









# Privacy-Preserving Anomaly Detection of Encrypted Smart Contract for Blockchain-Based Data Trading

Dajiang Chen , *Member, IEEE*, Zeyu Liao , Ruidong Chen , Hao Wang ,  
Chong Yu , *Graduate Student Member, IEEE*, Kuan Zhang , *Member, IEEE*,  
Ning Zhang , *Senior Member, IEEE*, and Xuemin Shen , *Fellow, IEEE*

**Abstract**—In a blockchain-based data trading platform, data users can purchase data sets and computing power through encrypted smart contracts. The security of smart contracts is important as it relates to that of the data platform. However, due to the inability to apply to detection rules with complex structures and the inefficiency of detection, existing malicious code detection methods are not suitable for the encrypted smart contracts in blockchain-based data trading platforms with high transaction rate requirements. In this article, a practical and privacy-preserving malicious code detection method is proposed for encrypted smart contract in blockchain-based data trading platform. Specifically, we design two kinds of miners to act as the malicious rule processor and the detector respectively for inspecting the encrypted smart contract. The rule processor generates an obfuscated map with the original open-source malicious rule set. The detector performs a malicious inspection algorithm by inputting the obfuscated map and the randomized tokens, where the latter is generated from smart contract. Then, we theoretically analyze the security syntax of the proposed method. The analysis results demonstrate the proposed scheme can achieve  $\mathcal{L}$ -secure against adaptive attacks. Extensive experiments are carried out through the open-source real rule sets, which show that the proposed scheme can reduce communication time and communication overhead.

**Index Terms**—Data trading platform, encrypted smart contract, malicious code detection, privacy-preserving.

Manuscript received 14 April 2023; revised 4 January 2024; accepted 10 January 2024. Date of publication 15 January 2024; date of current version 4 September 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB3106402, in part by NSFC under Grants 61872059 and 62002047, and in part by the Demonstration of Scientific and Technology Achievements Transform in Sichuan Province under Grant 2022ZHC0036. (*Corresponding author: Ruidong Chen.*)

Dajiang Chen, Zeyu Liao, and Hao Wang are with the Network and Data Security Key Laboratory of Sichuan Province, School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: djchen@uestc.edu.cn; zeyuliao.uestc@gmail.com; 201922090416@std.uestc.edu.cn).

Ruidong Chen is with the School of Computer Science Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: crdchen@163.com).

Chong Yu and Kuan Zhang are with the Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588 USA (e-mail: cyu6@huskers.unl.edu; kzhang22@unl.edu).

Ning Zhang is with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON N9B 3P4, Canada (e-mail: ning.zhang@uwindsor.ca).

Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: sshen@uwaterloo.ca).

Digital Object Identifier 10.1109/TDSC.2024.3353827

## I. INTRODUCTION

MASSIVE amounts of data is constantly being produced with the development of emerging technologies, e.g., Blockchain (BC) [1], [2], [3], Cloud/Fog/Edge Computing [4], Artificial Intelligence (AI), and 5 G Communication [5], [6], and their applications in Industrial Internet of Things (IIoT) [7], Internet of Medical Things (IoMT) [8], Smart City [9], etc. These data can be used for decision-making by leveraging Big Data analysis tools (e.g., AI algorithms and cloud computing) to benefit our economy and society [10], [11], [12]. As an emerging business model in the sharing economy, data trading can collect enough target data for users and bring value to the data collector/owner [13], [14], [15].

Existing Big Data trading systems face two major problems as follows. (1) How to efficiently protect the copyright of data; and (2) how to transfer copyright (or right to use) of data between two parties without trusted third party. As an alternative solution, blockchain technology is widely used in data copyright protection, and data trading between untrusted parties [16], [17], [18]. Actually, as a key enabling technology of data sharing and trading, blockchain has been used for data ownership confirmation, smart transaction contracts, and digital currency payments [16], [19], [20]. A blockchain-based coding platform allows developers to build distributed applications based on smart contracts [21], so that the programs can be executed automatically in full accordance with the contracts on the blockchain.

In this paper, a practical application scenario of blockchain-based data trading platform is considered. The platform consists of multiple data centers, a large number of data users, and smart contracts. Data centers maintain massive valuable data that can be traded and have a large amount of computing resources to perform AI algorithms for data mining [22], [23]. Data users purchase data sets and computing power via smart contracts, and analyze the purchased data sets through the corresponding data analysis algorithms (which are one part of the smart contract and executed by the data center), e.g., AI algorithms [24]. The corresponding computing results are return to data users based on the code logic of the smart contract, and data centers receive corresponding digital currency rewards. According to the consensus, the blockchain network guarantees that the status of the contract and the returned results are not tampered with.

On the one hand, for the security and privacy of the contact content, smart contract should be encrypted to ensure only the authorized parties (e.g., the corresponding data center) can obtain the plaintext of smart contract [25], [26]. On the other hand, due to the imperfect design of the blockchain-based platform itself, security vulnerabilities are not ruled out. Actually, regardless of environmental factors, inadvertent negligence or intentionally malicious behaviors of smart contract developers may raise serious security risks in the smart contract, which will cause irreparable losses to data security and user property [27]. As a result, it is necessary to ensure that the content of encrypted smart contract is legal before being published to the blockchain to prevent malicious activities. Accordingly, a practical and privacy-preserving encrypted smart contract inspection is required to support blockchain-based data trading platform.

In the existing literature, privacy-preserving encrypted traffic detection methods have emerged recently, and most of them introduce a third-party middlebox which performs detection during the transmission [31], [32], [33], [34]. In [31], a scheme named BlindBox is proposed, which ensures that the rules and plaintext data are visible only to the rule generator and endpoints respectively. It completes the detection of encrypted data without disclosing plaintext information. However, it incurs high computing/communication costs to implement garbled circuit [35] and oblivious transfer [36]. To improve the efficiency of the scheme above, an inspection scheme for encrypted traffic, namely PrivDPI, is presented [32]. In PrivDPI, the set of obfuscated rules can be generated from the rule set for malicious detection with an obfuscated rule generation algorithm, in which, the obfuscated rules obtained in preceding section can be reused to generate obfuscated rules in subsequent session. However, it requires high computing resource due to the usage of bilinear mapping. An encryption rule filter is designed for anomaly detection to store the encrypted action operations [33]. When the processed “content” option matches the randomized token successfully, the encrypted “action” of the rule can be restored and the corresponding operation should be executed. Later, a privacy-preserving inspection method of encrypted traffic is presented by using symmetric cryptographic techniques for IoT [34].

This paper aims to design a practical malicious code detection scheme of encrypted smart contract for blockchain-based trading platform. However, when privacy-preserving encrypted traffic detection meets blockchain-based trading platform, there are some technical challenges to be solved as follows. (1) *How to design a malicious code detection scheme that supports all forms of open-source Snort rules.* Note that, the rule used in existing schemes for malicious detection only consists single “content” option, while, in practice, a rule for malicious detection usually consists of multiple “content” options and an “action”, such as, Snort rules (please refer to Section III-B). (2) *How to design a malicious code detection scheme with high efficiency.* Existing schemes only perform rough sliding window processing on each rule for obfuscated rules generation, which destroys the semantic integrity of the rules and is inefficient. Moreover, most of existing works require high computing/communication costs [31], [32], [33], [34]. As a result, these schemes are not suitable for

the high transaction rate requirements of blockchain-based data trading platforms. (3) *How to design a malicious code detection scheme in a distributed way.* In blockchain-based data trading platforms, a centralized encryption smart contract anomaly detection scheme will face the problem of single point failure, so it is necessary to utilize the node resources of the peer-to-peer network for a fee to achieve distributed encryption smart contract anomaly detection. Accordingly, it is necessary to design an efficient and distributed malicious code detection scheme that supports malicious rules with multiple “content” options for encrypted smart contract in a blockchain-based trading platform.

In this paper, a practical malicious code detection scheme of encrypted smart contract is proposed by leveraging lightweight cryptography techniques. In the proposed scheme, a new obfuscated rule generation algorithm is designed to support all forms of open-source Snort rules for malicious detection. To further improve efficiency and scalability of inspection in a distributed way, in the proposed scheme, two kinds of miners are selected to perform encrypted smart contract inspection as the malicious rule processor and the detector, respectively. Specifically, one kind of miner is responsible for rule processing (i.e., rule processor) to get an obfuscated map with the original open-source malicious rule set. Another kind of miner who acts as the detector uses the obfuscated map to perform the malicious code inspection of the encrypted smart contract from the developer. The main objective of the proposed scheme includes (1) Privacy-preserving of Smart Contract: before the encrypted smart contract reaches the data center, other participants except the developer cannot obtain the code content; (2) Confidentiality of Rules: the original rule set is chosen by data center of the trading platform, and is visible only to the rule processor and data center; and (3) Reliable Detection: when the rule processor and the detector are honest but curious, the false negative and false positive errors of detection can be negligible. A new security syntax (e.g.,  $\mathcal{L}$ -secure) for encrypted smart contract inspection is defined formally. The formal proof of security proves that the proposed scheme is  $\mathcal{L}$ -secure against adaptive attacks.

The open-source Emerging Threats Snort Rules are utilized in proposed scheme. In Snort rules, a Snort rule can specify multiple “content” options, and the “content” options can be used in conjunction with “modifier” options to detect specific content in a traffic to trigger an appropriate action. The proposed scheme can be applicable to all forms of rules (the rule with single “content” option and the rule with multiple “content” options) in the existing open-source rule set. The proposed scheme analyzes and refines the rule structure, hides the “content” options, “modifier” options and “action” of each rule, and generates a small-capacity obfuscated map. When all the “content” options of a rule match and the corresponding “modifier” options match, the action operation of the rule can be restored. This greatly improves the accuracy of detection. We evaluate the performance of the proposed scheme with extensive experiments, and the experimental results demonstrate that the scheme has high detection rate, low time consumption/communication overhead. For example, the developer takes 285 ms to obtain randomized tokens with 5030 KB; the rule processor uses around 1987 ms to generate the obfuscated map generated by 5000 rules; and the

detector takes approximately 723 ms to detect smart contract (with size 25 KB) by using the obfuscated map from 4000 rules.

We summarize the contributions of this paper as follows.

- We propose a privacy-preserving and practical malicious code detection scheme of encrypted smart contract in blockchain-based data trading platform by leveraging lightweight cryptography techniques. As far as we know, it is a first privacy-preserving anomaly detection of encrypted data scheme that is suitable for all forms of open-source Snort rules in a distributed way.
- We define the security syntax of encrypted smart contract inspection scheme, i.e., the completeness and the  $\mathcal{L}$ -secure of a scheme. Given a leakage function  $\mathcal{L}$ , we prove the simplified version of the proposed scheme is complete and  $\mathcal{L}$ -secure against adaptive attacks.
- We conduct extensive simulation experiments to evaluate the performance of the proposed scheme on the open-source Emerging Threats Snort Rules. The results demonstrate that the proposed scheme can achieve high detection accuracy, small time cost, and low communication consumption.

## II. RELATED WORK

### A. Smart Contract

A smart contract is a computer program that is verified by means of information dissemination and can run autonomously on the blockchain. In [21], Nick introduced the concept of smart contract. When a smart contract is deployed in a block, trigger conditions and corresponding response rules are needed to be preset, and the relevant status will also be recorded in the smart contract. If the condition is triggered, the relevant action from the calling node will be executed in response. Only the behavior of modifying the contract state or value eventually is recorded in the blockchain, other behaviors are not.

As a programming platform with open source code, Ethereum enables the developers to design diverse distributed applications with smart contracts [28]. Ethereum can be used to guarantee behavior, design protocols, and process transactions such as enterprise management, voting applications, and e-commerce transactions [29]. Based on the deployed smart contract, Ethereum nodes can execute the contract in a decentralized manner. Generally, the smart contract is run by the local Ethereum virtual machine EVM of the node. EVM first interprets the contract program to obtain the bytecode file, and then executes it. When the transaction information related to the smart contract is stored in the blockchain, all nodes will run the contract according to the transaction information and change the state definitively.

### B. Encrypted Traffic Detection

In recent years, many novel encrypted traffic detection technologies were proposed. To solve the dilemma between privacy of encrypted traffic and content security, a privacy-preserving deep packet filtering protocol was designed to perform filtering function over encrypted traffic while guaranteeing the data and

rules privacy in software defined networks [30]. In [31], Blind-Box was proposed to realize deep packet inspection (DPI) over encrypted traffic. However, the usage of complex cryptography such as garbled circuit and oblivious transfer leads to the lower efficiency in obfuscated rule preparation phase and hinders the practical application.

On the basis of BlindBox, a scheme named Embark was proposed in [47]. In Embark, a novel encryption scheme, namely PrefixMatch, was introduced to realize DPI over encrypted traffic in a cloud-based middlebox, which enabled the cloud-based middlebox to detect whether an encrypted IP address is in a valid encrypted range. Later, a novel scheme BlindIDS was presented to further improve the performance of DPI over encrypted traffic by using pairing-based public key techniques [48]. However, its versatility is greatly reduced due to its incompatibility with TLS protocol. Subsequently, a more efficient scheme PrivDPI was proposed by utilizing bilinear mapping and a novelty obfuscated rule generation algorithm [32]. Yuan et al. designed an encryption rule filter to realize DPI over encrypted traffic, in which, an “action” will be restored and the corresponding operation will be executed, when the rule content matches the token successfully [37]. However, an inspection rule contains multiple fragments with different lengths generally. In this case, it is infeasible to realize DPI over encrypted traffic with the scheme in [37] by generating tokens of different sizes, as it will result in significant communication overhead for token set transmission. Other related works also include [38], [39], [40], [41], [42]. In [38], an efficient privacy-preserving deep packet inspection system was proposed in secure outsourced middleboxes. In [39], a privacy-preserving anomaly detection system was proposed for industrial control systems. Zhang et al. designed a blockchain-based privacy-preserving quality-aware incentive scheme [40]. In [41], a deep learning-based vulnerability detection framework was developed for smart contracts. Ivanov et al. proposed a real-time smart contract security testing approach, namely transaction encapsulation, to increase the vulnerability coverage [42].

### C. Searchable Symmetric Encryption

Searchable symmetric encryption supports the implementation of secure matching schemes that allow one party to outsource its data storage to another party in a private manner, while the data owner retains the ability to selectively search the data. Song et al. proposed a symmetric searchable encryption protocol in [43], which divides the plaintext file into “words” and encrypts the word collection. By scanning the entire ciphertext file and comparing the ciphertext words, the existence of the keyword in the ciphertext can be confirmed. In [44], Curtmola et al. standardized symmetric searchable encryption, and designed two schemes, namely SSE-1 and SSE-2, which can achieve indistinguishable security in non-adaptive/adaptive attack models. Later, Cash et al. presented a dynamic symmetric searchable encryption scheme to effectively search the encrypted database [45]. It is proved that the proposed scheme can achieve  $\mathcal{L}$ -secure against adaptive attacks.



TABLE I  
SUMMARY OF IMPORTANT NOTATIONS

Notation	Definition
$G(K_i)$	The pseudorandom function with key $K_i$ ;
$F(K_s)$	The pseudorandom function with key $K_s$ ;
$f(K_h)$	The pseudorandom function with key $K_h$ ;
$K_{AES}$	The AES symmetric key;
$M$	The plaintext of smart contract at the Client;
$\mathcal{W}$	The set of tokens $\{W_1, \dots, W_\ell\}$ for plaintext $M$ ;
$\mathcal{T}$	The set of tokens $\{T_1, \dots, T_\ell\}$ for plaintext $M$ ;
$\mathcal{R}$	The set of Snort rules $\{R_1, R_2, \dots, R_l\}$ ;
$seg_{i,j,k}$	The $k$ -th segmentation of $j$ -th content of $i$ -th rule;
$\mathcal{P}$	The collection of secret shares $\{p_1, \dots, p_n\}$ for processed content option $con$ ;
$\mathcal{Q}$	The collection of secret shares $\{q_1, \dots, q_n\}$ for action.

The main objective of this paper is to design a practical encryption smart contract anomaly detection scheme on the blockchain-based data trading platform. However, existing methods face some challenges when applied to encrypted smart contract anomaly detection on blockchain-based data trading platforms. (1) They are not suitable for the detection rules of complex structures, e.g., Snort rules; (2) their detection efficiency cannot meet the demand of data trading platform for efficient transaction speed; and (3) they cannot be directly applied to a peer-to-peer network environment. Different from existing methods, we propose a privacy-preserving and practical malicious code detection scheme of encrypted smart contract in blockchain-based data trading platform by leveraging lightweight cryptography techniques, which can be applied to all forms of open-source Snort rules in a peer-to-peer network environment.

### III. NOTIONS AND PRELIMINARIES

*Notions:* Let  $x$  and  $y$  be two bitstreams,  $x \oplus y$  be the bitwise XOR of  $x$  and  $y$ , and  $x\|y$  be the splicing of  $x$  and  $y$ . Random variables (RVs) are denoted by  $X, Y, K, \dots$ , and the realization of them are denoted by  $x, y, k, \dots$ . The main notations used in this paper are listed in Table I.

#### A. Preliminaries

By taking  $\lambda$  as a security parameter, several basic definitions are introduced as follows.

*Definition 1:* Let  $F : \{0, 1\}^{\alpha_1} \times \{0, 1\}^{\beta_1} \rightarrow \{0, 1\}^{\gamma_1}$  be an efficient and keyed function.  $F$  is a variable-input-length pseudo-random function, if the function

$$Adv_{F, \mathcal{A}}^{pdf}(\lambda) = \Pr[PRFReal_{\mathcal{A}}^F(\lambda)] - \Pr[PRFRand_{\mathcal{A}}^F(\lambda)] \quad (1)$$

is negligible for all probabilistic polynomial time (p.p.t.) adversary  $\mathcal{A}$ , in which,  $PRFReal$  and  $PRFRand$  are two games as follows. In  $PRFReal$ , the adversary  $\mathcal{A}$  first obtains key  $K$  from a dictionary with index  $j$ , and then, queries function  $F$  with outputting  $y$  and inputting an index  $K$  and  $x$ . In  $PRFRand$ , the adversary  $\mathcal{A}$  queries a random oracle  $R$  with outputting  $y$  and inputting an index  $j$  and  $x$ .

*Definition 2:*  $H : \{0, 1\}^{\alpha_2} \times \{0, 1\}^{\beta_2} \rightarrow \{0, 1\}^{\gamma_2}$  is called a collision-resistant hash function if the following experiment is negligible for any p.p.t. adversary  $\mathcal{A}$ .

TABLE II  
SUMMARY OF COMMON MODIFIERS IN SNORT RULES

Notation	Definition
<i>offset</i>	Set the location to start searching;
<i>depth</i>	Set the maximum depth of search;
<i>distance</i>	After a “content” is successfully matched, the next “content” is matched after at least <i>distance</i> bytes;
<i>within</i>	After a “content” is successfully matched, the next “content” is matched after at most <i>within</i> bytes;
<i>nocase</i>	It specifies that the “content” string is not case sensitive;
<i>http_header</i>	It limits the search in the header of extracted HTTP client request and server response, and the extracted header can be unformatted;
<i>http_client_body</i>	The modifier restricts the search in the body of the HTTP client request message;
<i>http_cookie</i>	It restricts the search in the cookie header fields of HTTP requests and responses.

- $\mathcal{A}$  succeeds if it outputs distinct  $x$  and  $x'$  with  $H_K(x) = H_K(x')$ , in which,  $H_K(\cdot) = H(K, \cdot)$  and key  $K$  is selected from  $\{0, 1\}^{\alpha_2}$  uniformly at random.

#### B. Snort Rules

*Snort* is an intrusion detection system, which analyzes and summarizes the known intrusion behaviors, summarizes the intrusion features, and forms rules. Its rule set *Snort Emerging Threats* is used in this paper as the initial rules to realize inspection. A rule includes two parties: header and options. *Header:* the rule header of Snort rules contains the response action and data flow direction, which is the content before the first “(”. *Options:* the rule options of Snort rules is the content between “(” and “)”, which are separated by “;”. A Snort rule can have multiple “content” option, and the “content” option can have multiple modifiers. Only when the packet matches one or several rule options successfully, can it be regarded as an insecure packet with intrusion behavior by Snort DPI system. We present a simple Snort rule as follows.

```

alert tcp $EXTERNAL_NET any
  → $HOME_NET 443
(msg : “openssl Heartbleed attack“;
content : “ftp.log”, fast_pattern, nocase)

```

The modifiers can be roughly divided into two categories. The first one is the modifiers such as *offset*, *depth*, *distance* and *within*, which explicitly limit the position of “content” with numbers. The second category is about content modifiers, such as “no case” which ignores case and a series of restrictive modifiers related to HTTP. When processing the packet payload, the location relationship can be clearly obtained from tokenization, but it is difficult to control the case format and HTTP related operations. Therefore, the proposed scheme focuses on the first type of modifiers. A list of important notations of *Snort* rule is shown in Table II.

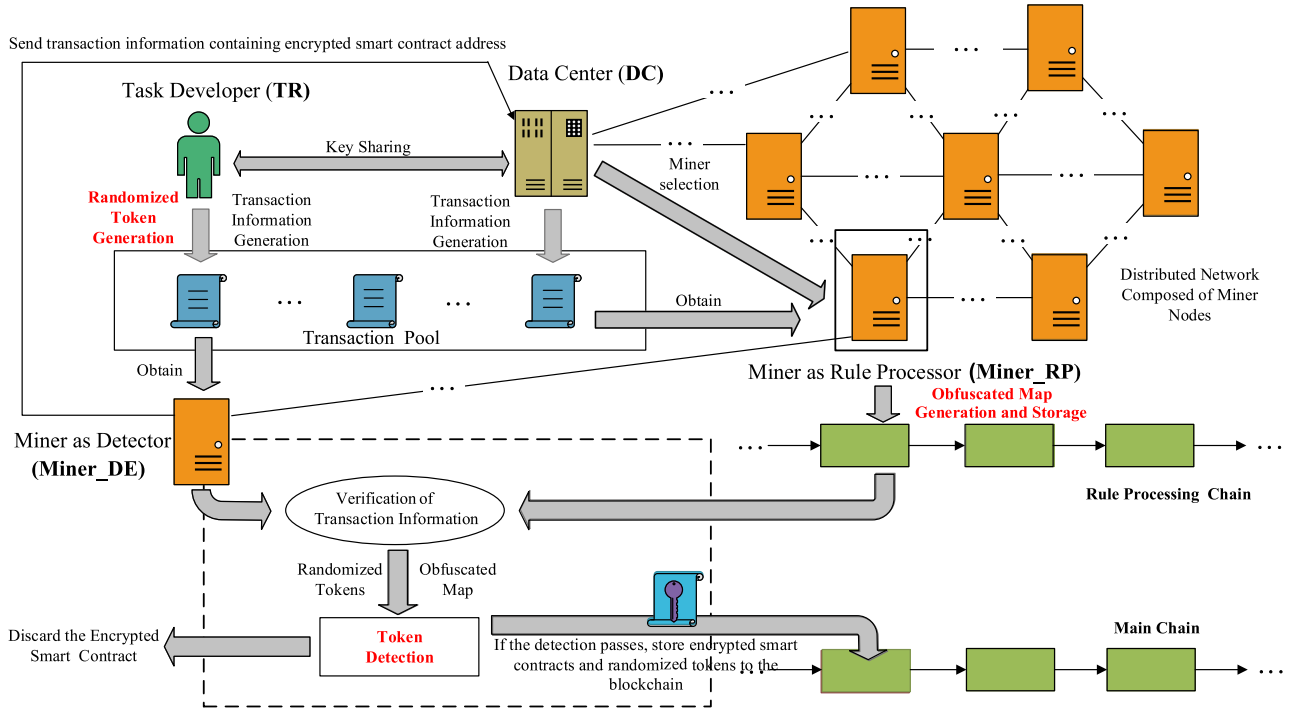


Fig. 1. System architecture.

#### IV. SYSTEM OVERVIEW

##### A. System Model

The system model is shown in Fig. 1, which includes four entities as follows. The data user, namely task requester ( $TR$ ), sends the encrypted smart contract for data analysis. The data center ( $DC$ ) sells its data and computing power to data user for data analysis. The miner ( $Miner_{RP}$ ) acts as a malicious rule processor, and the other miner ( $Miner_{DE}$ ) performs anomaly detection and data forwarding. Moreover, there are two blockchains in our system: main chain and rule processing chain.

$TR$ : When a task developer  $TR$  plans to send a smart contract on data trading platform to initiate a task request, who needs to generate a encrypted packet contained two data streams: an encrypted smart contract generated by symmetric key and a randomized token collection. Then the encrypted packet is sent to the trading pool by  $TR$ . If the randomized token collection passes the malicious detection, the encrypted packet will be packaged on blockchain. The encrypted packet can be obtained by  $DC$  from the blockchain.

$Miner_{RP}$ :  $Miner_{RP}$  is a miner who provides the processing of detection rules.  $Miner_{RP}$  generates the obfuscated rules from detection rules (e.g., Snort rules), with which,  $Miner_{DE}$  can detect malicious behavior if there is at least one “content” option in each rule to describe malicious code.

$Miner_{DE}$ :  $Miner_{DE}$  is a miner to monitor the encrypted packet from  $TR$  using the obfuscated rules issued by  $Miner_{RP}$ . If  $Miner_{DE}$  detects malicious data in the encrypted packet, it will interrupt the flow path and send the encrypted packet to  $Miner_{RP}$  for further analysis; otherwise, it will forensics

the encrypted packet and packages it with other transactions on blockchain.

$DC$ : After receiving the valid traffic from blockchain,  $DC$  first decrypts the encrypted packet, and then verifies the consistency of the content, i.e., whether the randomized token collection is generated correctly. If so,  $DC$  deploys the smart contract for the task of data mining; otherwise,  $TR$  is dishonest.

##### B. Threat Model

In this paper, the adversary model is considered as follows.

*Trust Assumption on Endpoints  $TR$  and  $DC$* . It is assumed that  $DC$  is credible without considering the problems caused by imperfect programming language and virtual environment. It is assumed that the  $TR$  can be malicious.  $TR$  wants to design a smart contract with malicious code and escape malicious detection. For instance,  $TR$  can generate fraudulent tokens to avoid the malicious detection.

*Semi-honest miner  $Miner_{DE}$  and  $Miner_{RP}$* . It is considered that the miners  $Miner_{DE}$  and  $Miner_{RP}$  are semi-honest, e.g., honest-but-curious. Specifically,  $Miner_{DE}$  must follow the steps of the proposed scheme to detect malicious code; but it wants to obtain the sensitive information from the smart contract, and attempts to infer the proprietary inspection rules from  $Miner_{RP}$ .

##### C. Design Goals

According to the above model, the design goals of the proposed system are as follows:

*Privacy-Preserving of Smart Contract:* This property requires that the privacy of smart contract of  $TR$  should be protected. Before the encrypted smart contract reaches the  $DC$ , other participants except the  $TR$  cannot obtain the code content.

*Confidentiality of Rules:* Only  $DC$  can choose the malicious rule set, and  $Miner_{RP}$  can learn and process the rules. In other words, other participants cannot know whether the plaintext contains the content that matches the rules.

*Efficiency and Accuracy:* To facilitate practical application, our system should have negligible false negative and false positive errors with low computation/communication cost.

## V. THE PROPOSED SCHEME

In this section, the proposed scheme is introduced, which consists of four phases: *Setup*, *Obfuscated Map Generation*, *Randomized Token Generation* and *Malicious Detection*.

### A. Setup

The setup phase is run by the  $TR$  and  $DC$  to exchange keys between them. These keys are used in subsequent phases for the processing of rules and smart contracts, as well as the selecting of the  $Miner_{RP}$ .

Assumed that  $TR$  and  $DC$  have a pair of public/private keys  $(PK_{TR}, SK_{TR})$  and  $(PK_{DC}, SK_{DC})$ , respectively, which can be generated by a Certificate Authority (CA).

*Key Distribution:*  $TR$  and  $DC$  negotiate a symmetric key  $K_{AES}$  for encrypting the smart contract with AES algorithm and a set of key  $K_{SET} = \{K_s, K_h, K_l\}$  for generating obfuscated map and randomized tokens by using *Diffie-Hellman* key distribution scheme [46].

*Miner<sub>RP</sub> Selection:* After obtaining the keys  $K_{AES}$  and  $K_{SET}$ , a miner should be selected from a set of miners  $\mathcal{A} = \{min_1, min_2, \dots, min_\delta\}$  as rule processor  $Miner_{RP}$  to process the original rule set. In order to provide the randomness of the miner choose,  $DC$  calculates the ID of miner from  $K_{AES}$ ,  $K_{SET}$ , and a time stamp  $ts$  as follows.

$$id_s = H(K_{AES} \| K_{SET} \| ts) \pmod{\delta} \quad (2)$$

According to (2),  $DC$  selects  $min_{id_s}$  as the rule processor. Moreover, for subsequent rule processing, the initial information of the transaction (i.e.,  $Inf_{DC}$  and  $Sig_{DC}$ ) needs to be put into the transaction pool of Rule Processing Chain (RP Chain) by  $DC$  in order to be shared with  $Miner_{RP}$ .

$$Inf_{DC} = Tran_{id} \| addr_{DC} \| addr_{min_{id_s}} \| Enc_{PK_{min_{id_s}}}(K_{SET}) \| ts \| id_s \quad (3)$$

$$Sig_{DC} = Sign_{SK_{DC}}(Inf_{DC}) \| Sign_{SK_{DC}}(Tran_{id} \| K_{SET} \| ts) \quad (4)$$

where  $Tran_{id}$  is the transaction ID,  $addr_{DC}$  and  $addr_{min_{id_s}}$  are the addresses of  $DC$  and miner respectively,  $PK_{min_{id_s}}$  is the public key of miner  $min_{id_s}$ ,  $Sign(\cdot)$  is the signature algorithm designed in [49] and  $Sig_{DC}$  denotes the signature of  $DC$ .

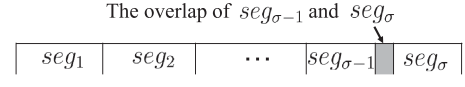


Fig. 2. Content segmentation.

### B. Obfuscated Rule Generation

Suppose that the open-source Snort rule set  $\mathcal{R} = \{R_1, R_2, \dots, R_\iota\}$  is selected as the detection rules, where the “content” option in each rule is used to match the smart contract to be inspected. An efficient obfuscated rule generation algorithm is proposed in this phase, which is based on the security framework of searchable symmetric encryption. Concretely,  $Miner_{RP}$  generates obfuscated map through the following steps.

*Key Acquisition:* As the rule processor, miner  $min_{id_s}$  first obtains  $Inf_{DC}$  and  $Sig_{DC}$  from the transaction pool of RP chain. Then  $min_{id_s}$  decrypts  $Inf_{DC}$  with its own private key  $SK_{min_{id_s}}$  to obtain the set of keys  $K_{SET} = (K_s, K_h, K_l)$ . Next,  $min_{id_s}$  verifies the validity of the signature  $Sig_{SK_{DC}}(Tran_{id} \| K_{SET} \| ts)$  from  $Tran_{id}$ ,  $K_{SET}$ , and  $ts$ . Meanwhile, the obtained signature of  $DC$  can be used as part of generated obfuscated rule later.

*Preliminary Treatment:* For each Snort rule  $R_i$ , extracting the content options and its modifier options to get the  $\{con_{i_j} : mod_{i_j}\}$  key value pair, where  $con_{i_j}$  is the  $j$ -th content of rule  $R_i$ , and the  $mod_{i_j}$  is the splicing of all modifiers corresponding to  $con_{i_j}$ . Note that, a Snort rule may contain multiple content options, and each content option may be modified by multiple modifier options. Moreover, each rule only has one header contained the security action (such as alert, log, pass, activate, and dynamic). Accordingly, rule  $R_i$  is initially processed as

$$R_i = \{con_{i_1} : mod_{i_1}; con_{i_2} : mod_{i_2}; \dots; con_{i_{\tau_i}} : mod_{i_{\tau_i}}; action_i\} \quad (5)$$

*Content Segmentation:* Taking the shortest length of all content options as  $len_{win}$ , all content options can be divided into several sub-contents with length  $len_{win}$ . As shown in Fig. 2, if the length of the last part is less than  $len_{win}$ , the last sub-content can be supplemented with the preceding bytes. Then, rule  $R_i$  is processed as

$$R_i = \{seg_{i_1 1}, seg_{i_1 2}, \dots, seg_{i_1 \sigma_{i_1}} : mod_{i_1}; \\ seg_{i_2 1}, seg_{i_2 2}, \dots, seg_{i_2 \sigma_{i_2}} : mod_{i_2}; \\ \dots \\ seg_{i_{\tau_i} 1}, seg_{i_{\tau_i} 2}, \dots, seg_{i_{\tau_i} \sigma_{i_{\tau_i}}} : mod_{i_{\tau_i}}; action_i\} \quad (6)$$

*Random ID Generation:* An ID of each content of the rules is defined to uniquely identify the rule for facilitating the subsequent operations as follows. The Snort rules  $\{R_1, R_2, \dots, R_\iota\}$  are reordered randomly, and the new sequence number of  $R_i$  is denoted by  $rid_{R_i}$ . Then, the random ID of the  $j$ -th content of  $R_i$  can be denoted by  $id_{i_j}$  and expressed as  $rid_{R_i} \| j \| b_{i_j}$ , where  $b_{i_j} \in \{0, 1\}$ , and  $b_{i_j} = 1$  if  $j = \sigma_{i_j}$ ;  $b_{i_j} = 0$ , otherwise. For example, a rule has 10 content options and its  $rid$  is 2. The

fifth and tenth content option of the rule can be expressed as  $id_{2_5} = 0002\|005\|0$  and  $id_{2_{10}} = 0002\|010\|1$ , respectively.

**Action and Modifiers Hiding:** In order to protect the privacy of rules,  $Miner_{RP}$  needs to hide the action and content options. The proposed system aims to restore the content option through its segments, and then restore the final action that declared in rule header after the matching occurs through all the content options in the rule. To realize the efficient hide the action and modifiers, an efficient  $(n, n)$  secret sharing scheme is utilized as follows.

For action hide, we target on the security requirement that, only when all related content options of a rule are matched, the corresponding action can be recovered; otherwise, nothing can be revealed. Specifically,  $\tau_i - 1$  random strings  $q_{i_1}, \dots, q_{i_{\tau_i-1}}$  are generated with the same length as action, and let  $q_{i_{\tau_i}} = q_{i_1} \oplus \dots \oplus q_{i_{\tau_i-1}} \oplus action_i$ . The collection secret sharing is defined as  $Q = \{q_{i_1}, \dots, q_{i_{\tau_i}}\}$ , where each  $q_{i_j}$  is a share of action. Then, the action can be restored as follows,

$$action_i = q_{i_1} \oplus \dots \oplus q_{i_{\tau_i-1}} \oplus q_{i_{\tau_i}} \quad (7)$$

Next, the ID of each original content  $id_{i_j}$ ,  $q_{i_j}$  and  $mod_{i_j}$  ( $j \in \{1, 2, \dots, \tau_i\}$ ) are first processed by a hash function  $H$ . The corresponding hash value, secret share  $q_i$  and the modifiers  $mod_{i_j}$  are spliced together as  $C_{i_j}$ , i.e.,

$$C_{i_j} = H(id_{i_j}, q_{i_j}, mod_{i_j}) \| q_{i_j} \| mod_{i_j}, \forall j \in \{1, 2, \dots, \tau_i\} \quad (8)$$

Similarly, given a content option  $con_{i_j}$  with  $\sigma_{i_j}$  segments, the corresponding  $C_{i_j}$  is also treated as a secret. Randomly choosing  $\sigma_{i_j} - 1$  strings  $\{p_{i_j}(1), \dots, p_{i_j}(\sigma_{i_j} - 1)\}$ , and the length of each bit string is the same bit length of  $C_{i_j}$ . Taking  $p_{i_j}(\sigma_{i_j}) = p_{i_j}(1) \oplus \dots \oplus p_{i_j}(\sigma_{i_j} - 1) \oplus C_{i_j}$ , we have

$$C_{i_j} = p_{i_j}(1) \oplus \dots \oplus p_{i_j}(\sigma_{i_j} - 1) \oplus p_{i_j}(\sigma_{i_j}) \quad (9)$$

Finally, rule  $R_i$  can be processed as follows.

$$\begin{aligned} &\{seg_{i_1 1}, \dots, seg_{i_1 \sigma_{i_1}} : id_{i_1} \| p_{i_1}(1), \dots, id_{i_1} \| p_{i_1}(\sigma_{i_1}); \\ &seg_{i_2 1}, \dots, seg_{i_2 \sigma_{i_2}} : id_{i_2} \| p_{i_2}(1), \dots, id_{i_2} \| p_{i_2}(\sigma_{i_2}); \\ &\dots \\ &seg_{i_{\tau_i} 1}, \dots, seg_{i_{\tau_i} \sigma_{i_{\tau_i}}} : id_{i_{\tau_i}} \| p_{i_{\tau_i}}(1), \dots, id_{i_{\tau_i}} \| p_{i_{\tau_i}}(\sigma_{i_{\tau_i}})\} \quad (10) \end{aligned}$$

**Obfuscated Map Generation:** An obfuscated map  $Obm$  is generated by  $Miner_{RP}$  as follows. For all  $\theta \in \{1, 2, \dots, \sigma_{i_j}\}$ ,  $(id_{i_j} \| p_{i_j}(\theta)) \oplus s$  is inserted into map  $Obm[loc]$ , where  $s = f(K_h, seg_{i_j \theta})$ ,  $loc = G(K_l, l)$  and  $l = F(K_s, seg_{i_j \theta})$ . To solve the hash collision problem, a variable-length hash bucket is built for each location. After obtaining the obfuscated map  $Obm$ ,  $Miner_{RP}$  puts the mapping table of  $Obm$  on RP chain as the body of a block. Specifically, the content of the block body contains

$$\begin{aligned} Inf_{RP} &= Tran_{id} \| |ts| \| |len_{win}| \| |Obm| \| |Sign_{SK_{DC}}| \\ &\times (Tran_{id} \| |K_{SET}| \| |ts|) \quad (11) \end{aligned}$$

and the signature

$$Sign_{RP} = Sign_{SK_{RP}}(Inf_{RP}) \quad (12)$$

---

### Algorithm 1: Obfuscated Rule Generation Algorithm.

---

**Require:** The Snort rule set  $\mathcal{R} = \{R_1, R_2, \dots, R_\iota\}$ , the shortest length  $len_{win}$  of all content options in rule set, and the initial information of transaction  $Inf_{DC}$  (i.e., Equ. (3)) from  $DC$  and the signature  $Sign_{DC}$  (i.e., Equ. (4)) of  $DC$ , where  $R_i = \{con_{i_1} : mod_{i_1}; con_{i_2} : mod_{i_2}; \dots; con_{i_{\tau_i}} : mod_{i_{\tau_i}}; action_i\}$ .

**Ensure:**  $Inf_{RP}$  and  $Sign_{RP}$ .

**(1) Key Acquisition.**

- Obtaining  $K_{SET} = (K_s, K_h, K_l)$  from  $Enc_{PK_{minid_s}}(K_{SET})$ ;
- Verifying the validity of the signature  $Sign_{SK_{DC}}$  with transaction ID  $Tran_{id}$ , keys  $K_{SET}$ , and time stamp  $ts$ .

**(2) Preliminary Treatment.**

- For each  $i \in \{1, \dots, \iota\}$ ,  $j \in \{1, \dots, \tau_i\}$ ,  $con_{i_j}$  is divided into sub-contents with length  $len_{win}$ :  $seg_{i_{j_1} 1}, seg_{i_{j_1} 2}, \dots, seg_{i_{j_1} \sigma_{i_{j_1}}}$ .

**(3) Random ID Generation.**

- The rules  $\{R_1, R_2, \dots, R_\iota\}$  are reordered randomly, and the new sequence number of  $R_i$  is denoted by  $rid_{R_i}$ ;
- The random ID of the  $j$ -th content of  $R_i$  is denoted by  $id_{i_j} = rid_{R_i} \| j \| b_{i_j}$ , where  $b_{i_j} \in \{0, 1\}$ , and  $b_{i_j} = 1$  if  $j = \sigma_{i_j}$ ;  $b_{i_j} = 0$ , otherwise.

**(4) Action and Modifiers Hiding.**

- For all  $i$ , randomly choosing  $\tau_i - 1$  bit strings  $q_{i_1}, \dots, q_{i_{\tau_i-1}}$  with the same length as  $action_i$ , and taking  $q_{i_{\tau_i}} = q_{i_1} \oplus \dots \oplus q_{i_{\tau_i-1}} \oplus action_i$ ;
- $\forall j \in \{1, 2, \dots, \tau_i\}$ , let  $C_{i_j} = H(id_{i_j}, q_{i_j}, mod_{i_j}) \| q_{i_j} \| mod_{i_j}$ ;
- For all  $C_{i_j}$ , randomly choosing  $\sigma_{i_j} - 1$  strings  $p_{i_j}(1), \dots$ , and  $p_{i_j}(\sigma_{i_j} - 1)$  with the same length of  $C_{i_j}$ , and taking  $p_{i_j}(\sigma_{i_j}) = p_{i_j}(1) \oplus \dots \oplus p_{i_j}(\sigma_{i_j} - 1) \oplus C_{i_j}$ , such that

$$C_{i_j} = p_{i_j}(1) \oplus \dots \oplus p_{i_j}(\sigma_{i_j} - 1) \oplus p_{i_j}(\sigma_{i_j});$$

- Finally, rule  $R_i$  can be processed as Equ. (10).

**(5) Obfuscated Map Generation.**

- For all  $\theta \in \{1, 2, \dots, \sigma_{i_j}\}$ ,  $(id_{i_j} \| p_{i_j}(\theta)) \oplus s$  is inserted into map  $Obm[loc]$ , where  $s = f(K_h, seg_{i_j \theta})$ ,  $loc = G(K_l, l)$  and  $l = F(K_s, seg_{i_j \theta})$ .
- Taking  $SK_{RP}$  as the secret key of  $Miner_{RP}$ , and computing

$$\begin{aligned} Inf_{RP} &= Tran_{id} \| |ts| \| |len_{win}| \| |Obm| \| |Sign_{SK_{DC}}| \\ &\times (Tran_{id} \| |K_{SET}| \| |ts|) \end{aligned}$$

and the signature  $Sign_{RP} = Sign_{SK_{RP}}(Inf_{RP})$ .

---

where  $SK_{RP}$  is the secret key of  $Miner_{RP}$ .

For the details of *Obfuscated Rule Generation Algorithm*, please refer to Algorithm 1.



### C. Randomized Token Generation

To realize the detection on encrypted smart contract,  $TR$  needs to generate a randomized token collection  $\mathcal{T}$ , besides the traditional encrypted information  $E_{K_{AES}}(M_{sc})$ , where  $M_{sc}$  denotes the plaintext of smart contract. To obtain  $\mathcal{T}$ ,  $TR$  first gets the  $Inf_{RP}$  and  $Sign_{RP}$  from the RP chain, and checks the validity of the signatures  $Sign_{DC}$  and  $Sign_{RP}$ . If so,  $TR$  divides the entire smart contract  $M$  into a token collection  $\mathcal{W} = \{W_1, W_2, \dots, W_\ell\}$ , i.e.,

$$\mathcal{W} = \{W_1, W_2, \dots, W_\ell\} = Token(M_{sc}) \quad (13)$$

by using a sliding-window-based tokenization algorithm  $Token(\cdot)$ , where each word  $W_i$  have the same length of sliding window  $len_{win}$ .

Then,  $TR$  utilizes pseudorandom functions and keys  $K_{SET}$  to generate randomized tokens  $\mathcal{T}$  as follows. For each  $\iota \in \{1, \dots, \ell\}$ ,  $TR$  computes  $T_i = F(K_s, W_\iota) \| f(K_h, W_\iota)$  by using two pseudorandom functions  $F$  and  $f$  with the keys  $K_s$  and  $K_h$ , respectively. Besides matching the rule's content options, it also needs to satisfy the corresponding requirements in the modifier options. Therefore, the starting position  $sp_\iota$  of  $W_\iota$  should be added as the following form.

$$T_\iota = F(K_s, W_\iota) \| f(K_h, W_\iota) \| sp_\iota \quad (14)$$

Next,  $TR$  can obtain the transaction information  $TransInf_{TR} = Inf_{TR} \| Sign_{TR}$  by computing

$$Inf_{TR} = Tran_{id} \| ts \| len_{win} \| id_s \| addr_{TR} \| addr_{DC} \| \mathcal{T} \| E_{K_{AES}}(M_{sc}) \quad (15)$$

$$Sign_{TR} = Sign_{SK_{TR}}(Inf_{TR}) \quad (16)$$

where  $addr_{TR}$  denotes the IP addresses of  $TR$ ,  $addr_{dest}$  indicates the IP address of  $DC$ ,  $\mathcal{T} = \{T_\iota\}_{\iota=1}^\ell$ , and  $SK_{TR}$  denotes the secret key of  $TR$ .

Finally,  $TR$  puts  $TransInf_{TR}$  into the transaction pool of main chain for subsequent token detection operations.

### D. Malicious Detection

In token detection phase, a miner  $Miner_{DE}$  is introduced as a detector to detect the malicious code in the smart contract. Before performing token detection, the following verification operations are required.

- $Miner_{DE}$  first obtains  $Inf_{TR}$  and  $Sign_{TR}$  from the transaction pool of main chain, and  $Inf_{RP}$  and  $Sign_{RP}$  from RP chain.
- And then,  $Miner_{DE}$  compares whether the  $len_{win}$  in  $Inf_{TR}$  and  $Inf_{RP}$  are the same: if so,  $Miner_{DE}$  continues with the following step; otherwise, the transaction is invalid.
- Next,  $Miner_{DE}$  checks the validity of the two signatures  $Sign_{TR}$  and  $Sign_{RP}$ : if so,  $Miner_{DE}$  continues with the following step; otherwise, the transaction is invalid.
- Finally,  $Miner_{DE}$  extracts the randomized token collection  $\mathcal{T}$  and obfuscated mapping table  $Obm$ .

After accomplishing the verification,  $Miner_{DE}$  performs the following steps.

- For  $\iota \in \{1, 2, \dots, \ell\}$ ,  $Miner_{DE}$  splits  $T_i$  into  $t'_\iota = F(K_s, W_\iota)$ ,  $t''_\iota = f(K_h, W_\iota)$  and  $L_\iota = sp_\iota$ , calculates the  $loc = G(K_l, t'_\iota)$ , and looks up the  $Obm$  mapping table through the location information  $loc$  to find the stored result  $Obm[loc] = (id_{i_j} \| p_{i_j}(\theta)) \oplus s$ .
- $Obm[loc]$  is XOR with  $t'_\iota$  to obtain the splicing of ID  $id_{i_j}$  and the share  $p_{i_j}(\theta)$ , where  $id_{i_j} = rid_{R_i} \| j \| b_{i_j}$ .
- $Miner_{DE}$  maintains a table  $\mathcal{T}_{con}$ , and stores the divided  $id_{i_j}$ ,  $p_{i_j}(\theta)$ , and  $sp_\iota$ .
- For each  $id_{i_j}$ , let  $\{id_{i_j}, p_{i_j}(\theta_\nu), sp_{\nu}\}_{\nu=1}^{\nu_0}$  be the set of triples with the same content ID  $id_{i_j}$ . If  $|sp_{\nu} - sp_{\nu-1}| = len_{win}$  for  $\nu < \nu_0$  and  $|sp_{\nu} - sp_{\nu-1}| \leq len_{win}$  for  $\nu = \nu_0$ , then,  $Miner_{DE}$  computes

$$\hat{C}_{i_j} = \hat{H}_{i_j} \| \hat{q}_{i_j} \| \widehat{mod}_{i_j} = \bigoplus_{\nu=1}^{\nu_0} p_{i_j}(\theta_\nu) \quad (17)$$

- $Miner_{DE}$  maintains a table  $\mathcal{T}_{rule}$  as follows. For each  $id_{i_j}$ , if (1)  $\widehat{mod}_{i_j}$  is a valid modifier and the starting position  $sp_{\nu_1}$  and the ending position  $sp_{\nu_0} + len_{win}$  conform to this modifier, and (2)  $\hat{H}_{i_j} = H(id_{i_j}, \hat{q}_{i_j}, \widehat{mod}_{i_j})$ , then  $rid_{R_i}$ ,  $j \| b_{i_j}$ ,  $\hat{H}_{i_j}$ ,  $\hat{q}_{i_j}$ , and  $\widehat{mod}_{i_j}$  are stored in table  $\mathcal{T}_{rule}$ .
- $Miner_{DE}$  maintains a table  $\mathcal{T}_{action}$  as follows. For each rule  $rid_{R_i}$ , if each element in  $\{1 \| 0, 2 \| 0, \dots, j_1 - 1 \| 0, j_1 \| 1\}$  is included in  $\mathcal{T}_{rule}$ , then  $Miner_{DE}$  restores  $action_i$  by computing

$$action_i = q_{i_1} \oplus q_{i_2} \oplus \dots \oplus q_{i_{j_1}} \quad (18)$$

Finally,  $rid_{R_i}$  and  $action_i$  are stored in table  $\mathcal{T}_{action}$  for  $Miner_{DE}$  to record the matching rules and actions.

If  $\mathcal{T}_{action}$  is empty after completing the detection, it means that the content of the smart contract is legal, and  $Inf_{TR}$  will be put on the blockchain for further data mining; otherwise, it is considered that the smart contract contains malicious information, and  $Miner_{DE}$  directly discards  $Inf_{TR}$ .

After receiving  $Inf_{TR}$ ,  $DC$  can obtain the smart contract  $M_{sc}$  from  $E_{AES}(M_{sc})$  with key  $K_{AES}$ . Then  $DC$  checks the consistency of the smart contract  $M_{sc}$  and the randomized tokens  $\mathcal{T}$ . Finally,  $DC$  performs AI algorithm in smart contract  $M_{sc}$  over its data set to get the final results, and return the results to  $DET$ .

## VI. SECURITY ANALYSIS

In this section, a novel security model will be introduced, and then the security theorem and its proof will be presented.

### A. Security Model

*Definition 3:* A privacy-preserving anomaly detection scheme  $\Pi = (Setup, ObMapGen, RaTokenGen, Match)$  over message space  $\mathcal{M}$  consists of a tuple of p.p.t. algorithms  $Setup$ ,  $ObMapGen$ ,  $RaTokenGen$  and  $Match$  as follows:

- $Setup(1^\lambda, KeyGen, KeyDist)$ : Let  $\lambda$  be a security parameter. Algorithm  $KeyGen$  outputs a key  $k = (K_{CS}, K_{RP})$ ; a key distribution protocol  $KeyDist$  can



distribute  $K_{RP}$  to rule processor, and  $K_{CS}$  to sender and receiver, securely.

- $ObMapGen(K_{RP}, \mathcal{R})$ : Algorithm  $ObMapGen$  outputs an obfuscated map  $Obm$  by inputting the keys  $K_{RP}$  and a rule  $\mathcal{R}$
- $RaTokenGen(k, W_1, W_2, \dots, W_\ell)$ : Algorithm  $RaTokenGen$  outputs a set of randomized tokens  $\{T_1, T_2, \dots, T_\ell\}$  by inputting the key  $k$  and a set of  $\ell$  original tokens  $\{W_1, W_2, \dots, W_\ell\}$  for message  $M \in \mathcal{M}$ .
- $Match(Obm, T_1, T_2, \dots, T_\ell)$ : Algorithm  $Match$  outputs a set of  $Ture/Flase$  values  $\{V_1, V_2, \dots, V_\ell\}$  by inputting the obfuscated map  $Obm$  and the set of randomized tokens  $T_1, T_2, \dots, T_\ell$ , where  $Ture$  (i.e., 1) indicates that it matches successfully;  $Flase$  (i.e., 0) means no match was found.

**Definition 4 (Completeness):** Scheme  $\Pi$  is completeness if for all efficient adversary  $\mathcal{Adv}$ , the following two equations

$$Adv_{\Pi, \mathcal{Adv}}^{fn-cor}(\lambda) = \Pr[FNCor_{\Pi}^{\mathcal{Adv}}(\lambda) = 1] \quad (19)$$

$$Adv_{\Pi, \mathcal{Adv}}^{fp-cor}(\lambda) = \Pr[FPCor_{\Pi}^{\mathcal{Adv}}(\lambda) = 1] \quad (20)$$

are negligible, where  $FNCor_{\Pi}^{\mathcal{Adv}}(\lambda)$  and  $FPCor_{\Pi}^{\mathcal{Adv}}(\lambda)$  are Experiment 1 and Experiment 2 respectively, as follows.

---

**Experiment 1:  $FNCor_{\Pi}^{\mathcal{Adv}}(\lambda)$ .**

---

- 1:  $k = (K_{CS}, K_{RP}) \leftarrow KeyGen(1^\lambda)$
  - 2:  $\mathcal{R} = \{R_1, \dots, R_{|\mathcal{R}|}\} \leftarrow \mathcal{Adv}(1^\lambda)$
  - 3:  $Obm \leftarrow ObMapGen(K_{RP}, \mathcal{R})$
  - 4:  $\mathcal{W} = \{W_1, W_2, \dots, W_\ell\} \leftarrow \mathcal{Adv}(1^\lambda)$
  - 5:  $\mathcal{T} = \{T_1, T_2, \dots, T_\ell\} \leftarrow RaTokenGen(K_{CS}, \mathcal{W})$
  - 6:  $Result \leftarrow Match(Obm, \mathcal{T})$
  - 7: if  $R_{i_0} = W_{j_0}$  (for some  $i_0$  and  $j_0$ ) and  $Result = 0$ , then  $b = 1$
  - 8: return  $b$
- 

---

**Experiment 2:  $FPCor_{\Pi}^{\mathcal{Adv}}(\lambda)$ .**

---

- 1:  $k = (K_{CS}, K_{RP}) \leftarrow KeyGen(1^\lambda)$
  - 2:  $\mathcal{R} = \{R_1, \dots, R_{|\mathcal{R}|}\} \leftarrow \mathcal{Adv}(1^\lambda)$
  - 3:  $Obm \leftarrow ObMapGen(K_{RP}, \mathcal{R})$
  - 4:  $\mathcal{W} = \{W_1, W_2, \dots, W_\ell\} \leftarrow \mathcal{Adv}(1^\lambda)$
  - 5:  $\mathcal{T} = \{T_1, T_2, \dots, T_\ell\} \leftarrow RaTokenGen(K_{CS}, \mathcal{W})$
  - 6:  $Result \leftarrow Match(Obm, \mathcal{T})$
  - 7: if  $R_{i_0} \neq W_{j_0}$  (for some  $i_0$  and  $j_0$ ) and  $Result = 1$ , then  $b = 1$
  - 8: return  $b$
- 

**Definition 5 ( $\mathcal{L}$ -secure Against Adaptive Attacks):** An encrypted data malicious detection scheme  $\Pi$  is  $\mathcal{L}$ -secure against adaptive attacks if, for all efficient adversary  $\mathcal{A}$ , there exists an efficient simulator  $\mathcal{S}$ , such that

$$Adv_{\Pi, \mathcal{A}, \mathcal{S}}^{adap}(\lambda) = |\Pr[Real_{\Pi}^{\mathcal{A}}(\lambda) = 1] - \Pr[Ideal_{\mathcal{L}, \mathcal{S}}^{\mathcal{A}}(\lambda) = 1]| \quad (21)$$

is negligible, where the games  $Real$  and  $Ideal$  are Game 1 and Game 2 respectively, as follows.

---

**Game 1:  $Real_{\Pi}^{\mathcal{A}}(\lambda)$ .**

---

- 1:  $k = (K_{CS}, K_{RP}) \leftarrow KeyGen(1^\lambda)$
  - 2:  $\mathcal{R} = \{R_1, \dots, R_{|\mathcal{R}|}\} \leftarrow \mathcal{A}(1^\lambda)$
  - 3:  $Obm \leftarrow ObMapGen(K_{RP}, \mathcal{R})$
  - 4:  $\mathcal{W} = \{W_1, W_2, \dots, W_\ell\} \leftarrow \mathcal{A}(1^\lambda)$
  - 5:  $\mathcal{T} = \{T_1, T_2, \dots, T_\ell\} \leftarrow RaTokenGen(K_{CS}, \mathcal{W})$
  - 6:  $b \leftarrow \mathcal{A}(Obm, \mathcal{T})$
- 

---

**Game 2:  $Ideal_{\mathcal{L}, \mathcal{S}}^{\mathcal{A}}(\lambda)$ .**

---

- 1:  $\mathcal{R} = \{R_1, \dots, R_{|\mathcal{R}|}\} \leftarrow \mathcal{A}(1^\lambda)$
  - 2:  $\widetilde{Obm} \leftarrow \mathcal{S}(\mathcal{L}(\mathcal{R}))$
  - 3:  $\mathcal{W} = \{W_1, W_2, \dots, W_\ell\} \leftarrow \mathcal{A}(1^\lambda)$
  - 4:  $\mathcal{T} = \{T_1, T_2, \dots, T_\ell\} \leftarrow RaTokenGen(K_{CS}, \mathcal{W})$
  - 5:  $b \leftarrow \mathcal{A}(\widetilde{Obm}, \mathcal{T})$
- 

In  $Real_{\Pi}^{\mathcal{A}}(\lambda)$ , the challenger calls  $KeyGen(1^\lambda)$  to output  $k = \{K_{CS}, K_{RP}\}$ . The adversary  $\mathcal{A}$  selects a rule  $R$  for the challenger to create an obfuscated map  $Obm$  via  $ObMapGen(K_{RP}, R)$ . Then  $\mathcal{A}$  adaptively transmits a polynomial number of strings, which is extracted from the data packet, i.e.,  $\{W_1, W_2, \dots, W_\ell\}$ . After that, the challenger responds to  $\mathcal{A}$  with the corresponding tokens  $T_1, T_2, \dots, T_\ell$ . Finally,  $\mathcal{A}$  outputs a decision bit  $b$  with inputting the tokens and  $Obm$ .

In  $Ideal_{\mathcal{L}, \mathcal{S}}^{\mathcal{A}}(\lambda)$ ,  $\mathcal{A}$  selects a rule set  $\mathcal{R}$ , and a simulator  $\mathcal{S}$  generates  $\widetilde{Obm}$  based on leakage information  $\mathcal{L}(\mathcal{R})$ . Then  $\mathcal{A}$  adaptively transmits a polynomial number of strings, which is extracted from the data packet, i.e.,  $\{W_1, W_2, \dots, W_\ell\}$ . After that,  $\mathcal{A}$  obtains  $T_1, T_2, \dots, T_\ell$  from an oracle, which runs  $RaTokenGen$  with inputting  $\{W_1, W_2, \dots, W_\ell\}$  and  $K_{CS}$ . Finally,  $\mathcal{A}$  outputs a bit  $b$ . The  $\mathcal{L}$  mentioned above is a state leakage function which describes what information is to be leaked from the inputs.

## B. Security Theorem

In our security analysis, we assume that the symmetric encryption algorithm  $(E_K, D_K)$  and the public key encryption algorithm  $(Enc_{PK}, Dec_{SP})$  are secure, which can be considered as pseudorandom permutations.

Here, we simplify our scheme  $\Pi_0$  as follows. Assumed that  $F : \mathcal{K}_s \times \{0, 1\}^{l_1} \rightarrow \{0, 1\}^{l_2}$ ,  $G : \mathcal{K}_l \times \{0, 1\}^{l_2} \rightarrow \{0, 1\}^{l_3}$  and  $f : \mathcal{K}_h \times \{0, 1\}^{l_4} \rightarrow \{0, 1\}^{l_5}$  are three pseudorandom functions; and  $H : \{0, 1\}^{l_6} \rightarrow \{0, 1\}^{l_7}$  is a hash function;

- $Setup(1^\lambda, KeyGen, KeyDist)$ : It generates a secure key tuple  $k = (K_s, K_h, K_l)$  between  $TR$  and  $DC$ , and sends key  $k$  to  $Miner_{RP}$  securely.
- $ObMGen(k, \mathcal{R})$ : It calculates obfuscated map  $Obm$  with key  $k$  and the set of rules  $\mathcal{R}$ . The form of each map entry is  $Obm[loc] = (id||p) \oplus s$ , where  $id$  is the random ID of each rule,  $p$  is the secret share used to restore the content options for a single rule  $con$ ,  $s = f(K_h, seg)$ ,  $loc = G(K_l, l)$  is the input of obfuscated map, and  $l = F(K_s, seg)$ . Here,  $con = H(id, q, Mod)||q||Mod$ , where  $Mod$  is the modifier collection of a single content option and  $q$  is the secret share used to restore rule header  $action$  for subsequent operation.

**Experiment 3:**  $FNCor_{\Pi_0}^{Adv}(\lambda)$  (resp.  $FPCor_{\Pi_0}^{Adv}(\lambda)$ ).

---

```

1:  $k = (K_{CS}, K_{RP}) \leftarrow KeyGen(1^\lambda)$ 
2:  $\mathcal{R} \leftarrow Adv(1^\lambda)$ 
3:  $Obm \leftarrow \emptyset$ 
4: for  $i \in \{1, \dots, |\mathcal{R}|\}$  do
5:  $id \leftarrow idGen(R_i)$ 
6:  $C_i = \{con_i: mod_i; action_i\} \leftarrow ruleSplit(R_i)$ 
7:  $Seg = \{seg_{i1}, \dots, seg_{i\sigma_i} : mod_i; action_i\} \leftarrow$ 
    $contentSplit(C_i)$ 
8: for  $\theta \in \{1, \dots, \sigma_i\}$  do
9:  $s \leftarrow f(K_h, seg_\theta)$ 
10:  $mapContent \leftarrow (id||p) \oplus s$ 
11:  $l \leftarrow F(K_s, seg_\theta)$ 
12:  $loc \leftarrow G(K_l, l)$ 
13:  $Obm \leftarrow Obm \cup (loc, mapContent)$ 
14:  $\mathcal{W} = \{W_1, W_2, \dots, W_\ell\} \leftarrow Adv(1^\lambda)$ 
15:  $\iota \in \{1, 2, \dots, \ell\}$  do  $T_\iota \leftarrow F(K_s, W_\iota) || f(K_h, W_\iota) || sp_\iota$ 
16:  $Result \leftarrow Match(Obm, \mathcal{T})$ , where  $\mathcal{T} = \{T_1, \dots, T_\ell\}$ 
17: if  $R_{i_0} = W_{j_0}$  (for some  $i_0$  and  $j_0$ ) and  $Result = 0$ , then
    $b = 1$  (17: if  $R_{i_0} \neq W_{j_0}$  (for some  $i_0$  and  $j_0$ ) and
    $Result = 1$ , then  $b = 1$ )
18: return  $b$ 

```

---

- $TokenGen(k, W_1, \dots, W_\ell)$ : It generates a set of randomized tokens  $\{T_1, \dots, T_\ell\}$ , where  $T_i = F(K_s, W_i) || f(K_h, W_i) || sp_i$ , where  $sp_i$  is the starting position of  $W_i$ .
- $Match(T_1, \dots, T_\ell, Obm)$ : Outputs a set of Ture/Flase values:  $\{Ind_1, \dots, Ind_\ell\}$ , where  $Ind_j = Ture$  if an action is restored; otherwise,  $Ind_j = Flase$ .

*Remark:* In the subsequent of this paper, each rule in  $\mathcal{R}$  contains only one content option and one response operation, and the length of the content option is the same as that of the plaintext token. Moreover, the length of response operation of all rules in  $\mathcal{R}$  is equal.

*Theorem 1:* The proposed Scheme  $\Pi_0$  is completeness if  $F$ ,  $G$  and  $f$  are secure PRF and the hash function  $H$  utilized in the proposed scheme is *collision-resistant*.

*Proof:*  $FNCor_{\Pi_0}^{Adv}(\lambda)$  and  $FPCor_{\Pi_0}^{Adv}(\lambda)$  are shown in Experiment 18. It is clear that

$$Adv_{\Pi_0, \mathcal{A}}^{fn-cor}(\lambda) = \Pr[FNCor_{\Pi_0}^{\mathcal{A}}(\lambda) = 1] = 0 \quad (22)$$

We define (1)  $BAD_F$  be the event that there is a  $W_{\iota_0}$  and  $seg_{\theta_0}$  ( $\theta_0 \in \{1, \dots, \sigma_{i_0}\}$  for some  $i_0$  in  $\{1, \dots, |\mathcal{R}|\}$ ) with  $W_{\iota_0} \neq seg_{\theta_0}$  such that  $F(K_s, W_{\iota_0}) = F(K_s, seg_{\theta_0})$ ; (2)  $BAD_f$  be the event that there is a  $W_{\iota_0}$  and  $seg_{\theta_0}$  ( $\theta_0 \in \{1, \dots, \sigma_{i_0}\}$  for some  $i_0$  in  $\{1, \dots, |\mathcal{R}|\}$ ) with  $W_{\iota_0} \neq seg_{\theta_0}$  such that  $f(K_h, W_{\iota_0}) = f(K_h, seg_{\theta_0})$ ; (3)  $BAD_G$  be the event that there is a  $W_{\iota_0}$  and  $seg_{\theta_0}$  ( $\theta_0 \in \{1, \dots, \sigma_{i_0}\}$  for some  $i_0$  in  $\{1, \dots, |\mathcal{R}|\}$ ) with  $W_{\iota_0} \neq seg_{\theta_0}$  such that  $G(K_l, F(K_s, W_{\iota_0})) = G(K_l, F(K_s, seg_{\theta_0}))$ ;  $BAD_H$  be the event that  $H(id_{i_0}, q', Mod_{i_0}) = H(id_{i_1}, q'', Mod_{i_1})$  with  $(id_{i_0}, q', Mod_{i_0}) \neq (id_{i_1}, q'', Mod_{i_1})$  for some  $id_{i_0}$   $id_{i_1}$ ,  $q'$ ,  $q''$ ,  $Mod_{i_0}$ , and  $Mod_{i_1}$  in obfuscated rule generation phase.

**Game 4:**  $Input(\mathcal{R}, W_0, \dots, W_\ell) // \mathbf{G}_0$ .

---

```

1:  $k = (K_s, K_h, K_l) \leftarrow Setup(1^\lambda)$ 
2:  $Obm \leftarrow \emptyset$ 
3: for  $\iota \in \{1, 2, \dots, \ell\}$  do
4:  $T_\iota \leftarrow F(K_s, W_\iota) || f(K_h, W_\iota) || sp_\iota$ 
5: for  $i \in \{1, \dots, |\mathcal{R}|\}$  do
6:  $id \leftarrow idGen(R_i)$ 
7:  $C_i = \{con_i: mod_i; action_i\} \leftarrow ruleSplit(R_i)$ 
8:  $Seg = \{seg_{i1}, \dots, seg_{i\sigma_i} : mod_i; action_i\} \leftarrow$ 
    $contentSplit(C_i)$ 
9: for  $\theta \in \{1, \dots, \sigma_i\}$  do
10:  $s \leftarrow f(K_h, seg_\theta)$ 
11:  $mapContent \leftarrow (id||p) \oplus s$ 
12:  $l \leftarrow F(K_s, seg_\theta)$ 
13:  $loc \leftarrow G(K_l, l)$ 
14:  $Obm \leftarrow Obm \cup (loc, mapContent)$ 
15: return  $b \leftarrow \mathcal{A}(Obm, T_1, \dots, T_\ell)$ 

```

---

Here,  $H$  can be regarded as a keying hash function, in where, the random ID  $id$  and random number  $q$  are used as a key.

$FPCor_{\Pi_0}^{\mathcal{A}}(\lambda) = 1$  happens only when one of the events  $BAD_F$ ,  $BAD_f$ ,  $BAD_G$ , and  $BAD_H$  happens. Note that, adversary  $\mathcal{A}$  does not know the keys  $(K_{CS}, K_{RP})$  in Scheme  $\Pi_0$ . Therefore, from the fact that  $F$ ,  $G$  and  $f$  are secure PRF and the hash function  $H$  used in the scheme is *collision-resistant*, we have the probability of  $BAD_F$ ,  $BAD_f$ ,  $BAD_G$ , and  $BAD_H$  are negligible.

$$\begin{aligned} Adv_{\Pi_0, \mathcal{A}}^{fp-cor}(\lambda) &= \Pr[FPCor_{\Pi_0}^{\mathcal{A}}(\lambda) = 1] \\ &\leq \Pr[BAD_F] + \Pr[BAD_f] + \Pr[BAD_G] + \Pr[BAD_H] \\ &\leq negl(\lambda) \end{aligned} \quad (23)$$

*Theorem 2:* Scheme  $\Pi_0$  is  $\mathcal{L}$ -secure against adaptive attacks if  $F$ ,  $G$  and  $f$  are secure PRF.

*Proof:* We consider games  $\mathbf{G}_0$ ,  $\mathbf{G}_1$ , and  $\mathbf{G}_2$ , in which,  $\mathbf{G}_0$  is used to compute a distribution identical to  $Real_{\Pi_0}^{\mathcal{A}}(\lambda)$ , and  $\mathbf{G}_2$  is used to compute a distribution identical to  $Ideal_{\mathcal{L}, \mathcal{S}}^{\mathcal{A}}(\lambda)$ .

The first game  $\mathbf{G}_0$  computes  $Obm$  and randomized tokens  $T_i$  as specified in the adaptive game. It selects a key set  $k = (K_s, K_h, K_l)$  via  $Setup(1^\lambda)$ . For each  $W_i$ ,  $\mathbf{G}_0$  computes  $F(K_s, W_i) || f(K_h, W_i) || sp_i$  as  $T_i$ . Moreover, it computes obfuscated map  $Obm$  using key set  $k$  for each original rule. Accordingly, for any p.p.t. adversary  $\mathcal{A}$

$$\Pr[G_0] = \Pr[Real_{\Pi_0}^{\mathcal{A}}(\lambda) = 1] \quad (24)$$

In Game  $\mathbf{G}_1$ , the outputs of tree random oracles  $RF(\emptyset, \cdot)$ ,  $RF'(\emptyset, \cdot)$  and  $RF''(\emptyset, \cdot)$  are used to replace that of  $F(K_s, \cdot)$ ,  $f(K_h, \cdot)$  and  $G(K_l, \cdot)$  in  $\mathbf{G}_0$ , respectively. This means that all of the  $T_i$  are uniform and independent strings.

Let Game  $G'_0$  be the game which utilizes  $RF(\emptyset, \cdot)$  to replace  $F(K_s, \cdot)$  in Game  $G_0$ , and let Game  $G'_1$  be the game which uses  $RF'(\emptyset, \cdot)$  to replace  $f(K_h, \cdot)$  in Game  $G'_0$ . Then, from that fact

**Game 5:**  $Input(\mathcal{R}, W_0, \dots, W_\ell) // \mathbf{G}_1$ .

---

```

1:  $k = (K_s, K_h, K_l) \leftarrow Setup(1^\lambda)$ 
2:  $Obm \leftarrow \emptyset$ 
3: for  $\iota \in \{1, 2, \dots, \ell\}$  do
   $T_\iota \leftarrow RF(\emptyset, W_\iota) \| RF'(\emptyset, W_\iota) \| sp_\iota$ 
4: for  $i \in \{1, \dots, |\mathcal{R}|\}$  do
5:  $id \leftarrow idGen(R_i)$ 
6:  $C_i = \{con_i : mod_i; action_i\} \leftarrow ruleSplit(R_i)$ 
7:  $S_{eg} = \{seg_{i1}, \dots, seg_{i\sigma_i} : mod_i; action_i\} \leftarrow$ 
   $contentSplit(C_i)$ 
8: for  $\theta \in \{1, \dots, \sigma_i\}$  do
9:  $s \leftarrow RF'(\emptyset, seg_\theta)$ 
10:  $mapContent \leftarrow (id \| p) \oplus s$ 
11:  $l \leftarrow RF(\emptyset, seg_\theta)$ 
12:  $loc \leftarrow RF''(\emptyset, l)$ 
13:  $Obm \leftarrow Obm \cup (loc, mapContent)$ 
14: return  $b \leftarrow \mathcal{A}(Obm, T_1, \dots, T_\ell)$ 

```

---

**Game 6:**  $Input(\mathcal{R}, W_0, \dots, W_\ell) // \mathbf{G}_2$ 


---

```

1:  $k = (K_s, K_h, K_l) \leftarrow Setup(1^\lambda)$ 
2:  $Obm \leftarrow \emptyset$ 
3: for  $\iota \in \{1, 2, \dots, \ell\}$  do
   $T_\iota \leftarrow F(K_s, W_\iota) \| f(K_h, W_\iota) \| sp_\iota$ 
4: for  $i \in \{1, \dots, |\mathcal{R}|\}$  do
5:  $id \leftarrow idGen(R_i)$ 
6:  $C_i = \{con_i : mod_i; action_i\} \leftarrow ruleSplit(R_i)$ 
7:  $S_{eg} = \{seg_{i1}, \dots, seg_{i\sigma_i} : mod_i; action_i\} \leftarrow$ 
   $contentSplit(C_i)$ 
8: for  $\theta \in \{1, \dots, \sigma_i\}$  do
9:  $s \leftarrow RF'(\emptyset, seg_\theta)$ 
10:  $mapContent \leftarrow (id \| p) \oplus s$ 
11:  $l \leftarrow RF(\emptyset, seg_\theta)$ 
12:  $loc \leftarrow RF''(\emptyset, l)$ 
13:  $Obm \leftarrow Obm \cup (loc, mapContent)$ 
14: return  $b \leftarrow \mathcal{A}(Obm, T_1, \dots, T_\ell)$ 

```

---

that  $F$ ,  $f$  and  $G$  are pseudorandom functions, we can obtain that

$$|\Pr[G_0] - \Pr[G'_0]| \leq \text{negl}(\lambda) \quad (25)$$

$$|\Pr[G'_0] - \Pr[G'_1]| \leq \text{negl}(\lambda) \quad (26)$$

$$|\Pr[G'_1] - \Pr[G_1]| \leq \text{negl}(\lambda) \quad (27)$$

for any efficient adversary  $\mathcal{A}$ . As a result, for any efficient adversary  $\mathcal{A}$ , we can obtain that

$$|\Pr[G_0] - \Pr[G_1]| \leq \text{negl}(\lambda) \quad (28)$$

Note we consider Game  $G_2$ . In this game, random tokens  $T_0, \dots, T_\ell$  are generated by  $F(K_s, \cdot)$  and  $f(K_h, \cdot)$  with inputting  $W_0, \dots, W_\ell$ .

Note that, if  $F(K_s, \cdot)$  and  $f(K_h, \cdot)$  are pseudo-random functions, then  $\widehat{F}(K_s, K_h, \cdot) (= (F(K_s, \cdot), f(K_h, \cdot)))$  is also a pseudo-random function. Then, we claim that,

$$|\Pr[G_1] - \Pr[G_2]| \leq \text{negl}(\lambda) \quad (29)$$

Actually, if there is an efficient adversary  $\mathcal{A}_1$  such that

$$|\Pr[G_1] - \Pr[G_2]| > \text{negl}(\lambda) \quad (30)$$

then an efficient adversary  $\mathcal{B}_1$  exists to satisfy

$$Adv_{\widehat{F}, \mathcal{B}_1}^{\text{pdf}}(\lambda) > \text{negl}(\lambda) \quad (31)$$

Here, the adversary  $\mathcal{B}_1$  has access to an oracle  $RF(\cdot, \cdot)$ ,  $RF'(\cdot, \cdot)$ , and  $RF''(\cdot, \cdot)$ .  $\mathcal{B}_1$  runs  $\mathcal{A}_1$  to get  $(\mathcal{R}, W_0, \dots, W_\ell)$ , and then uses output of  $\mathcal{A}_1$  as its own.

An effective simulator  $\mathcal{S}$  takes the information output by the leakage function  $\mathcal{L}$  as the initial input. Specifically,  $\mathcal{L}(\mathcal{R}) = \{|loc|, |mapContent|, |e|, \{|d|\}_{|e|}\}$ , where  $|loc|$  is the length of the input of the obfuscated map,  $|mapContent|$  is the length of the value in the obfuscated map,  $|e|$  is the number of entries in the obfuscated map, and  $\{|d|\}_{|e|}$  is the size of the corresponding bucket for each entry.  $\mathcal{S}$  simulates and constructs the obfuscated map  $\widetilde{Obm}$  through the leakage function. Compared  $\mathcal{S}$  with  $G_2$ ,

the keys and values in  $\widetilde{Obm}$  are random independent strings so that

$$\Pr[G_2] = \Pr[Ideal_{\mathcal{L}, S}^{\mathcal{A}}(\lambda) = 1] \quad (32)$$

Accordingly, we have

$$\begin{aligned} Adv_{\Pi_0, \mathcal{A}, S}^{\text{adap}}(\lambda) &= |\Pr[Real_{\Pi_0}^{\mathcal{A}}(\lambda) = 1] - \Pr[Ideal_{\mathcal{L}, S}^{\mathcal{A}}(\lambda) = 1]| \\ &= |\Pr[G_0] - \Pr[G_2]| \\ &= |\Pr[G_0] - \Pr[G_1]| + |\Pr[G_1] - \Pr[G_2]| \\ &\leq \text{negl}(\lambda) \end{aligned} \quad (33)$$

Consequently, if  $F$ ,  $f$  and  $G$  are secure PRFs, the scheme PESCO  $\Pi_0$  is correct and  $\mathcal{L}$ -secure against adaptive attacks.  $\blacksquare$

## VII. PERFORMANCE EVALUATION

In this section, we conduct extensive experiments to discuss the performance of our scheme.

### A. Experimental Settings

In our experiments, a desktop computer (AMD Ryzen 7 4800H) is used with 64-bit Linux operating system, Radeon Graphics 2.90 GHz and 16.0 GB of RAM. Moreover, all experiments are conducted with the libraries of Python. The pseudo-random functions  $F$ ,  $G$ ,  $f$  and  $H$  are implemented by SHA1, SHA224, SHA256 and MD5 respectively. Both of them are built on the Python library *Crypto*. Moreover, the size of the function code of a smart contract is set to 5~110 KB. The size of the sliding window is determined according to the length of the shortest ‘‘content’’ option in all selected rules. In our experiments, the size of the sliding window is set to 2 bytes, as the shortest length of ‘‘content’’ option of Snort rule set used in our experiments is 2 bytes.



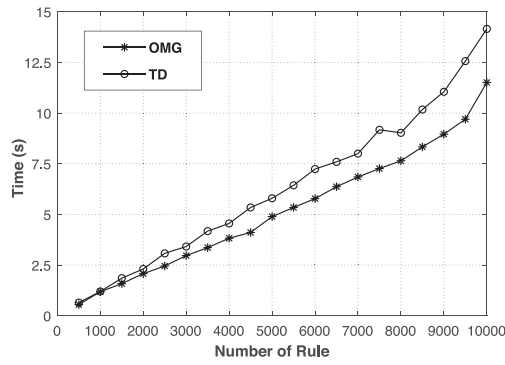


Fig. 3. Time consumption versus rule number.

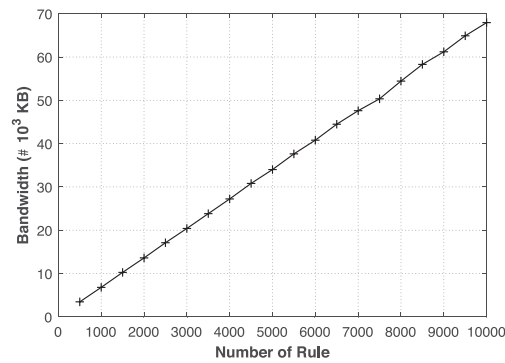


Fig. 4. Communication cost of obfuscated map versus rule number.

### B. Exploration of System Parameters

We now discuss the impact of system parameters on computation cost and communication cost of the proposed scheme. We first conduct an experiment to discuss the effect of the size of Snort rule set  $|\mathcal{R}|$  and the size of smart contract file  $|M|$  on the overhead of the proposed scheme. In our experiment, the size of rule set is chosen from 500 to 10000 with step 500, and the size of smart contract  $M$  is set as 15 KB. The time cost of the obfuscated map generation and token detection against the varied size of original rule set is first studied. As shown in Fig. 3, the time cost of both obfuscated map generation and token detection increases approximately linearly with the increase in the size of rule set. Then, the communication overhead of the proposed method versus the varied size of original rule set is discussed. From Fig. 4, it can be seen that, the communication overhead of the proposed scheme the bandwidth consumption of the generated obfuscated map increases linearly as the cardinality of rule set increases. Specifically, when the size of the smart contract is fixed at 15 KB, the time cost is around 112 ms and the bandwidth consumption is 5316 KB in the randomized token generation phase.

Then, the overhead of the randomized token generation and token detection under different size of smart contract (from 5 KB to 110 KB with step 5 KB) is discussed. Fig. 5 plots the time cost of two main phases under different size of smart contract. It can be seen that, the time cost is mainly concentrated on the token detection phase. The reason is that, with the increase of the

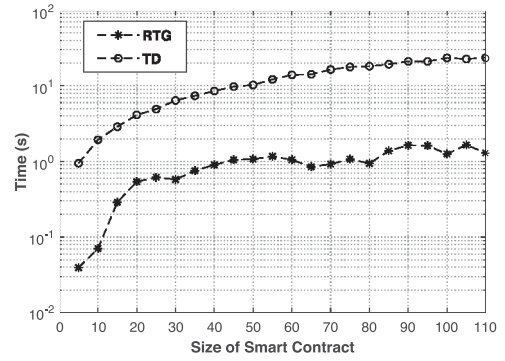


Fig. 5. Time consumption versus smart contract size.

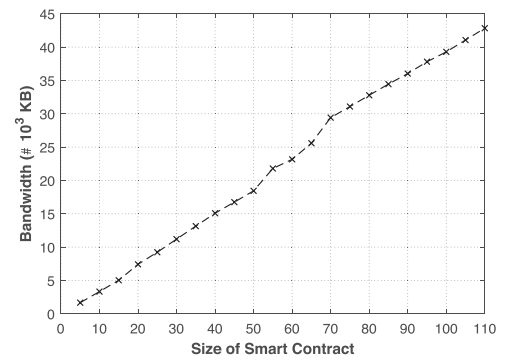


Fig. 6. Communication cost of token set versus smart contract size.

size of smart contract file, the number of the tokens increases, which further leads to the increase of time consumption of token detection. In Fig. 6, the communication overhead of two main phases under different size of smart contract is presented. From Fig. 6, we have that the communication overhead of randomized token generation increases with the increase of smart contract size linearly. Specifically, when the cardinality of the rule set is fixed at 3000, the time consumption of the obfuscated map generation phase is 1927 ms and the bandwidth consumption is 14938 KB.

### C. Performance Analysis

An experiment is carried out to discuss the overall performance of our scheme by taking the size of smart contracts from 5 KB to 110 KB with step 5 KB. In this experiment, four rule sets containing 4000 rules are selected. These sets have no intersection of rules and are all from suricata-5.0. It should be noted that the main difference between these rule sets is the length of the shortest “content” options, which are 2, 3, 4 and 5 bytes respectively. According to the shortest length from small to large, the four sets are defined as  $ruleset_1$ ,  $ruleset_2$ ,  $ruleset_3$  and  $ruleset_4$  respectively.

Fig. 7(a) shows the time cost of randomized token generation phase under different sizes of smart contract. From Fig. 7(a), we have the result that for four rule sets with different sliding windows (i.e. 2, 3, 4 and 5), the time consumption decreases with the increase of the size of sliding window. Meanwhile, for

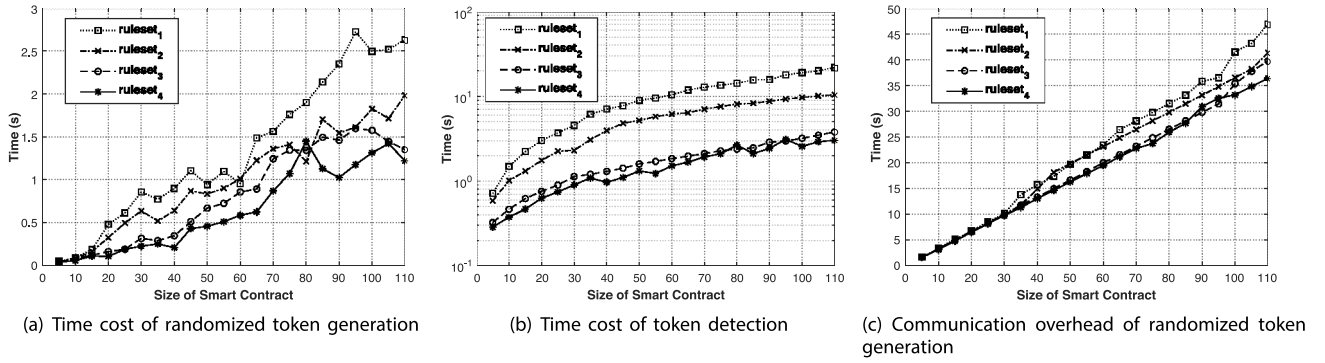


Fig. 7. Time and communication cost of the proposed scheme versus varying size of smart contract in different rule set.

TABLE III  
TIME CONSUMPTION AND COMMUNICATION COST OF OBFUSCATED MAP  
GENERATION PHASE

Num of Rule Set	Time (ms)	Bandwidth (KB)
<i>ruleset<sub>1</sub></i>	1974	8,488
<i>ruleset<sub>2</sub></i>	2296	14,623
<i>ruleset<sub>3</sub></i>	2631	16,042
<i>ruleset<sub>4</sub></i>	3132	20,125

different rule sets, the time cost at this phase shows an upward trend as the size of the smart contract increases. Fig. 7(b) shows the time cost of obfuscated map generation phase. It can be seen that, the time cost of this phase raises with increasing the size of the sliding window or the size of smart contract. Fig. 7(c) shows the communication overhead of randomized token generation phase of the proposed scheme by changing the size of sliding window. As shown in this figure, with the increase of sliding window size, the communication overhead of randomized token generation phase decreases. The reason is that, for the larger sliding window, the fewer randomized tokens are generated, which reduces the bandwidth. Meanwhile, as the increase of smart contract size, the bandwidth corresponding to each experimental rule set in randomized token generation is increasing.

As shown in Table III, since the rules and sliding windows in the three rule sets are fixed, the time cost and communication overhead corresponding to each rule set in the obfuscated map generation phase is constant. Specifically, the communication cost corresponding to *ruleset<sub>1</sub>*, *ruleset<sub>2</sub>*, *ruleset<sub>3</sub>* and *ruleset<sub>4</sub>* is 8,488 KB, 14,623 KB, 16,042 KB and 20,125 KB, respectively.

#### D. Comparison With Existing Schemes

We conduct an experiment to compare the time and communication overhead of our scheme with existing schemes (i.e., PrivDPI [32] and PETI [34]). In this experiment, the rule length *ruleLen* of the proposed scheme and PETI [34] are set to 8 bytes and 16 bytes, and the rule length *ruleLen* of PrivDPI [32] is set to 8 bytes, respectively; the number of rules is 6000 for *ruleLen* = 8 bytes, and 3000 for *ruleLen* = 16 bytes; and the size of the smart contract is set to 19.2 KB. Since the ORG phase

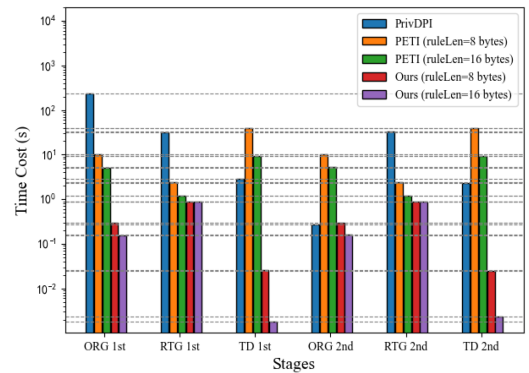


Fig. 8. Comparison of the time consumption of the proposed scheme with existing schemes.

of PrivDPI in the subsequent session can reuse the obfuscated rules in the first session to reduce time and communication consumption [32], we consider two consecutive sessions in anomaly detection of encrypted smart contract to fairly compare the performance of the above schemes.

Fig. 8 shows the comparison of the time consumption of the proposed scheme with existing schemes in two consecutive sessions. It can be found that, (1) the computational overhead of our scheme is significantly smaller than that of PrivDPI and PETI in all three phases of two sessions; and (2) there is not much difference between the first and second execution of the proposed scheme, while the computational overhead of the ORG phase during the second session of PrivDPI is significantly reduced due to the fact that PrivDPI utilizes the results of the first session for simplicity of computation.

Fig. 9 plots the comparison of the communication cost of the proposed scheme with existing schemes in two consecutive sessions. It can be seen that, (1) in the ORG phase, the communication cost of the proposed scheme and PETI is almost the same, which are less than that of PrivDPI; (2) in the TD phase, there is no communication cost for all three schemes; and (3) the proposed scheme has a considerable advantage over privDPI and PETI in the RTG phase of the first session, and the communication overhead of PrivDPI in the RTG phase of the second session is extremely small due to the utilization of the results of the first session.

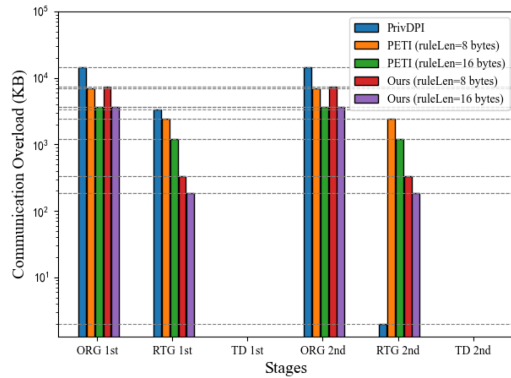


Fig. 9. Comparison of the communication cost of the proposed scheme with existing schemes.

### E. Discussion

In our system, it is assumed that the miners  $Miner_{DE}$  and  $Miner_{PR}$  are semi-honest. A feasible approach based on the admission and punishment mechanism to eliminate this assumption is as follows. Each miner participating in the task of *Obfuscated Rule Generation* and *Malicious Detection* needs to have a certain access mechanism and pay a certain amount of deposit for the penalty in case of dishonesty. When dishonest event occurs, data center first detects the honesty of  $Miner_{PR}$ . Note that, in the proposed protocol,  $Miner_{PR}$  needs to process the matching rules into obfuscated map, and store it and the corresponding signature on the RP chain. If  $Miner_{PR}$  is a dishonest node and generates a fake obfuscated map for a transaction, the data center can generate a real obfuscated map using the real rule set and key, and then compare it with the fake obfuscated map on the rule processing chain, so as to find out the evidence of dishonesty of  $Miner_{PR}$ . Then, data center verifies the honesty of  $Miner_{DE}$ . If node  $Miner_{PR}$  is honest while  $Miner_{DE}$  is dishonest in the detection process. The data center can extract the electronic contract and decrypt the transaction content; and then use the real rule set to detect the decrypted electronic contract. If the detection result is inconsistent with the system detection result, then  $Miner_{DE}$  is dishonest. When a miner acts dishonestly, his qualification as a miner may be revoked or he may be fined depending on the severity of the behavior.

## VIII. CONCLUSION

In this paper, a blockchain-based data trading platform has been considered, in which, data users can purchase data set and computing power through encrypted smart contracts. A privacy-preserving encrypted smart contract detection system has been proposed in the blockchain-based data trading platform with the help of two kinds of miners. One kind of miner acts as a rule processor to generate an obfuscated map with the original open-source malicious rule set; and another kind of miner acts as a detector to perform malicious inspection by inputting the obfuscated map and the randomized tokens of smart contract. We have defined the security syntax of encrypted smart contract inspection, and proved that the proposed scheme is

$\mathcal{L}$ -secure against adaptive attacks. Experimental results demonstrate that our scheme can achieve malicious code detection with high detection accuracy, low time cost and low communication overhead. The assumption that all miners are semi-honest in the proposed scheme is relatively strong, which weakens the application in the practical peer-to-peer network environment. In the future, we would like to continue working on this topic and propose a privacy-preserving anomaly detection scheme for encrypted smart contracts under the dishonest miner model.

## REFERENCES

- [1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, 2008, Art. no. 21260.
- [2] Y. Liu, J. Liu, Q. Wu, H. Yu, Y. Hei, and Z. Zhou, "SSHC: A secure and scalable hybrid consensus protocol for sharding blockchains with a formal security framework," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 2070–2088, May/Jun. 2022.
- [3] M. Song, Z. Hua, Y. Zheng, H. Huang, and X. Jia, "Blockchain-based deduplication and integrity auditing over encrypted cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 6, pp. 4928–4945, Nov./Dec. 2023, doi: [10.1109/TDSC.2023.3237221](https://doi.org/10.1109/TDSC.2023.3237221).
- [4] Y. Chen, J. Zhao, Y. Wu, J. Huang, and X. Shen, "QoE-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 769–784, Jan. 2024, doi: [10.1109/TMC.2022.3223119](https://doi.org/10.1109/TMC.2022.3223119).
- [5] N. Zhang, P. Yang, J. Ren, D. Chen, L. Yu, and X. Shen, "Synergy of big data and 5G wireless networks: Opportunities, approaches, and challenges," *IEEE Wirel. Commun.*, vol. 25, no. 1, pp. 12–18, Feb. 2018.
- [6] W. Wu et al., "AI-native network slicing for 6G networks," *IEEE Wirel. Commun.*, vol. 29, no. 1, pp. 96–103, Feb. 2022.
- [7] Y. Jiang and Y. Zhong, "IIoT data sharing based on blockchain: A multileader multifollower stackelberg game approach," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4396–4410, Mar. 2022.
- [8] M. Kumar et al., "ANAF-IoMT: A novel architectural framework for IoMT-Enabled smart healthcare system by enhancing security based on RECC-VC," *IEEE Trans. Ind. Inform.*, vol. 18, no. 12, pp. 8936–8943, Dec. 2022.
- [9] X. Li et al., "Big data analysis of the internet of things in the digital twins of smart city based on deep learning," *Future Gener. Comput. Syst.*, vol. 128, pp. 167–177, 2022.
- [10] W. Dai, C. Dai, K. -K. R. Choo, C. Cui, D. Zou, and H. Jin, "SDTE: A secure blockchain-based data trading ecosystem," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 725–737, 2019.
- [11] D. Chen, S. Jiang, N. Zhang, L. Liu, and K. -K. R. Choo, "On message authentication channel capacity over a wiretap channel," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 3107–3122, 2022.
- [12] N. Zhang, N. Lu, N. Cheng, J. W. Mark, and X. S. Shen, "Cooperative spectrum access towards secure information transfer for CRNs," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 11, pp. 2453–2464, Nov. 2013.
- [13] S. Sedkaoui and M. Khelifaoui, *Sharing Economy and Big Data Analytics*, Hoboken, NJ, USA: Wiley, 2020.
- [14] X. Yang, R. Lu, J. Shao, X. Tang, and A. A. Ghorbani, "Achieving efficient secure deduplication with user-defined access control in cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 591–606, Jan./Feb. 2022.
- [15] Y. Li, L. Li, Y. Zhao, N. Guizani, Y. Yu, and X. Du, "Toward decentralized fair data trading based on blockchain," *IEEE Netw.*, vol. 35, no. 1, pp. 304–310, Jan./Feb. 2021.
- [16] L. D. Nguyen, I. Leyva-Mayorga, A. N. Lewis, and P. Popovski, "Modeling and analysis of data trading on blockchain-based market in IoT networks," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6487–6497, Apr. 2021.
- [17] D. Zhang, J. Le, X. Lei, T. Xiang, and X. Liao, "Secure redactable blockchain with dynamic support," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: [10.1109/TDSC.2023.3261343](https://doi.org/10.1109/TDSC.2023.3261343).
- [18] J. Zhang, Y. Ye, W. Wu, and X. Luo, "Boros: Secure and efficient off-blockchain transactions via payment channel hub," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 407–421, Jan./Feb. 2023.
- [19] C. Li et al., "Blockchain-based data trading in edge-cloud computing environment," *Inf. Process. Manage.*, vol. 59, no. 1, pp. 1–22, 2022.



- [20] B. An, M. Xiao, A. Liu, Y. Xu, X. Zhang, and Q. Li, "Secure crowdsensed data trading based on blockchain," *IEEE Trans. Mobile Comput.*, vol. 22, no. 3, pp. 1763–1778, Mar. 2023.
- [21] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. [Online]. Available: <http://firstmonday.org/ojs/index.php/fm/article/view/548/469>
- [22] L. Ale, N. Zhang, H. Wu, D. Chen, and T. Han, "Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5520–5530, Jan. 2019.
- [23] D. Chen et al., "MAGLeak: A learning-based side-channel attack for password recognition with multiple sensors in IIoT environment," *IEEE Trans. Ind. Inform.*, vol. 18, no. 1, pp. 467–476, Jan. 2022.
- [24] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 620–636, Jan./Feb. 2023.
- [25] A. Kosba et al., "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy*, 2016, pp. 839–858.
- [26] S. Steffen, B. Bichsel, M. Gersbach, P. Tsankov, and M. Vechev, "zkay: Specifying and enforcing data privacy in smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1759–1776.
- [27] P. Pal and K. Sudharsana, "WiP: Criminal smart contract for private key theft in end to end encrypted applications," in *Proc. Int. Conf. Inf. Syst. Secur.*, 2019, pp. 21–32.
- [28] G. Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.
- [29] Y. Wu, S. Tang, B. Zhao, and Z. Peng, "BPTM: Blockchain-based privacy-preserving task matching in crowdsourcing," *IEEE Access*, vol. 7, pp. 45605–45617, 2019.
- [30] J. Shi and Y. Zhang, "Privacy-preserving network functionality outsourcing," 2015, *arXiv:1502.00389*.
- [31] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep packet inspection over encrypted traffic," in *Proc. SIGCOMM Conf.*, London, U.K., 2015, pp. 213–226.
- [32] J. Ning, G. S. Poh, J. Loh, J. Chia, and E. Chang, "PrivDPI: Privacy-preserving encrypted traffic inspection with reusable obfuscated rules," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, London, U.K., 2019, pp. 1657–1670.
- [33] X. Yuan, X. Wang, J. Lin, and C. Wang, "Privacy-preserving deep packet inspection in outsourced middleboxes," in *Proc. IEEE 35th Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
- [34] D. Chen et al., "Privacy-preserving encrypted traffic inspection with symmetric cryptographic techniques in IoT," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 17265–17279, Sep. 2022.
- [35] A. C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci.*, 1986, pp. 162–167.
- [36] M. Naor and B. Pinkas, "Oblivious transfer with adaptive queries," in *Proc. Annu. Int. Cryptol. Conf.*, 1999, pp. 573–590.
- [37] D. Yuan, Q. Li, G. Li, Q. Wang, and K. Ren, "PriRadar: A privacy-preserving framework for spatial crowdsourcing," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 299–314, 2020.
- [38] M. Deng, K. Zhang, P. Wu, M. Wen, and J. Ning, "DCDPI: Dynamic and continuous deep packet inspection in secure outsourced middleboxes," *IEEE Trans. Cloud Comput.*, vol. 11, no. 4, pp. 3510–3524, Fourth Quarter 2023, doi: [10.1109/TCC.2023.3293134](https://doi.org/10.1109/TCC.2023.3293134).
- [39] S. Gao et al., "Privacy-preserving industrial control system anomaly detection platform," *Secur. Commun. Netw.*, vol. 2023, 2023, Art. no. 7010155.
- [40] C. Zhang, M. Zhao, L. Zhu, W. Zhang, T. Wu, and J. Ni, "FRUIT: A blockchain-based efficient and privacy-preserving quality-aware incentive scheme," *IEEE J. on Sel. Areas Commun.*, vol. 40, no. 12, pp. 3343–3357, Dec. 2022.
- [41] C. Sendner et al., "Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–18.
- [42] N. Ivanov and Q. Yan, "TxT: Real-time transaction encapsulation for ethereum smart contracts," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1141–1155, 2023.
- [43] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, Oakland, CA, USA, 2000, pp. 44–55.
- [44] R. Curtmola et al., "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.
- [45] D. Cash et al., "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2014, pp. 1–16.
- [46] A. Joux, "A one round protocol for tripartite Diffie-Hellman," *J. Cryptol.*, vol. 17, no. 4, pp. 263–276, 2004.
- [47] C. Lan et al., "Embarc: Securely outsourcing middleboxes to the cloud," in *Proc. 13th USENIX Symp. Networked Syst. Des. Implementation*, 2016, pp. 255–273.
- [48] S. Canard et al., "BlindIDS: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 561–574.
- [49] M. Bellare and P. Rogaway, "The exact security of digital signatures-How to sign with RSA and Rabin," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Heidelberg, 1996, pp. 399–416.



**Dajiang Chen** (Member, IEEE) received the PhD degree in information and communication engineering from the University of Electronic Science and Technology of China, in 2014. He is currently an associate professor with the School of Information and Software Engineering, University of Electronic Science and Technology of China (UESTC). He was a postdoc research fellow with the BBCR group, Department of Electrical and Computer Engineering, University of Waterloo, Canada, from 2015 to 2017. He served as the workshop chair for BDEC-SmartCity'19 (in conjunction with IEEE WiMob 2019). He also served as a Technical Program Committee Member for IEEE Globecom, IEEE ICC, and IEEE VTC. His current research interests include physical layer security, secure channel coding, and machine learning and its applications in wireless network security and wireless communications.



**Zeyu Liao** received the BS degree in software engineering from the School of information and Software Engineering, University of Electronic Science and Technology of China, in 2021. He is currently working toward the postgraduate degree with the School of Information and Software Engineering, University of Electronic Science and Technology of China. His research interests include security and privacy protection in wireless networks.



**Ruidong Chen** received the PhD degree in information and communication engineering from the University of Electronic Science and Technology of China, in 2019. He is currently an associate research fellow with the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include blockchain, and security and privacy protection in information systems, software, networking, and databases.



**Hao Wang** received the BS degree in software engineering from the School of information and software Engineering, University of Electronic Science and Technology of China, in 2019. He is currently working toward the postgraduate degree with the School of Information and Software Engineering, University of Electronic Science and Technology of China. His research interests include security and privacy protection in different application scenarios of Wireless networks (e.g., IoT, Edge Computing, and Internet of Vehicles).

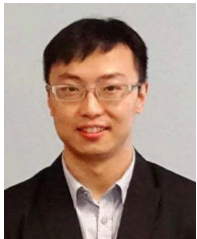


**Chong Yu** (Graduate Student Member, IEEE) received the BSc degree in communication engineering and the MSc degree in communication and information system from Northeastern University, Shenyang, China, in 2015 and 2017, respectively. She is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, Omaha, NE, USA. Her research interests include intelligent Internet of Things, cybersecurity, intelligent vehicle, cloud/edge computing, and machine learning.



**Kuan Zhang** (Member, IEEE) received the BSc degree in communication engineering and the MSc degree in computer applied technology from Northeastern University, China, in 2009 and 2011, respectively, and the PhD degree in electrical and computer engineering from the University of Waterloo, in 2016. He has been an assistant professor with the Department of Electrical Communication Engineering, University of Nebraska-Lincoln, since 2017. He was also a postdoctoral fellow with the Broadband Communications Research (BBCR) group, University of Waterloo from

2016 to 2017. His research interests include security and privacy for mobile social networks, cloud/edge computing, and cyber physical systems.



**Ning Zhang** (Senior Member, IEEE) received the PhD degree in electrical and computer engineering from the University of Waterloo, Canada, in 2015. He is an associate professor with the Department of Electrical and Computer Engineering, University of Windsor, Canada. After that, he was a postdoc research fellow with the University of Waterloo and University of Toronto, Canada, respectively. His research interests include connected vehicles, mobile edge computing, wireless networking, and machine learning. He is a Highly Cited Researcher (Web of

Science). He received an NSERC PDF award in 2015 and 6 Best Paper Awards from IEEE Globecom in 2014, IEEE WCSP in 2015, IEEE ICC in 2019, IEEE ICC in 2019, IEEE Technical Committee on Transmission Access and Optical Systems in 2019, and *Journal of Communications and Information Networks* in 2018, respectively. He serves as an associate editor of *IEEE Internet of Things Journal*, *IEEE Transactions on Cognitive Communications and Networking*, and *IEEE Systems Journal*; and a guest editor of several international journals, such as *IEEE Wireless Communications*, *IEEE Transactions on Industrial Informatics*, and *IEEE Transactions on Cognitive Communications and Networking*.



**Xuemin (Sherman) Shen** (Fellow, IEEE) received the BSc degree from Dalian Maritime University, China, in 1982, and the MSc and PhD degrees electrical engineering from Rutgers University, New Jersey, USA, in 1987 and 1990. He is a University professor and the associate chair for Graduate Studies, Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on wireless resource management, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He is the elected IEEE

ComSoc VP Publication, was a member of IEEE ComSoc Board of Governor, and the Chair of Distinguished Lecturers Selection Committee. He served as the Technical Program Committee Chair/Co-Chair for IEEE Globecom'16, Infocom'14, IEEE VTC'10 Fall, and Globecom'07, etc. He also serves/served as the editor-in-chief for *IEEE Internet of Things Journal*, *IEEE Network*, *Peer-to-Peer Networking and Application*, and *IET Communications*; a founding area editor for *IEEE Transactions on Wireless Communications*; and an associate editor for *IEEE Transactions on Vehicular Technology* and *IEEE Wireless Communications*, etc. He received the IEEE ComSoc Education Award, the Joseph LoCicero Award for Exemplary Service to Publications, the Excellent Graduate Supervision Award in 2006, and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He is a registered professional engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, and a distinguished lecturer of IEEE Vehicular Technology Society and Communications Society.