBMITSCORE	Submit reputation score	111,284	15.44
DEPLOYMENT	Deploy contract	3,654,843	506.99

Regarding the key registration, RSA is implemented with a 270 bytes long public key, encoded in X.509 format; ECDSA generates 64 bytes long public key while EdDSA produces 32 bytes key. Because of the limitation in storing bytes in Solidity, we have to split the long keys into an array of 32 bytes. We then save the key-length information so that we can reformat the arrays back to the original form. A lengthy public key such as RSA is a disadvantage because it consumes more gas than the rest. Finally, the HMAC secret key is represented as a commitment hash using the KECCAK-256 algorithm. Because the algorithm always returns a fixed 32 bytes length hash string, we can save only the hash (without length information) in the contract, resulting in cheaper gas consumption.

During daily operations, nodes must submit their signatures (as parts of the chainsig element) to the smart contract when using PAIR mode. The gas consumption for this step depends on the cryptographic algorithms. HMAC (in SHA-256) produces a shorter signature length than others. EdDSA comes in second place, which is slightly more efficient than ECDSA. Meanwhile, RSA generates longer signatures than the alternatives, thus, consuming the most gas. Aside from signature, this method also stores additional information such as the signature length, signature algorithm, timestamp, and task identifier. Therefore, this method consumes a lot of gas.

The smart contract saves the timestamp, the reporter address, and the task identifier when reporting a malicious behavior. Meanwhile, revealing HMAC and adding root hash only deals with a 32-byte hash; hence, they consume less gas than reporting cases. The adding validation score is expensive because we track the root hash to prevent validators from adding multiple scores for a given user, rendering a double scoring impossible. Finally, contract deployment is the most expensive operation among all methods. This behavior is expected since we need to store the smart contract's bytecode in the blockchain. SIGNORA contract produces about

3.6 million gas, which is still within a boundary of Ethereum block size (about 15-30 million gas). Thus, it can be appropriately deployed in the public network.

In conclusion, some of the gas costs are manageable. For example, the contract deployment and identity registration are performed only once during the contract's lifetime. Thus, higher gas metrics can be considered investments. Similarly, the key registration also happens less frequently (one in a very long time), assuming that the corresponding secret key is not leaked. The reporting procedure is an optional step that nodes can take when receiving malicious chainsig elements. When malicious activities rarely happen, calls to this method are not frequent.

Moreover, even though some methods produce high gas consumption and must be executed frequently, several steps can be taken off-chain to save transaction costs. For example, the keeper nodes can adjust the internal parameters on how frequently they form a root hash and anchor it to the blockchain. If the root hash can be produced once in a long duration, it can cut the gas cost. The validator can aggregate the evaluation score for each node off-chain and then submit the aggregated scores to the contract. Finally, adding a chainsig element to the smart contract is very expensive to be performed frequently. Therefore, we provide an alternative of `PAIR` mode, which is `FULL` mode.

Note that if developers decide to use permissioned/private blockchains instead of public ones, the gas consumption will have less impact on the system. Still, this metric is an excellent evaluation of the system's efficiency. Finally, the transaction fee for executing the method's call is subject to market economic evaluations; thus, the total fee can fluctuate.

3) REPUTATION SYSTEM

a: MEASURING CONTRIBUTION SCORES

We set the bounty scores β to 50 or 100. This parameter setting is merely used as an example. Developers can tweak this value according to their desired use cases. We then create a simulation to show how generator, relayer, and keeper nodes can improve their reputation scores.

Based on Equation 6, generators' contribution scores are calculated based on the number of data they can produce within a given validation time window. Therefore, we see linear growth in Figure 8(a); generators obtain scores as more data being generated. Similarly, keepers nodes also improve their contribution scores based on the number of stored chainsig elements (c.f., Equation 8). Hence, we can see the same linear growth in Figure 8(c).

Unlike previous scenarios, relayer nodes must process and retransmit the received data to the next node as quickly as possible to obtain the best scores. Figure 8(b) shows that the higher the processing delay, the smaller contribution scores that a relayer node can get from the system. In this case, we set t_{exp} from Equation 7 to 60 seconds. If node takes longer than 60 seconds to process data, they will receive 0 scores.

TABLE 5. List of configurations used to calculate contribution scores. Node 1 is the generator node; Node 2 and Node 3 are relayers, while Node 4 acts as the keeper node.

Parameters	Node 1	Node 2	Node 3	Node 4
Number of data generated	100	-	-	-
Number of data relayed	-	100	100	-
Processing delay (seconds)	-	600	180	-
Number of leaf hashes L	-	-	-	100
Number of chainsig element E	-	-	-	4

b: MEASURING DECAY OF REPUTATION SCORES

The reputation score for nodes and managers will expire over time. The developer can control how fast the score expires using t_{exp} parameter. We first set one initial reputation score (z_n in Equation 9) to 1000. After that, we set the t_{exp} to either 30 or 60 seconds. Similar to previous evaluations, these numbers are chosen merely as examples. We then call the GetReputation($\cdot$) method every second and record the result to see how the reputation score evolves.

Figure 9(a) shows that shorter duration of t_{exp} makes the reputation score to reduce to zero more quickly than longer duration of t_{exp}. Furthermore, a low value of t_{exp} also makes the scores decrease more drastically in the early seconds (having a deeper slope in the first 20 seconds) than the high value of t_{exp}. This indicates that the system will quickly ignore this particular score in lower t_{exp}. Scores closer to zero will have less impact when it is summed together in Equation 9.

c: MEASURING DYNAMIC OF REPUTATION SCORES

Four nodes (n_1, n_2, n_3, n_4) are deployed in the system. n_1 is a generator node. n_2 and n_3 are relayer nodes. n_4 is a keeper node. We also have two managers (m_1, m_2). m_1 owns n_1 and n_4 while m_2 governs n_2 and n_3.

At each validation time window, n_1 generates 100 data, which are relayed by n_2 and n_3 to n_4. n_2 is considered a slow processing node that processes data with a 10 minutes delay, while n_3 can process data quickly with a 5 minutes delay. This processing delay corresponds to $|t_n - t_{n-1}|$ in Equation 7, and we previously set the t_{exp} for Equation 7 to 20 minutes. Because we have four nodes, the number of chainsig elements becomes 4. For each chainsig, n_4 creates one receipt resulting in 100 leaf hashes. The summary of the simulation parameters is shown in Table 5.

Contribution scores (i.e., the generation, delivery, and storage scores) are then calculated based on Equation 6, 7, and 8 for all nodes. The resulting scores are then submitted to the smart contract by calling SubmitScore($\cdot$) method at the end of each validation time window. Note that the bounty β parameter is set to 100, and the validation is performed every 20 minutes.

We present the simulation over 3 hours duration, including (i) a 2-hour active period (where we validate nodes, calculate contribution scores, and add the scores to the smart contract every 20 minutes) and (ii) a 1-hour non-active period, where no scores will be added. During this simulation, we call the GetReputation($\cdot$) method every minute and plot nodes'

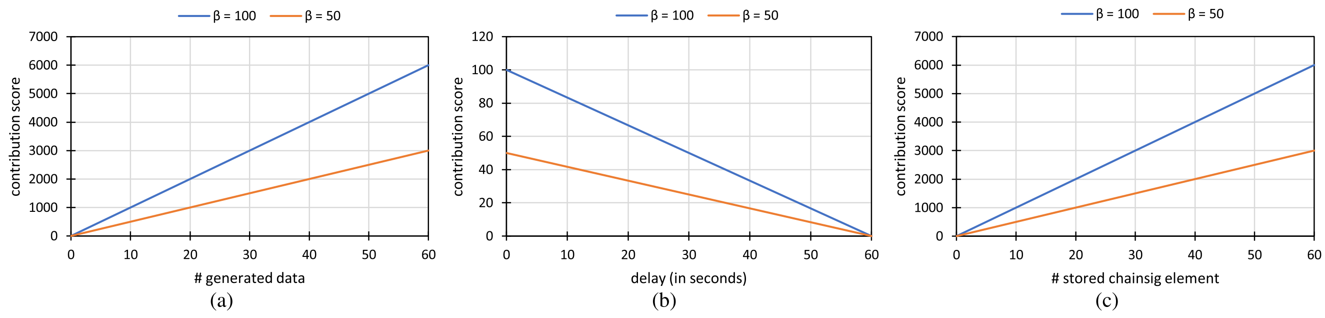

FIGURE 8. The results of contribution score evaluation: for generator (a), relayer (b), and keeper (c) nodes.

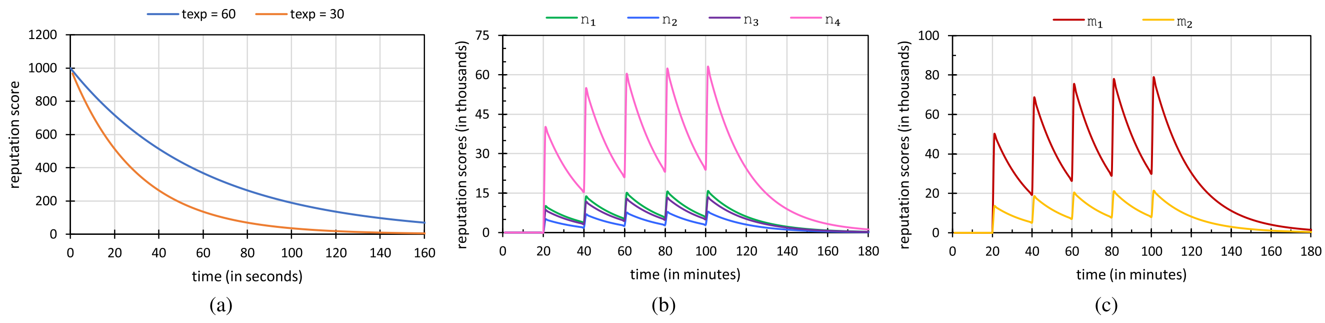

FIGURE 9. The results of reputation system evaluation: the decay of reputation score (a), node reputation (b), and manager reputation (c).

and managers' reputation overtime in Figure 9(b) and 9(c). Note that, t_{exp} for Equation 9 is configured with the value of 1200 seconds.

Several observations can be made from those figures. First, nodes and managers are initiated with zero reputation scores. Hence, we ensure that new nodes always have the lowest scores possible.

Second, the reputation scores for managers are exactly the sum of reputation scores from all managers' nodes (c.f., Equation 10).

Third, nodes and managers improve their reputation scores as we add contribution scores during the active period (at $t \leq 120$). They obtain the scores every 20 minutes, hence, their scores are updated accordingly at $t = 21, 41, 61, 81, 101$, and 121.

Fourth, the resulting reputation scores form the sawtooth wave due to the stored contribution scores decay over time (c.f. Equation 9). The highest scores (peak) happen right after we update the scores at $t = 21, 41, 61, 81, 101$, and 121. The reputation scores put attention only on the recent score updates. Old scores will contribute less to the equation, and therefore, with static score updates, nodes can see drastic peak increases at $t = 0$ to $t \leq 61$, but later only see steady peak increases at $t = 61$ to $t \leq 121$. This indicates that new nodes are rewarded with a good amount of reputation scores at the beginning to catch up with the scores of existing nodes. However, at some point, the only way to maintain their scores is by actively conducting honest behavior to receive the same amount of contribution scores at regular intervals.

Finally, similar to Figure 9(a), the reputation scores keep decreasing towards zero values during the non-active period (at $t > 121$). Given a very long non-active period (e.g., at $t \gg 180$), all nodes' scores will be much closer to zero. This behavior is intentional as we want to incentivize active behaviors from nodes. We should treat existing nodes as new nodes when they are inactive for a long time.

4) WHITE-BOX TESTING

Once the smart contract has been deployed in the blockchain network, it cannot be patched if a bug is found. Hence, unit testing becomes crucial in smart contract development. We have developed 49 unit test scenarios to ensure that SIGNORA smart contract works as intended. The test scenarios include not only primary cases such as registering user, chainsig, root hash, or scores but also more complex scenarios such as implementing access control when calling methods (e.g., only the manager can add nodes) and time-limited restrictions on methods' call (e.g., submitting scores can only be done in validation stage).

5) DEPENDENCY ON BLOCKCHAIN NETWORK

In this study, we do not propose a new blockchain network. Instead, our framework resides on top of the existing Ethereum blockchain network, whether it is a public or permissioned blockchain. Therefore, we inherit the security and performance properties of the actual implementation of the blockchain network. Because of this reasoning, we refrain from analyzing the underlying peer-to-peer network performance such as throughput (in terms of transactions per second), latency, CPU, RAM, and storage consumption. Those metrics can be varied depending on the blockchain consensus algorithm (c.f. [28]) and have been intensely investigated in [29].

VI. DISCUSSION

This section discusses several limitations and pointers to deploying our proposal in production cases.

A. HYBRID MODE

FULL mode is cheaper to operate in public blockchain because of having lesser on-chain processing compared to PAIR mode. However, it requires many network resources to transmit chainsig off-chain (from genesis to the latest), especially in a higher number of pipeline nodes. On the other hand, PAIR mode can reduce the off-chain network resources and have a stronger integrity guarantee than FULL mode at the expense of higher on-chain cost. Ideally, if the same organization owns all nodes inside a pipeline, we can safely use FULL mode to generate chainsig because all nodes are trusted. However, suppose they belong to different organizations. In that case, it is more beneficial (if economic cost is not the issue) to use PAIR mode because we can provide a better non-repudiation guarantee and limit the possibility of malicious behavior through on-chain processing. As a trade-off, a hybrid approach between FULL and PAIR mode can then be made out of this trend. For example, given $n_1, \ldots, n_5$ pipeline, if n_1 and n_2 belongs to m_1, n_3 comes from m_2, and m_3 deploy n_4 and n_5. Then, we can put FULL mode from n_1 to n_2 and from n_4 to n_5. Meanwhile, the PAIR mode is built on n_2, n_3, and n_4.

B. PERMISSIONED BLOCKCHAIN

On permissioned blockchain, we suggest using PAIR mode on all pipeline nodes rather than using FULL mode because we do not need to pay the transactions fee when calling the smart contract methods; thus, extending chainsig becomes affordable. This makes FULL mode less attractive in comparison.

C. ORDER OF PIPELINE NODES

Because chainsig performance drops on a higher number of chains when using FULL mode, more resource-constrained devices should not be put in the latter chain order. On the other hand, the order does not matter in PAIR mode because every node transmits about the same amount of bytes to the subsequent nodes.

D. ONLY ON ESSENTIAL DATAFLOW

Based on our evaluation results, the generations and validations of chainsig and receipt produce overheads that may hinder day-to-day operational tasks. Therefore, we suggest limiting SIGNORA use case only to be employed on "essential dataflow" [3] such as control sequence data that have a massive impact on the pipeline workflow. This way, SIGNORA can protect the system with minimal overhead.

E. CRYPTOGRAPHY ALGORITHMS

The chosen cryptography algorithm to use should depend on the actual use cases. For example, asymmetric cryptography becomes mandatory if the system needs a solid non-repudiation guarantee. Therefore, we suggest using either ECDSA or EdDSA because they produce better performance and lower signature bytes with the same security strength as RSA. However, RSA should only be used if those two options cannot be implemented due to backward compatibility or policy issues. On the other hand, if the system can accept the non-repudiation guarantee of HMAC key distribution through the proposed commitment hash, we suggest using HMAC if the delivered data is small (less or equal than 10 KB) since it can significantly improve the performance.

F. DATA LEAKAGE

Blockchain transparency can benefit multiple parties to quickly audit the correctness of the data stored in the network. However, this transparency also brings privacy issues in which all nodes that connect to the blockchain network can see the data. In our case, observers (valid blockchain nodes) can pinpoint the participants' identity and status. For example, to which organizations they belong (by observing the CAs), the current reputation scores (by calling methods in the smart contract), and the number of pipeline nodes that participants have (by monitoring the mapping in the smart contract). Furthermore, if using PAIR mode, the chainsig metadata is also revealed in the blockchain; therefore, the relationship between pipeline nodes can be leaked as well. Depending on the production requirements, this kind of data publicity may not be tolerable. Therefore, a thought-out design for constructing the blockchain network must be implemented by considering permissioned blockchain networks if necessary.

G. TRUSTED IDENTITIES

Our proposal is not fully decentralized because it requires trusted CA and validators to enable trust among multiple organizations. The trust is enforced easily in a centralized manner, for example, using the current PKI-based certificate management [30] for TLS/SSL. Meanwhile, decentralized solution such as PGP [31] or Web of Trust [32] is still not widely used.

H. MOTIVATION TO VALIDATE

The reputation system is functional as long as it is reliable. Some policy enforcements or incentives are required for participants to use the reputation system and drive system usability. For example, the Openly Operated standard [33], which requires service providers to be transparent and audited regularly, is a good starting point to motivate participants to

[3] We leave the definition of "essential dataflow" to the developers as it differs from one use case to another.

audit their system using SIGNORA. When enforced correctly, this standard will drive the validation requests and make the reputation system alive because of a higher number of contribution score updates.

VII. CONCLUSION

This paper proposed SIGNORA, a framework to provide dataflow integrity provisioning on an untrusted data pipeline. SIGNORA combined the concept of a chain of signatures with blockchain receipt, where involved participants took turns producing signatures of data they were currently processing. The hash of the data and signatures were then anchored in the blockchain for a stronger integrity guarantee through blockchain receipt.

Through the results of our experiments, we have shown that SIGNORA can provide dataflow integrity provisioning in multiple scenarios of data payload size with reasonable overhead. The cost of smart contract methods has also been analyzed, and several off-chain solutions have been addressed to reduce costs. Furthermore, we have shown that the reputation system can adapt to the history of nodes' activities by increasing their scores when they actively perform honest behavior and reducing their scores when they become inactive.

Still, developers must make crucial design decisions when deploying SIGNORA in their use case. For example, using a permissioned blockchain allows developers to eradicate the cost of storing data in the blockchain, but the system becomes more centralized. Furthermore, the chain of signatures should only be employed to provide integrity only on essential dataflow to reduce the overall processing overhead, especially in a high number of pipeline nodes.

For future works, we plan to improve our performance further and solve JSON marshaling issues by refactoring the code to be more efficient and employing an in-memory cache to speed up overall data processing. The vertical scalability of our code will also be investigated to reap the multi-threading benefits of the deployed hardware. Furthermore, SIGNORA can be implemented in Trusted Execution Environment (TEE) to solve the blockchain oracle issues further (e.g., make a trustable off-chain timestamp) that are still present in the current system.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Decentralized Bus. Rev., White Paper, Oct. 2008, Art. no. 21260. [Online]. Available: https://www.debr.io/article/21260-bitcoin-a-peer-to-peer-electronic-cash-system
[2] V. Buterin, "A next-generation smart contract and decentralized application platform," White Paper, 2014, vol. 3, no. 37. [Online]. Available: https://nft2x.com/wp-content/uploads/2021/03/EthereumWP.pdf
[3] W. Vaugan, J. Bukowski, and S. Wilkinson. (2016). *Chainpoint: A Scalable Protocol for Anchoring in the Blockchain and Generating Blockchain Receipts*. Accessed: Dec. 14, 2021. [Online]. Available: https://bit.ly/33s2CRM
[4] A. Guo, "Blockchain receipts: Patentability and admissibility in court," *Chicago-Kent J. Intellectual Property*, vol. 16, no. 2, p. 440, 2017.
[5] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng, "CrowdBC: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1251–1266, Jun. 2018.
[6] K. R. Özyilmaz, M. Doğan, and A. Yurdakul, "IDMoB: IoT data marketplace on blockchain," in *Proc. Crypto Valley Conf. Blockchain Technol. (CVCBT)*, Jun. 2018, pp. 11–19.
[7] J. R. Douceur, "The sybil attack," in *Proc. Int. Workshop Peer-to-Peer Syst.*, Cham, Switzerland: Springer, 2002, pp. 251–260.
[8] V. Bralić, H. Stančić, and M. Stengård, "A blockchain approach to digital archiving: Digital signature certification chain preservation," *Records Manage. J.*, vol. 30, no. 3, pp. 345–362, Feb. 2020.
[9] J.-P. Abegg, Q. Bramas, T. Brugière, and T. Noel, "Distributed publish/subscribe protocol with minimum number of encryption," in *Proc. 23rd Int. Conf. Distrib. Comput. Netw.*, Jan. 2022, pp. 117–123.
[10] S. Brotsis, N. Kolokotronis, K. Limniotis, S. Shiaeles, D. Kavallieros, E. Bellini, and C. Pavué, "Blockchain solutions for forensic evidence preservation in IoT environments," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2019, pp. 110–114.
[11] M. Hossain, Y. Karim, and R. Hasan, "FIF-IoT: A forensic investigation framework for IoT using a public digital ledger," in *Proc. IEEE Int. Congr. Internet Things (ICIOT)*, Jul. 2018, pp. 33–40.
[12] J. Zhao, G. Zhao, B. Pan, M. Zhou, L. Song, B. Wang, and J. Zhang, "An intelligent service method for grid spatio-temporal big data based on Beidou," in *Proc. 3rd Int. Conf. Inf. Technol. Electr. Eng.*, Dec. 2020, pp. 574–578.
[13] A. H. Lone and R. N. Mir, "Forensic-chain: Blockchain based digital forensics chain of custody with PoC in hyperledger composer," *Digit. Invest.*, vol. 28, pp. 44–55, Mar. 2019.
[14] L. Huang, Z. Liu, F. Xu, and J. Gu, "An RDF data set quality assessment mechanism for decentralized systems," *Data Intell.*, vol. 2, no. 4, pp. 529–553, Oct. 2020.
[15] Y. Li and S. Xu, "Design and implementation of a scalable distributed DNS system," in *Proc. IEEE 6th Int. Conf. Comput. Commun. Syst. (ICCCS)*, Apr. 2021, pp. 528–535.
[16] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 468–477.
[17] S. Shetty, V. Red, C. Kamhoua, K. Kwiat, and L. Njilla, "Data provenance assurance in the cloud using blockchain," in *Disruptive Technologies in Sensors and Sensor Systems*, vol. 10206. Washington, DC, USA: International Society for Optics and Photonics, 2017, Art. no. 102060I.
[18] Y. Zhang, S. Wu, B. Jin, and J. Du, "A blockchain-based process provenance for cloud forensics," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Dec. 2017, pp. 2470–2473.
[19] Y. Kwon, E.-K. Lee, and J. Jo, "Performance evaluation of data provenance system with blockchain-based cloud environment," *Int. J. Eng. Res. Technol.*, vol. 13, no. 12, pp. 4245–4250, 2020.
[20] X. Liang, J. Zhao, S. Shetty, and D. Li, "Towards data assurance and resilience in IoT using blockchain," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2017, pp. 261–266.
[21] M. Baldi, F. Chiaraluce, M. Kodra, and L. Spalazzi, "Security analysis of a blockchain-based protocol for the certification of academic credentials," 2019, *arXiv:1910.04622*.
[22] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, and S. Muralidharan, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, Apr. 2018, pp. 1–15.
[23] M. Bordewes. (2021). *How do RSA and ECDSA Differ in Signing Performance*. Accessed: Dec. 9, 2021. [Online]. Available: https://bit.ly/33lIMaB
[24] S. Touset. (2021). *SHA-512 Faster Than SHA-256*. Accessed: Dec. 9, 2021. [Online]. Available: https://bit.ly/3ypfuDv
[25] T. Suite. (2022). *Trufflesuite/Ganache: A Tool for Creating a Local Blockchain for Fast Ethereum Development*. Accessed: Feb. 15, 2022. [Online]. Available: https://bit.ly/33y36pU
[26] T. Suite. (2022). *Trufflesuite/Truffle: A Tool for Developing Smart Contracts. Crafted With the Finest Cacaos*. Accessed: Feb. 15, 2022. [Online]. Available: https://bit.ly/3H10CxY

[27] Concourse Open Community. (2022). *ETH Gas Station*. Accessed: Feb. 14, 2022. [Online]. Available: https://bit.ly/3JGcpU9
[28] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Proc. Int. Workshop Open Problems Netw. Security*, Cham, Switzerland: Springer, 2015, pp. 112–125.
[29] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "BLOCKBENCH: A framework for analyzing private blockchains," in *Proc. ACM Int. Conf. Manage. Data*, May 2017, pp. 1085–1100.
[30] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X. 509 internet public key infrastructure online certificate status protocol-OCSP," Internet Eng. Task Force (IETF), Wilmington, DE, USA, Tech. Rep. RFC 2560, 1999.
[31] S. Garfinkel, *PGP: Pretty Good Privacy*, Sebastopol, CA, USA: O'Reilly, 1995.
[32] G. Caronni, "Walking the web of trust," in *Proc. IEEE 9th Int. Workshops Enabling Technol., Infrastruct. Collaborative Enterprises (WET ICE)*, Jun. 2000, pp. 153–158.
[33] J. Lin and R. Dewan. (2022). *Openly Operated: The Internet's Transparency Standard*. Accessed: Feb. 17, 2022. [Online]. Available: https://bit.ly/3IcyXv6

YUSTUS EKO OKTIAN received the bachelor's degree in electrical engineering from Petra Christian University, Indonesia, in 2013, and the master's and doctoral degrees in computer engineering from Dongseo University, South Korea, in 2016 and 2021, respectively. He is currently a Postdoctoral Researcher with Pusan National University, South Korea. His research interests include network security, distributed computing, blockchain, the Internet-of-Things (IoT), and software-defined networking (SDN).

SHINWOOK HEO received the bachelor's degree from Pusan National University, South Korea, where he is currently pursuing the Ph.D. degree. He is a Senior Researcher with Smart M2M. He has conducted various projects, such as the IoT standard security technology development, smart city platform development, and crypto implementation. His current research interests include convergence of the IoT and blockchain.

HOWON KIM (Member, IEEE) received the Ph.D. degree from the Pohang University of Science and Technology (POSTECH), in 1999. From 1998 to 2008, he was at the Electronics and Telecommunications Research Institute (ETRI). From July 2003 to June 2004, he visited as a Postdoctoral Researcher at the Chair for the Communication Security Group (COSY), Ruhr-University Bochum, Germany. He is currently a Professor with the Chief of Energy IoT Research Center and the Chief of Blockchain Platform Research Center, Department of Computer Engineering, PSU. His research interests include blockchain platform, cryptography, deep learning, and security chip design.