Hindawi
Security and Communication Networks
Volume 2022, Article ID 7206494, 14 pages
https://doi.org/10.1155/2022/7206494

Research Article

ROS-Ethereum: A Convenient Tool to Bridge ROS and Blockchain (Ethereum)

Shenhui Zhang,[1,2] **Ming Tang**,[1,2] **Xiulai Li**,[1,3,4] **Boyi Liu**,[2,5] **Bo Zhang**,[6] **Fei Hu**,[2,7] **Sirui Ni**,[2,8] **and Jieren Cheng**[1,3]

[1]*School of Computer Science and Technology, Hainan University, Haikou 570228, China*
[2]*RobAI-Lab, Hainan University, Haikou 570228, China*
[3]*Hainan Blockchain Technology Engineering Research Center, Haikou 570228, China*
[4]*Hainan Hairui Zhong Chuang Technol Co. Ltd., Haikou 570228, China*
[5]*The Hong Kong University of Science and Technology, Hong Kong 999077, China*
[6]*Hainan Vocational College of Political Science and Law, Haikou 570228, China*
[7]*School of Civil Engineering and Architecture, Hainan University, Haikou 570228, China*
[8]*School of Information and Communication Engineering, Hainan University, Haikou 570228, China*

Correspondence should be addressed to Shenhui Zhang; freak01716@163.com and Boyi Liu; by.liu@ieee.org

Received 23 January 2022; Accepted 15 March 2022; Published 18 April 2022

Academic Editor: Helena Rifà-Pous

Robot Operating System (ROS) has received widespread utilization with the development of robotics, self-driving, etc., recently. Meanwhile, the other technology blockchain is frequently applied to various fields with its trustworthy characteristics and immutability in data storage. However, ROS has no ability to interact with the blockchain, which hinders research in related fields. Therefore, we wonder if we can develop a convenient tool to bridge ROS and blockchain. Inspired by this, we propose ROS-Ethereum. It bridges ROS and Ethereum, a widely used blockchain platform. ROS-Ethereum is based on the User Datagram Protocol (UDP) communication mechanism and the SM algorithm family along with Ethereum technology. Simply put, ROS-Ethereum allows users to invoke the contract when interacting with the blockchain, which makes this process easier and safer. We conduct experiments in real robots to verify the effectiveness of ROS-Ethereum and evaluate it from the following metrics: (1) the encryption efficiency and stability of the algorithm and (2) ROS-Ethereum transaction response time and packet loss rate.

1. Introduction

Robot Operating System (ROS) is a type of operating system consisting of huge nodes running on computers to control the robot. It provides a standard approach to connecting to all the sensors in robots. The ROS nodes which are based on the information collected by these sensors to develop can communicate via the subscription and publication of topics or in a client/server (C/S) mode. This type of flexible structure permits ROS with extremely excellent flexibility and randomness and makes each of the nodes exchange their information at a high frequency.

Researchers in the ROS community now begin to consider the problems exposed in ROS and even other robotic systems constructed by other frames, such as how to improve the security of communication in the robotic operating system [1–3] and how to deal with the typical problem in the field of the cluster of robots: Byzantine fault tolerance [4]. With a certain size of the robot cluster, some nodes with bad efficiency inevitably appear in the process of task execution, even affecting the normal work of other nodes in the cluster. In addition, on account of the transparency of the ROS communication mechanism, the issues of credibility communication between the commercial robots also gain a wide range of attention. To address these problems exposed by ROS, researchers seem to simultaneously select blockchain as the solution to these problems, aiming at these problems exposed by ROS.

Blockchain technology originated from Bitcoin [5] proposed by Satoshi Nakamoto. Vitalik Buterin introduced the concept of smart contracts into blockchain later, proposing the second-generation blockchain named Ethereum [6]. The original intention of blockchain technology is intended to solve the problems of transaction trust crises in the financial sector. As the technology develops, the blockchain has gradually evolved into the storage structure of distributed encryption combined with a point-to-point distributed network [7–9], cryptographic algorithm [10–13], consensus algorithm [14–17], and other technologies. Its superior performance in information storage security and transaction trustworthiness arouses the researchers studying other fields to introduce the excellent characteristics of blockchain into their fields. To satisfy the demands of ROS researchers to interact with the blockchain, this paper proposes a novel interaction tool: ROS-Ethereum, based on the UDP communication technology, Ethereum, Web3, and SM algorithm family. It is the first tool that bridges ROS to Ethereum. And with the help of the SM algorithm family, ROS-Ethereum is born with excellent security in both attack resistance and data transmission.

ROS-Ethereum enables users to monitor the nodes that deliver key information and then preprocess the information (timestamp, format conversion, etc.) in the ROS execution. It hardly limits what type of key information and supports most types of messages interacting with the blockchain. The algorithmic encryption system based on the SM algorithm family can effectively guarantee the security of data transmission while the Ethereum control module interacts with Ethereum via Web3 interfaces to call smart contracts on the blockchain. The superiorities of ROS-Ethereum are shown as follows:

(1) ROS-Ethereum provides convenient user interfaces, blocking the complicated implementation of blockchain and the details of the encryption algorithm. The users are just required to input the names of topics or nodes to be monitored to interact with Ethereum.

(2) ROS-Ethereum has high safety performance. This interactive tool introduces a symmetric key plain text encryption mechanism, providing a shield for the information against the third party.

The main contributions of the paper are as follows:

(1) Providing the ROS-based robot with a convenient, fast, and secure Ethereum interaction tool for the first time.

(2) Playing an important role in promoting the combination of blockchain and ROS, and producing new applications.

(3) Proposing a secure, efficient, and encrypted interaction scheme bridging ROS and blockchain represented by Ethereum.

The rest of this paper is organized as follows: Section 2 describes the related work, Section 3 formalizes our methodology and introduces our framework, and Section 4 analyzes the results of the experiment. The end is a summary.

2. Related Works

In the present national and international study, with the increasing research on robot swarm communication and applications of blockchain technology, the latency of them was measured and analyzed. Besides, the authors have also identified parts of links between them. These studies provide some good examples for our research in this paper and allow us to optimize the top of the research we do. Most themes of these studies focus on blockchain-based communication within robot swarm, secure information sharing through the blockchain network, and behavior control and optimization of robot swarm via blockchain technology. Works with similarities are listed in Table 1.

2.1. Blockchain-Based Communication within Robot Swarm.

Ming Li et al. described the possible ways of building wireless mesh networks to enable multiple robots within every cluster to form mobile self-organizing networks [18]. John Harris et al. proposed an approach working on a generic robot swarm communication network architecture through scalable algorithms for autonomous cluster deployment and ROBOTRAK (a socket-based cluster monitoring and control) [19]. Exchange and sharing of information data such as electronic medical records through blockchain networks were presented by Abichandani et al. [1]. Alexandre Pacheco et al. showed that robotic clusters used decentralized mobile self-organizing networks to achieve a high-throughput communication framework and use blockchain as a security layer to neutralize Byzantine robots [4].

2.2. Secure Information Sharing through Blockchain Network.

To combat the COVID-19 pandemic, a conceptual, trustworthy framework, which can facilitate the sharing of information, goals, and representation, was proposed by Alsamhi et al. to increase intelligence, decentralization, and autonomous operations of connected multirobot collaboration in the blockchain network [20]. Meanwhile, in the case of mutual collaboration in edge-assisted multirobot systems, Jianan Li et al. adapt alike ideas and proposed a blockchain-based collaborative edge knowledge inference (BCEI) framework for edge-assisted multirobot systems [21]. The architecture of knowledge sharing of the Internet of Things (IoT) platform with the help of the Ethereum blockchain was presented by Xia Qi [22]. Ilya Afanasyev and others firstly presented the analysis and application of blockchain-based multi-intelligent body robotic systems [23].

2.3. Behavior Control and Optimization of Robot Swarm via Blockchain Technology.

Siddarth Jain et al. proposed a real-time-based grid search and ad hoc networks for swarm robot optimization in grid navigation [24]. Distributed Bayesian hypothesis testing was used as a novel collective decision-making strategy to solve the collective perception problem in Qihao Shan et al. [25]. Aryo Jamshidpey et al. proposed a multirobot overlay approach, in which the robots

Table 1: Focus of works.

Focus	Related works
Blockchain-based communication within a robot swarm	[1, 4, 18, 19]
Secure information sharing through a blockchain network	[20–23]
Behavior control and optimization of robot swarm via blockchain technology	[24–29]

self-organize a dynamic ad hoc communication network for planning and coordination control [26]. Trung T. Nguyen et al. utilized robot clusters to form a decentralized peer-to-peer network and used blockchain technology to execute different transactions in their study, demonstrating that the performance of decision-making outperforms classical methods [27]. Anbar et al. analyze the security of intrusion detection systems based on blockchain and made a conclusion about the background of integration of blockchain technology and intrusion detection along with future directions and trending topics in intrusion detection based on blockchain technology [28]. Baothman et al. built an imperial example for a blockchain electronic voting (e-voting) application using digital identity management for fulfilling immutable, transparent, and secure distributed blockchain features which proves that it is time-saving and easy to use [29].

These studies provided a solid foundation for the application of ROS-Ethereum.

3. Methodology

ROS-Ethereum is equipped with a user-friendly framework. Users are just required to input names of topics that transmit key data to configure files. Afterward, process nodes within ROS-Ethereum will monitor the assigned topic or nodes and the encrypted key data which the users need in some specific time (or a period of time) is stored on the chain with persistence and encryption for later interactive use. The framework of ROS-Ethereum is shown in Figure 1.

3.1. SM Algorithm Family. The encryption system of ROS-Ethereum is based on SM2, SM3, and SM4. SM2 elliptic curve public key crypto algorithm involves SM2-1 elliptic curve digital signature algorithm, SM2-2 elliptic curve key exchange protocol, and SM2-3 elliptic curve public key encryption algorithm which are used to implement digital signature, key negotiation, and data encryption, respectively. The SM3 hash algorithm is applied to the generation and verification of digital signatures and message authentication codes as well as the generation of random numbers in commercial applications. Concerning message m with a length of l ($l < 2$) bit, the SM3 hash algorithm is filled and iteratively compressed to generate a hash value. The hash value has a length of 256 bits, which has high security. The SM4 algorithm is a block algorithm. The structure of the decryption algorithm is the same as that of the encryption algorithm, except that the use order of the round key is reversed, and the decryption round key is the reverse order of the encryption round key. SM4 has high security and can effectively resist a variety of attacks. In this system, we utilize SM4 to encrypt the data communication between the client and the server and use SM2 for digital signature. The data sending end uses SM2 to sign the raw data and encrypts the whole data block including the signature through SM4.

SM4 Key Expansion: the variable size of word type is 32 bits. The variable MK is the initial key, an array of four 32-bit long elements for a total of 128 bits. The variable rK is an array containing thirty-two 32-bit long elements generated by the 128-bit initial key for the SM4 cipher algorithm. The variables FM and MK are publicly known, where FK is the system parameter and MK is the fixed parameter. Their values are as follows:

$$FK = [\mathrm{3B1BAC6}, \mathrm{56AA3350}, \mathrm{677D9197}, \mathrm{B27022DC}],$$
$$CK = \begin{bmatrix} \mathrm{00070E15}, \mathrm{1C232A31}, \mathrm{383F464D}, \mathrm{545B6269}, \\ \mathrm{70777E85}, \mathrm{8C939AA1}, \mathrm{A8AFB6BD}, \mathrm{C4CBD2D9}, \\ \mathrm{E0E7EEF5}, \mathrm{FC030A11}, \mathrm{181F262D}, \mathrm{343B4249}, \\ \mathrm{50575E65}, \mathrm{6C737A81}, \mathrm{888F969D}, \mathrm{A4ABB2B9}, \\ \mathrm{C0C7CED5}, \mathrm{DCE3EAF1}, \mathrm{F8FF060D}, \mathrm{141B2229}, \\ \mathrm{30373E45}, \mathrm{4C535A61}, \mathrm{686F767D}, \mathrm{848B9299}, \\ \mathrm{A0A7AEB5}, \mathrm{BCC3CAD1}, \mathrm{D8DFE6ED}, \mathrm{F4FB0209}, \\ \mathrm{10171E25}, \mathrm{2C333A41}, \mathrm{484F565D}, \mathrm{646B7279} \end{bmatrix}. \tag{1}$$

The pseudocode of the SM4 key expansion algorithm is shown in Algorithm 1.

3.2. Encryption Mechanism. To ensure security during the data transmission, we designed the certificate authentication key exchange based on the SM algorithm. Firstly, it can perform the authentication of ID and temporary interaction number N between the server and the client through an identifier. Secondly, the server uses SM2 to encrypt and signal symmetric keys and deliver them to the client. Finally, the client decrypts the key K, recovering and verifying the integrity of the key. The executive process is shown in Figure 2.

As to data communication, ROS-Ethereum encrypts the original and signature of data in communication between the client and the server through SM4, keeping the transmission in the form of ciphertext to guarantee the privacy of communication data. The user selectively specifies the topic that needs to be captured. After ROS-Ethereum captures the text, it will be encrypted, using the SM4 key immediately. The ciphertext will be placed in the loop queue waiting for the sending process reading. Before the client sends messages, some pretreatment operations are firstly performed on the data ciphertext which is firstly encrypted with SM4, such

```
(i) Input: the 128-bit initial key $MK$
(ii) listalistaOutput: the round key array of length 32, each element of size 32 bits $rK$
(iii) Initialization: convert $MK$ to an array of length 4, each element of size 32 bits
(iv) Begin
(v) word $K[36]$
(vi) word $tmp$
(vii) For $i = 0:4$
(viii) $K[i] = MK[i] \oplus FK[i]$
(ix) For $i = 0:32$
(x) $tmp = K[i+1] \oplus K[i+2] \oplus K[i+3] \oplus CK[i]$
    $tmp = tmp \oplus (tmp \lll 13) \oplus (tmp \lll 23)$
    $rK[i] = K[i+4] = K[i] \oplus tmp$
(xi) End
(xii) SM4 cipher: The variable size of byte type is 8 bits. The variables $X$ and $Y$ are plaintext and ciphertext pairs, and they are both an
      array containing four 32-bit long elements for a total of 128 bits. The SM4 cipher algorithm works for both encryption and
      decryption; the only difference is that the order of the round keys is reversed. The pseudocode of the SM4 cipher algorithm is
      shown in Algorithm 2.
```

Algorithm 1: The pseudocode of the SM4 key expansion algorithm.

```
(i) Input: the 128 bit plaintext $X$ and the round key array of 32 bits each $rK$
(ii) Output: the 128 bit ciphertext $Y$
(iii) Initialization: convert $X$ to an array of length 4, each element of size 32 bits
(iv) Begin
(v) word state$[36]$
(vi) byte $tmp1[4]$
(vii) word $tmp2$
(viii) For $i = 0:4$
(ix) state$[i] = X[i]$
(x) For $i = 0:32$
(xi) $(tmp[0], tmp[1], tmp[2], tmp[3]) = \text{state}[i+1] \oplus \text{state}[i+2] \oplus \text{state}[i+3] \oplus rK[i]$
(xii) For $j = 0:4$
(xiii) $tmp[j] = Sbox(tmp(j))$
(xiv) $tmp2 = (tmp1[0], tmp1[1], tmp1[2], tmp1[3])$
(xv) $tmp2 = tmp2 \oplus (tmp2 \lll 2) \oplus (tmp2 \lll 10) \oplus (tmp2 \lll 18) \oplus (tmp2 \lll 24)$
(xvi) state$[i+4] = \text{state}[i] \oplus tmp2$
      For $i = 0:4$
      $Y[i] = \text{state}[35-i]$
(xvii) End
(xviii) SM2 KDF : KDF is the key derivation function used to derive key data from a shared secret bit string. The bitstring type is a bit
        string, and the integer type is an integer. The  operation implies the operation of converting the type to a bit string for
        concatenation. $\lceil X \rceil$ is the smallest integer greater than or equal to $X$, and $\lfloor X \rfloor$ is the largest integer less than or equal to $X$. The
        variable $Z$ is the shared secret bit string, and the variable klen is the length of derived key data. The variable $K$ is the output key
        data bit string of length klen. The function $Hv$ is the cryptographic hash function whose output is exactly $v$ bits in length. The
        pseudocode of KDF is shown in Algorithm 3.
```

Algorithm 2: The pseudocode of the SM4 cipher algorithm.

as increasing timestamp and message format Jason conversion. After the pretreatment is completed, the encryption module will call the message-converted message to the second encryption and the ciphertext is sent by the UDP client to the server underthe specified port. The relevantillustrations of the conceptare shown in Figure 3.

3.3. Ethereum Setup. Through Ethereum Virtual Machine (EVM), which is a Turing complete virtual machine that processes and handles all transactions carried out in the Ethereum, a private Ethereum network where every robot which runs an independent ROS system has a relating Ethereum account is set up on the server with the help of Geth. And due to the features of a private blockchain, the identity of every robot is known and verifiable.

(1) Geth : Geth client is used to build the private Ethereum blockchain. Ethereum clients are software that can establish P2P communication channels with other users, sign and broadcast transactions, mine, deploy, and interact with intelligent contracts. So far,

```
(i) Input: the shared secret bit string $Z$, the length of derived key data $klen$
(ii) Output: key data bit string of length $K$
(iii) Begin
(iv) bitstring $ct = 0x00000001$
(v) bitstring $Ha[\text{upper}]$
(vi) bitstring $Ha_{-}$
(vii) upper = $\lceil klen/v \rceil$
(viii) For $i = 0$ : upper
(ix) $Ha[i] = Hv[Z||t++]$
(x) If $klen/v$ is the type of Integer
(xi) $Ha_{-} = Ha[\text{upper} - 1]$
(xii) Else
(xiii) left_len = $(klen - (v * \lfloor klen/v \rfloor))$
(xiv) $Ha_{-} = Ha[\text{upper} - 1][0\text{: left\_}len]$
(xv) $K = 0$
(xvi) For $i = 0$: upper − 1
      $K = K||Ha[i]$
      $K = K||Ha_{-}$
(xvii) End
(xviii) SM2 Encrypt: The variable $M$ is the message to be encrypted, and the variable $C$ is the encrypted ciphertext, both of which are
      bit string types. The variable $PB$ is the public key of the point type, and the variable $k$ is the random integer generated by the
      random number generator. The pseudo of SM2 encrypt is shown in Algorithm 4.
```

Algorithm 3: The pseudocode of the SM2 key derivation function algorithm.

```
(i) Input: the bit string type message $M$, the public key $PB$, and the random number $k$
(ii) Output: the bit string type ciphertext $C$
(iii) Begin
(iv) $C1 = k * G$
(v) $S = h * PB$
(vi) If $S$ equals $O$
(vii) throw error and exit
(viii) $tmp = k * PB$
(ix) $t = \text{KDF}(tmp.x||tmp.y, klen)$
(x) $C2 = M \oplus t$
     $C3 = Hash(tmp.x\|M\|tmp.y)$
     $C = \text{bitstring}(C1)\|C2\|C3$
     End
(xi) SM2 Decrypt: The variable $C$ is the ciphertext to be decrypted, and the variable $M$ is the decrypted message, both of which are bit
     string types. The variable $C1$, $C2$, and $C3$ are bit strings extracted from variable $C$ according to certain rules. The variable $dB$ is the
     private key of the integer type. The pseudocode of SM2 decrypt is shown in Algorithm 5.
```

Algorithm 4: The pseudocode of the SM2 encrypt algorithm.

Geth is one of the most popular Ethereum clients, which is written in the Golang language. There are two types of accounts in Geth : Externally Owned Accounts (EOAs) and Contract Accounts (Smart Contracts). ROS-Ethereum has a built-in Contract Account managed by the administrator in its private Ethereum blockchain, and the rest of the robot-bound accounts are EOA accounts. An Ethereum account is defined by a pair of keys including public keys and private keys, and each account is indexed by its address which is derived from the first 20 bytes of the SHA3 hashed public key.

(2) Truffle Development Environment: truffle is an Ethereum resource management channel, development, and testing environment that makes development on Ethereum simple. Truffle can compile, link, and deploy smart contracts. And Truffle can manage the binary files generated by contract compilation. Truffle uses the package management provided by EthPM and NPM and uses the ERC190 standard. Users can directly test the contract through the console. The built-in automation script of Truffle ensures that each node can only call the methods meant for it, ensuring the security of communication between Ethereum nodes.

```
(i) Input: the bit string type ciphertext $C$, the private key $dB$
(ii) Output: the bit string type message $M$
(iii) Initialization: extract byte strings $C1$, $C2$, and $C3$ from $C$
(iv) Begin
(v) $C1 = \text{Point}(C1)$
(vi) $S = h * C1$
(vii) If $S$ equals $O$
(viii) throw error and exit
(ix) $tmp = dB \ * C1$
(x) $t = \text{KDF}(tmp.x||tmp.y, klen)$
(xi) If $t$ equals 0
(xii) throw error and exit
(xiii) $\_M = C2 \oplus t$
(xiv) $U = Hash(tmp.x \| \_M \| tmp.y)$
(xv) If $U$ not equals $C3$
     throw error and exit
     $M = \_M$
     End
```

Algorithm 5: The pseudocode of the SM2 decrypt algorithm.

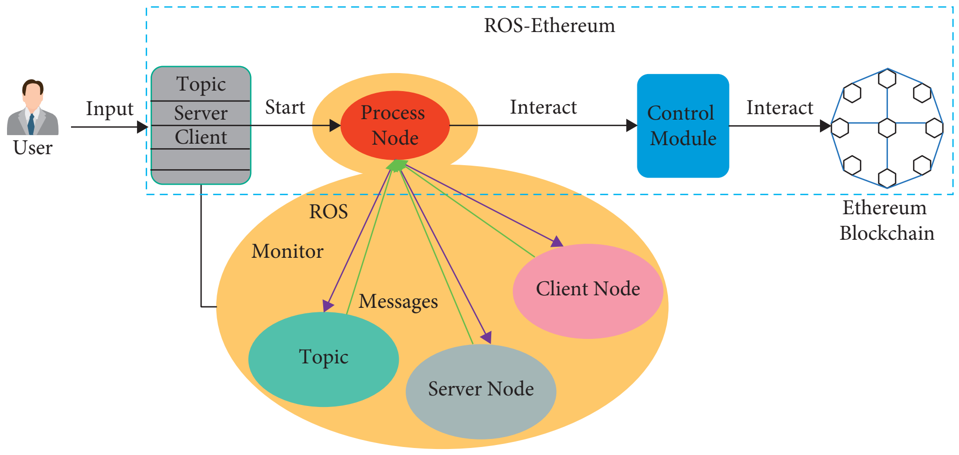


Figure 1: The framework of ROS-Ethereum.

(3) Architecture of Ethereum network: considering the actual scenario, building an Ethereum node on the robot will encounter the bottleneck of Edge Computing. Therefore, the Ethereum network composed of 3 nodes was implemented using a host equipped with a CPU (Intel(R) Core(TM) i9-10900 KF @3.70Ghz), memory (64 GB DDR4 3200 MHz), and a GPU (NVIDIA Geforce RTX3090). The robot only needs to maintain a connection with the ROS Master running on the host. Keeping the ROS system separated from the Ethereum network can avoid the bottleneck of Edge Computing. The structure of the Ethereum private chain is shown in Figure 4. Among the 3 Ethereum nodes, one node is a miner node. This node was also designated as the bootstrap node and was responsible for forming the overlay network with other nodes. 4 ROS-Melodic robots are associated with two EOA accounts in Node2 and Node3, respectively, to obtain the ability to interact with the Ethereum network.

(i) Consensus Algorithm: consensus algorithms equip the distributed ledger of the blockchain with immutability, automation, and anonymous transactions without third parties. Without consensus algorithms, blockchain can only be an inefficient distributed storage database. The existence of consensus algorithms endows blockchain technology with core meaning and value. Current consensus algorithms mainly include Proof of Work (Pow), Proof of Stake (PoS), Proof of Authority (PoA), and Practical Byzantine Fault Tolerance (PBFT). Ethereum currently uses the PoW consensus algorithm. With ETH2.0 coming soon, the consensus algorithm of Ethereum will be changed from PoW to PoS. Since Ethereum has not yet fully supported PoS, in our experiments, we evaluated the effect of PoW and PoA algorithms on the time consumption of knowledge on-chain and query. The biggest difference between PoW and PoA is that PoW relies on mining to verify blocks and cannot specify the block

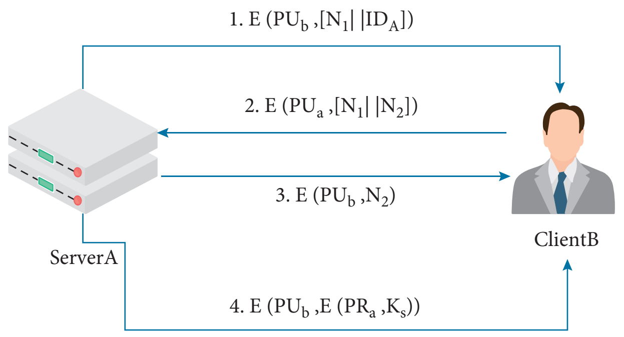

1. E (PU$_b$,[N$_1$| |ID$_A$])
2. E (PU$_a$,[N$_1$| |N$_2$])
3. E (PU$_b$,N$_2$)
4. E (PU$_b$,E (PR$_a$,K$_s$))
ServerA
ClientB

Figure 2: The process of key exchange.

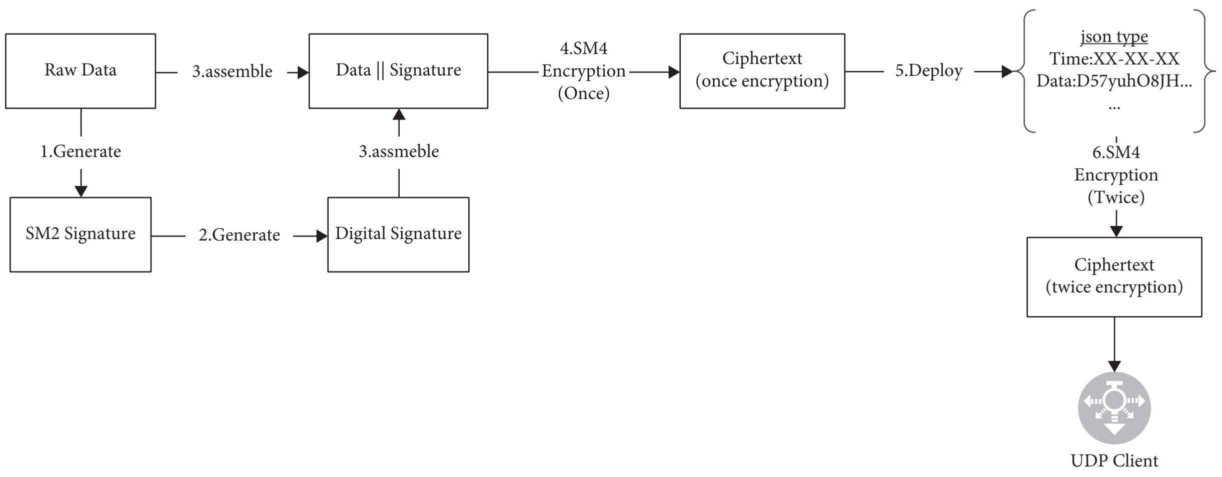


Figure 3: The process of encryption of raw data.

times, while PoA relies on preauthorized nodes to verify blocks without mining operations. Therefore, the block time can be specified.

(ii) Interaction Scheme: the blockchain end of ROS-Ethereum automatically obtains the message to be processed for interaction. The way to interact with Ethereum is mainly through smart contracts and contracts are developed using Solidity which is unique to Ethereum contract development. The pseudocode of the process of interaction with the blockchain of ROS-Ethereum is shown in Algorithm 6.

```
(i) Initialization: Token which can prove the identity of robot $Tk$. Ciphertext $Cp$
(ii) Begin
(iii) Process of Registration:
(iv) Addr = Registration ($Tk$)
(v) Process of ciphertext storage:
(vi) If Addr is not registered or Addr does not have enough ETH
(vii) throw error and exit
(viii) SetData ($Tk, Cp$, Addr)
(ix) Process of data checking:
(x) If Addr is not registered
(xi) throw error and exit
(xii) CheckData ($Tk, Cp$)
(xiii) Process of ETH allocating:
      If Addr is not registered or ETH of Addr > 5
      throw error and exit
      AllocETH (Addr)
      End
```

Genesis Block: genesis block is the first block in the blockchain. Some important parameters defining the blockchain were included in the genesis block. Normally, the file which is used to initialize the genesis block is written in JSON and is shared by all Ethereum nodes. Since we evaluate two consensus algorithms, we prepare genesis files for two consensuses each other. The files we used are depicted in Figures 5 and 6.

Some key parameters include the following:

(i) GasLimit: this parameter is the maximum amount of gas that the user of Ethereum is willing to spend for a particular transaction. While gas is a unit used to measure the effort required for a particular computation in an Ethereum Virtual Machine (EVM), gas price is the value the transaction sender

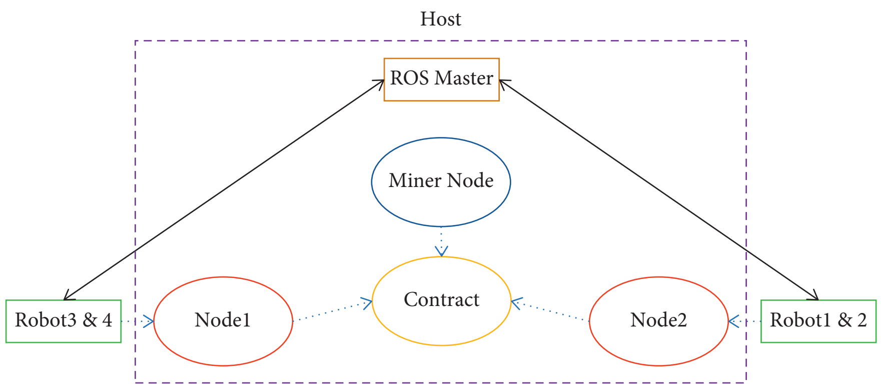


Figure 4: Architecture of private Ethereum network used in experiments.

```
{
    "config": {
        "chainId": 666,
    },
    "nonce": "0x0",
    "timestamp": "0x5ddf8f3e",
    "extraData": "0x0",
    "gasLimit": "0xffffffff",
    "difficulty": "0x4cccc8",
    "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x0000000000000000000000000000000000000000",
    "alloc": { },
    "number": "0x0",
    "gasUsed": "0x0",
    "parentHash": "0x0"
}
```

Figure 5: Genesis file of a PoW based private Ethereum network.

is willing to pay per gas unit and is measured in GWei. Ether is the token used to pay for gas.

(ii) Difficulty: this parameter represents the difficulty of generating blocks in the blockchain, that is, the difficulty of calculating the hash value of the next block. The higher the difficulty is set, the longer it takes to find blocks.

(iii) Alloc: this parameter is used to specify the number of preset accounts in the blockchain and the amount of ether allocated to these preset accounts. The specified amount of ether balance cannot be lower than the preset GasLimit; otherwise, there will be a situation where the amount of ether in an account is not enough to pay for the transactions it needs to execute. In addition, it is worth mentioning that the Alloc parameter only exists in private blockchains.

(iv) Both the PoW and PoA based private Ethereum networks share the same set of GasLimit, Difficulty, and Alloc. All EOAs will be created and allocated balances after the completion of the initialization of the genesis block. Meanwhile, a contract account in the miner node will be created for contract deployment. After the deployment, the contract account will turn into a miner, which provides Ethers for other EOAs. Though a private Ethereum network is not part of the public blockchain, we set gas prices to a default value to simulate the true trading situation. Figure 7 depicts the workflow of ROS-Ethereum.

4. Implementation and Simulation Results Evaluation

4.1. Implementation of ROS-Ethereum in Robot. Initialize the Ethereum private chain network in the PC. To facilitate experiments, we set 10 Ethereum accounts with 100 ETH coins. The 10 accounts are shown in Figure 8.

After initialization, the smart contract is deployed to the blockchain. Block information is shown in Figure 9.

Next, we run the ‘easy_register’ Python package to register an account named ‘test_account’. The registered account will be tied to an existed Ethereum account automatically for later interaction. The information on command lines is shown in Figure 10.

```
{
  "config": {
    "clique": {
      "period": 20,
      "epoch": 30000
    }
  },
  "nonce": "0x0",
  "timestamp": "0x62232e59",
  "extraData":
"0x0000000000000000000000000000000000000000000000000000000000000000d2f01c1f7
341f67105fff23aab886ce0fdff561b0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
00000000",
  "gasLimit": "0xffffffff",
  "difficulty": "0x4cccc8",
  "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000",
  "alloc": {
    "d2f01c1f7341f67105fff23aab886ce0fdff561b": {
      "balance":
"0x200000000000000000000000000000000000000000000000000000000000000"
    }
  },
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

FIGURE 6: Genesis file of a PoA based private Ethereum network.

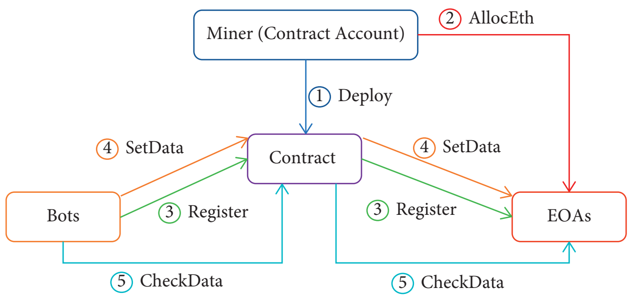


FIGURE 7: Workflow of our Ethereum network used in experiments. All bots are tied with an EOA and can interact with the blockchain through smart contract functions. The smart contracts featured functions that can only be called by preset EOAs. In this case, bots firstly need to register to get the tied EOA and then can call other functions.

```
>eth.accounts
["0x188a57f3bc189d9118f07ce09ff91d11ef6b204f",
"0xcc1361e603c3957090c1c377f3a9f6a188006f23",
"0xd607672b17219b7f160d6889fae67ed81bd6e98e",
"0xbf1e52c2315e9fa73fb1f0f1f0d3703318af46df",
"0xaa667eae5b399593a491126bc5a34b109c7b65b1",
"0x7c4c066df9f51c5233fa874c9a581a205d445593",
"0x4d0e8c72386c00ec9fa7930200b41fafe71318cf",
"0x896f864ef83de0d70f95ee9a87adccd508b101e6",
"0x8b465b93553cde50c5ddd5d3f58c836da567530d",
"0x39bded73cf2fe342592e6083064d0404dfc575b5"]
```

FIGURE 8: 10 accounts are initialized for the Ethereum network, and each account comes with 100 Ethernet coins for demonstration.

Finally, we input the configuration file and monitor the/robot_1/odom topic in the robot (this topic is mainly responsible for the relative position, moving line speed, etc.). After the obtained message is stored in the cache, the blockchain operation module will reach the ciphertext and call the corresponding decryption process to decrypt the secondary encryption ciphertext. Finally, the decrypted messages will be used to interact with Ethereum. Figure 11 shows the ciphertext in the cache and the prompt message of a successful interaction transaction.

4.2. Environment Setup of the System. We run experiments in five parts to verify the practicability of ROS-Ethereum: User_end messages sending rates, packet loss rate of message reception at blockchain_end, encryption and decryption efficiency and stability of SM algorithm family, the blockchain interaction ability of ROS-Ethereum in the high

```
{
   blockHash: "0x6411331b39faaed9c380d047bc3a6d08d480f00ac1e0ffe8126ee1e1d0d69b0a",
   blockNumber: 61,
   from: "0x3ba2b5549678974e4e48dec33a65d863cdb2aa68",
   gas: 2573980,
   gasPrice: 100000000,
   hash: "0xdc363b0f3a7c048420f24e0e41de92a15e280b336cd69b0f17e389e21c11f17b",
   input: “bytecode of contract”,
   nonce: 2,
   r: "0x1e376d0f809663743012cc052964278104be081bb9cab9854006bfa187e67fd3",
   s: "0x640690016bcecf013b8591c01a060d6e9b13015fe8abb359d47f0145eca0c333",
   to: null,
   transactionIndex: 0,
   v: “0x558”,
   value: 0
}
```

Figure 9: The prompt information after the successful contract deployment and the transaction packaging information generated by the contract deployment.

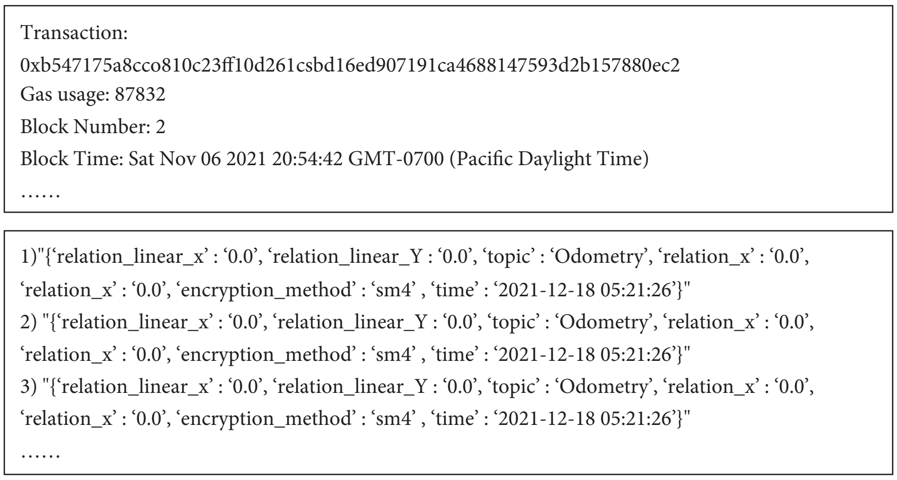
```
(base) root@ubuntu:/home/zsh# conda activate py2_env
(py2_env) root@ubuntu:/home/zsh# cd $catkin_ws/src/listen_node/scripts/Utlimate
(py2_env) root@ubuntu:/home/zsh/catkin_ws/src/listen_node/scripts/Utlimate# python
register.py
welcome to ROS-Etherem!
----------------------------Warning----------------------------------
The name you enter is important, please keep that for future using
----------------------------------------------------------------------
For the first use, Please input your username
test_account
Please input your password
zsh
Please input your sm4 key for encryption.
123456
Your username is ‘test_account’.
Register successfully! Please keep your username, password and sm4_key carefully!
```

Figure 10: ‘test_account’ is created and related information such as password and tied Ethereum account number are stored in the Redis cache. The red bold font indicates the input value.

concurrency scenario, and effects of consensus algorithms. We ran our experiments on 4 ROS-Melodic robots with a ROS Master on the host. For the same configuration of robots, we only evaluate the performance of one robot. The detailed environment configuration is shown in Table 2.

4.3. Experiments of User_End. The experiment of the user_end mainly focuses on the response time of the message sending. The user_end information of ROS-Ethereum adopts the SM4 symmetrical encryption algorithm to encrypt and the encryption performance of the encryption algorithm largely affects the rate of messaging. It will cause the users to fail to send messages and even lose them if the response time is too long. To make the experiments results more adaptive, we write the Python file to analog the process of message delivery and use it to build bash files to simulate a process of sending 10,000 messages in a sequence. Write the time consumption of successful transmission and SM4 encryption to two CSV files, respectively. The table obtained after the washing analysis of this 10000 sets of data is shown in Table 3:

Since UDP is a transport protocol that can broadcast message broadcasts without establishing a connection, there is an advantage of a high transmission rate (32 subtle), but there are also a few packet losses (0.14%). The response time transmitted by the message is largely dependent on the encryption efficiency of the SM4 encryption algorithm. The average response time is 0.852 ms, and SM4 encryption just costs 0.82 ms. From the experimental result, the time of a single user’s message is relatively low, which is in



```
Transaction:
0xb547175a8cco810c23ff10d261csbd16ed907191ca4688147593d2b157880ec2
Gas usage: 87832
Block Number: 2
Block Time: Sat Nov 06 2021 20:54:42 GMT-0700 (Pacific Daylight Time)
……
```

```
1)"{'relation_linear_x' : '0.0', 'relation_linear_Y : '0.0', 'topic' : 'Odometry', 'relation_x' : '0.0',
'relation_x' : '0.0', 'encryption_method' : 'sm4' , 'time' : '2021-12-18 05:21:26'}"
2) "{'relation_linear_x' : '0.0', 'relation_linear_Y : '0.0', 'topic' : 'Odometry', 'relation_x' : '0.0',
'relation_x' : '0.0', 'encryption_method' : 'sm4' , 'time' : '2021-12-18 05:21:26'}"
3) "{'relation_linear_x' : '0.0', 'relation_linear_Y : '0.0', 'topic' : 'Odometry', 'relation_x' : '0.0',
'relation_x' : '0.0', 'encryption_method' : 'sm4' , 'time' : '2021-12-18 05:21:26'}"
……
```

FIGURE 11: The upper part is the ciphertext of the user message stored in the cache at the blockchain end and Ethereum records the relevant transaction information about the interaction. The lower part indicates the decrypted information.

TABLE 2: Configuration information.

Item	Rules
CPU	Intel(R) core(TM) i9-10900 KF 10 core @3.70 GHz
Memory	64 GB DDR4 memory
Hard disk	200G ESSD
GPU	NVIDIA GeForce RTX3090
System environment	Ubuntu18.04 LTS
ROS	Melodic
Programing language	Python3.8, Python2.7, java
Blockchain environment	Ethereum1.x Geth v1.9.25 Web.js v5.21.0 Solidity ^0.6.0

TABLE 3: Experimental results of message sending in user_end.

Item	Results
Messages send successfully	9986
All time consumed	8.508s
SM4 encryption time consumed	8.188s
Average time consumed	0.852 ms
SM4 encryption time peak	1.23 ms
SM4 encryption time minimum	0.50 ms

microsecond levels. In actual use, the message transmission interval (second level) is much greater than the response time. Therefore, it can meet the normal using requirements of the users. The extreme time required for SM4 encryption is between 0.50 ms and 1.23 ms, with good encryption efficiency and stability.

4.4. Experiments of Blockchain_End. The UDP server consists of data reception, data parsing, and blockchain interaction in 3 parts. We write the bash scripts simulating user links to create 300, 500, and 700 user analog processes, respectively. Experiments consist of the packet loss rate of the data reception module, the parsing efficiency of the data parsing module to the ciphertext sent by the user (reflected in the processing time), and the blockchain interaction (reflected interface call response time) under different system loads. The experimental results are shown in Table 4.

4.4.1. Packet Loss Rate. Our experiment sets the simulated process of each user analog to send 10 messages. The second column in Table 4 is the number of messages which are accepted successfully, and the third list is the number of the received message successfully going to Ethereum. The experiments results show that ROS-Ethereum can maintain a lower packet loss rate, even in high concurrent scenes. After analysis, we believe that the loss of data is mainly due to the unreliability of the UDP transport protocol.

Table 4: Experimental results of packet loss rate in Blockchain_end.

System loading	Receive successfully	To chain successfully
300	2994	2993
500	4986	4986
700	6991	6991
Average	99.77%	99.98%

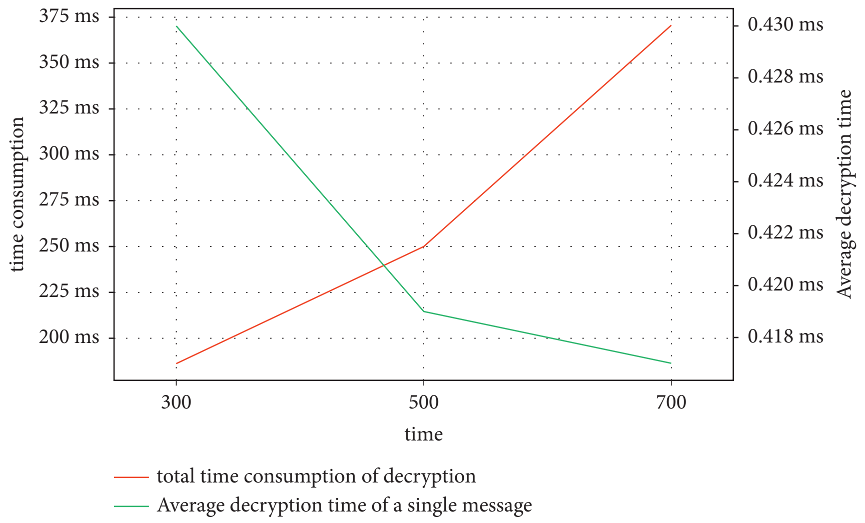


Figure 12: It takes 186.34 ms for 2994 messages sent by 300 user processes to be fully decrypted, 249.88 ms for 4986 messages sent by 500 user processes, and 370.29 ms for 6991 messages sent by 700 user processes. The average decryption time of a single message is 0.422 ms. C. The ability of blockchain interaction.

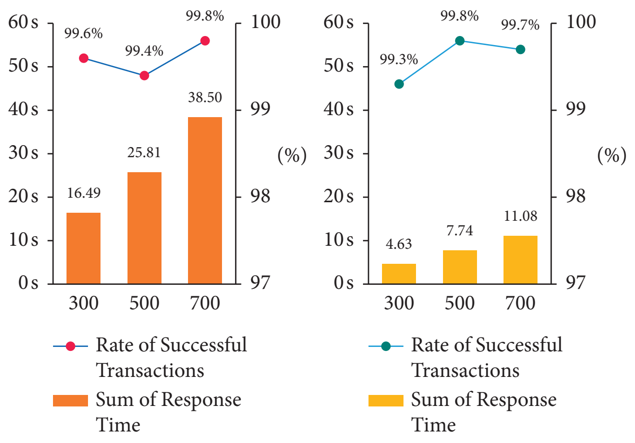


Figure 13: Left: total response time and success rate of transactions in seconds for Proof of Work (PoW). Right: total response time and success rate of transactions in seconds for Proof of Authority (PoA).

4.4.2. Time Consumption of Data Decryption. For a different number of messages sent in different load conditions, we set 8 processes to decrypt ciphertexts stored in 8 corresponding buffers at the same time. The experimental results are shown in Figure 12. Although the increase in the total number of messages under different loads makes the overall data parsing time long, the decryption efficiency of single messages is not very fluctuated by it and the average decryption time is maintained at 0.422 ms. The extreme time required for single message decryption lies between 0.311 ms and 0.78 ms, so it can be concluded that the SM4 algorithm has good decryption efficiency and stability.

Users and blockchain interactions are the core of ROS-Ethereum. It is the core issue of our attention whether ROS-Ethereum can maintain the normal interaction of users and Ethereum under the high concurrent scenario and how many users are supported by this scheme to interact. So we assume that all users' analog processes initiated 9 interactive requests to Ethereum in different system loads. (Mainly through the first three interfaces in the above contract, 1 is a request to register the identity in the Ethereum network, 4 times for the data upper chain request and 4 times for query requests.) We set 8 request processing processes, testing the total response time and success rate of transactions under three different concurrency parts. At the same time, we also evaluate the effect of different consensus algorithms (PoA and PoW) on the response time under the condition of the same block difficulty and GasLimit. The experimental results are shown in Figure 13.

ROS-Ethereum maintains good interaction using both algorithms in high concurrent scenes. For the success rate of transactions, both consensus algorithms reached more than 99%, and it was observed that different consensus algorithms have no significant gap in the success rate of transactions. However, it was obvious that PoA has a clear edge over PoW on the response time. This behavior follows the fact that PoW relies on the mining process to verify transactions while PoA verifies transactions through preset nodes' voting which causes longer blocking time and verifies time of transactions of PoW. We can draw a conclusion that PoA is a faster option for information transfer in multirobots.

5. Conclusion

This paper proposes a blockchain-interaction-scheme: ROS-Ethereum. With regard to the research based on ROS and blockchain, ROS-Ethereum has good applicability and scalability. Benefiting from the SM algorithm-based encrypted transmission system we design, ROS-Ethereum has excellent safety performance and higher encryption efficiency. ROS-Ethereum provides the ROS application developers with a good choice to interact with blockchain when they need it. In the future, we will further promote the interaction capabilities of ROS-Ethereum and support more types of ROS messages to broaden the applicabilities of ROS-Ethereum through the expansion of user interfaces of ROS-Ethereum.

Data Availability

The data are available at https://github.com/RobAI-Lab/ROS-Chain.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Key Research and Development Program of Hainan Province (Grant no. ZDYF2020033), Young Talents' Science and Technology Innovation Project of Hainan Association for Science and Technology (Grant no. QCXM202007), Hainan Provincial Natural Science Foundation of China (Grant nos. 2019RC107, 2019RC098, and 621RC612), and Innovation and Entrepreneurship Training Program for College Students (Grant no. 202110589017X).

References

[1] P. Abichandani, D. Lobo, S. Kabrawala, and W. McIntyre, "Secure communication for multiquadrotor networks using Ethereum blockchain," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1783–1796, 2020.

[2] M. Ai Duhayyim, F. N, R. Marzouk et al., "Integration of fog computing for health record management using blockchain technology," *Computers, Materials & Continua*, vol. 71, no. 2, pp. 4135–4149, 2022.

[3] M. Uddin, I. Memon, I. Ali, J. Memon, M. Abdelhaq, and R. Alsaqour, "Hyperledger fabric blockchain: secure and efficient solution for electronic health records," *Computers, Materials & Continua*, vol. 68, no. 2, pp. 2377–2397, 2021.

[4] A. Pacheco, V. Strobel, and M. Dorigo, "A blockchain-controlled physical robot swarm communicating via an ad-hoc network," in *Proceedings of the International Conference on Swarm Intelligence*, pp. 3–15, Springer, Belgrade, Serbia, July 2020.

[5] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[6] V. Buterin, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[7] J. A. Donet Donet, C. Pérez-Solà, and J. Herrera-Joancomartí, "The bitcoin P2P network," *Financial Cryptography and Data Security*, in *Proceedings of the International conference on financial cryptography and data security*, pp. 87–102, Springer, Barbados, March 2014.

[8] F. Cornelli, E. Damiani, S. D. C. Di Vimercati, S. Paraboschi, and P. Samarati, "Choosing reputable servents in a P2P network," in *Proceedings of the 11th international conference on World Wide Web*, pp. 376–386, Honolulu HI USA, May 2002.

[9] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in Bitcoin P2P network," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 15–29, Scottsdale AZ USA, November 2014.

[10] M. Szydlo, "Merkle tree traversal in log space and time," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 541–554, Springer, Interlaken, Switzerland, May 2004.

[11] H. Gilbert and H. Handschuh, "Security analysis of SHA-256 and sisters," in *Proceedings of the International workshop on selected areas in cryptography*, pp. 175–193, Springer, Ottawa, Canada, August 2003.

[12] D. Boneh, "Twenty years of attacks on the RSA cryptosystem," *Notices of the AMS*, vol. 46, no. 2, pp. 203–213, 1999.

[13] F. Amounas and E. H. El Kinani, "ECC encryption and decryption with a data sequence," *Applied Mathematical Sciences*, vol. 6, no. 101, pp. 5039–5047, 2012.

[14] M. H. DeGroot, "Reaching a consensus," *Journal of the American Statistical Association*, vol. 69, no. 345, pp. 118–121, 1974.

[15] L. J. Bourgeois III, "Performance and consensus," *Strategic Management Journal*, vol. 1, no. 3, pp. 227–248, 1980.

[16] A. Fink, J. Kosecoff, M. Chassin, and R. H. Brook, "Consensus methods: characteristics and guidelines for use," *American Journal of Public Health*, vol. 74, no. 9, pp. 979–983, 1984.

[17] J. E. Innes, "Consensus building: clarifications for the critics," *Planning Theory*, vol. 3, no. 1, pp. 5–20, 2004.

[18] M. Li, K. Lu, H. Zhu, M. Chen, S. Mao, and B. Prabhakaran, "Robot swarm communication networks: architectures, protocols, and applications," in *Proceedings of the 2008 Third*

International Conference on Communications and Networking in China, pp. 162–166, IEEE, Hangzhou, China, August 2008.

[19] M. Li, J. Harris, M. Chen et al., "Architecture and protocol design for a pervasive robot swarm communication networks," *Wireless Communications and Mobile Computing*, vol. 11, no. 8, pp. 1092–1106, 2011.

[20] S. H. Alsamhi and B. Lee, "Blockchain-empowered multi-robot collaboration to fight COVID-19 and future pandemics," *Ieee Access*, vol. 9, pp. 44173–44197, 2020.

[21] J. Li, J. Wu, J. Li, A. K. Bashir, M. J. Piran, and A. Anjum, "Blockchain-based trust edge knowledge inference of multi-robot systems for collaborative tasks," *IEEE Communications Magazine*, vol. 59, no. 7, pp. 94–100, 2021.

[22] Q. Xia, E. Sifah, A. Smahi, S. Amofa, and X. Zhang, "BBDS: blockchain-based data sharing for electronic medical records in cloud environments," *Information*, vol. 8, no. 2, p. 44, 2017.

[23] Y. Nishida, K. Kaneko, S. Sharma, and K. Sakurai, "Suppressing chain size of blockchain-based information sharing for swarm robotic systems," in *Proceedings of the 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 524–528, IEEE, Takayama, Japan, November 2018.

[24] S. Jain, M. Sawlani, and V. K. Chandwani, "Ad-hoc swarm robotics optimization in grid based navigation," in *Proceedings of the2010 11th International Conference on Control Automation Robotics & Vision*, pp. 1553–1558, IEEE, Singapore, December 2010.

[25] Q. Shan and S. Mostaghim, "Collective decision making in swarm robotics with distributed Bayesian hypothesis testing," *Lecture Notes in Computer Science*, in *Proceedings of the International Conference on Swarm Intelligence*, pp. 55–67, Springer, Barcelona, Spain, October 2020.

[26] A. Jamshidpey, W. Zhu, M. Wahby, M. Allwright, M. K. Heinrich, and M. Dorigo, "Multi-robot coverage using self-organized networks for central coordination," *Lecture Notes in Computer Science*, in *Proceedings of the International Conference on Swarm Intelligence*, pp. 216–228, Springer, Barcelona, Spain, October 2020.

[27] T. T. Nguyen, A. Hatua, and A. H. Sung, "Blockchain approach to solve collective decision making problems for swarm robotics," in *Proceedings of the International Congress on Blockchain and Applications*, pp. 118–125, Springer, Seoul, Korea, May 2019.

[28] S. Al-E'mari, M. Anbar, Y. Sanjalawe, S. Manickam, and I. H. Hasbullah, "Intrusion detection systems using blockchain technology: a review, issues and challenges," *Computer Systems Science and Engineering*, vol. 40, no. 1, pp. 87–112, 2022.

[29] F. Baothman, K. Saeedi, and K. Aljuhani, "Computational intelligence approach for municipal council elections using blockchain," *Intelligent Automation & Soft Computing*, vol. 27, no. 3, pp. 625–639, 2021.