Computer Modeling in Engineering & Sciences

Tech Science Press

Doi:10.32604/cmes.2025.072602

ARTICLE

ORTHRUS: A Model for a Decentralized and Fair Data Marketplace Supporting Two Types of Output

Su Jin Shin[1] **and Sang Uk Shin**[2,*]

[1]Department of Information Security, Graduate School, Pukyong National University, Busan, 48513, Republic of Korea
[2]Division of Computer Engineering and Artificial Intelligence, Pukyong National University, Busan, 48513, Republic of Korea

*Corresponding Author: Sang Uk Shin. Email: shinsu@pknu.ac.kr

Received: 30 August 2025; Accepted: 30 October 2025; Published: 26 November 2025

ABSTRACT: To reconstruct vehicle accidents, data from the time of the incident—such as pre-collision speed and collision point—is essential. This data is collected and generated through various sensors installed in the vehicle. However, it may contain sensitive information about the vehicle owner. Consequently, vehicle owners tend to be reluctant to provide their vehicle data due to concerns about personal information exposure. Therefore, extensive research has been conducted on secure vehicle data trading models. Existing models primarily utilize centralized approaches, leading to issues such as single points of failure, data leakage, and manipulation. To address these problems, this paper proposes ORTHRUS, a blockchain-based vehicle data trading marketplace that ensures transparency, traceability, and decentralization. The proposed model accommodates two categories of output data: the original data and the computed result from the function. Additionally, in the proposed model, data owners retain control over their data, enabling them to directly choose which types of data to provide. By employing Multi-party computation (MPC) technique, MOZAIK architecture, and the random leader selection technique, the proposed scheme, ORTHRUS, guarantees the input privacy and resistance to pre-collusion attacks. Furthermore, the proposed model promotes fairness by identifying dishonest behavior among participants by enforcing penalties and rewards through the implementation of smart contracts.

KEYWORDS: Blockchain; privacy; data sharing; MPC

1 Introduction

As digital technologies advance, the volume of data is experiencing exponential growth [1]. Consequently, challenges associated with data collection, storage, and management have become increasingly prominent. In response to these challenges, data sharing and trading have commenced through user interactions. This phenomenon has given rise to the concept of "data space". A data space is defined as a distributed system that is governed by a framework ensuring trust and data sovereignty [2]. It is essential for such a system to provide a wide variety of data types and formats, along with tools for searching, querying, and managing all data contained within the data space. This paper presents a vehicle data space specifically tailored for the Internet of Vehicles (IoV) environment, with a particular focus on vehicle data. The IoV constitutes a dynamic network infrastructure that facilitates the connection of vehicles, users, and various smart devices to the internet [3]. Within the IoV ecosystem, vehicles are capable of collecting and generating diverse types of data, encompassing both vehicle-specific information and environmental data, through interactions with onboard sensors and external factors. Each entity within the IoV ecosystem engages in the sharing or trading of its collected and generated vehicle data, thereby creating new value and enhancing

user services. However, current methodologies for data sharing and trading are predominantly centralized, which presents several challenges, including a single point of failure, data leakage and manipulation, as well as concerns regarding data confidentiality. To mitigate these challenges, research has been undertaken to explore data sharing and trading within a blockchain framework. Blockchain technology enables reliable data trading by providing several benefits, including transparency, traceability, decentralization, and non-repudiation [4,5]. However, security vulnerabilities in the data storage process continue to exist, resulting in challenges such as data leakage, tampering, and breaches of privacy. Furthermore, entities currently seeking to trade vehicle data rely on the data trading marketplace they participate in. This means vehicle data owners lack ownership, control, or management rights over their own data. Consequently, issues such as privacy protection, data confidentiality, and data resale arise. Additionally, relying on a data trading marketplace means adhering to the data provision methods supported by that model. This means that data owners cannot trade their data in the manner they desire. If a data owner desires to trade data in a specific manner, they must seek out a data trading marketplace that supports that method. This hinders user convenience. Furthermore, users seeking diverse data computation results must participate in multiple data trading marketplace. This necessitates providing their information to various marketplaces, thereby creating privacy-related concerns. In this paper, we introduce ORTHRUS, a blockchain-based model for vehicle data trading that prioritizes privacy protection. The proposed model accommodates two categories of output data: the original data and the computed result from the function. Scenarios that necessitate these types of output data include:

- Original data: One of the most common forms of raw vehicle data is video data. For example, video data recorded from dashcams can serve as objective evidence in legal proceedings for various cases, such as actual traffic accidents and vehicle theft. In addition, actual traffic accident videos or image data recorded by the dashcam can be used as training data for autonomous vehicle traffic accident detection models. The dashcam video data that perform these functions must not be forged, altered, or modified, and must remain original.
- The computed result from the function: The computed result from the function is a value obtained from the aggregation of multiple vehicle data points, rather than relying on data from a single vehicle. This is crucial for autonomous driving services, including motion planning, vehicle route prediction, and driver behavior prediction, among others. For instance, to enhance driving security across diverse traffic scenarios, autonomous vehicles employ predictive techniques and decision-making algorithms. To support these algorithms, it is necessary to utilize and compute a variety of vehicle data, including vehicle speed, brake pressure, and driver behavior.

1.1 Our Contributions

The contributions in this paper are as follows:

1. Ensure fairness: The proposed model promotes fairness by identifying dishonest behavior among participants utilizing commitment and signature techniques. Additionally, it enforces penalties and rewards through the implementation of smart contracts.
2. Protecting input privacy: Input privacy refers to the concealment of transaction data supplied by the data owner from both the data requestor and the computation node responsible for executing the computational process [6]. The proposed model limits the access and acquisition of the original data by all participants, allowing only the computation result of a function to be disclosed as the final output. Futhermore, access to the original data is granted to legitimate participants only when the provision of the original data is permitted.

3. Preventing pre-collusion attacks: The proposed model mitigates the risk of pre-collusion attacks by employing random leader selection techniques to randomly designate computation nodes for participation in the computation process. Specifically, it is probably very low to predict in advance that a given node is selected as a computation node. Consequently, even if malicious nodes were to collude, the likelihood of their selection for participation in the actual computation process remains exceedingly low.
4. Ensure data sovereignty: Data sovereignty pertains to the allocation of control over personal data to the individual data owner [7]. In the proposed model, data owners retain authority over their data and possess the ability to determine the method of data delivery.

This paper is structured as follows: Section 2 presents the cryptographic techniques utilized in the proposed model, while Section 3 delineates the operational procedures of the model in a step-by-step manner. Section 4 discusses the implementation of the proposed model, along with the results and analysis derived from this implementation. Finally, Section 5 offers a conclusion to the paper.

1.2 Related Work

Khan et al. [8] introduced BELIEVE, a blockchain platform based on multi-party computation (MPC) [9], which serves as a framework for efficient verification via location identification and encryption. BELIEVE employs an adaptive strategy that empowers data sources to autonomously ascertain the timing of verification and the specifics of their trajectories. Additionally, it utilizes smart contracts to facilitate the verification of mobility data. The platform also guarantees privacy through an MPC-based consensus mechanism among proximate mobile peers, thereby eliminating the need to share original data.

Li et al. [10] introduced a Blockchain-based Speed Advisory System (BSAS), which enhances the existing Consensus-based Speed Advisory System (CSAS) by integrating blockchain technology. This integration aims to establish a trustworthy framework that prioritizes user privacy. BSAS employs cryptographic primitives to safeguard the confidentiality of the CO_2 emission cost function, thereby offering a reliable and fully decentralized vehicle speed recommendation system. Furthermore, it suggests an optimal speed that minimizes emissions and implements a value-driven incentive mechanism to motivate vehicle users to engage with the service.

Golob et al. [6] proposed a decentralized data marketplace that prioritizes input and output privacy within a blockchain framework. This marketplace guarantees payment fairness by automatically compensating data providers with cryptocurrency tokens. Additionally, it employs two privacy techniques—secure multi-party computation (MPC) and robust differential privacy guarantees—to facilitate a privacy-centric data exchange.

Yuan et al. [11] proposed TRUCON, a blockchain-based data sharing mechanism designed to facilitate congestion control and data deduplication within Internet of Vehicles (IoV) environments. TRUCON employs the Kademlia algorithm, enabling source vehicles to restrict the number of reference vehicles involved in the transmission of traffic data. Additionally, it incorporates a Cuckoo Filter-based deduplication technique to mitigate unnecessary data sharing. This approach enhances both the integrity and reliability of data sharing, while simultaneously improving network efficiency.

Sadiq et al. [12] proposed a data trading model based on the Internet of Vehicles (IoV) that guarantees secure data storage and the privacy of vehicles within a blockchain framework. The model employs a digital signature scheme grounded in elliptic curve bilinearity to ensure the authenticity and integrity of the trading data. Additionally, it utilizes smart contracts that are autonomously executed between trading parties to address payment disputes and ensure the fulfillment of payment obligations.

2 Preliminaries

This chapter provides an overview of Multi-Party Computation (MPC) [9] and the MOZAIK architecture [13] as they pertain to ORTHRUS. Additionally, it outlines the fundamental cryptographic primitives utilized in this context.

2.1 MPC

First introduced by Yao in the 1980s, secure multi-party computation (MPC) is a versatile cryptographic technique that enables multiple distributed participants to collaboratively compute arbitrary functions while maintaining the confidentiality of their private inputs and outputs [9]. This process ensures privacy, as each participant remains unaware of the input data of the other participants, and the output of the computation cannot be deduced from the input data of others. Additionally, consistency is assured in that honest participants will obtain identical computational results. The ORTHRUS employs Multi-Party Computation (MPC) grounded in the principles of Secret Sharing, facilitating participant interaction while maintaining data confidentiality. Secret Sharing is a method that protects confidential input data within various MPC frameworks, enabling the division of secret input data or multiple confidential inputs into several shares [14]. This paper adopts the SPDZ protocol as a representative technique based on Secret Sharing for the implementation of MPC.

SPDZ Protocol

In the SPDZ protocol, the values exchanged among all participants are shares produced through a secret sharing technique, which are subsequently utilized for circuit evaluation [15]. The SPDZ framework is comprised of three distinct phases: an offline phase, an online phase, and a phase dedicated to output and verification.

- Offline phase: This phase, commonly referred to as the preprocessing stage, produces a set of shared random numbers utilized for the purpose of masking confidential input data [15]. Additionally, it executes a multiplication operation to create a predominant multiplication triple $(< a >, < b >, < c >)$ and a random mask value.
- Online phase: This phase entails the collaborative computation of arithmetic circuits utilizing secretly shared input data, which encompasses operations such as addition, multiplication, and Message Authentication Codes (MAC) [15].
  1. Addition: Given secretly shared values x, y and a publicly known constant c, each party's local computation performs the following operations.
     - $< x > + < y >$ can be calcuated from $(x_1 + y_1, \ldots, x_n + y_n)$
     - $c \cdot < x >$ can be calculated from $(c \cdot x_1, \ldots, c \cdot x_n)$
     - $c + < x >$ can be calculated from $(c + x_1, x_2, \ldots, x_n)$
  2. Multiplication: Given a value that is shared in secrecy, the objective is to compute the product of these values, which includes the multiplication of triples. We define a multiplication triple as the ordered set $(< a >, < b >, < c >)$, where $(< a >, < b >, < c >)$ represents the secret numbers that have been preshared among the participants. Furthermore, it is assumed that the multiplication triples are generated in advance during an offline phase.
     - Compute $<\epsilon>=< x > - < a >$, then reveal $<\epsilon>$
     - Compute $< \delta >=< y > - < b >$, then reveal $< \delta >$
     - Finally, compute $< x > \cdot < y >=< c > + <\epsilon> \cdot < b > + < \delta > \cdot < a > + \ \epsilon \cdot \delta$
  3. MAC: The SPDZ protocol employs the Message Authentication Code (MAC) technique to protect messages from dishonest parties [15]. The MAC key is established as a global variable that remains

unknown to the participants and is shared in a confidential manner. All secret data are encoded in a manner that incorporates the MAC, rendering the resultant value challenging to alter without the consensus of the other participants. Consequently, by verifying the MAC, it is possible to detect dishonest behavior and ensure the integrity of the data.

- Output and Verification phase: This phase involves the verification of all published numerical data and final output values through the use of Message Authentication Code (MAC). Verification can be conducted in batches utilizing a randomized linear combination. In the event of a discrepancy during the MAC verification phase, the protocol will be terminated.

2.2 MOZAIK Architecture

The primary algorithms of the MOZAIK architecture are developed utilizing secret sharing techniques [13] and the AEAD (Symmetric Authenticated Encrypted with Associated Data) algorithm [16].

The secret sharing mechanism employed in the MOZAIK architecture is based on threshold-based secret sharing. This method involves partitioning a single source of data into n shares, which are then distributed among the participants. The original data can be reconstructed only when the number of gathered shares exceeds a certain threshold. Two algorithms that facilitate this process are the partitioning algorithm $share(d) \rightarrow [[d]]_1, \ldots, [[d]]_n$ and the reconstruction algorithm $Recon([[d]]_1, \ldots, [[d]]_n) = d$. Here, d are the original data, while n denotes the number of participants.

The AEAD [16] is a cryptographic technique that combines encryption of the original data with the generation of an authentication tag alongside the ciphertext. This authentication tag serves to verify both the integrity and confidentiality of the data. The underlying algorithm is represented by the encryption algorithm $Enc(k, d) \rightarrow (c, t)$ and the decryption algorithm $DecAndVerify(k, Enc(k, d)) = d$. Where k is a symmetric key, c represents the ciphertext, and t signifies the authentication tag.

The main algorithms of the MOZAIK architecture based on this are the key generation algorithm $keyGen()$, the distributed decryption algorithm $DistDec()$, and the computation algorithm $Infer()$.

- $KeyGen()$: The algorithm uses the symmetric key k and the partitioning algorithm $share()$ to generate the appropriate share of keys $k_1, \ldots, k_n$ for the number of compute nodes of MPC.
- $DistDec()$: We use the ciphertext c and the authentication tag t as a public input, and the key share k_i as a private input provided to each participant. The algorithm verifies the authentication tag t, decrypts the ciphertext c, and returns the plaintext share $d_1, \ldots, d_n$. If verification of the authentication tag t fails, it returns $\perp$. This distributed decryption is implemented by running $DecAndVerify(k_i, c, t) \rightarrow d_n$.
- $Infer()$: It analyzes data from share $d_1, \ldots, d_n$ in the original data. Then, it performs some computation based on the analyzed values to obtain the resulting values $x_1, \ldots, x_n$.

2.3 Cryptographic Primitives

The proposed model applies the following cryptographic primitives.

1. A cryptographically secure hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$, where λ is the security parameter [17].
2. A semantically secure symmetric cipher consisting of $(SymKGen, SymEnc, SynDec)$ [18].
3. An asymmetric cipher that is semantically secure against a chosen ciphertext attack consisting of $(AsymKGen, AsymEnc, AsymDec)$ [19,20].
4. An existential unforgeability under chosen message attack (EU-CMA) secure electronic signature technique consisting of a polynomial-time algorithm $(SignKGen, SignVerify)$ [21].

3 ORTHRUS

In this chapter, we introduce ORTHRUS, a model for vehicle data transactions that preserves privacy while accommodating two types of output data within a public blockchain framework. Fig. 1 shows an overview of the model proposed in this paper.

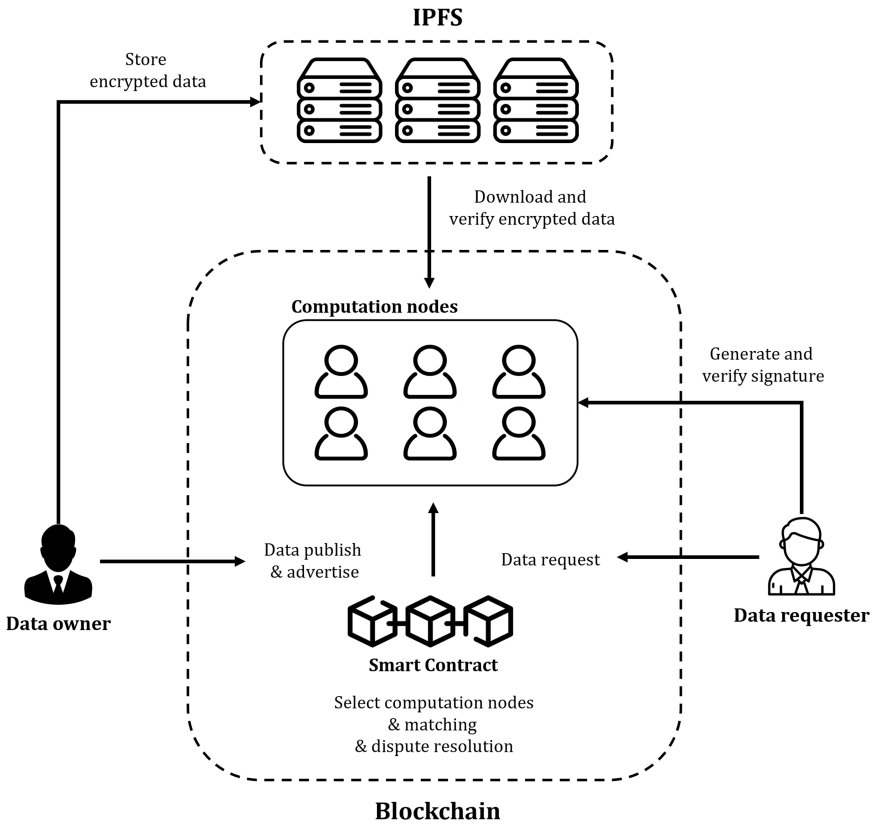

Figure 1: Overview of the proposed model

The ORTHRUS framework comprises several key components, including the Data Owner (*DO*), Data Requester (*DR*), Computation Node (*CN*), Smart Contract (*SC*) and the Interplanetary File System (IPFS). The *DO* is a participant who deploys the smart contracts defined in the proposed model on the blockchain and engages in trading their own data. Conversely, the *DR* refers to a participant who, within the blockchain peer-to-peer network, acquires data from a *DO* holding the required information in exchange for a fair price. The *CN*s are the entities actively engaged in computation, selected at random by the smart contract to participate in the computational process. These nodes serve as actual computation participants and are incentivized based on the correctness of their computations at the conclusion of the transaction. Each time a transaction is initiated, these *DO*, *DR*, and *CN* generate key pairs for transactions and signatures, respectively, utilizing a cryptographically secure public-key algorithm. The proposed model is designed based on a blockchain. The blockchain used here supports *SC*. The *SC* is a stateful ideal functionality, allowing all components, including attackers, to expose their internal state. Furthermore, the *SC* selects *CN* to participate in the actual computation process by employing a randomized leader selection technique, such as that used in Algorand [22]. Upon receiving a dispute resolution request from component of the proposed model, the *SC* initiates a dispute resolution process to identify dishonest participants, who are then subject to penalties such as deposit forfeiture and a reduction in their reputation score. The ORTHRUS uses IPFS, a scalable

peer-to-peer data storage platform [8]. To mitigate storage costs, we store ciphertexts and encrypted secret key shares within IPFS and subsequently publish the addresses returned by IPFS on the blockchain [12].

3.1 Assumptions and Threat Model

The ORTHRUS is built on top of a public blockchain, thereby inheriting security properties such as integrity, tamper-resistance, and others from the underlying blockchain platform that hosts the smart contracts. The system assumes a synchronous network model with authenticated peer-to-peer channels, which limits the adversary's ability to manipulate communication [23]. Specifically, while an adversary may delay any message passed between honest parties by a known upper bound $\triangle$, they cannot delete, reroute, or modify the message. The threat model considered in this paper is defined as follows:

- The main components of ORTHRUS, *DO*, *DR*, and *CN*, do not trust each other and are assumed to act in their own self-interest and act rationally accordingly.
- The *CN*, which performs the actual computation and validation in ORTHRUS, is not a fully trusted entity.
- In order to use the randomized leader selection technique in ORTHRUS, it is assumed that the number of nodes on the blockchain network that want to participate in the computation process is sufficient, and the randomized leader selection technique is also assumed to be secure because it inherits the security of the technique used.
- We consider only static corruption [24] and standalone [25] models, where an adversary can only compromise any object before the protocol is executed, that is, an adversary can only compromise a protocol participant before the protocol is executed.
- The cryptographic techniques mentioned in Section 2 are assumed to be secure against a Probabilistic Polynomial Time (P.P.T.) attacker.
- The security of MPC technique applied to ORTHRUS is guaranteed in the presence of subthreshold adversarials by relying on admissible Threshold Adversarial Structures to ensure the security of MPC. This ensures output correctness and input privacy. Furthermore, the accuracy and verifiability of the output results depend on the applied MPC technique. In the proposed model, the threshold or the protocol's computation process can change based on the attacker model considered by the applied SPDZ protocol.

The ORTHRUS employs a reputation mechanism to enhance fairness and prevent Byzantine behavior [26]. The following studies exist on reputation mechanisms. Zhang et al. [26] proposed a blockchain-based Cyber Threat Intelligence (CTI) model combining consortium blockchains with distributed reputation management systems. This model prevents bribery attacks and offers faster performance than existing Byzantine Fault Tolerance (BFT) models. Liu et al. [27] proposed BtRal, a blockchain and trust-based reputation evaluation incentive mechanism. It introduced a multidimensional reputation evaluation method. It prevents malicious or pre-collusive evaluation attacks using multifactor moderation. It also designed and proposed rewards and penalties through reputation-based tokens. While reputation mechanisms themselves are an important research topic actively studied in the field of data marketplaces to enhance fairness and security, this paper focuses on secure and trustworthy data trading. Existing, well-established reputation mechanisms can be utilized to implement the proposed model. However, the specific details of the reputation mechanism itself are beyond the scope of this paper.

3.2 ORTHRUS Architecture

The ORTHRUS system begins with the deployment of the *DO* smart contract, followed by pre-processing, matching, and the actual computation steps for data trading. Since ORTHRUS accommodates

two types of output data, the internal behavior during the computation phase is contingent upon the output data type requested by the *DR*. In this paper, the output data type refers to the specific data type requested by the *DR*, which can be categorized into two types: original data ($flag_{XX} = 1$) and the computation result for the requested function ($flag_{XX} = 0$). When a smart contract receives a request for dispute resolution from a *DO* or a participant in the transaction process, such as *DR* or *CN*, it initiates a dispute resolution step, and identifies the dishonest participant and penalizes them by confiscating their deposit and reducing their reputation score. The detailed process for each step in the ORTHRUS is discussed in Appendix A. Fig. 2 shows the processing flow of transactions and state variables handled by smart contracts in the proposed model.

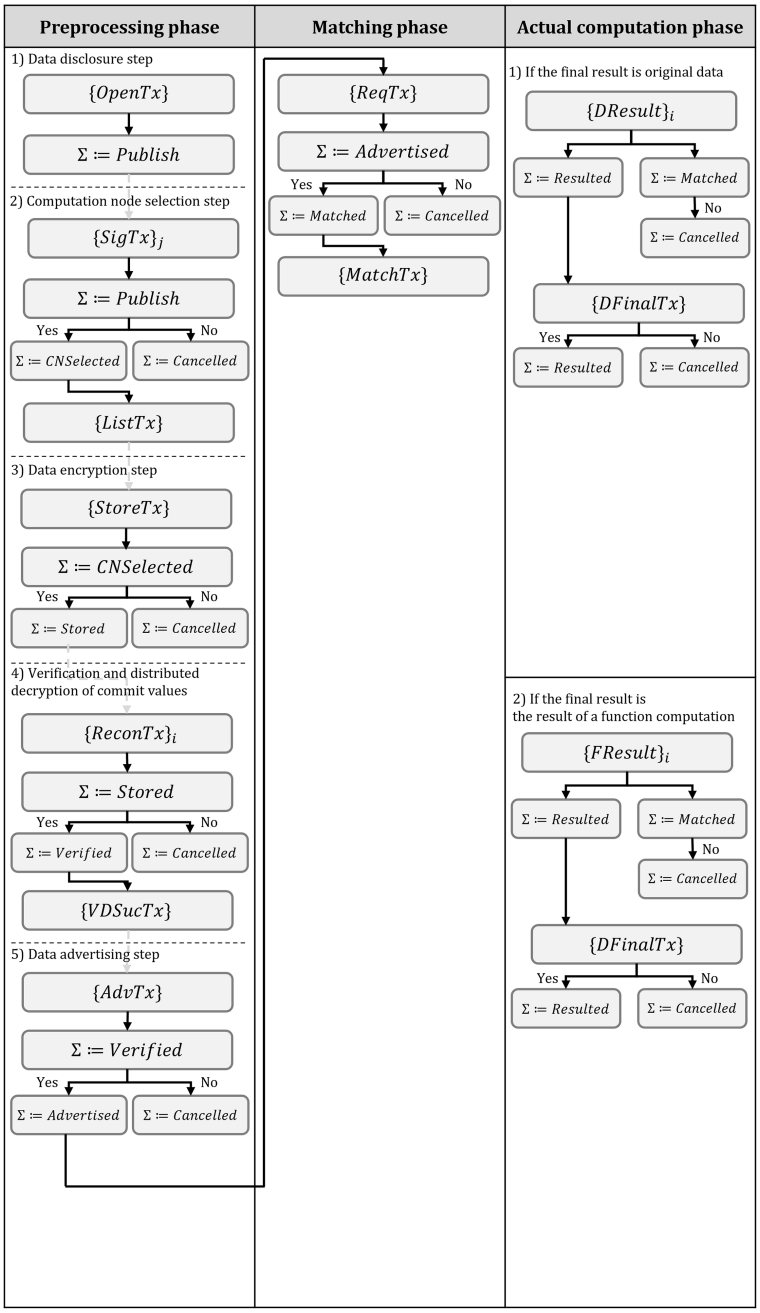

Figure 2: The processing flow of transactions and state variables handled by smart contracts in the proposed model

3.2.1 Preprocessing Phase

The preprocessing phase consists of a data disclosure step, a compute node selection step, a data encryption step, a commit value verification and distributed decryption step, and a data ad step. The detailed algorithms for each step are described in Appendix A.

- Data disclosure step:

  The *DO* defines the data it wishes to trade as the original data D and publishes a transaction $\{OpenTx\}$ on the blockchain through the *SC*. This transaction includes the data provision method $flag_{DO}$ and information about D. In this context, $flag_{DO} = 1$ indicates that the *DO* provides both D and the result of function computation as the final output of the proposed model, while $flag_{DO} = 0$ signifies that the *DO* provides only the result of function computation.
- Computation node selection step:

  Among all the participating nodes BN_j, $j \in \{1, \ldots, M\}$ in the blockchain, only those nodes that wish to engage in the computation process generate their signatures based on $\{OpenTx\}$. Then, BN_j forwards $\{SigTx\}_j$ with its signature to the *SC*. Here, M represents the total number of nodes participating in the blockchain network. The *SC* receives this information and generates a hash value based on the signature. It then calculates this as a probability value and compares it to the reference probability value P. The *SC* selects the N nodes with the smallest values among those that are less than or equal to P as the computing nodes. The *SC* then updates the information of the selected nodes to the list L of computing nodes. Finally, the *SC* publishes $\{ListTx\}$, which contains L and various other information, to the blockchain network. After this step, the nodes designated as computing nodes are denoted as CN_i, $i \in \{1, \ldots, N\}$, where N is the total number of selected computation nodes.
- Data encryption step:

  The *DO* generates a secret key K using the $KeyGen()$ function, which is one of the key algorithms in the MOZAIK Architecture. Based on this key, it creates secret key shares K_i for $i \in \{1, \ldots, N\}$. Next, the *DO* encrypts each secret key share K_i using a public key encryption algorithm to produce the encrypted secret key share EK_i, and it generates a commit value Com_{EK_i} for verification purposes. The *DO* then creates a ciphertext *ED* using the AEAD encryption algorithm $AEAD.Enc()$ and generates a corresponding commit value Com_{ED}. For each pair of EK_i and Com_{EK_i}, the *DO* stores them in IPFS and receives the storage location URI_{EK-Com}. Subsequently, the *DO* stores *ED* and Com_{ED} in IPFS, obtaining the storage location URI_{ED-Com}. Finally, the *DO* submits a transaction $\{StoreTx\}$ to the *SC*, which includes this information.
- Verification and distributed decryption of commit values:

  The CN_i obtains the values stored in IPFS based on what is published on the blockchain. Then, the CN_i generates the commit value Com'_{ED} for *ED* and checks if it matches the value obtained via URI_{ED-Com}. If there is a mismatch, the CN_i sends $\{FailEDTx\}_i$, $i \in \{1, \ldots, N\}$ to the *SC* to resolve the dispute, and if it matches, the CN_i generates a commit value Com'_{EK_i} for EK_i and performs the same process. Similarly, if there is a mismatch, the CN_i requests the *SC* to resolve the dispute by sending $\{FailEKTx\}_i$. If they match, the CN_i performs MOZAIK Architecture's $DistDec()$ on *ED* to obtain $D_i / \perp \leftarrow DistDec(K_i, ED)$. The CN_i sends Success to $\{ReconTx\}_i$, $i \in \{1, \ldots, N\}$, if D_i is obtained, and Fail to the *SC*, if $\perp$ is obtained.
- Data advertising step:

  To sell its data, the *DO* sends an advertising transaction $\{AdvTx\}$ containing information about its data to the *SC*, which then publishes it on the blockchain.

3.2.2 Matching Phase

First, the *DR* selects a tra nsaction that contains the data it wants and how it wants the data to be provided from among the $\{AdvTx\}$ posted on the blockchain and delivers a data request transaction $\{ReqTx\}$ to the *SC*. At this time, the *DR*′ request function, *F*, is included only if the *DR* wants data as a result of the function calculation ($flag_{DR} = 0$). Upon receiving this, *SC* checks whether the *DR*′s account balance is sufficient and checks whether the data type provided by the *DO* matches the data type desired by the *DR*. If it does, it posts a matching transaction $\{MatchTx\}$ on the blockchain network. At this time, a parameter called $flag$ defines the behavior of the proposal model. $Flag = 1$ means that the final output of the proposal model will be the original data, while $flag = 0$ means that the output of the function will be the result of the calculation.

3.2.3 Actual Computation Phase

The steps are described separately for the cases where the final result is the original data and for the cases where the final result is the result of a function computation.

1. If the final result is original data ($flag = 1$)
   This means that when $flag_{DO} = flag_{DR} = 1$, the final output of the proposed model is the original data type.
   - Encryption and signature generation step:
     $CN_i, i \in \{1, \ldots, N\}$ performs public key encryption on D_i to generate encrypted data ED_i. Then, it generates a signature $Sign_{ED_i}$ for it and publishes the transaction $\{DResult\}_i$ containing it on the blockchain.
   - Verification step:
     The DR obtains ED_i and Sig_{ED_i} through the published transaction $\{DResult\}_i$, $i \in \{1, \ldots, N\}$. Based on this, *DR* performs the signature verification algorithm $Verify()$ to verify the signature. If verification fails, *DR* identifies the CN_i that caused the result, records the information in $VCNlist$, and delivers the verification failure transaction $\{DVerifySigTx\}$ to *SC*. The receiving *SC* performs the dispute resolution steps. If verification is successful, *DR* decrypts ED_i and obtains D_i'. The *DR* then performs the reconstruction algorithm $Recon()$ to obtain $D^{'}$. The *DR* generates a hash value $H(D')$ for the obtained $D^{'}$ and checks if it matches the $H(D)$ contained in $\{AdvTx\}$. If there is a mismatch, *DR* returns $\{DFailTx\}$ to *SC* to resolve the dispute. If they match, *DR* passes $\{DTrueTx\}$ to *SC*.
2. If the final result is the result of a function computation ($flag = 0$) This means that $flag_{DO} = 1$ or 0 and $flag_{DR} = 0$. The final output of the proposed model is of the type function computation Output.
   - Function computation step:
     $CN_i, i \in \{1, \ldots, N\}$ generates a function computation result share X_i with D_i as input to *F*. The CN_i creates a computation result share EX_i encrypted with public key encryption on X_i. The CN_i generates a signature Sig_{EK_i} for the generated value. The CN_i then publishes a transaction $\{FResult\}_i$ containing this information on the blockchain.
   - Verification and reconstruction step:
     DR obtains EX_i and Sig_{EX_i} from $\{FResult\}_i$, $i \in \{1, \ldots, N\}$ posted on the blockchain and performs signature verification on them. If verification fails, *DR* identifies the CN_i that caused the result and forwards the transaction $\{FVerifySigTx\}$ that contains it to *SC*. The receiving *SC* performs the dispute resolution steps. If verification is successful, the DR decrypts EX_i to obtain X_i. The *DR* then aggregates X_i over a threshold to obtain X through the reconstruction algorithm $Recon()$. The *DR* delivers $\{FfinalTx\}$, the transaction resulting from the reconstruction, to *SC*.

Upon receiving it, *SC* distributes data payments and incentives from the DR' account to each participant, increases each participant's reputation score, and terminates the transaction.

3.2.4 Dispute Resolution Phase

The dispute resolution phase is performed by participants requesting *SC* to resolve disputes in the event that (1) ED's commit value verification fails, (2) EK_i's commit value verification fails, (3) ED_i's signature verification fails, (4) D's hash value verification fails, or (5) EX_i's signature verification fails. The detailed dispute resolution process is discussed in Appendix B.

1. ED's commit value verification fails

   This step is the dispute resolution request phase triggered by CN_i, where $i \in \{1, \ldots, N\}$, due to a failed verification of the ED's commit value. At this point, CN_i transmits $\{FailEDTx\}_i$ to the *SC*. Upon receiving this, the *SC* verifies whether the transaction's timestamp is within T_{Store} and performs verification for up to $i \in \{1, \ldots, l\}$ transactions that arrived within T_{Store} and are below the threshold. After verifying all l transactions, the *SC* changes the state variable Σ to *'Cancelled'* and terminates the transaction only if a dishonest DO's record exists or if the value of mCN_i exceeds the threshold. If a dishonest DO's record exists, the *SC* confiscates that DO's deposit and distributes it among the honest participants. Then, the *SC* lowers the reputation score of that *DO* and terminates the trading.
2. EK_i's commit value verification fails

   This step is the dispute resolution request phase triggered by CN_i, where $i \in \{1, \ldots, N\}$, due to a failed verification of the commit value EK_i. The CN_i transmits $\{FailEKTx\}_i$ to the *SC*. Upon receiving this, the *SC* verifies whether the transaction's timestamp is within T_{Store}. The *SC* performs verification on up to l transactions where $i \in \{1, \ldots, l\}$ and the transactions arrived within T_{Store}. After verifying all l transactions, the *SC* handles the situation identically to the ED's commit value failure scenario.
3. ED_i's signature verification fails

   This step is the dispute resolution request phase triggered by DR's signature verification failure for ED_i, where $i \in \{1, \ldots, N\}$. The *DR* transmits $\{DVerifySigTx\}$ to the *SC*. Upon receiving this, the *SC* verifies the corresponding $i \in \{1, \ldots, l\}$ transactions by checking whether the transaction timestamp is within $T_{DResult}$ and whether the number of CN_i included in $VCNlist$ is below the threshold. After verifying all l transactions, the *SC* changes the state variable Σ to *'Cancelled'* and terminates the trading only if there exists a record from a dishonest *DR* or if the value of mCN_i is above the threshold. If a record from a dishonest *DR* exists, the *SC* confiscates that DR's deposit and distributes it to the honest participants. Then, the *SC* lowers the reputation score of the *DR* in question and terminates the trading.
4. D's hash value verification fails

   This step is the dispute resolution request phase triggered by DR's failure to verify D's hash value. The *DR* transmits $\{DFailTx\}$ to *SC*. Upon receiving this, *SC* verifies whether the transaction's timestamp falls within T_{finish} and performs validation for each transaction. If *DR* denies, *SC* confiscates DR's deposit and lowers their reputation score. Then, the *SC* changes Σ to *'Cancelled'* and terminates the trading. If the *DO* denies, the *SC* performs the same process and then terminates the trading.
5. EX_i's signature verification fails

   This step is the dispute resolution request phase triggered by DR's signature verification failure for EX_i where $i \in \{1, \ldots, N\}$. *DR* transmits $\{FVerifySigTx\}$ to *SC*. Upon receiving this, *SC* verifies whether the transaction's timestamp is within T_{Data} and whether the number of CN_i included in $VCNlist$ is below the threshold. Then, the *SC* performs verification for the transactions where $i \in \{1, \ldots, l\}$. After verifying all l transactions, the *SC* proceeds identically to the ED_i's signature verification fails step.

4 Implementations and Analysis of ORTHRUS

In this chapter, we design the ORTHRUS based on the Ethereum blockchain network environment to examine gas usage. We also describe security analysis, such as fairness and resistance to pre-collusion attacks.

4.1 Implementation

In this paper, we designed ORTHRUS based on the Ethereum blockchain network environment with smart contracts. We utilized the Ganache test network to create accounts for participants and deploy smart contracts in ORTHRUS. In ORTHRUS, smart contracts perform transaction validation, dispute resolution steps, punishments, and rewards. We used the Solidity language to implement ORTHRUS' smart contracts that operate in an on-chain environment, and JavaScript for the rest.

The ORTHRUS implementation consists of an off-chain part that generates signatures, a main smart contract part, and a smart contract part that performs dispute resolution steps. The smart contract part that connects to the Ganache environment and performs the main operation is deployed first, followed by the smart contract that performs the dispute resolution step. Various signature values used in the dispute resolution process are generated through the off-chain part using the Elliptic Curve Digital Signature Algorithm (ECDSA) method, and the smart contract uses them to perform verification. In addition, hash values for signatures in the main smart contract part are generated using keccask256.

4.1.1 Analysis of ORTHRUS Implementation Result

In this paper, N computational nodes are randomly selected from the nodes participating in the blockchain network, and the selected computational nodes perform cryptographic processing and computation of functions based on the MPC technique. To implement this, in the ORTHRUS implementation program, the total number of nodes participating in the blockchain network is set to 15, and the number of computing nodes to be selected, N, is set to 3 and 5 to compare the MPC computation efficiency and the smart contract gas cost, respectively. The analysis is limited to on-chain efficiency.

First, the gas usage in the main smart contract part is 1,875,501 when the actual number of compute nodes is 3. If we set the gas price at 20 gwei to calculate the total cost, 0.03751002 ETH is consumed. The gas usage of the smart contract that performs the dispute resolution step is 2,029,864, and if we set the gas price to the same value to calculate the total cost, 0.04059728 ETH is consumed. When the actual number of computing nodes is 5, the gas usage of the main smart contract is 1,979,069, for a total cost of 0.03958138 ETH. The gas consumption of the smart contract that performs the dispute resolution step is 2,586,162, totaling 0.05172324 ETH. From this we can see that as the number of computing nodes increases, the gas usage and cost of deploying smart contracts increases. Figs. 3 and 4 show the results of gas usage as a function of the number of compute nodes. Fig. 5 presents a quantitative comparison based on Figs. 3 and 4.

The main computational overhead off-chain is the distributed decryption process between peers, which involves the computation of AEAD′s $DecAndVerify()$ algorithm over MPC. Among the existing works on this algorithm, reference [28] introduces various modes of distributed decryption. Among them, the Jolteon-AES and Espeon-AES modes are introduced as the best performers. Based on the communication between two participants in [28], we analyzed the offline and online execution times of distributed decryption according to the message length, and found that the Jolteon mode has the best performance and practicality, with an 8 bytes message taking about 2.5 s and a 128 bytes message taking about 6.5 s. In addition, the time to decrypt a 128-byte message in Jolteon mode was measured to be about 0.07 s for 3 players and 0.11 s for 5 players. Based on this, if ORTHRUS also performs distributed decryption in Jolteon mode, assumes a message length of 128 bytes, and sets the number of computational nodes as players to 3 and 5, the

decryption time is expected to be about 0.07 and 0.11 s, respectively, which is sufficiently feasible for a real system environment.

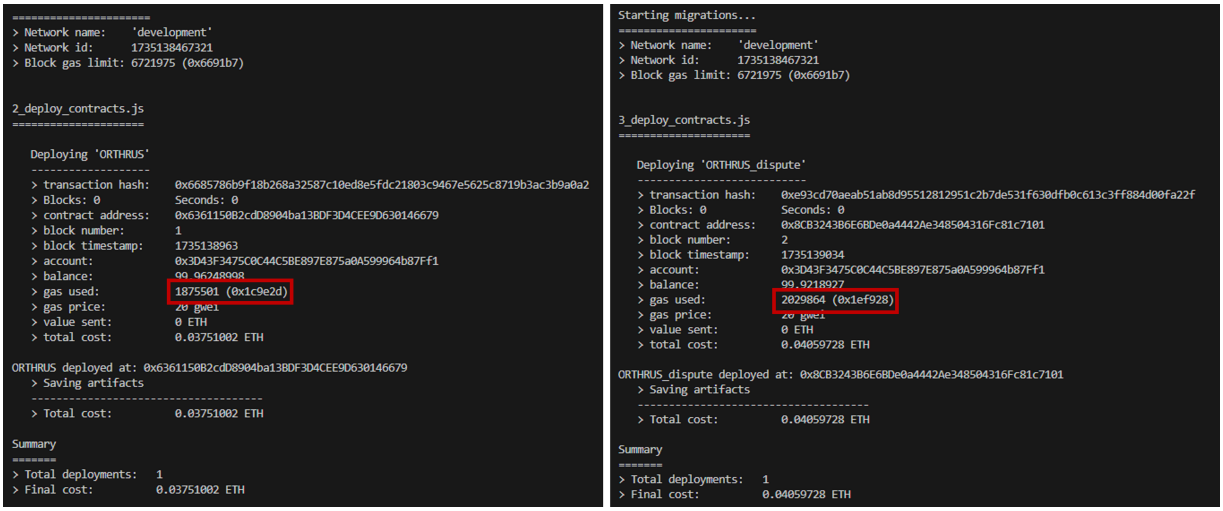

Figure 3: When the number of computation nodes is 3, gas usage

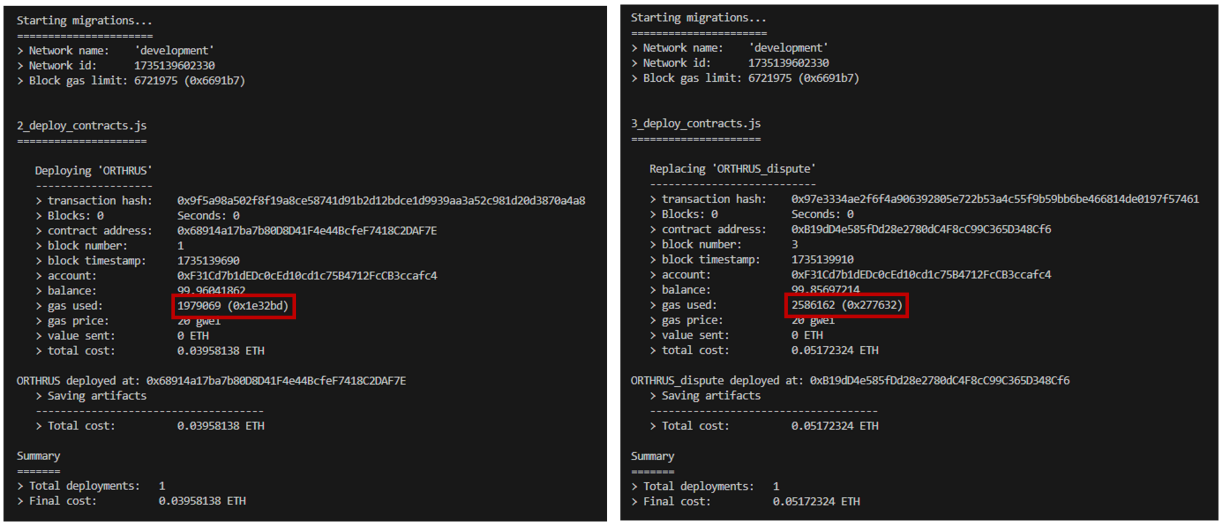

Figure 4: When the number of computation nodes is 5, gas usage

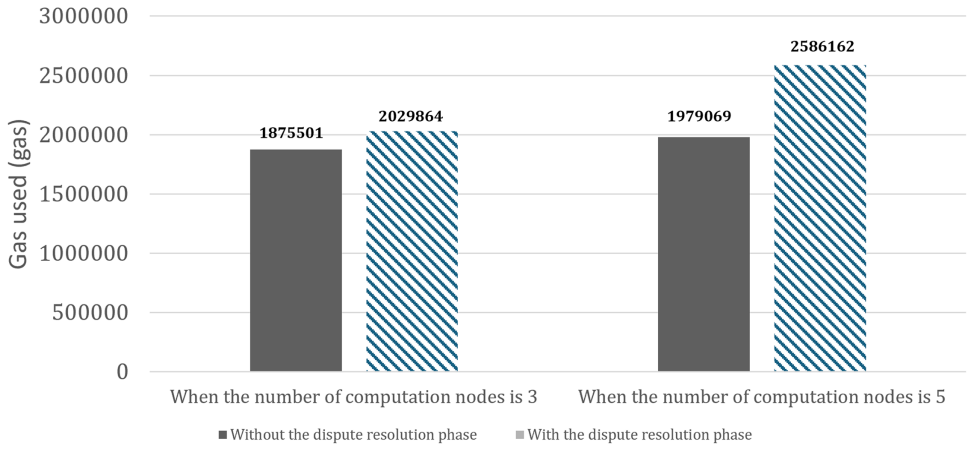

Figure 5: Comparison of gas usage based on the number of computation nodes

4.1.2 Anaylsis of ORTHRUS Storage Overhead

This section analyzes the storage overhead of ORTHRUS on the blockchain. The following data is stored on-chain for each phase.

1. Pre-processing phase:
   - Data Publishing Step: $(OpenTx_{ID}, ID_{DO}, PK_{DO}, sPK_{DO}, T_{Open}, T_{Apply}, meta, flag_{DO}, P, N)$
   - Computation node selection step: $(ListTx_{ID}, T_{List}, L, OpenTx_{ID})$
   - Data Encryption Step: $(StoreTx_{ID}, ID_{DO}, T_{Store}, URI_{ED-Com}, URI_{EK-Com}, OpenTx_{ID})$
   - Verification and distributed decryption of commit values: $(ReconTx_{ID}, ID_{CN_i}, Success/Fail, StoreTx_{ID}, OpenTx_{ID})$, $(VDSucTx_{ID}, OpenTx_{ID}, StoreTx_{ID})$
   - Data advertising step: $(AdvTx_{ID}, ID_{DO}, T_{Adv}, meta, reP_{DO}, Per_{DO}, Price_O, Price_F, Deposit_{DO}, flag_{DO})$
2. Matching phase: $(ReqTx_{ID}, ID_{DR}, PK_{DR}, sPK_{DR}, reP_{DR}, Per_{DR}, Deposit_{DR}, Inc, flag_{DR}, F, AdvTx_{ID})$, $(MatchTx_{ID}, ID_{DO}, ID_{DR}, T_{finish}, flag, AdvTx_{ID}, ReqTx_{ID}, StoreTx_{ID})$
3. Actual computation phase:
   - The output result is the original data: $(DResult_{ID}, ID_{CN_i}, ED_i, Sig_{ED_i}, T_{DResult}, MatchTx_{ID})$, $(DTrueTx_{ID}, ID_{DR}, MatchTx_{ID})$
   - The output result is the result of a function computation: $(FResult_{ID}, ID_{CN_i}, T_{FResult}, EX_i, Sig_{EX_i}, MatchTX_{ID})$, $(FfinalTx_{ID}, ID_{DR})$

Note that the values stored during the actual computation step are only those values that would be stored if the verification result is successful.

To compute storage complexity, we assume that the proposed protocol has key lengths of cryptographic primitives with 128-bit security; that is, we can choose bit lengths for 128-bit symmetric ciphers, 256-bit hash functions, and 256-bit elliptic curve pairings. Accordingly, in the proposed protocol PK is 512 bits long and SK is 256 bits long. The signature is generated with a length of 512 bits because it utilizes the ECDSA scheme. The *ID* of the participant and the *ID* of the transaction are assumed to be 160 bits, the $flag$ is 1 bit because it is either 0 or 1, the probability is 32 bits and the price, various timestamps, and integer values are 64 bits. In addition, the meta-data published in the data disclosure step contains the hash value $H(D)$ of D, so we define the hash function as 256 bits. In addition, the encrypted values have ED_i and EX_i, which are variable depending on the length of D and which F is computed, so we exclude the values of ED_i and EX_i to compute the total storage overhead.

The storage overhead of the preprocessing step is 5026 bits, the matching step is 2786 bits, and the final result is 1536 bits if the final result is the original data and 1376 bits if the final result is the result of a function computation. Therefore, the on-chain storage overhead seems to be a bit high, as 9348 bits (about 1.14 KB) are required when the final result is the original data, and 9188 bits (about 1.12 KB) are required when the final result is the result of a function computation. However, storage overhead can be reduced by using off-chain storage, that is, storing the actual data in off-chain storage while keeping its metadata on-chain. For example, if the output result is the original data in the actual computation step, the storage overhead can be reduced by storing only $DResult_{ID}$, ID_{CN_i} the location of the stored data (the address of the off-chain storage), and the hash value of the stored data off-chain on the on-chain and storing the remaining values on the off-chain.

4.2 Security Analysis

This section describes the security of the ORTHRUS with the following lemmas and theorem.

Lemma 1. *If all participants are honest, ORTHRUS satisfies completeness based on the assumptions and threat model in Section 3.1.*

Proof. The completeness of the ORTHRUS is guaranteed when *DO*, *DR*, and *CN* follow the protocol with honesty.

- Case 1: If *DR* requests the original data as the final result and all the participants are honest, *DR* will obtain D_i' during the validation of the actual computation step of the proposed model. The *DR* collects and reconstructs the obtained D_i' above the threshold to obtain $D^{'}$, and delivers $\{DTrueTx\}$ to the SC as a result of hash value verification. The *DO* receives $Price_O$ for the data, and each *CN* receives a share of Inc as a reward for computing the data correctly.
- Case 2: If *DR* requests the output of the function computation as the final output and all participants are honest, then during the verification of the actual computation step of the proposed model, *DR* will obtain X_i'. The *DR* aggregates the obtained X_i' over a threshold and reconstructs them to generate X, and then delivers $\{FfinalTx\}$ to *SC*. The *DO* receives the payment for the data $Price_O$ or $Price_F$, and each *CN* receives a share of the incentive Inc.

□

Lemma 2. *Assuming that the random leader selection technique applied in ORTHRUS is secure, based on the assumptions in Section 3.1 and the threat model, ORTHRUS is secure against prior collusion attack between the attacker and each participant.*

Proof. In a proposal model, an attacker can only compromise a participant before executing the protocol in that model. Therefore, in order to perform a pre-collusion attack, the attacker must select the blockchain nodes to perform the collusion attack before the proposal protocol is executed. Furthermore, the selected blockchain node must be selected as a *CN* participant in the actual computation of the proposed protocol. However, based on the unpredictability of the output of the hash function and the signature technique applied in the proposal protocol, it is computationally difficult for an attacker to predict which blockchain nodes will be selected as *CN*s before the proposal protocol is executed. Furthermore, the collusion attack among *CN*s during the execution of the proposal protocol is based on the security of Algorand [22], the random leader selection technique used for the selection of *CN*. In other words, if the hash function, signature technique, and Algorand [22] are secure, the probability of success of the attacker's collusion attack with the blockchain nodes in the proposed model is negligible. ORTHRUS also allows collusion between *CN*s and *DO*s and collusion between *CN*s and *DR*s.

- Case 1. CN_i, $i \in \{1, \ldots, N\}$ and *DO* colluding in advance.

During the validation of the commit value in the preprocessing phase, it can try to manipulate the validation results of Com_{ED} and Com_{EK_i}. However, CN_i cannot decrypt EK_i and ED during the distributed decryption process, so a collusion attack is not possible.

- Case 2. CN_i and DR colluding in advance.

  The DR in the actual computation phase may attempt to manipulate the verification result while verifying Sig_{ED_i} or Sig_{EK_i}. In this case, in order to manipulate the signature verification result, the DR must colludes with CN_i above the threshold. However, since the proposed model relies on the attacker threshold structure, which is a fundamental characteristic of MPC, the probability that a DR collides with a CN_i above the threshold is negligible.

As such, the probability of such a pre-collusion attack succeeding against ORTHRUS is negligible if the applied hash function, signature technique, and random leader selection technique Algorand [22] are secure. □

Lemma 3. *ORTHRUS assumes that the cryptographic techniques underlying the proposed model are secure, given the assumptions in Section 3.1 and the threat model. Thus, ORTHRUS satisfies fairness even if DO or DR is compromised by a nonadaptive P.P.T. adversary.*

Proof. Fairness satisfies both DR fairness and DO fairness.

- DR fairness: DR fairness means that even in the presence of dishonest DOs, honest DRs only pay for validly acquired data. We assume that an attacker can compromise the DOs that provide data to the honest DR and the CNs that help in the computation. The following are scenarios in which an attacker compromises DO or CN.
  1. To disrupt the normal flow of the protocol, an attacker may attempt to corrupt $CN_i, i \in \{1, \ldots, N\}$ to intentionally send dispute resolution requests $\{FailEDTx\}_i$ or $\{FailEKTx\}_i$. However, SC that receives the dispute resolution request can determine that the dispute resolution request was intentionally sent by CN_i by generating and comparing the commit values. Therefore, fairness is ensured by SC penalizing CN_i.
  2. An attacker could attempt to compromise DO so that an invalid Com_{ED} or Com_{EK} is stored in IPFS. In this case, the validation process of CN_i's commit value will result in a dispute resolution request, and SC can perform the dispute resolution process to determine whether DO has stored the wrong values. Thus, fairness is ensured by allowing SC to punish DO who performed the malicious behavior.
  3. An attacker could compromise DO, causing it to store an invalid EK_i in URI_{EK-Com} or not disclose the correct location where EK_i is stored. As a result, a $DistDec()$ on $CN_i, i \in \{1, \ldots, N\}$ will result in $\perp$, which will pass $\{ReconTx\}_i$ with "*Fail*" to the SC. Upon receiving it, SC determines that DO is malicious and punishes the DO to ensure fairness.
  4. An attacker can compromise DO, causing DO to release an incorrect $H(D)$ to the blockchain. In response, DR will send $\{DFailTx\}$ to SC as a sign of verification failure during the verification process of the actual computation phase, and SC will carry out the dispute resolution phase. Through the dispute resolution process, SC determines that DO has posted an incorrect $H(D)$ and determines that DO is a malicious actor. This ensures fairness by penalizing the malicious DO.
- DO fairness: DO fairness means that DO who act honestly receives a fair amount of money for the valid data they provide to the DR. In response, an attacker can compromise the fairness of the honest DO by allowing DR and CN, who did not pay the legitimate payment, to obtain the data provided by DO in the process of trading.

1. The attacker can attempt to compromise the *CN* and obtain the original data *D*. $CN_i, i \in \{1, \ldots, N\}$ decrypts EK_i using his private key SK_{CN_i} to obtain K_i. Since K is required to decrypt *ED*, CN_i may attempt to share their acquired K_i and reconstruct K. However, the proposed model is secure against collusion attacks among CN_i due to the security property described in Lemma 2. Therefore, a sub-threshold malicious CN_i cannot restore K and thus cannot obtain D provided by the honest *DO*.
2. An attacker can compromise the *CN* and attempt to reconstruct D by gathering D_i above a threshold. When $CN_i, i \in \{1, \ldots, N\}$ decrypts the *ED* with K_i, it obtains D_i, and CN_i may want to share its D_i with other CN_i to collect more than a threshold number of D_i and obtain D. However, a malicious CN_i below the threshold cannot recover D because Lemma 2 prevents collusion attacks between CN_i.
3. By compromising the *DR*, an attacker can attempt to obtain D provided by an honest *DO* without paying a fair price. The *DR* can obtain the D provided by the honest *DO* through verification in the actual computation phase. The *DR* may attempt to avoid paying fairly by not sending a $\{DTrueTx\}$ or a $\{DFailTx\}$ to the *SC* as a result of the verification of D. In this case, the *SC* verifies that the transaction has not arrived within the transaction end time T_{finish} defined by itself, and determines that *DR* has not intentionally sent the transaction to *SC*. The *SC* can decide the *DR* to be a dishonest participant and enforce punishment to ensure fairness.

As long as the cryptographic primitives applied in the proposed protocol, such as secure hash functions, public key ciphers, etc. cannot be compromised by an attacker, the fairness of the *DO* is guaranteed.

□

Lemma 4. *ORTHRUS satisfies confidentiality for non-adaptive P.P.T. adversaries based on the assumptions and threat model in Section 3.1.*

Proof. The secret key share K_i is stored in IPFS in encrypted form EK_i, and the stored location URI_{EK-Com} is included in the transaction and published on the blockchain. An attacker can obtain EK_i via URI_{EK-Com} and attempt to decrypt it. However, the attacker does not know the private key of CN_i, which is required to decrypt EK_i, that is, in order for P.P.T attackers to compromise the confidentiality of D, they must compromise the public-key cryptographic algorithm applied to the proposal model. Therefore, if the cryptographic mechanism applied to the proposal model is secure, the confidentiality of the proposal model is guaranteed. In another case, an adversary can attempt to obtain D by compromising CN_i above a threshold. However, since the proposal model inherits the fundamental property of MPC that the presence of malicious participants does not affect the protocol, the attacker cannot obtain D from an honest *DO* in the proposal model. □

Lemma 5. *Based on the assumptions and the threat model in Section 3.1, if the proposal protocol operates in the case where ORTHRUS output data is the result of a function computation, the original data provided by DO is not disclosed to any participants, that is, the proposal model guarantees input privacy.*

Proof: In the trading process of the proposed protocol when the output data is the result of the function computation, an attacker may try to obtain D, from participants without paying the rightful price.

1. If the attacker wants to obtain D via *ED* and $EK_i, i \in \{1, \ldots, N\}$: The attacker can obtain *ED* and EK_i through URI_{ED-Com} and URI_{EK-Com}, but cannot decrypt EK_i because he cannot obtain the private key CN_i' SK_{CN_i}, which is needed to decrypt EK_i. Even if the adversary wants to compromise CN_i to obtain

meaningful data, the proposed model inherits the basic characteristics of MPC, making it difficult to compromise CN_i above a threshold, so the adversary cannot obtain meaningful data D.

2. DR and CN_i want to obtain D maliciously: If the final output of the proposed protocol is a function computation output, DR decrypts EX_i provided by CN_i in the actual computation phase to obtain the function computation output share X_i. This can be collected and reconstructed on a threshold to obtain the function computation result X. However, D provided by DO cannot be obtained due to the characteristics of the function requested by DR.
3. Each CN_i colludes to obtain D: CN_i may attempt to reconstruct K by colluding with other CN_is except itself in order to restore K, which is necessary for decrypting ED. However, CN_i can't obtain D because, according to Lemma 2, pre-collusion attacks are difficult.

As such, if the cryptographic mechanisms and MPC techniques applied in the proposed model are secure, the likelihood of an attacker compromising input privacy is negligible. □

Lemma 6. *Based on the assumptions and threat model in Section 3.1, the ORTHRUS satisfies Timeliness if either DO or DR is honest and the SC is secure.*

Proof: Timeliness means that an honest component can always reach a point in the protocol where it can stop in a finite amount of time with guaranteed fairness. For a proposal model with an SC and at least one honest participant, the following termination cases exist:

- No abort: The proposal model ends when $\{DFailTx\}$ or $\{DTrueTx\}$ is received if all parties are honest and the final output of the proposal model is the original data, or when $\{FfinalTx\}$ is received if the final output of the proposal model is the result of a function computation, or when T_{finish} expires. In this case, DO and DR are guaranteed to have obtained what they want, and the proposal model is terminated.
- Cancellation at the computation node selection stage: In the computation node selection stage, T_{Apply} means the deadline to apply to participate in the computation node specified by DO, that is, only applications that arrive within T_{Apply} are recognized as valid. If it does not arrive within T_{Apply}, the blockchain node that sent the intention to participate will not be able to participate in the proposed protocol. In addition, DO has not yet provided its data.
- Cancel in the data storage stage: At the stage of storing encrypted values in IPFS, T_{Open} means the data sale validity period defined by DO, that is, if T_{Open} has expired, it means that DO has not provided a $\{StoreTx\}$, in which case DO cannot participate in the proposal model. In this case, the DO has not provided their data, and the CN hasn't performed any calculations, so they don't receive any incentives.
- Revocation in the verification phase of $CN_i, i \in \{1, \ldots, N\}$: In the phase where CN_i verifies the commit values of ED and EK_i, T_{Store} means the deadline to receive the commit value verification result of CN_i defined by DO. If T_{Store} has expired, this means that CN_i has not provided the commit verification results. In this case, SC cancels the transaction process.
- Cancel at the data request stage: At the stage where DR publishes a data request to the blockchain network, including the data type he wants, T_{Adv} means the validity time of the DO' data advertisement defined by DO. If T_{Adv} expires, it means that DR has not provided $\{ReqTx\}$, in which case SC cancels the transaction. At this time, the fairness of DO is guaranteed because DO has not provided data to DR, and the fairness of DR is guaranteed because DR has not paid.
- Cancellation in the encryption phase of $CN_i, i \in \{1, \ldots, N\}$: In the phase where CN_i encrypts the value it has obtained and delivers it, including a signature for the encrypted value, $T_{DResult}$ or $T_{FResult}$ means the time at which the signature generated by CN_i is valid. If $T_{DResult}$ or $T_{FResult}$ has expired, it means that CN_i has not provided $\{DResult\}_i$ or $\{FResult\}_i$, in which case SC cancels the transaction.

□

Theorem 1: *If the cryptographic primitives underlying ORTHRUS are secure and the blockchain and smart contracts are secure, then ORTHRUS satisfies completeness, resistance to pre-collusion attacks, fairness, confidentiality, input privacy, and timeliness.*

Proof: Guaranteed by Lemma 1, 2, 3, 4, 5, and 6. □

The ORTHRUS is secure against second-resell attacks. A second reselling attack occurs when a malicious buyer purchases data from the blockchain and then resells the same data for their own profit. If duplicate data are found, the seller is verified to be the owner of the original data and the seller is verified to be the buyer in the previous transaction. If the current seller was the buyer in the previous transaction, the resale transaction can be invalidated. In ORTHRUS, DO's activities are recorded on the blockchain using transactions. In addition, commitment techniques, signature techniques, hash functions, etc. are applied to verify the values shared between each participant in the ORTHRUS transaction process. Based on the techniques applied, ORTHRUS prevents secondary resale within the platform. However, it does not prevent the resale of the data in other ways or on other platforms for personal gain.

4.3 Comparative Analysis with Existing Research

In this section, we compare three of the existing studies mentioned in Section 1 with the ORTHRUS based on the security analysis criteria.

Li et al. [10] proposed a BSAS to recommend a consensus speed for a group of vehicles based on a blockchain. The BSAS computes a polynomial to obtain the optimal rate, where each user's secret value is unknown to the others. BSAS randomly selects operators, but it does not guarantee against collusion attacks among other users. Therefore, if collusion among users is successful, the secret value used to compute the polynomial can be known. In other words, if the trustworthiness of the polynomial or the trustworthiness of the computation participants is not guaranteed, the security of the secret value is not guaranteed. Yuan et al. [11] proposed TRUCON, a blockchain-based data sharing mechanism with congestion control in IoV environments. TRUCON has a trust factor and there is no punishment or reward process. In addition, message forwarding is done between vehicles and RSUs, but it does not consider detailed privacy issues and malicious behavior of dishonest participants. Sadiq et al. [12] proposed a vehicle data transaction model based on a consortium blockchain with trusted authorities. It stores unencrypted original data in IPFS and utilizes the same unencrypted original data for data transactions. It does not take into account the existence of dishonest participants, and there is no mechanism for punishment and reward.

In contrast, ORTHRUS is designed on a public blockchain with no trust factor, ensuring full decentralization. In addition, the nodes that participate in the actual computation process are selected using a randomized leader selection technique, making it difficult to predict the probability that a node that has colluded in malicious behavior in advance will participate in the computation process. This prevents collusion attacks. Since ORTHRUS inherits the security of the applied cryptographic techniques, if the final output is the original data, participants other than DR who have paid a reasonable amount of money are not permitted to access D. If the final result is the result of a function computation, then all participants cannot access and obtain the original data D. The ORTHRUS enforces punishments such as deposit forfeiture and reputation score reduction through smart contracts to participants who engage in malicious behavior. Conversely, if the transaction is terminated legitimately, they receive a share of the data payment and incentives, and their reputation score increases. Finally, the ORTHRUS supports both original data and function computation result as final outputs, overcoming the problem of existing vehicle data trading methods that depend on the type and delivery method of final outputs supported by the registered trading platform. The ORTHRUS allows data owners to provide their own data in the type and manner they want. At

the same time, when the data transaction process begins, the rights to the data provided by the data owner are not transferred, and the data owner retains control over the original data. In other words, the data owner sets the type of data provision and the transaction method, ensuring data sovereignty. In this way, ORTHRUS, which supports two types of data as the end result and guarantees data sovereignty, presents a revolutionary way to trade vehicle data, enabling users to obtain all desired data types within a single platform, instead of using multiple platforms to obtain the desired data types. Table 1 below shows a comparison of features with existing studies.

Table 1: Comparison of features between ORTHRUS and existing models

	Data sovereignty	Fairness	Input privacy	Decentralized	Preventing pre-collusion
Li et al. [10]	X	O	△	O	△
Yuan et al. [11]	X	△	X	X	X
Sadiq et al. [12]	X	X	X	X	X
ORTHRUS	O	O	O	O	O

Note: *O*: satisfied, *X*: not satisfied, △: partially satisfied.

5 Conclusion

In an IoV environment, vehicles collect and generate data through interactions with various sensors and peripherals. This data is then utilized for user convenience systems or transportation systems. However, existing vehicle data sharing and transaction models are designed in a centralized way due to the trust factor, leading to various problems such as single point of failure, data leakage, and privacy protection. To solve these problems, blockchain-based models have emerged, but vehicle owners' sensitive information contained in vehicle data is still not protected, and data owners are not guaranteed data sovereignty due to the reliance on participatory data transaction models.

In this paper, we proposed ORTHRUS, a blockchain-based model for vehicle data trading that prioritizes privacy protection. The ORTHRUS guarantees data sovereignty by allowing data owners to trade data in the way they want. In addition, unlike previous studies, ORTHRUS supports two types of outputs: original data and function computation results. By employing MPC technique, MOZAIK architecture, and the random leader selection technique, the proposed scheme, ORTHRUS, guarantees the input privacy and resistance to pre-collusion attacks. Furthermore, the proposed model promotes fairness by identifying dishonest behavior among participants by enforcing penalties and rewards through the implementation of smart contracts. Finally, the distributed decryption performance of the proposed model depends on the performance of the applied MPC technique. Currently, one of the most important and challenging issues in MPC is performance optimization, and for MPC to be deployed in commercial services, its performance problems must be resolved. Even in existing services applying MPC techniques, the number of computational servers used ranges from a minimum of two to a maximum of five. The ORTHRUS system proposed in this paper also conducted performance analyses using only three and five computational nodes, similar to these services. Therefore, limitations exist regarding computational efficiency and service scalability. Consequently, future research will focus on optimizing MPC performance in ORTHRUS implementations involving a larger number of computational nodes. This future research is expected to further enhance the practicality of the proposed ORTHRUS model and contribute to the advancement of privacy-preserving computing technology in distributed environments.

Acknowledgement: Not applicable.

Funding Statement: This work was supported by the IITP (Institute of Information & communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) grant funded by the Korea government (Ministry of Science and ICT) (IITP-2025-RS-2020-II201797), and was supported as a 'Technology Commercialization Collaboration Platform Construction' project of the INNOPOLIS FOUNDATION (Project Number: 1711202494).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Su Jin Shin, Sang Uk Shin; data collection: Su Jin Shin; analysis and interpretation of results: Su Jin Shin, Sang Uk Shin; draft manuscript preparation: Su Jin Shin. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

Appendix A ORTHRUS's Pseudo-code

The ORTHRUS consists of a preprocessing phase, a matching phase, and an actual computation phase. Appendix A presents each stage of the ORTHRUS in pseudo-code. Additionally, Table A1 explains the symbols used in the algorithm.

Table A1: Notation and meanings used in the proposed model

Notation	Meaning	Notation	Meaning
ID_{XX}	ID of XX	SK_{XX}	Private key of XX
PK_{XX}	Public key of XX	sSK_{XX}	Signature verification key of XX
sPK_{XX}	Signing key of XX	D	Data provided by DO
$meta$	Description of data D	$flag_{DO}$	Data type provided by DO
$flag_{DR}$	Data type desired by DR	P	Base probability value to be used in the random leader selection technique
N	Total number of computation nodes	URI	Location of data stored on IPFS
F	DR's computation request function	$Deposit_{XX}$	Deposit of the XX
reP_{XX}	XX's reputation score	Per_{XX}	XX's desired opponent's reputation score
Inc	Incentive	$Price_O$	If $flag_{DO} = 1$, data fee
$Price_F$	If $flag_{DO} = 0$, data fee	$VCNlist$	A list of information about CN_i instances whose submitted values led to verification failure
L	A list containing information about the selected computation nodes	T_{Open}	Data sales period defined by DO
T_{Apply}	Deadline for application for participation in the computation node defined by DO	T_{Compt}	Time period during which a component that has submitted a computation node participation request remains online

(Continued)

Table A1 (continued)

Notation	Meaning	Notation	Meaning
T_{Store}	Deadline for *DO* to receive $\{TrueTx\}_i$ from CN_i	T_{Adv}	Validity period of advertising transactions distributed by *DO*
$T_{DResult}$	Validity period of the signature generated by CN_i (if the output is original data)	$T_{FResult}$	Validity period of the signature generated by CN_i (if the output is function computation result)
T_{list}	Time period during which the computation node must remain online, as defined by the *SC*	T_{finish}	Trading end time defined by *SC*

Appendix A.1 Preprocessing Phase

- Data publish step:
  Upon receiving $\{OpenTx\}$ from *DO*, *SC* operates as described in Algorithm A1.
- Computation node selection step:
  Blockchain participating nodes generate their signatures based on $\{OpenTx\}$. They transmit $\{SigTx\}_j$, $j \in \{1, \ldots, M\}$, including this signature, to the *SC*. The receiving *SC* operates as described in Algorithm A2. Here, M denotes the total number of nodes participating in the blockchain network.
- Data encryption step:
  The DO transmits $\{StoreTx\}$ to *SC*. Upon receiving it, the *SC* verifies the transaction using Algorithm A3.
- Verification and distributed decryption of commit values:
  Algorithm A4 shows the process where CN_i performs distributed decryption and the *SC* receives the results.
- Data advertising step: Algorithm A5 is the process by which *DO* distributes transactions to the blockchain network for data sales.

Algorithm A1: Data publish step

1: **// Performed by *SC***
2: **Input:** $\{OpenTx\} := \{OpenTx_{ID}, ID_{DO}, PK_{DO}, sPK_{DO}, T_{Open}, T_{Apply}, meta, flag_{DO}, P, N\}$ from *DO*
3: **if** $T_{Open} \geq T_{now}$ and $T_{Open} \geq T_{Apply} \geq T_{now}$ **then**
4: Set status variable Σ = Publish
5: Broadcast $\{OpenTx\}$ to all nodes
6: **end if**

Algorithm A2: Computation node selection step

1: **// Performed by *SC***
2: **Input:** $\{SigTx\}_j := \{SigTx_{ID}, ID_{BN_j}, PK_{BN_j}, sPK_{BN_j}, SIG_j(\{OpenTx\}), Deposit_{BN_j}, T_{Compt}, OpenTx_{ID}\}$ from BN_j

3: **for** each $j \in \{1, \ldots, M\}$ **do**
4: **if** $Timestamp \leq T_{Apply}$ **then**
5: **if** $T_{Compt} \geq T_{Open}, T_{now}$ **then**
6: **if** status variable Σ = Publish **then**
7: **for** each valid transaction $j \in \{1, \ldots, l\}$ **do**
8: Generate $H(SIG_j)$
9: Compute probability $.H(SIG_j)$
10: Select N nodes with value matching $P \geq .H(SIG_j)$
11: Update computation node list L
12: **end for**
13: Set status variable Σ = CNSelected
14: Broadcast $\{ListTx\}$ including L to all nodes
15: **end if**
16: **end if**
17: **end if**
18: **end for**

Algorithm A3: Data encryption step

1: // **Performed by *DO***
2: $K, K_i \leftarrow KeyGen()$ ▷ $i \in \{1, \ldots, N\}$
3: $ED \leftarrow AEAD.Enc(K, D)$
4: $r_C \leftarrow H(T_{Apply} || PK_{DO} || PK_{CN_i})$ and $Com_{ED} \leftarrow Comm(ED, r_C)$
5: $EK_i \leftarrow PK.Enc(PK_{CN_i}, K_i)$
6: $r_{EK_i} \leftarrow H(T_{Open} || PK_{DO} || PK_{CN_i})$ and $Com_{EK_i} \leftarrow Comm(EK_i, r_{EK_i})$
7: **for** each $i \in \{1, \ldots, N\}$ **do**
8: Generate pair (EK_i, Com_{EK_i})
9: Store (EK_i, Com_{EK_i}) on IPFS
10: Receive storage location $(URI_{EK-Com})_i$
11: **end for**
12: Store ED, Com_{ED} on IPFS
13: Receive storage location URI_{ED-Com}
14: Sends $\{StoreTx\} := \{StoreTx_{ID}, ID_{DO}, T_{Store}, URI_{ED-Com}, (URI_{EK-Com})_i, OpenTx_{ID}\}$ to SC
15:
16: // **Performed by *SC***
17: **Input:** $\{StoreTx\}$ from DO
18: **if** $Timestamp \leq T_{Open}$ **then**
19: **if** status variable Σ = CNSelected **then**
20: Set status variable Σ = Stored
21: Broadcast $\{StoreTx\}$ to all nodes
22: **end if**
23: **end if**

Algorithm A4: Verification of the commit value and distributed decryption step

1: // **Performed by** CN_i
2: **Input:** URI_{ED-Com}, $(URI_{EK-Com})_i$ from $\{StoreTx\}$
3: Obtains ED and Com_{ED} via URI_{ED-Com}
4: Obtains the key pair (EK_i, Com_{EK_i}) via URL_{EK-Com_j}
5: $r'_C \leftarrow H(T_{Apply}||PK_{DO}||PK_{CN_i})$ and $Com'_{ED} \leftarrow Comm(ED, r'_C)$
6: **if** $Com'_{ED} \neq Com_{ED}$ **then**
7: Sends $\{FailEDTx\}$ to SC
8: **else**
9: $r'_{EK_i} \leftarrow H(T_{Open}||PK_{DO}||PK_{CN_i})$ and $Com'_{EK_i} \leftarrow Comm(EK_i, r'_{EK_i})$
10: **if** $Com'_{EK_i} \neq Com_{EK_i}$ **then**
11: Sends $\{FailEKTx\}_i$ to SC
12: **end if**
13: Sends $\{TrueTx\}$ to SC
14: **end if**
15: $K'_i \leftarrow PK.Dec(SK_{CN_i}, EK_i)$
16: $D_i / \perp \leftarrow DistDec(ED, K_i)$
17: Sends $\{ReconTx\}_i := \{ReconTx_{ID}, ID_{CN_i}, \textit{"Success/Fail"}, StoreTx_{ID}, OpenTx_{ID}\backslash\}$ to SC
18:
19: // **Performed by** ***SC***
20: **Input:** $\{ReconTx\}_i$ from CN_i
21: **for** each $i \in \{1, \ldots, N\}$ **do**
22: **if** status variable Σ = Stored **then**
23: **if** $Timestamp \leq T_{Store}$ **then**
24: $mCount = mCount + 1$
25: **end if**
26: **end if**
27: **end for**
28: **if** "*Fail*" in $\{ReconTx\}_i$ and $mCount \geq$ threshold **then**
29: Perform the punishment procedure against the DO
30: **end if**
31: Set status variable Σ = Verified
32: Broadcast $\{VDSucTx\} := \{VDSucTx_{ID}, OpenTx_{ID}, StoreTx_{ID}\}$ to all nodes

Algorithm A5: Data advertising step

1: // **Performed by** ***SC***
2: **Input:** $\{AdvTx\} := \{AdvTx_{ID}, ID_{DO}, T_{Adv}, meta, reP_{DO}, Per_{DO}, Price_O, Price_F, Deposit_{DO}, flag_{DO}\}$ from DO
3: **if** $T_{Adv} \leq T_{Open}$ **then**
4: **if** status variable Σ = Verified **then**
5: **if** the DO's account balance is sufficient **then**
6: Set status variable Σ = Advertised

(Continued)

Algorithm A5 (continued)

7: **end if**
8: **end if**
9: **end if**
10: Broadcast $\{AdvTx\}$ to all nodes

Appendix A.2 Matching Phase

Algorithm A6 shows the process where *SC* matches *DR* and *DO*. Here, if $flag = 1$, the final result value of the proposed model is the original data. If $flag = 0$, it is the result value of the function computation.

Algorithm A6: Matching phase

1: **// Performed by *SC***
2: **Input:** $\{ReqTx\} := \{ReqTx_{ID}, ID_{DR}, PK_{DR}, sPK_{DR}, reP_{DR}, Per_{DR}, Deposit_{DR}, Inc, flag_{DR}, (F), AdvTx_{ID}\}$ from *DR*
3: **if** $Timestamp \leq T_{Adv}$ **then**
4: **if** status variable Σ = Advertised **then**
5: **if** the *DR*′s account balance is sufficient **then**
6: **if** $flag_{DO} == flag_{DR}$ **then**
7: Set status variable Σ = Matched
8: **end if**
9: **end if**
10: **end if**
11: **end if**
12: **Broadcast** $\{MatchTx\} := \{MatchTx_{ID}, ID_{DO}, ID_{DR}, T_{finish}, flag, AdvTx_{ID}, ReqTx_{ID}, StoreTx_{ID}\}$ to all nodes

Appendix A.3 Actual Computation Phase

This process executes differently depending on the $flag$ value.

- If the final result is original data ($flag = 1$):

  This proceeds in two stages: (1) the encryption and signature generation step and (2) the verification step.

  1. The encryption and signature generation step: The CN_i transmits $\{DResult\}_i$, which includes the encrypted value of D_i, to *SC*. Upon receiving this, *SC* verifies it as described in Algorithm A7.
  2. The verification step: The *DR* generates a hash value $H(D')$ for *D*. It then transmits the result of comparing this with $H(D)$ stored by *DO* to *SC*. The *SC* receives this and verifies it using Algorithm A8.

- If the final result is the result of a function computation ($flag = 0$):

  This process consists of (1) the function computation step and (2) the verification and reconstruction step.

  1. The function computation step: The CN_i computes the function *F* requested by *DR* and transmits it to *SC*. The detailed process is as described in Algorithm A9.
  2. The verification and reconstruction step: The *DR* performs verification on EX_i and Sig_{EX_i}, and transmits the results to *SC*. The detailed process is as described in Algorithm A10.

Algorithm A7: Encryption and signature generation step

1: **// Performed by *CN_i***
2: **for** each $i \in \{1, \ldots, N\}$ **do**
3: $ED_i \leftarrow PK.Enc(PK_{DR}, D_i)$
4: $Sig_{ED_i} \leftarrow Sign(sSK_{CN_i}, ED_i)$
5: Sends $\{DResult\}_i := \{DResult_{ID}, ID_{CN_i}, ED_i, Sig_{ED_i}, T_{DResult}, MatchTx_{ID}\}$ to *SC*
6: **end for**
7:
8: **// Performed by *SC***
9: **Input:** $\{DResult\}_i$ from CN_i
10: **if** status variable Σ = Matched **then**
11: **for** $i \in \{1, \ldots, N\}$ **do**
12: **if** $T_{DResult} \leq T_{Adv}$ **then**
13: $mCount \leftarrow mCount + 1$
14: **end if**
15: **end for**
16: **if** $mCount \geq threshold$ **then**
17: Set status variable Σ = Resulted
18: Broadcast $\{DResult\}_i$ to all nodes
19: **else**
20: Set status variable Σ = Cancelled
21: Perform the punishment procedure against the CN_i
22: **end if**
23: **end if**

Algorithm A8: Verification step

1: **// Performed by *DR***
2: Obtain ED_i and Sig_{ED_i} via $\{DResult\}_i$ ▷ $i \in \{1, \ldots, N\}$
3: $1/0 \leftarrow Verify(sPK_{CN_i}, Sig_{ED_i}, ED_i$
4: **if** Verify result =1 **then**
5: Identify CN_i
6: Record in *VCNlist*
7: Broadcast $\{DVerifySigTx\} := \{DVerifySigTx_{ID}, ID_{DR}, VCNlist, T_{DVerifySig}, MatchTx_{ID}\}$ to all nodes
8: **else**
9: $D'_i \leftarrow PK.Dec(SK_{DR}, ED_i)$
10: $D' \leftarrow Recon(D'_i)$ ▷ Number of $D'_i \geq threshold$
11: Generate *H(D')*
12: Obtain *H(D)* via $\{AdvTx\}$
13: **if** $H(D') == H(D)$ **then**
14: Send $\{DFailTx\} := \{DFailTx_{ID}, ID_{DR}, SK_{DR}, H(D'), MatchTx_{ID}\}$
15: **else**
16: Send $\{DTrueTx\} := \{DTrueTx_{ID}, ID_{DR}, MatchTx_{ID}\}$ to the *SC*
17: **end if**

(Continued)

Algorithm A8 (continued)

18: **end if**
19:
20: // **Performed by *SC***
21: **Input:** $\{DTrueTx\} := \{DTrueTx_{ID}, ID_{DR}, MatchTx_{ID}\}$ from DR
22: **if** status variable Σ = Resulted **then**
23: **if** $Timestamp \leq T_{finish}$ **then**
24: Trading ends normally
25: **else**
26: Punishment for DR
27: Set status variable Σ = Cancelled
28: The trading is terminated
29: **end if**
30: **end if**

Algorithm A9: Function computation step

1: // **Performed by CN_i, $i \in \{1, \ldots, N\}$**
2: $X_i \leftarrow F(D_i)$
3: $EX_i \leftarrow PK.Enc(PK_{DR}, X_i)$
4: $Sig_{EX_i} \leftarrow Sign(sSK_{CN_i}, EX_i)$
5: Send $\{FResult\}_i := \{FResult_{ID}, ID_{CN_i}, T_{FResult}, EX_i, Sig_{EX_i}, MatchTx_{ID}\}$ to SC
6:
7: // **Performed by *SC***
8: **Input:** $\{FResult\}_i$ from CN_i
9: **if** status variable Σ = Matched **then**
10: **for** $i \in \{1, \ldots, N\}$ **do**
11: **if** $Timestamp \leq T_{Adv}$ **then**
12: **if** $T_{FResult} \leq T_{Adv}$ **then**
13: $mCount \leftarrow mCount + 1$
14: **end if**
15: **end if**
16: **end for**
17: **if** $mCount \geq threshold$ **then**
18: Set status variable Σ = Resulted
19: Broadcast $\{FResult\}_i$ to all nodes
20: **else**
21: Set status variable Σ = Cancelled
22: Perform the punishment procedure against the CN_i
23: The trading is terminated
24: **end if**
25: **end if**

Algorithm A10: Verification and reconstruction step

1: **// Performed by *DR***
2: Obtains EX_i and Sig_{EX_i} from $\{FResult\}_i$
3: $1/0 \leftarrow Verify(sPK_{CN_i}, Sig_{EX_i}, EX_i)$
4: **if** Verify result = 1 **then**
5: Identify CN_i
6: Record in *VCNlist*
7: Broadcast $\{FVerifySigTx\} := \{FVerifySigTx_{ID}, ID_{DR}, VCNlist, T_{FVerifySig}, MatchTx_{ID}\}$ to *SC*
8: **else**
9: $X_i' \leftarrow PK.Dec(SK_{DR}, EX_i)$
10: $X \leftarrow Recon(X_i')$
11: Broadcast $\{FfinalTx\} := \{FfinalTx_{ID}, ID_{DR}\}$ to *SC*
12: **end if**
13:
14: **// Performed by *SC***
15: **Input:** $\{FfinalTx\}$ from *DR*
16: **if** status variable Σ = Resulted **then**
17: **if** $Timestamp \leq T_{finish}$ **then**
18: The trading ends normally
19: **else**
20: Punishment for *DR*
21: Set status variable Σ = Cancelled
22: The trading is terminated
23: **end if**
24: **end if**

Appendix B Pseudo-Code of the Dispute Resolution Phase

The dispute resolution phase proceeds when the *SC* receives a dispute resolution request from other participants under the following conditions: (1) *ED*′s commit value verification fails, (2) EK_i′s commit value verification fails, (3) ED_i′s signature verification fails, (4) *D*′s hash value verification fails, or (5) EX_i′s signature verification fails. (3) and (4) are dispute resolution phase that occur when ORTHRUS provides the original data as the final result. (5) is a dispute resolution phase that occurs when ORTHRUS provides the function computation result as the final result.

Appendix B.1 Verification Failure of ED′s Commit Value

This process occurs when CN_i, where $i \in \{1, \ldots, N\}$, fails to verify the commit value for *ED* and sends $\{FailEDTx\}_i$ to *SC* as a dispute resolution request. Algorithm A11 details this process.

Algorithm A11: Verification failure of *ED*′s commit value

1: **// Performed by *SC***
2: **Input:** $\{\{FailEDTx\}_i\}$ from CN_i, $i \in \{1, \ldots, N\}$
3: **if** $Timestamp \leq T_{Store}$ **then**

(Continued)

Algorithm A11 (continued)

4: **for** $i \in \{1, \ldots, l\}$ **do**
5: Obtain ED and Com_{ED} via URI_{ED-Com} from $\{StoreTx\}$
6: $r''_C \leftarrow H(T_{Apply}||PK_{DO}||PK_{CN_i})$
7: $Com''_{ED} \leftarrow Comm(ED, r''_C)$
8: Obtain Com'_{ED} via $\{\{FailEDTx\}_i\}$
9: Compare Com''_{ED}, Com_{ED}, and Com'_{ED} with each other
10: **if** $Com''_{ED} == Com_{ED}$ **then**
11: Identify CN_i
12: Punishment for CN_i
13: $mCN_i \leftarrow mCN_i + 1$
14: **else**
15: **if** $Com''_{ED} \neq Com_{ED} \& Com''_{ED} == Com'_{ED}$ **then**
16: Record DO information
17: **end if**
18: **end if**
19: **end for**
20: **if** DO information is recorded **then**
21: Punishment for DO
22: Set status variable Σ = Cancelled
23: The trading is terminated
24: **else**
25: **if** $mCN_i \geq threshold$ **then**
26: Set status variable Σ = Cancelled
27: The trading is terminated
28: **end if**
29: **end if**
30: **end if**

Appendix B.2 Verification Failure of EK_i's Commit Value

This process occurs when CN_i, where $i \in \{1, \ldots, N\}$, fails to verify the commit value for EK_i and sends $\{FailEKTx\}_i$ to SC as a dispute resolution request. Algorithm A12 details this process.

Algorithm A12: Verification failure of EK_i's commit value

1: **// Performed by *SC***
2: **Input:** $\{\{FailEKTx\}_i\}$ from CN_i, $i \in \{1, \ldots, N\}$
3: **if** $Timestamp \leq T_{Store}$ **then**
4: **for** $i \in \{1, \ldots, l\}$ **do**
5: Obtain EK_i, Com_{EK_i} via URI_{EK-Com} from $\{StoreTx\}$
6: $r''_{EK_i} \leftarrow H(T_{Open}||PK_{DO}||PK_{CN_i})$
7: $Com''_{EK_i} \leftarrow Comm(EK_i, r''_{EK_i})$
8: Obtain Com'_{EK_i} via $\{FailEKTx\}_i$
9: Compare Com''_{Ek_i}, Com_{Ek_i} and Com'_{EK_i} with each other
10: **if** $Com''_{EK_i} == Com_{EK_i}$ **then**

(Continued)

Algorithm A12 (continued)

Identify CN_i
Punishment for CN_i
$mCN_i \leftarrow mCN_i + 1$
else
if $Com''_{EK_i} \neq Com_{EK_i}$&$Com''_{EK_i} == Com'_{EK_i}$ **then**
Record *DO* information
end if
end if
end for
if *DO* information is recorded **then**
Punishment for *DO*
Set status variable Σ = Cancelled
The trading is terminated
else
if $mCN_i \geq threshold$ **then**
Set status variable Σ = Cancelled
The trading is terminated
end if
end if
end if

Appendix B.3 Verification Failure of ED_i's Signature

This process occurs when *DR*, fails to verify the signature for ED_i, where $i \in \{1, \ldots, N\}$ and sends $\{DVerifySigTx\}$ to *SC* as a dispute resolution request. Algorithm A13 details this process.

Algorithm A13: Verification failure of ED_i's signature

// Performed by *SC*
Input: $\{\{DVerifySigTx\}$ from $DR\}$
if $Timestamp \leq T_{DResult}$ **then**
if number of CN_i in $VCNlist \in threshold$ **then**
for $i \in \{1, \ldots, l\}$ **do**
Obtain ED_i and Sig_{ED_i} from CN_i in *VCNlist*
$1/0 \leftarrow Verify(sPK_{CN_i}, Sig_{ED_i}, ED_i)$
if Verification result = 1 **then**
Identify *DR* and record *DR* information
else
Identify CN_i
Punishment for CN_i
$mCN_i \leftarrow mCN_i + 1$
end if
end for
if *DR* information is recorded **then**
Punishment for *DR*
Set status variable Σ = Cancelled

(Continued)

Algorithm A13 (continued)

19: The trading is terminated
20: **else**
21: **if** $mCN_i \geq threshold$ **then**
22: Set status variable Σ = Cancelled
23: The trading is terminated
24: **end if**
25: **end if**
26: **end if**
27: **end if**

Appendix B.4 Verification Failure of D′s Hash Value

This process occurs when *DR*, fails to verify the hash value for *D* and sends $\{DFailTx\}$ to *SC* as a dispute resolution request. Algorithm A14 details this process.

Algorithm A14: Verification failure of *D′*s hash value

1: **// Performed by *SC***
2: **Input:** $\{\{DFailTx\}$ from $DR\}$
3: **if** $Timestamp \leq T_{finish}$ **then**
4: Obtain ED_i from $\{DResult\}_i$ and Obtain SK_{DR} from $\{DFailTx\}$
5: $D_i'' \leftarrow PK.Dec(SK_{DR}, ED_i)$
6: $D'' \leftarrow Recon(D_i'')$
7: Generate *H(D")*
8: Obtain *H(D')* via $\{DFailTx\}$ and *H(D)* via $\{AdvTx\}$
9: Compare *H(D")*, *H(D')* and *H(D)*
10: **if** $H(D'') == H(D)$ **then**
11: Identify *DR* and punishment for *DR*
12: Set status variable Σ = Cancelled
13: The trading is terminated
14: **else**
15: **if** $H(D'') \neq H(D)$ & $H(D'') == H(D')$ **then**
16: Identify *DO* and punishment for *DO*
17: Set status variable Σ = Canclled
18: The trading is terminated
19: **end if**
20: **end if**
21: **end if**

Appendix B.5 Verification Failure of EX_i's Signature

This process occurs when *DR*, fails to verify the signature for EX_i and sends $\{FVerifySigTx\}$ to *SC* as a dispute resolution request. Algorithm A15 details this process.

Algorithm A15: Verification failure of EX_i's signature

// **Performed by *SC***
Input: $\{\{FVerifySigTx\}$ from $DR\}$
if $Timestamp \leq T_{Data}$ **then**
if number of CN_i in $VCNlist \leq threshold$ **then**
  for $i \in \{1, \ldots, l\}$ **do**
    Obtain EX_i and Sig_{EX_i} from CN_i in *VCNlist*
    $1/0 \leftarrow Verift(sPK_{CN_i}, Sig_{EX_i}, EX_i)$
if Verification result = 1 **then**
  Identify *DR* and Record *DR* information
else
  Identify CN_i and punishment for CN_i
  $mCN_i \leftarrow mCN_i + 1$
end if
  end for
  if *DR* information is recorded **then**
    Punishment for *DR*
    Set status variable Σ = Cancelled
    The trading is terminated
  else
    if $mCN_i \geq threshold$ **then**
Set status variable Σ = Cancelled
The trading is terminated
    end if
  end if
end if
    end if

References

1. Farayola OA, Olorunfemi OL, Shoetan PO. Data privacy and security in it: a review of techniques and challenges. Comput Sci IT Res J. 2024;5(3):606–15. doi:10.51594/csitrj.v5i3.909.
2. Fabianek C, Krenn S, Loruenser T, Siska V. Secure computation and trustless data intermediaries in data spaces. arXiv:2410.16442. 2024. doi:10.48550/arXiv.2410.16442.
3. Karim SM, Habbal A, Chaudhry SA, Irshad A. Architecture, protocols, and security in IoV: taxonomy, analysis, challenges, and solutions. Secur Commun Netw. 2022;2022(1):1131479. doi:10.1155/2022/1131479.
4. Luo Y, You W, Shang C, Ren X, Cao J, Li H. A cloud-fog enabled and privacy-preserving IoT data market platform based on blockchain. Comput Model Eng Sci. 2024;139(2):2237–60. doi:10.32604/cmes.2023.045679.
5. Lan C, Li H. BC-PC-Share: blockchain-based patient-centric data sharing scheme for PHRs in cloud computing. Comput Model Eng Sci. 2023;136(3):2985–3010. doi:10.32604/cmes.2023.026321.
6. Golob S, Pentyala S, Dowsley R, David B, Larangeira M, De Cock M, et al. A decentralized information marketplace preserving input and output privacy. In: DEC '23: Proceedings of the Second ACM Data Economy Workshop; 2023 June 18; Seattle, WA, USA. p. 1–6.
7. Gabrielli S, Krenn S, Pellegrino D, Pérez Baún JC, Pérez Berganza P, Ramacher S, et al. Kraken: a secure, trusted, regulatory-compliant, and privacy-preserving data sharing platform. Cham, Switzerland: Springer International Publishing; 2022. p. 107–30.

8. Khan JA, Wang W, Ozbay K. BELIEVE: privacy-aware secure multi-party computation for real-time connected and autonomous vehicles and micro-mobility data validation using blockchain—a study on New York City data. Transp Res Rec. 2024;2678(3):410–21. doi:10.1177/03611981231180200.
9. Zhao C, Zhao S, Zhao M, Chen Z, Gao CZ, Li H, et al. Secure multi-party computation: theory, practice and applications. Inf Sci. 2019;476(1):357–72. doi:10.1016/j.ins.2018.10.024.
10. Li J, Li S, Cheng L, Liu Q, Pei J, Wang S. BSAS: a blockchain-based trustworthy and privacy-preserving speed advisory system. IEEE Trans Vehicular Technol. 2022;71(11):11421–30. doi:10.1109/TVT.2022.3189410.
11. Yuan M, Xu Y, Zhang C, Tan Y, Wang Y, Ren J, et al. Trucon: blockchain-based trusted data sharing with congestion control in internet of vehicles. IEEE Trans Intell Transp Syst. 2022;24(3):3489–500. doi:10.1109/TITS.2022.3226500.
12. Sadiq A, Javaid N, Samuel O, Khalid A, Haider N, Imran M. Efficient data trading and storage in internet of vehicles using consortium blockchain. In: 2020 International Wireless Communications and Mobile Computing (IWCMC); 2020 Jun 15–19; Limassol, Cyprus. p. 2143–8. doi:10.1109/iwcmc48107.2020.9148188.
13. Marquet E, Moeyersons J, Pohle E, Van Kenhove M, Abidin A, Volckaert B. Secure key management for multi-party computation in Mozaik. In: 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW); 2023 Jul 3–7; Delft, Netherlands. p. 133–40.
14. Chattopadhyay AK, Saha S, Nag A, Nandi S. Secret sharing: a comprehensive survey, taxonomy and applications. Comput Sci Rev. 2024;51(2):100608. doi:10.1016/j.cosrev.2023.100608.
15. Damgård I, Keller M, Larraia E, Pastro V, Scholl P, Smart NP. Practical covertly secure MPC for dishonest majority-or: breaking the SPDZ limits. In: 18th European Symposium on Research in Computer Security; 2013 Sep 9–13; Egham, UK. p. 1–18.
16. Rogaway P. Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM Conference on Computer and Communications Security; 2002 Nov 18–22; Washington, DC, USA. p. 98–107.
17. Al-Kuwari S, Davenport JH, Bradford RJ. Cryptographic hash functions: recent design trends and security notions. In: 6th China International Conference on Information Security and Cryptology (Inscrypt 2010); 2010 Oct 20–23; Shanghai, China. p. 133–50.
18. Bellare M, Desai A, Jokipii E, Rogaway P. A concrete security treatment of symmetric encryption. In: Proceedings 38th Annual Symposium on Foundations of Computer Science; 1997 Oct 20–22; Miami Beach, FL, USA. p. 394–403.
19. Benaloh J. Dense probabilistic encryption. In: Proceedings of the Workshop on Selected Areas of Cryptography; 1994; Kingston, ON, Canada. p. 120–45.
20. Naor M, Yung M. Public-key cryptosystems provably secure against chosen ciphertext attacks. In: STOC90: 22nd Annual ACM Symposium on Theory of Computing; 1990 May 13–17; Baltimore, MD, USA. p. 427–37.
21. Goldwasser S, Micali S, Rivest RL. A digital signature scheme secure against adaptive chosen-message attacks. SIAM J Comput. 1988;17(2):281–308. doi:10.1137/0217017.
22. Chen J, Algorand MS. arXiv:1607.01341. 2016. doi:10.48550/arXiv.1607.01341.
23. Kosba A, Miller A, Shi E, Wen Z, Papamanthou C. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy (SP); 2016 May 22–26; San Francisco, CA, USA. p. 839–58.
24. Abe M, Ohkubo M. A framework for universally composable non-committing blind signatures. Int J Appl Cryptogr. 2012;2(3):229–49. doi:10.1504/IJACT.2012.045581.
25. Lindell AY. Adaptively secure two-party computation with erasures. In: Paper Session Presented at: Cryptographers' Track at the RSA Conference; 2009 April 20–24; San Francisco, CA, USA. p. 117–32.
26. Zhang X, Miao X, Xue M. A reputation-based approach using consortium blockchain for cyber threat intelligence sharing. Secur Commun Netw. 2022;2022(1):7760509. doi:10.1155/2022/7760509.
27. Liu Y, Liu Z, Zhang Q, Su J, Cai Z, Li X. Blockchain and trusted reputation assessment-based incentive mechanism for healthcare services. Future Gener Comput Syst. 2024;154(3):59–71. doi:10.1016/j.future.2023.12.023.
28. Bhati AS, Pohle R, Abidin A, Andreeva E, Preneel B. Let's Go Eevee! A friendly and suitable family of AEAD modes for IoT-to-cloud secure computation. In: CCS '23: ACM SIGSAC Conference on Computer and Communications Security; 2023 Nov 26–30; Copenhagen Denmark. p. 2546–60.