Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

OWL: A data sharing scheme with controllable anonymity and integrity for group users

Yongxin Zhang [a,1], Zijian Bao [a,1], Qinghao Wang [a,c], Ning Lu [c,d], Wenbo Shi [c], Bangdao Chen [a], Hong Lei [b,a,*]

[a] SSC Holding Company Ltd., Chengmai, 571924, China
[b] The School of Cyberspace Security, Hainan University, Haikou, 570228, China
[c] The College of Computer Science and Engineering, Northeastern University, Shenyang, 110819, China
[d] The School of Computer Science and Technology, Xidian University, Xi'an, 710071, China

ARTICLE INFO

Keywords:
Blockchain
Controllable anonymity
Data sharing
Data integrity
State channel

ABSTRACT

As our society becomes digital and communication technology develops, global collaboration is an inevitable trend, where data-sharing is a critical component of cooperation across organizations. Identity privacy and data integrity are vital issues in data-sharing. Existing works struggle to address these problems simultaneously, either privacy leaking or privacy abuse. In this work, we proposed OWL, a data-sharing scheme that (1) provides users on-demand anonymity and (2) allows users to verify data integrity while preserving anonymity. To achieve (1) OWL enables controllable anonymity that allows de-anonymity for the malicious while keeping anonymity to the honest providers based on traceable ring signature technology. To achieve (2), OWL designs a data integrity auditing scheme that uses vector commitment to verify data integrity without privacy leakage. Furthermore, OWL employs the blockchain to store immutable auxiliary information for the integrity and controllable anonymity. We also employ the state channel to resolve the performance bottleneck of blockchain and design methods to improve the usage of the state channel for group users. We prove that OWL achieves controllable anonymity and integrity. Finally, we implement the experiment to evaluate the efficiency of OWL.

1. Introduction

In the evolving global economy, data regarded as the digital oil has been recognized as a central issue for promoting societal development. Statista, an international authoritative organization, forecasts that the big data market size revenue will come to \$ 103 billion in 2027.[2] Realizing data-sharing across various consumers and enterprises is not only a prerequisite for further exerting data value, but also an inevitable choice for the in-depth development of digital society [1,2]. Many groups, teams, and organizations have adopted project hosting platforms, such as Github, BitBucket and Springloops, to share data internally or externally.

As shown in Fig. 1, the typical data-sharing model is relatively simple: the group of data providers that have limited storage resources outsource the data to the data-sharing platform (DSP). The DSP is based on the cloud with high storage capacity, and the data consumers request it to obtain the data for further work. The **data integrity** and **provider privacy** matter in this model [3,4].

- **Data integrity**. It is the main concern of data outsourcing [5]. The data in the DSP may be a risk of loss or corruption [6]. For example, Amazon, Dropbox, and Tencent Cloud lost their data due to natural disasters, software vulnerabilities and human error [7]. The use of incomplete data can cause economic damage for data consumers and reputation damage for data providers.
- **Provider privacy**. The provider's identity is also confidential, since they could disclose some significant information about the shared (even encrypted) data [8]. For example, once a data consumer finds that the shared data is a market user's shopping records, the consumer may infer certain private information about that user, such as potential shopping preferences, rough dwelling area, etc.

Many solutions have been proposed to address the above limitations in data-sharing schemes for group users, but few have considered both issues. To guarantee the data integrity, previous works employed the provable data possession (PDP) technology and a trusted third-party auditor to verify the data integrity for users by pre-generated tags

* Corresponding author at: The School of Cyberspace Security, Hainan University, Haikou, 570228, China.
E-mail address: leiluono1@163.com (H. Lei).
[1] The first two authors contributed equally to this work.
[2] Big data market size revenue forecast worldwide from 2011 to 2027.

https://doi.org/10.1016/j.comcom.2023.07.022
Received 27 July 2022; Received in revised form 3 July 2023; Accepted 15 July 2023
Available online 18 July 2023

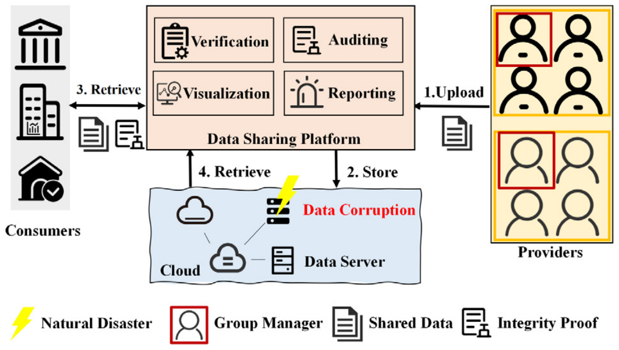


Fig. 1. Typical data sharing system with centralized trust.

with the user's private key [6,7,9,10]. Some studies [11–16] realized integrity and anonymity simultaneously but they relied on centralized trust. Hu et al. modified the data structure of shared data to provide strong anonymity for providers [8]. Nevertheless, it is a huge risk for the data sharing environments. [5,15,16]. In the aforementioned example, a data consumer may obtain a fake record that indicates an inexistent product in the market to generate a mistaken judgement, while the market cannot identify who violates the rules. Overall, a data integrity auditing scheme for group users should reach *anonymity* and *traceability*, simultaneously. Moreover, we require that *traceability* should **not** violate the *anonymity* of honest providers, and we call it *controllable anonymity*. Accordingly, a question emerges: *how to achieve controllable anonymity without centralized trust while maintaining data integrity for data-sharing?*

1.1. Proposed scheme

To answer this question, this work proposes OWL, a data sharing scheme for group users with controllable anonymity and integrity. We desire OWL to perform as a real **owl** that opens an eye to the providers and closes an eye to the honest providers (controllable anonymity), catches the mouse (the malicious provider), and protects the crops in the farmland (data integrity). There are three technical challenges that underpin the design of OWL.

1. **Anonymity *vs.* Integrity**. As mentioned earlier, the data integrity adopts pre-generated tags with the user's private key. The provider's public key is required for verification, which may disclose the identity or the relationship between the public key and the identity [8]. We employ vector commitment (VC) technology to achieve the data integrity without using the private key. Compared to the homomorphic linear authenticators [6,7,9,10], VC generates auditing information using public parameters rather than the private key, resulting in anonymity.
2. **Trust *vs.* Efficiency**. To remove the centralized trust, we apply traceable ring signature (TRS) technology to replace the group signature (delete the manager) used in [5,12,15]. Nevertheless, the DSP and consumers require a trustworthy third party to collect some public parameters to verify the TRS signature and VC commitment. Some schemes [17–20] employed blockchain as a trusted auditor to perform auditing. Hence, we utilize the blockchain as a trust bulletin board to log information of integrity and controllable anonymity [21]. Unfortunately, the poor throughput of blockchain [22–24] is a bottleneck for data updates. To this end, we adopt the state channel (SC) [23] to offload and aggregate data update operations.
3. **Efficiency *vs.* Conflicts**. Although the SC improves the throughput, the frequent data update of a group of suppliers implies concurrent conflicts. For example, two data suppliers may update the same data to independent versions. This fork prohibits other providers within the group from proceeding with the following operation. We propose methods to solve this predicament in Section 6.

In broad terms, the contributions made by this work can be described as follows:

- OWL allows a group of data providers to share sensitive data with controllable anonymity leveraging the traceable data structures we designed, traceable ring signature and blockchain. It preserves the anonymity of providers from the outside world, and it is able to catch the malicious provider inside the group while maintaining the anonymity of the honest providers.
- We design a data integrity auditing scheme that leverages vector commitment and blockchain to enable the DSP and data consumers to verify the integrity of data. Furthermore, it leaks no identity privacy of providers than traditional integrity auditing schemes by avoiding the usage of private keys.
- OWL utilizes the state channel to increase the blockchain throughput. To address the conflict issues that arise when a group of providers shares a single state channel, we propose schemes to (1) alleviate the contradiction between the high-frequency data update and the inefficiency throughput of blockchain and (2) resolve the concurrent conflict of multiple providers. We also develop a strategy for dealing with the online assumption problem and linear overhead of the state channel.
- We conduct comprehensive security analysis and combine these with existing building blocks to test the overhead of OWL. The results demonstrate the effectiveness and efficiency of our proposed schemes.

1.2. Layout

The rest of this paper is organized as follows. In Section 2, we introduce the related work. In Section 3, we give the system model, formal workflow, threat model and design goals. Section 4 gives the security assumptions. In Section 5, we present the detailed construction. In Section 6, we present the methods for state channel predicament. Section 7 gives the security analysis of our scheme. In Section 8, we carry out experiments to evaluate the overhead in our scheme. The results show that our scheme is efficient. We list the limitations of OWL in Section 9. We finally present the conclusion in Section 10.

2. Related works

In this section, we review literature related to the data sharing schemes, including data integrity and anonymity. Data sharing is an important step to explore the potential value of data. It enables data providers to transfer data to data consumers that are with more computing and analytical power via a cloud-based platform. Research on data security of data sharing includes confidentiality, anonymity, integrity, accountability, access control, and so on [25–28]. Our work focuses on the integrity of shard data and the anonymity of a group of providers.

2.1. Data integrity

Under the assumption of a cloud storage model for resource-limited data providers (data owners). Ateniese et al. [6] first presented the concept of provable data possession (PDP) and constructed homomorphic linear verifiable (HLA) tags on the basis of RSA-based signatures as auxiliary information to check the integrity of data on remote servers. Based on this, scholars have proposed various schemes to enhance it, such as dynamic operation [9] and outsourced verification [10]. Shen et al. introduced a sanitizer that sanitizes the data block to hide sensitive information while keeping the integrity of shared data. By using this method, the data in the cloud can be shared with others preserving the sensitive information [11]. Ambika and Moses use factorial prime-based data control to guarantee the data integrity with dynamic member management under a group manager [12]. Song et al. suggested a public data integrity verification approach for cloud data with asynchronous revocation for group users based on HLA tags and proxy signatures. The HLA tags protect the integrity, and the revoked users' old signatures are updated by a fresh signature using proxy re-signature [13]. However, these schemes rely on centralized trust. Work [11] involves a trustful sanitizer, the manager oversees the whole group in work [12]. The cloud server that generates the new signature is also a centralized trust in work [13]. With the advent of blockchain technology, auditing schemes [17–20] resort to it to eliminate the centralized trust and records integrity information immutably.

OWL also leverages blockchain as bullion broad to record the integrity proofs of shared data. With this trustful information, consumers and DSP can verify the data integrity via vector commitment.

2.2. Anonymity

It is deficient to provide content confidentiality through message encryption alone for data sharing under multi-roles involved. The sophisticated attackers use various means to steal the user's identity information. For example, the attacker may use a public key used in data integrity to infer the identity of the provider [8]. Some studies guarantee strong anonymity for providers to publish data [29,30]. Unfortunately, strong anonymity is detrimental to data sharing since the malicious provider may serve the devil while escaping punishments. Some works have been proposed to limit this uncontrollable anonymity [31–33]. Yan et al. proposed a rights distribution center to hide user's identity when verifying the data integrity though hash function [14]. However, they brought in a trust role rights distribution center that records the data activities made by users. Huang et al. proposed a blockchain-based data sharing scheme for multiple groups with anonymity and traceability via group signature [15]. The verification information is saved on the blockchain to help users verify the data integrity. In work [16], a privacy-preserving scheme with an identity-trackable character is proposed for the smart grid, where a trustful anchor is given to trace the abnormal entities. Nevertheless, the manager of group in work [15] and the anchor in work [16] have excessive power to expose every member's identity. The centralized trust assumption brings potential risk to the actual execution of data sharing. For example, the manager may shelter the malicious provider. Tan addressed the security and privacy issues in edge computing environment, and proposes an IoT group association and update mechanism, which protects identity privacy and security through an identity verification mechanism [34]. Xu et al. proposed a privacy-preserving sanitizable signature scheme to solve the problem that traditional digital signatures cannot meet the diversity and privacy requirements of medical data applications, which realizes the mutability of messages without the cooperation of the original signer, and improves the privacy and flexibility [35].

OWL combines traceable ring signature and blockchain to accomplish controllable anonymity that can catch the malicious provider and preserve the privacy of honest providers in a single group without a centralized manager.

3. Overview

In this section, we introduce the system model, threats, assumptions and design goals of OWL.

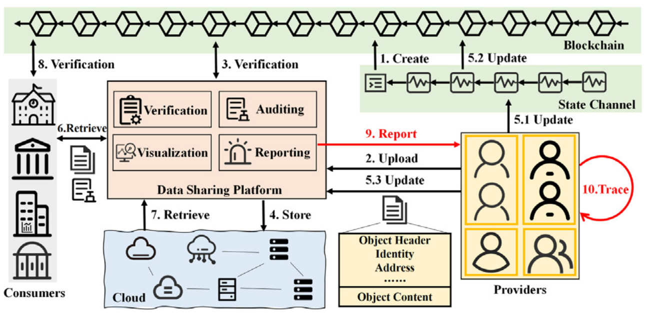


Fig. 2. System model of OWL.

3.1. System model

Fig. 2 shows the system model of OWL, which involves four parties:

- **Consumers** are willing to get the wanted data file from a reputable and trustworthy data sharing platform.
- **Providers** prefer to present in the form of a group. They tend to resort to a platform to share their data. The majority is honest, while the minority is malicious and seeks to shift blame.
- **Data sharing platform (DSP)** conscientiously provides data services for consumers and providers. Once illegal data is discovered on the platform, DSP will immediately delete the data and notify the corresponding provider group. Note that the underlying storage component is a cloud storage model.
- **Blockchain** consists of nodes from all over the world, maintaining a tamper-proof evidence-perpetuation platform via a consensus algorithm. The blockchain can run smart contracts, which support **state channel**. OWL uses it to resolve contradictions lying to frequent updates and insufficient blockchain throughput and relieve concurrent conflict among providers.

The formal definition is given as follows:

- **Setup**$(1^{\lambda}, \ell)$. Take as input security parameters λ and ℓ, all entities generate their cryptographic parameters for blockchain keys (pk, sk), TRS keys (tpk, tsk) and VC public parameters pp.
- **Creating**$(sk, tpk, tsk, \mathsf{pp})$. The providers create state channel SC in blockchain, generate TRS signature σ and VC commitment C for the encrypted data m.
- **Uploading**(m, σ, C). The DSP verifies the σ and C of m sent by a provider.
- **Sharing**(m, σ, C, Λ). The consumer verifies the σ, C and Λ of m from the DSP.
- **Updating**(m', σ', C'). The provider updates the m to m' with TRS signature σ' and VC commitment C'. The blockchain records σ' and C'. The DSP verifies the σ' and C'.
- **Tracing**(σ, tpk, tsk). Given the signature σ of illegal data, the providers jointly execute the trace algorithm of TRS and find the malicious provider.
- **Correction**. The providers expel the malicious provider and recovery data m. The rest of providers create a new state channel SC in blockchain, generate TRS signature σ and VC commitment C for the encrypted data m. After saving the parameters in blockchain, the provider send them to the DSP, who verifies the σ and C of m.

3.2. Threat model and assumption

Our design purpose is to disclose dishonest behavior while preserving the privacy of the honest entities.

Data providers share data in the form of groups. The majority of data providers are honest, and the minority of data providers in each group intentionally or unintentionally upload incorrect or illegal data. When such an incident occurs, honest providers take actions to find and expel the malicious provider, and then provide the correct and legitimate data to the DSP.

Consumers are honest to send requests the DSP for data. They are equipped with anomaly detection module, which can detect incorrect or illegal data. Consumers will report the accident to the DSP.

Additionally, DSP delivers data from providers to consumers. We view it as a typical curious but honest role that provides data-sharing services but desires to catch users' identity privacy without being detected [6,7]. We assume the threats to the integrity of sharing data on DSP can be both internal and external attacks, e.g., hardware failures, software vulnerabilities [20,36]. The DSP may deliberately hide the data loss incidents to protect its reputation [37]. The adversary $\mathcal{A}$ with probabilistic polynomial time ($\mathcal{PPT}$) computing ability is interested in the privacy of data or identities of providers.

Furthermore, we assume a secure append-only blockchain with the following features [38]:

1. Immutability, no one can tamper with the data on it.
2. Accessibility, everyone can access the blockchain to get the entire or partial data.
3. Pseudonym, the possibility that adversary $\mathcal{A}$ infers the entity's real identity through a pseudonym is negligible.
4. Smart contract, a program that runs in the blockchain, and everyone can publish a transaction to establish a smart contract or modify the state of contracts. OWL employs a smart contract to implement state channel that is accessible for everyone and modifiable for the providers who establish it.

We assume that the existing communication technologies enable reliable transmission, and some anonymous communication tools [39, 40] are adopted by the entities to protect the identity privacy in the network layer.

3.3. Design goals

In accordance with the analysis of the data-sharing system practices, we prioritize the desirable goals of OWL.

- **Sharing data integrity**. Integrity guarantees that the data-sharing platform faithfully stores the data is of utmost critical. Compared to the conventional data integrity schemes in cloud storage, we need to ensure that the integrity information is accessible.
- **Controllable anonymity**. Most data sharing systems give greater power to control user privacy, and some schemes emphasize privacy overly. Hence, it is desirable to propose an on-demand anonymity mechanism for data-sharing, where the honest can be protected instead of the malicious *provider*.
- **On-chain efficiency**. Inherent cost due to on-chain storage and computation is inevitable for the goal of controllable anonymity. Considering the criticized throughput of blockchain obstructs the efficiency of sharing data updating. We also need an efficient way of updating data among providers as few as possible to interact with the blockchain.

4. Preliminaries

4.1. RSA assumption

Let $k \in \mathbb{N}$ be the security parameter, N a random RSA modulus of length k, z be a random element in $\mathbb{Z}_N$, and e be an $(\ell+1)$-bit prime (for a parameter ℓ). Then we say that the RSA assumption holds if for any $\mathcal{PPT}$ adversary $\mathcal{A}$ the probability

$$\Pr[y \leftarrow \mathcal{A}(N, e, z) : y^e = z \mod N]$$

is negligible on the security parameter k.

Table 1
Summary of notations.

Notations	Meanings
$\mathbb{F}_R$	The finite field with prime p.
G	The base point of elliptic curve with order o.
pk/sk	The public/private key pairs of providers for *BC*.
pks	The public key set of all providers for *BC*.
$addr$	The *BC* address of smart contract.
K	The symmetric encryption (AES-256) key
$\lambda, \mathsf{k}, \ell$	The security parameters.
$\mathbb{G}$	The multiplicative group.
q	The prime order of $\mathbb{G}$.
g	The generator of $\mathbb{G}$.
H, H', H''	The distinct one-way functions.
n	The number of providers in one group.
h	The hash value for *TRS*.
tpk/tsk	The public/private key pairs of providers for *TRS*.
$tpks$	The public key set of all providers for *TRS*.
σ	The signature generated by provider with tsk.
L	The tag used in *TRS*.
$issue$	The core component of L, enables the traceability.
A_0, A_1	The elements for *TRS*.
a_i, b_i	The intermediate parameter for *TRS*.
w_i, c_i, z_i	The random number for *TRS*.
p_1, p_2	The $\mathsf{k}/2$-bit prime number for *VC*.
N	The composite number for *VC*.
e_i	The $q(\ell+1)$-bit primes for *VC*.
a	The random number for *VC*.
S_i	The intermediate parameter for *VC*.
v	The size of data block.
pp	The public parameters for *VC*.
C	The commitment of data.
aux	The auxiliary information used for *VC*.
Λ	The proof of data integrity for *VC*.

4.2. Bilinear pairing

Given two cyclic groups of large prime order q, G_1 and G_T. Let g_1 and g_2 be the generators of G_1 and G_T, respectively. A cryptographic bilinear map is a map e: $G_1 \times G_1 \rightarrow G_T$ satisfying the three properties as follows.

1. *Bilinear*: for $\forall P, Q \in G_1$ and $\forall x, y \in Z_q^*$, $e(P^x, Q^y) = e(P, Q)^{xy}$;
2. *Non-degenerate*: $\exists g_1 \in G_1$, then $e(g_1, g_1) \neq 1$;
3. *Computable*: the map e can be computed efficiently.

4.3. Decisional Diffie–Hellman assumption (DDH)

Given a quad $(g, g^x, g^y, g^z) \in \mathbb{G}_1$, where $(x, y, z) \leftarrow \mathbb{Z}_q^*$, and $g \in \mathbb{G}_1$ is a generator. For any $\mathcal{PPT}$ adversary $\mathcal{A}$ and a security parameter λ, the advantage

$$\begin{aligned} Adv_{\mathcal{A}}^{DDH}(\lambda) = &|\Pr[\mathcal{A}(g, g^x, g^y, g^{xy}) = 1]| \\ &-|\Pr[\mathcal{A}(g, g^x, g^y, g^z) = 1]|. \end{aligned}$$

is negligible on the security parameter λ.

5. A data integrity auditing scheme with controllable anonymity

OWL ensures the integrity of sharing data leveraging VC and blockchain. We also design data structures along with TRS to realize the controllable anonymity. SC are used to improve the throughput of blockchain. In this section, we introduce the basic usage of SC in OWL, and we focus on the conflicts of providers within one state channel in Section 6. For the sake of readability, let us give the notations in Table 1.

5.1. Building blocks

The details are shown in Appendix, including TRS, VC and SC.

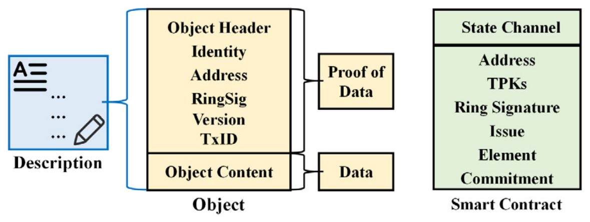


Fig. 3. Traceable data structures for OWL.

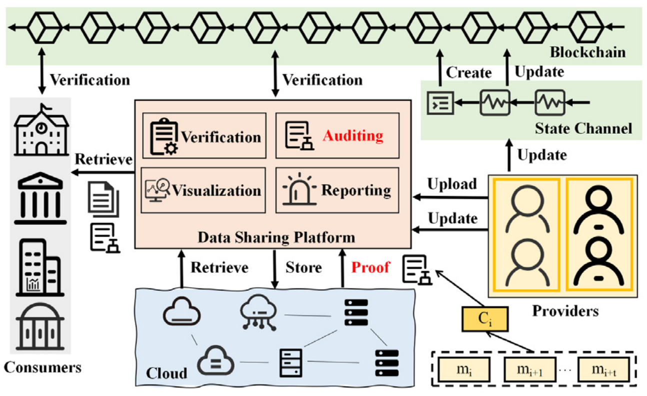


Fig. 4. A data integrity auditing scheme with controllable anonymity.

5.2. *Traceable data structures for* OWL

We design traceable data structure (*TDS*) to assist in data sharing, as shown in Fig. 3. It contains description and object. The former is plaintext, briefly introduces the content of object and attracts the consumer. The latter consists of an object header $\mathrm{obj_h}$ and an object content $\mathrm{obj_c}$. The $\mathrm{obj_h}$ contains auxiliary information. The *Identity* is the unique identification of this data. The *Address* is the state channel address of this data. The *RingSig* is the traceable ring signature. The *Version* is the version number of this object. The *TxID* is the transaction ID of this object version. The first three terms are used to ensure *integrity* and *controllable anonymity*, as described in the next subsection. The last two terms are used to keep *freshness* and *validity* explained in Section 6.2. The $\mathrm{obj_c}$ is the encrypted content for sharing.

Correspondingly, we use state channels (also a smart contract of blockchain) to record information so that consumers and the DSP can verify the obj in case of data corruption. The entries of a state channel in OWL contains $Address$, $RingSig$, $issue$, $Element$, $Commitment$ and $TPKs$. The $Address$ is the address of smart contract. The $RingSig$ is the traceable ring signature of this obj. The $issue$ is the random number used in the $RingSig$. The $Element$ records the elements used in the $RingSig$, *i.e.* A_0. The $issue$ and A_0 are useful to trace the malicious provider. The $Commitment$ is the commitment used for data integrity. The $TPKs$ records all traceable ring signature public keys of this group for the obj, they are used to verify the $RingSig$.

5.3. *A concrete scheme*

Here, we introduce the data integrity auditing scheme that enables DSP and consumers to verify the data integrity via the VC and blockchain. The scheme is shown in Fig. 4 and is defined by a collection of phases as follows:

Setup$(1^\lambda, \ell)$:

1. Let λ, k, ℓ be security parameters.
2. We adopt elliptic curves cryptography (ECC) algorithm to generate blockchain public/private key pair as Bitcoin and Ethereum. In a finite field $\mathbb{F}_R$ with prime p, we choose a curve $y^2 = x^3 + ax + b$, let G be a base point of this curve with order $o, o > 2^{160}$. Let provider P_i randomly chooses a number sk in $[1, o-1]$ and calculates $pk = skG$. The BC public key of P_i is pk and the private key is sk. We let $pks = (pk_1, \ldots, pk_n)$ be an ordered public-key list for n providers. We also use the public key pk represent the address $addr$ in blockchain.
3. Let $\mathbb{G}$ be a multiplicative group of prime order q and let g be a generator of $\mathbb{G}$. Let $H : \{0,1\}^* \to \mathbb{G}$, $H' : \{0,1\}^* \to \mathbb{G}$, and $H'' : \{0,1\}^* \to \mathbb{Z}_q$ be distinct one-way functions. Provider P_i picks up random element $tsk_i \in \mathbb{Z}_q$ and computes $y_i = g^{tsk_i}$. The TRS public key of P_i is $tpk_i = \{g, y_i, \mathbb{G}\}$ and the corresponding private key is tsk_i. We let $tpks = (tpk_1, \ldots, tpk_n)$ be an ordered public-key list for n providers.
4. The n providers randomly choose two k/2-bit primes p_1 and p_2, set $N = p_1 p_2$, and choose $q(\ell+1)$-bit primes $e_1, \ldots, e_q$ that do not divide $\phi(N)$. For $i = 1, \ldots, q$, there are $S_i = a^{\prod_{j=1, j\neq i}^{q} e_j}$. Hence, $\mathsf{pp} = \{N, a, S_1, \ldots, S_q, e_1, \ldots, e_q\}$. The message space is $M = \{0,1\}^\ell$.
5. The provider P_i generates a symmetric encryption (AES-256) key K to send to others in this group.

The basic process is shown in Algorithm 1:

Algorithm 1 Setup Phase

Input: k, n, ℓ, q
Output: $\{(pk_i, sk_i)|_{i\in n}\}$, $\{(tpk_i, tsk_i)|_{i\in n}\}$, pp

```
PROVIDERS:
1: $(pk_i, sk_i) \leftarrow \mathsf{SC.Setup}(1^k, n)$
2: $(tpk_i, tsk_i) \leftarrow \mathsf{TRS.KGen}(1^k, n)$
3: $\mathsf{pp} \leftarrow \mathsf{VC.KGen}(1^k, \ell, v)$
```

Creating$(sk, tpk, tsk, \mathsf{pp})$:

1. The providers have created the data $data$ and encrypt it with key K (*i.e.*, $\mathrm{obj_c}$) and leveraged OWL for sharing. Then, they need to generate the $\mathrm{obj_h}$, including *Identity*, *Address* and *RingSig*.
2. They use a one-way function $Identity \leftarrow H''(\mathrm{obj_c})$.
3. One provider P_i generate tag $L = \{issue, tpks\}$, where $issue$ is a random number. P_i calculates the hash value $h = H(L)$, compute signature $\sigma_i = h^{tsk_i}$, sets $A_0 = H'(L, \mathrm{obj_c})$ and $A_1 = \left(\frac{\sigma_i}{A_0}\right)^{\frac{1}{i}}$. For all $j \neq i$, P_i compute $\sigma_j = A_0 A_1^j$. We set $\sigma_N = (\sigma_1, \ldots, \sigma_n)$. Then, P_i randomly chooses $w_i \leftarrow \mathbb{Z}_q$ and sets $a_i = g^{w_i}, b_i = h^{w_i}$. For all $j \neq i$, P_i randomly chooses $z_j, c_j \leftarrow \mathbb{Z}_q$ and sets $a_j = g^{z_i} y_i^{c_j}, b_j = h^{z_j} \sigma_j^{c_j}$. We set $a_N = (a_1, \ldots, a_n)$ and $b_N = (b_1, \ldots, b_n)$. P_i sets $c = H''(L, A_0, A_1, a_N, b_N)$ Then, P_i computes $c_i = c - \sum_{j\neq i} c_j$ mod q and $z_i = w_i - c_i tsk_i$ mod q. We set $c_N = (c_1, \ldots, c_n)$ and $z_N = (z_1, \ldots, z_n)$. The signature of $\mathrm{obj_c}$ is $\sigma = (A_1, c_N, z_N)$ on $issue$. We set $RingSig = \sigma$.
4. The provider P_i compute commitment $C = S_1^{\mathrm{obj_{c1}}} S_2^{\mathrm{obj_{c2}}} \ldots S_q^{\mathrm{obj_{cq}}}$ and the auxiliary information $\mathsf{aux} = \{\mathrm{obj_{c1}}, \ldots, \mathrm{obj_{cq}}\}$.
5. The provider P_i constructs a transaction to create the state channel with parameters $(pks, tpks, \sigma, C, \mathsf{pp}, issue, A_0)$ to get the address of state channel $addr_{SC}$ and the transaction ID $txid$ to fill $\mathrm{obj_h}$. We set $Address = addr_{SC}$ and $TxID = txid$. Then the $Version = x$, where x is a random number.

The basic process is shown in Algorithm 2:

Uploading$(\mathrm{obj}, \sigma, C)$:

1. The provider sends the obj to the DSP with σ, C.
2. The DSP extracts the address of state channel from $\mathrm{obj_h}$.

Algorithm 2 Creating Phase

```
Input:   obj, q, pp, $\{(pk_i, sk_i)|_{i\in n}\}$, $\{(tpk_i, tsk_i)|_{i\in n}\}$
Output:  $C$, $aux$, $\sigma$
PROVIDERS:
 1: $issue \leftarrow PRNG()$
 2: $L := \{issue, tpks\}$
 3: $\sigma \leftarrow$ TRS.Sign(obj, $L$, $sk_j$)
 4: $addr \leftarrow$ SC.Open($pks, tpks, \sigma, issue$, pp)
 5: $(C, aux) \leftarrow$ VC.Com(pp, $m_1, m_2, ..., m_q$)
 6: fill $\mathrm{obj_h}$ with contract address $addr$.
```

3. The DSP accesses the blockchain to fetch the parameters $(tpks, \sigma^{BC}, C^{BC}, \mathrm{pp}, issue)$ according to *Address*. Then, the DSP compare the σ of $\mathrm{obj_h}$ and the σ^{BC} of state channel.
4. If they are equal, the DSP parses the $\sigma = (A_1, c_N, z_N)$. The DSP parses $L = (issue, tpks)$, computes $h = H(L)$, $A_0 = H'(L, \mathrm{obj_c})$, $\sigma_i = A_0 A_1^i$, $a_i = g^{z^i} y_i^{c_i}$ and $b_i = h^{z_i} \sigma_i^{c_i}$. Then, the DSP checks whether $H''(L, \mathrm{obj_c}, A_0, A_1, a_N, b_N) = \sum_n c_i \mod \mathrm{q}$.
5. If they are equal, the DSP stores the data and makes the description information public.

The basic process is shown in Algorithm 3:

Algorithm 3 Uploading Phase

```
PROVIDERS:
 1: send the obj and its description to DSP.
DSP:
 1: get $L$ from contract according to $addr$.
 2: compare the $\sigma$ in $\mathrm{obj_h}$ and $\sigma^{BC}$ in blockchain.
 3: $b \leftarrow$ TRS.Verf(obj, $L$, $\sigma$)
 4: if $b = 0$ then
 5:     return false
 6: else
 7:     store the obj and the description.
 8: end if
```

Sharing(m, σ, C, Λ):

1. According to the description, the consumer can find and request the desired data from the DSP.
2. The DSP generates the integrity proof. For $i \in |\mathrm{q}|$, the DSP computes $\Lambda_i = \sqrt[e_i]{\prod_{j=1,j\neq i}^{\mathrm{q}} S_j^{m_j}} \mod N$, and sends $(\mathrm{obj}, C, \Lambda_i)$ to the consumer.
3. The consumer accesses the blockchain to fetch the parameters $(tpks, \sigma^{BC}, C^{BC}, \mathrm{pp}, issue)$ according to *Address* of $\mathrm{obj_h}$. Then, the consumer compares the σ of $\mathrm{obj_h}$ and the σ^{BC} of state channel.
4. The consumer verifies the proof. The consumer checks whether $S_i^{m_i} \Lambda_i^{e_i} = C$.
5. If they are equal, the consumer parses the $\sigma = (A_1, c_N, z_N)$. The consumer parses $L = (issue, tpks)$, computes $h = H(L)$, $A_0 = H'(L, \mathrm{obj_c})$, $\sigma_i = A_0 A_1^i$, $a_i = g^{z^i} y_i^{c_i}$ and $b_i = h^{z_i} \sigma_i^{c_i}$. Then, the DSP checks whether $H''(L, \mathrm{obj_c}, A_0, A_1, a_N, b_N) = \sum_n c_i \mod \mathrm{q}$.
6. If they are equal, the consumer sends the public key pk_c to the DSP. The DSP sends it to the providers. Every provider can encrypt K with pk_c to get $\{K\}_{pk_c}^{enc}$, and sends it the DSP. The DSP sends $\{K\}_{pk_c}^{enc}$ to the consumer. The consumer decrypts it with private key sk_c to obtain K. Finally, the consumer can decrypt $\mathrm{obj_c}$ to get the plaintext.

The basic process is shown in Algorithm 4:

Updating$(\mathrm{obj_c'}, \sigma', C')$:

Algorithm 4 Sharing Phase

```
CONSUMER:
 1: send request for retrieving the target obj.
DSP:
 1: $proof \leftarrow$ VC.Open(obj, $i$, $aux$)
 2: send the obj with $proof$ to the CONSUMER.
CONSUMER:
 1: $b_1 \leftarrow$ VC.Verf($C$, obj, $i$, $proof$)
 2: $b_2 \leftarrow$ TRS.Verf(obj, $L$, $\sigma$)
 3: if $(b_1 = 0 || b_2 = 0)$ then
 4:     return error
 5: end if
```

1. With updated data $\mathrm{obj_c'}$, the provider needs to re-generate new ring signature, new commitment and update the state channel. Then, the provider sends new data to the DSP.
2. The provider P_i generate new $issue'$ to build new tag $L' = issue', tpks$. P_i calculates the hash value $h' = H(L')$, compute signature $\sigma_i' = h'^{tsk_i}$, sets $A_0' = H'(L', \mathrm{obj_c'})$ and $A_1' = \left(\frac{\sigma_i'}{A_0'}\right)^{\frac{1}{i}}$. For all $j \neq i$, P_i compute $\sigma_j' = A_0' A_1'^j$. We set $\sigma_N' = (\sigma_1', \ldots, \sigma_n')$. Then, P_i randomly chooses $w_i' \leftarrow \mathbb{Z}_q$ and sets $a_i' = g^{w_i'}, b_i' = h'^{w_i'}$. For all $j \neq i$, P_i randomly chooses $z_j', c_j' \leftarrow \mathbb{Z}_q$ and sets $a_j' = g^{z_i'} y_i'^{c_j'}, b_j' = h'^{z_j'} \sigma_j'^{c_j'}$. We set $a_N' = (a_1', \ldots, a_n')$ and $b_N' = (b_1', \ldots, b_n')$. P_i sets $c' = H''(L', A_0', A_1', a_N', b_N')$ Then, P_i computes $c_i' = c' - \sum_{j\neq i} c_j' \mod \mathrm{q}$ and $z_i' = w_i' - c_i' tsk_i' \mod \mathrm{q}$. We set $c_N' = (c_1', \ldots, c_n')$ and $z_N' = (z_1', \ldots, z_n')$. The new signature of $\mathrm{obj_c'}$ is $\sigma' = (A_1', c_N', z_N')$ on $issue'$. We set $RingSig' = \sigma'$.
3. The provider P_i update commitment $C' = C \cdot S_i^{\mathrm{obj_c'} - \mathrm{obj_c}}$.
4. The provider P_i constructs a transaction to update the state channel with parameters $(\sigma', C', issue', A_0')$ to get a new transaction ID $txid'$ and update $\mathrm{obj_h'}$ with $\sigma', C', txid', x+1$. Then, P_i sends the new data $\mathrm{obj_c'}$ to the DSP with parameters C'.
5. The DSP verifies the new data and signature as explained in **uploading** phase.

The basic process is shown in Algorithm 5:

Algorithm 5 Updating Phase

```
PROVIDERS:
 1: $issue' \leftarrow$ PRNG()
 2: $L' := \{issue', tpks\}$
 3: $\sigma' \leftarrow$ TRS.Sign(obj$'$, $L'$, $sk_j$)
 4: $C' \leftarrow$ VC.Update($C, m, m'$)
 5: SC.Update($\sigma', issue', C'$)
 6: send obj$'$ to DSP.
DSP:
 1: get $L'$ from contract according to $addr$.
 2: compare the $\sigma'$ in $\mathrm{obj_h}$ and $\sigma_b'$ in blockchain.
 3: $b \leftarrow$ TRS.Verf(obj$'$, $L'$, $\sigma'$)
 4: if $b = 0$ then
 5:     return false
 6: else
 7:     store the data and the description.
 8: end if
```

Here, we focus on the primary updating process, ignoring the updating conflict, which is explained in Section 6.2.

Tracing(σ, tpk, tsk):

1. The consumer deployed an analysis module and an anomaly detection module can catch the illegal data and inform the DSP with *Identity* of obj.
2. The DSP should stop the accessibility of illegal data and sends the obj_h to the providers.
3. The providers access the blockchain to obtain the parameters $(tpks, \sigma, issue, A_0)$. Compare the $issue$ in the blockchain and it in the obj_h.
4. If they are equal, they extract the A_1 from $\sigma = (A_1, c_N, z_N)$ and compute the $\sigma_i = A_0 A_1^i$ for all $i \in [1, n]$. Then, they generate a random data m_t, set $h = H(L)$ and compute $\sigma_i' = h^{tsk_i}$. For all $i \in [1, n]$, if $\sigma_i = \sigma_i'$, the i is the order number of the malicious provider.

Algorithm 6 Tracing Phase

```
Input: obj
Output: $tpk_{mal}$
CONSUMER:
1: alert DSP that obj is illegal or incorrect.
DSP:
1: check the obj.
2: delete the description and $\text{obj}_\text{c}$.
3: inform the PROVIDERS with $\text{obj}_\text{h}$.
PROVIDERS:
1: extract $issue$ from $\text{obj}_\text{h}$.
2: $L := \{issue, tpks\}$
3: for $i \in \{\text{PROVIDERS}\}$ do
4:     $\sigma_i \leftarrow \text{TRS.Sign}(str, L, tsk_i)$
5:     $\sigma'.add(\sigma_i)$
6: end for
7: $tpk_{mal} \leftarrow \text{TRS.Trace}(L, \sigma, \sigma')$
```

Correction:

1. The providers expel the malicious provider, correct the data, rebuild the ring. Suppose that initially n providers were in the group and now there are only $n-1$ providers. Provider P_i picks up random element $tsk_i \in \mathbb{Z}_q$ and computes $y_i = g^{tsk_i}$. The TRS public key of P_i is $tpk_i = \{g, y_i, \mathbb{G}\}$ and the corresponding private key is tsk_i. We let $tpks = (tpk_1, \ldots, tpk_{n-1})$ be an ordered public-key list for $n-1$ providers.
2. The rest of providers create a new state channel SC in blockchain, generate TRS signature σ and VC commitment C for the encrypted data m, following the process explained in **creating**.
3. After saving the parameters in blockchain, the provider send them to the DSP, who verifies the σ and C of m, following the process explained in **uploading**.

6. Methods for state channel predicament

OWL uses state channel to reduce the interaction with blockchain for on-chain efficiency. However, there are some inherent drawbacks to using state channels that get in the way, such as update time, concurrent conflict, linear overhead and online assumption. We propose methods to solve them. They consist of: (1) an event-driven state channel update mechanism, (2) a concurrent conflict resolution strategy of multi-users' state channel and (3) a state channel scheme to relax the assumption.

6.1. An event-driven state channel update mechanism

To achieve consistency in an untrusted environment, the consensus mechanism causes the low throughput of the blockchain, which contradicts the high-frequency update of data in OWL. There is a mass of research [22–24,41–45] to improve the scalability of blockchain. Considering practicality, security, and efficiency, we adopt the state channel as a buffer to relieve this contradiction. All providers in one group make the state channel up. They update the data status in the channel with high-frequency off-chain and change the channel status with low-frequency on-chain according to the method described below.

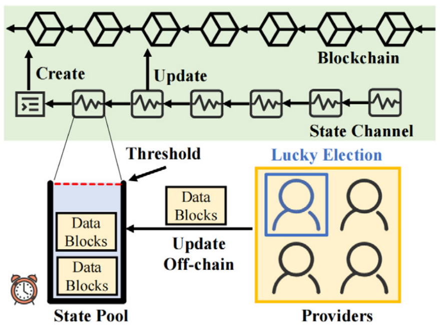


Fig. 5. The event-driven update mechanism of OWL. All members maintain an update pool, which is affected by the number and time of modified data blocks. Through lucky election, a temporary leader is selected to determine the updated data block when there is a **write–write** conflict. The leader updates the data on-chain once the threshold is reached.

A direct solution is to upload the data consistency label to the blockchain periodically, and change the upload frequency by adjusting the length of the epoch. Unfortunately, the fixed duration method can hardly meet the demand of dynamic data update. To adapt to the update demand and interact with the blockchain as little as possible, we define the data update validity (DUV), affected by the proportion of updated content (Pro) and priority (Pri) to trigger the on-chain operation.

$$DUV = \frac{\sum_1^n \left(al_i * Pro_i * Pri_i\right)}{\sum_1^n \left(Pro_i * Pri_i\right)} * 100\%$$

where i denotes the order number of data block, al_i denotes whether the ith data block is altered, if altered, $al_i = 1$ otherwise, $al_i = 0$. If the DUV is greater than a threshold $T = \alpha * LoT$, then the channel can be updated on-chain, where LoT denotes the length of time since last update on-chain and the α is the impact factor. The process is shown in Fig. 5.

6.2. Concurrent conflict resolution strategy of multi-users' state channel

Although we employ state channels to update the data, the concurrent conflict problem within a group remains unresolved in OWL. The read operation does not change the data and hence does not cause conflicts. Therefore, we focus on **read–write** and **write–write** conflicts.

Read–Write. At first, we point out the reason for this conflict. In **updating phase**, updating the information in the blockchain and the updating operation to DSP have different time costs. Generally, the blockchain has the old obj_h when DSP obtains the newer one, causing the conflict to verify the data. We use the *Version* and *TxID* to solve it. The *Version* distinguishes between new obj and old obj, we call it *freshness*. The *TxID* enables entities to access blockchain to obtain the validity. For example, obj_i with $Version = i$ is outdated than the obj_{i+1} with $Version = i + 1$, but the *TxID* is used to prove that the obj_i is just outdated but legal, we call it *validity*. On the one hand, providers update data according to the *freshness*. On the other hand, consumers are able to verifies the data based on *validity*. For example, the open-source desktop operating system Ubuntu has many types of version.[3] At the time of writing, the 22.04 LTS is the latest version, but we can still adopt the old version such as 20.04 TLS and 18.04 TLS.

[3] Ubuntu Desktop.

Table 2
Comparison of schemes.

	OWL	[5]	[8]	[11]	[12]	[13]	[14]	[15]	[16]	[27]	[28]
Data confidentiality	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Anonymity	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
Traceability	✓	✓	×	×	✓	✓	✓	✓	✓	×	×
Integrity	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Decentralized trust	✓	×	✓	×	×	×	×	×	×	✓	✓

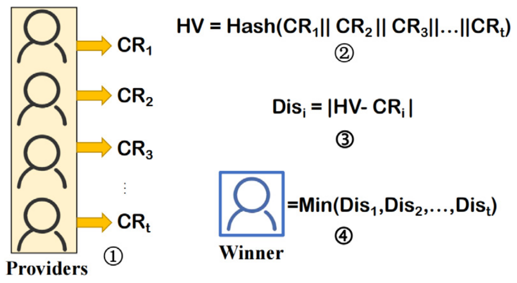


Fig. 6. A brief sketch of Randomness-based Lucky Election. ①, providers generate competitive random number CR. ②, providers calculate hash value HV with CRs. ③, providers calculate the distance between HV and CR. ④, the one who has the smallest distance wins the game. In fact, to identify each other, these steps also need to be accompanied by their respective signatures.

Write–Write. We also relieve conflicts within the state channel. Although version number has a prominent effect when different, providers often start with the same version number. A naïve scheme is to use a lock to restrain the operation of providers. It is inefficient, especially since linearizability is feasible when operations on unrelated data blocks. If these concurrent operations could not be linearized, what should we do with the group without leaders?

The existing consensus algorithm is blooming, but they target a wide range of users. However, the number of providers is limited in OWL. Consequently, we propose a randomness-based election protocol named *lucky election*, as shown in Fig. 6. It uses random to compete for a winner who decides the final version of the content in a conflict. To retain the fairness of selection, we compete by comparing the digital distance. The winner owns the minimum digital distance. In detail, after each provider updates the content, $provider_i$ needs to generate a competitive random number CR and sign it, *i.e.*, $signele_i = Sign(object_i, CR_i, sk_i)$. When it occurs to conflict, the relative providers should share the $object_i$, $signele_i$ and CR_i and calculate the hash value. Suppose there $provider_1$, $provider_3$ and $provider_4$ in this conflict, so the value $HV = hash(CR_1, CR_3, CR_4)$. Then, everyone is able to calculate the distance $dis_i = |HV - CR_i|$. The minimal dis_i implies the $provider_i$ wins the game. Of course, discard the signature $signele_i$ and random number CR_i when there is no conflict.

6.3. Linear overhead and online assumption of state channel

In traditional state channel designs, updating one channel state needs the assent of all members inside this channel (*i.e.*, the signature to validate the state) [23,43]. The overhead for communication and calculation grows linearly with the number of channel participants. The primary difference is that their updates are only available off-chain. We need to synchronize the intermediate states on-chain as the auxiliary information for data integrity auditing. Worse, the state channel further potentially requires all users to remain online when updating, which is impractical. In the age of global collaboration, a group of a dozen people from various time zones is a usual expectation. It is common that some providers are working (updating the data) while others are resting (sleeping or on their vacation), so it is hard to reach an agreement timely among all members under global collaboration.

We modify the constraints for state channel updates to reduce update overhead and the mandatory negative impact of online assumptions. First, we refer to reference. [45] to distinguish channel state changes by introducing a state field, laying the foundation for subsequent state transformations. Second, we loose the constraints of the state update. We use TRS signature σ to replace everyone's signatures so that each provider could update the channel. Finally, to prevent the malicious provider from shutting the channel, we demand that all providers participate in the establishment and closure of state channels.

7. Security analysis

In this section, we analyze the following security features of our scheme, namely: correctness, data confidentiality, anonymity and controllable anonymity. Table 2 gives the comparison with existing works.

Theorem 1 (*Correctness*). *In* OWL, *correctness indicates that the data consumer and the DSP can obtain the result 1 when they execute* VC.Veri *in data integrity verification, if the RSA assumption holds.*

Proof. In setup phase, n providers randomly choose two k/2-bit primes p_1 and p_2, set $N = p_1 p_2$, and choose $q(\ell + 1)$-bit primes $e_1, \dots, e_q$ that do not divide $\phi(N)$. For $i = 1, \dots, q$, there are $S_i = a^{\prod_{j=1, j\neq i}^{q} e_j}$. Hence, $\mathsf{pp} = \{N, a, S_1, \dots, S_q, e_1, \dots, e_q\}$.

In creating phase, one provider compute commitment $C = S_1^{m_1} S_2^{m_2} \dots S_q^{m_q}$ and the auxiliary information $\mathsf{aux} = \{m_1, \dots, m_q\}$. Then, $\{C, \mathsf{aux}\}$ are sent to the DSP.

In sharing phase, the DSP computes proof

$$\Lambda_i = \sqrt[e_i]{\prod_{j=1, j\neq i}^{q} S_j^{m_j}} \mod N.$$

And the consumer needs to verify the proof If $C = S_i^{m_i} \Lambda_i^{e_i} \mod N$, then the VC.Veri() output 1. The detail is proved in Eq. (1)

$$\begin{aligned} S_i^{m_i} \Lambda_i^{e_i} &= S_i^{m_i} \left(\sqrt[e_i]{\prod_{j=1, j\neq i}^{q} S_j^{m_j}} \right)^{e_i} \mod N \\ &= S_i^{m_i} \left(\prod_{j=1, j\neq i}^{q} S_j^{m_j} \right) \mod N \\ &= S_1^{m_1} S_2^{m_2} \dots S_q^{m_q} \mod N \\ &= C. \end{aligned} \tag{1}$$

Theorem 2 (*Data Confidentiality*). OWL *guarantees data confidentiality from the DSP.*

Proof. In setup phase, providers generate the symmetric encryption algorithm (AES-256) key K. In creating phase, the provider encrypts the data with K. By using this method, the shared $\mathrm{obj_c}$ is ciphertext. The work [46] had proved that the AES is a resistant cryptographical primitive to quantum attacks, such as quantum square attacks and quantum DS-MITM attacks. In our work, the adversary $\mathcal{A}$ with $\mathcal{PPT}$ has weaker computing capacity than quantum attackers. As a result, the $\mathcal{PPT}$ adversary $\mathcal{A}$ cannot obtain the plaintext of $\mathrm{obj_c}$ without the K.

On the other hand, our scheme uses blockchain to record parameters $(tpks, \sigma^{BC}, C^{BC}, \mathsf{pp}, issue, A_0)$. The $tpks$ and pp are public parameters that are not related to the data. Therefore, they do not reveal the data. The commitment is generated by $C = S_1^{m_1} S_2^{m_2} \dots S_q^{m_q}$. If the $\mathcal{PPT}$ adversary $\mathcal{A}$ can reverse the data by C, we can conclude the adversary breaks the RSA assumption, which contradicts the threat assumption. The $issue$ is generated by the cryptographically secure pseudorandom number generator that satisfies features. The random number $issue$ should be indistinguishable from 'true random' numbers according to specified statistical tests. The $\mathcal{PPT}$ adversary $\mathcal{A}$ cannot calculate and guess it from any given sub-sequence or value. The A_0 is calculated by the hash function $A_0 = H'(L, \mathrm{obj_c})$, where $L = issue, tpks$ do not leak privacy. The $\mathrm{obj_c}$ is the ciphertext of data. Based on the hash function feature, one-way, once a hash value has been generated, the $\mathcal{PPT}$ adversary $\mathcal{A}$ cannot convert it back into the original data. Therefore, the $\mathcal{PPT}$ adversary $\mathcal{A}$ cannot use the A_0 to guess the secret key or something else important.

Hence, the data confidentiality holds.

Theorem 3 (*Anonymity*)**.** *For the purpose of protecting the privacy of providers,* OWL *is able to restrict the following behaviors of the consumer and the DSP: (1) learn the identity from the data integrity information. (2) learn the identity from the signature.*

Proof. The consumer and the DSP cannot learn the identity of providers in the uploading, sharing updating and tracing phases.

1. The consumer cannot get the providers' identities. If the consumer abides by the protocol, he/she can fetch the desired data obj from DSP along with integrity proof Λ generated by the DSP and the signature σ signed by one of the providers. The verification of integrity as described in Appendix takes the data obj, the commitment C, and the proof Λ as the parameters, none of them contains the identity information of providers. Hence, the consumer cannot learn the identity from the data integrity information. On the other hand, the signature σ is generated by one of the providers using the traceable ring signature which guarantees anonymity proved by the work [32] under the decisional Diffie–Hellman assumption in the random oracle model. In other words, the adversary is with probabilistic polynomial time ($\mathcal{PPT}$) computing ability cannot destroy the anonymity. Therefore, OWL ensure the anonymity of providers from consumers.
2. The DSP cannot obtain the providers' identities. As the above demonstration, the DSP cannot destroy the anonymity from the signature σ. Besides, as the assumption defined in Section 3.2, some anonymous communication tools, such as Tor are used to protect the identity of providers in the network layer. Additionally, the DSP needs to generate commitment C for consumers to prove the integrity of data. The commitment of integrity as described in Appendix takes the data obj and the public parameters pp as the parameters, none of them contains the identity information of providers. Hence, the DSP cannot learn the identity from the commitment process.

Overall, OWL guarantees the anonymity of providers from consumers and the DSP.

Theorem 4 (*Controllable Anonymity*)**.** OWL *has controllable anonymity for the malicious provider while keeping the anonymity of honest providers even the illegal data are lost.*

Proof. OWL relies on the *tag-linkablity* of TRS as proved by the work [32] to trace the malicious provider. The immutability of blockchain are utilized in our work to record the *tag* to realize the *linkablity* when the target data are lost. We will first prove how TRS achieves *tag-linkablity* at high level, and then prove how we use blockchain to achieve controllable anonymity while protecting the honest providers.

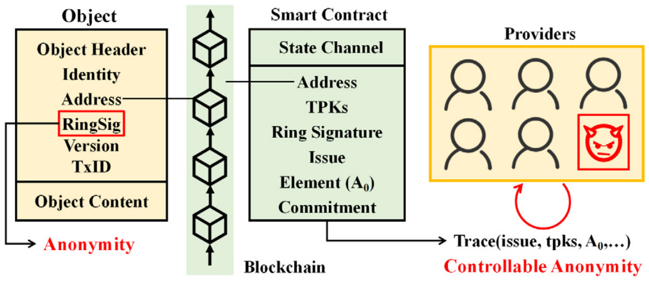


Fig. 7. Controllable anonymity for providers.

Tag-linkablity. The TRS signature σ is generated by the TRS.Sign() with parameters message m, tag $L = \{issue, tpks\}$ and the signer's private key tsk_i. The detail are given in the work [32]. We give a high level description. The signature is a triple tuple $\sigma = \{A_1, c_N, z_N\}$, the element A_1 is a computation result with parameter $issue$ and the private key tsk_i, and the other two elements c_N and z_N are computation results with parameters obj, L and tsk_i. We reduce them with a algorithm R to represent the computation program, $A_1 = \mathsf{R}_{\mathsf{A}_1}(A_0, issue, tsk_i)$, $c_N = \mathsf{R}_{\mathsf{c_N}}(m, issue, tsk_i)$ and $z_N = \mathsf{R}_{\mathsf{z_N}}(m, issue, tsk_i)$. When the provider finds the illegal data m with signature σ, the n providers will execute the tracing program along with m and L to trace the real signer. Hence, we can obtain the signatures $\sigma'_1, \dots, \sigma'_n$. By comparing σ and all new signatures, we can catch the signer s if

$$\begin{aligned}\sigma &= (A_1, c_N, z_N)\\ &= (\mathsf{R}_{\mathsf{A}_1}(A_0, issue, tsk_s), \mathsf{R}_{\mathsf{c_N}}(\mathrm{obj}, issue, tsk_s), \mathsf{R}_{\mathsf{z_N}}(\mathrm{obj}, issue, tsk_s))\\ &= (\mathsf{R}_{\mathsf{A}_1}(A_0, issue, tsk_j), \mathsf{R}_{\mathsf{c_N}}(\mathrm{obj}, issue, tsk_j), \mathsf{R}_{\mathsf{z_N}}(\mathrm{obj}, issue, tsk_j))\\ &= (A'_1, c'_N, z'_N)\\ &= \sigma'_j.\end{aligned}$$

If the obj is missing, TRS is also able to catch the malicious signer s by comparing the element A_1, as follows:

$$\begin{aligned}A_1 &= \mathsf{R}_{\mathsf{A}_1}(A_0, issue, tsk_s)\\ &= \mathsf{R}_{\mathsf{A}_1}(A_0, issue, tsk_j)\\ &= A'_1 \ in \ \sigma'_j.\end{aligned}$$

In nature, TRS uses the $issue$ and A_0 to catch the signer, which requires the same $issue$ and A_0 used to execute tracing procedure. To make sure the availability of $issue$ and A_0, OWL **stores it in the blockchain state channel**, as explained in Section 5.2. Besides, providers is supposed to generate a new $issue$ when they upload or update the data (required in **uploading and updating phases**) to avoid that the same $issue$ and A_0 links the signer by the outsiders of this group. The brief flow is shown in Fig. 7.

In summary, OWL realizes controllable anonymity.

8. Evaluation

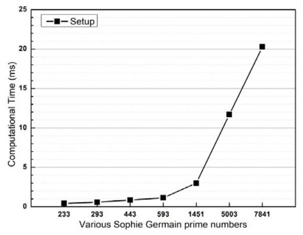
In this section, we focus on the extra overhead for controllable anonymity and integrity. To understand the improvements and trade-offs of OWL's design in decentralized trust, especially, our evaluation answers these questions:

Section 8.1: How does the effect of cryptographic tools, such as traceable ring signature and vector commitment?

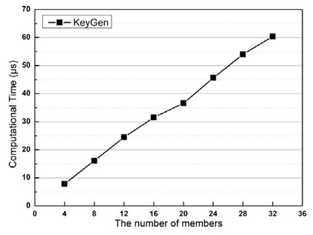
Section 8.2.1: How does the effect of *event-driven update* compare to *epoch-based update*?

Section 8.2.2: How does blockchain smart contracts affect performance?

Our evaluation was conducted on a laptop with an Ubuntu 20.04 operating system, an Intel Core i7-10510U processor and 8 GB of



(a) Setup with several variable Sophie Germain prime numbers



(b) KeyGen under variable members

Fig. 8. The TRS computational time of two algorithms, Setup() and KGen().

Table 3
Comparison of storage overhead with data block size = 4 kB (Unit = kB).

	OWL	BLS-based [10][a]	RSA-based [10][a]	[9][a]
Space	0.616	243	223	280

[a]Tolerance rate is 99% and the total file size is 1 GB.

memory. We implemented traceable ring signature and vector commitment with the Python3.8 and encryption algorithm library PyCryptome. OWL uses blockchain to store auxiliary information for controllable anonymity and integrity. In our implementation, we used Hyperledger Fabric v2.3.2 as blockchain to run state channel smart contract coded via IBM blockchain platform. We also evaluated the performance with Hyperledger Caliper v0.4.2. We implemented a proof of concept of scheme event-driven state channel update mechanism proposed in Section 6.1.

8.1. Microbenchmarks

8.1.1. Traceable ring signature

The TRS enables OWL to trace the malicious *provider*. Here, we focus on its computational overhead to evaluate the availability. TRS consists of five algorithms, Setup(), KGen(), Sign(), Verf() and Trace(). We first tested the variable Sophie Germain prime numbers that is used to generate its group to map the operation into cryptographic computation in Setup(). As shown in Fig. 8(a), the larger the number, the longer it takes, and the higher the security. Next, we tested effect of the variable group member numbers, which leads to the linear public/private key pairs (tpk, tsk). In the case of 32 members, it only costs about 60.37 ms, given in Fig. 8(b).

Consider the impact of data size and number of members, we tested the rest algorithms, respectively. Especially, we divided Trace() into Trace1() and Trace2(). The former uses the message used in Sign(), while the latter uses arbitrary message to execute it. As shown in Fig. 9, all overhead increases as the data size and the number of members increase. Notably, the Trace2() costs almost half as much as Trace1(), consequently the tracing costs are independent of the original data. In other words, without the original data, not only can it be traced back, but it is also faster.

8.1.2. Vector commitment

We focus on the overhead of vector commitment of each phase in OWL, adopting scheme in [47] to evaluate it. We worked on the size of modulus N is 1024 bits, and all results are the averages of 1000 trials. Refer to the experiments in work [48], Fig. 10 shows the computational time in phases, $Verify$, $Update$ and $UpdateProof$ for 2^{20} bits file under various data block size, 512 Bytes, 1 kB, 2 kB and 4 kB. We concluded that the overhead of providers (phases $KeyGen$, $Commit$) is huge, while the overhead of consumers (phase $Verify$) is quite small. We listed them in Table 3.

8.2. Macrobenchmarks

8.2.1. Event-driven update mechanism

Considering the urgency of update and the throughput of blockchain, we defined the following two metrics: (1) DUV, which measures the value of each on-chain operation. (2) Rounds, which represent the number of on-chain operation within a certain period.

Here, we used eight datasets to compare the difference between event-driven mechanism and epoch-based mechanism. We suppose there is one group of providers, the data consists of 10,000 blocks, and each block has its priority (1 to 4). Every dataset consists of massive records, which denote the modified time and corresponding block numbers.

The eight datasets are composed of four open datasets and four datasets that we generated according to certain rules. For the first four datasets, we used the original time as the modification time and generate some modified data blocks (1 to 10) for each term. For the last four datasets, we set the time to go through one month $(30 * 24 * 3600$ s), and aggregated 100,000 modification operations into records according to variable distributions, and each modification involves 1 to 10 data blocks. The following is the description of the datasets.

(a) **Check-Ins**. This dataset contains check-ins in NYC and Tokyo collected for about 10 months (from 12 April 2012 to 16 February 2013). It contains 227,428 check-ins in New York city and 573,703 check-ins in Tokyo [49].
(b) **FDC**. This dataset contains 4,187,024 records in “San Francisco Fire Department Calls” and “San Francisco Elevation Data” for about 17 years (from 12 April 2000 to 29 October 2016) [50].
(c) **UCI**. This dataset contains 1,067,372 transactions occurring for a UK-based and registered, non-store online retail from 1 December 2009 to 9 December 2011 [51]. The minimum unit of time for this dataset is minutes and the rest seven datasets are seconds.
(d) **MA**.[4] This data set contains 110,528 medical appointments from 10 November 2015 to 8 June 2016.
(e) **DP** (Poisson). A dataset satisfying Poisson distribution, consists of 6454 records.
(f) **DN** (normal). A dataset satisfying Normal distribution, consists of 98,091 records, $\lambda = 3.6K * 24 * 15$, and $\sigma^2 = 500K$
(g) **DU** (uniform). A dataset satisfying Uniform distribution, consists of 97,240 records.
(h) **DZ** (zipf). A dataset satisfying Zipf distribution, consists of 6013 records, $\alpha = 1.5$.

We also set variable parameters for event-driven mechanism (Eved) and epoch-based mechanism (Epob), respectively. For event-driven mechanism, we set 4 thresholds denotes low, medium, high and extreme. For epoch-based mechanism, we set 4 time epochs: (a) 1 h, (b) 6 h, (c) 12 h and (d) 24 h. Of course, if no update occurs within the time interval, there will be no on-chain operation. For comparison, we also set the baseline scheme Base, that is, any updates will trigger the on-chain operation. Results are shown in Fig. 10.

We used black squares to represent rounds and red circles to represent average duv. In terms of the overall trend, Eved has more $rounds$ and less avr_duv, and Epob acts contrariwise. Base has the most rounds and the least avr_duv. Back to the rounds, whether it is Eved or Epob, these schemes effectively reduce the number of on-chain operation. Even Eved-low with the most rounds, is still far less than Base. More rounds means more on-chain operations, which means more throughput pressure for blockchain. However, from the perspective of efficiency and profit, we cannot directly determine which one is perfect. Recall that the first four open datasets covers tourism, public security, payment and medical industry. Each industry has variable efficiency needs, according to their various impact on

[4] Medical Appointment No Shows.

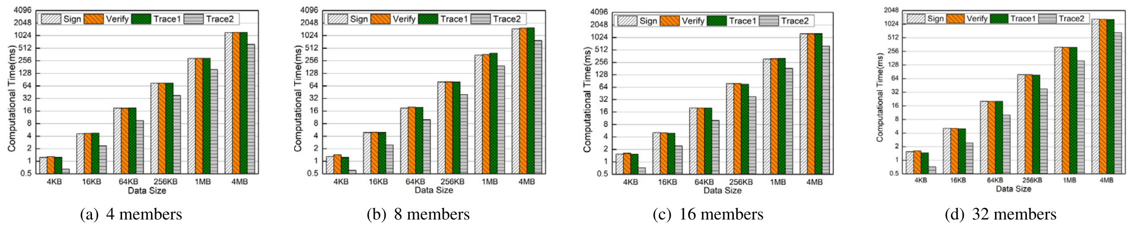


Fig. 9. The TRS computational time of four algorithms, Sign(), Verf(), Trace1() with same message, Trace2() with arbitrary message (1 byte in our evaluation), under variable size of data when the group members are 4, 8, 16 and 32 (Sophie Germain prime number $q = 443$).

Table 4
Performance metrics for ReadState and UpdateState.

Name	Succ	Fail	Send rate (TPS)	Max latency (s)	Min latency (s)	Avg latency (s)	Throughput (TPS)
ReadState	9960	0	353.8	0.19	0.00	0.02	353.7
UpdateState	13 161	1039	156.2	2.07	0.03	0.66	156.1

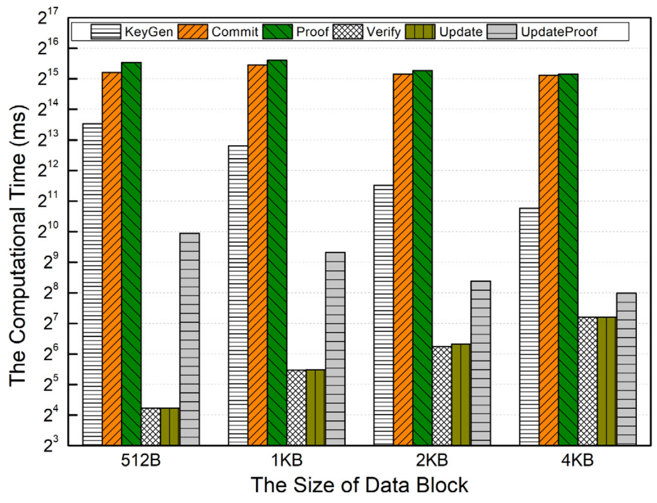


Fig. 10. The computational time of VC under variable size of data block.

society. Based on the distribution of polylines, we can put Figs. 11(a) and 11(c) as a group and put Figs. 11(b) and 11(d) as the other group. The first group is closer to the field of people's livelihood, and we argue that they have lower requirements for efficiency and are more suitable for Epob, which has a higher duv for each transaction. The second group is more relevant to the field of life security, and we all agree that they have higher requirements for efficiency. Although the data value is measured by duv, the potential value is much higher than the value we set. Hence, these fields prefer Eved.

8.2.2. Blockchain

The Hyperledger Fabric network contains one order-node and two peer-node in two organization, respectively, and delay = 0.01, retry = 100. We developed smart contract with functions CreateState, ReadState and UpdateState, the latter two functions are the goals to be tested. The benchmark in Caliper consists of 10 workers for one round with transaction duration 30 s.

The throughput and latency is shown in Table 4. The ReadState does not involve write operations, so the throughput is higher, reaching about 353 TPS with 100% success rate. The UpdateState, however, modifies the state of the data, triggers consensus behavior, reducing the TPS to 156. It is worth noting that there 1039 transactions of UpdateState fails to execute due to **key collisions** in the high read–write concurrency. In detail, when you submit a transaction, the peer in Fabric generates a read–write set. This read–write set is then used when the transaction is committed to the ledger. It contains the identity and version (**key**) of the variables to be read/written. If, during the time between set creation and committing, a different transaction $TX2$ was committed and changed the version of the variable, the original transaction $TX1$ will be rejected during committal because the version when read is not the current version. This experiment reflects the fact that multiple changes to the same data in a short time (less than the consensus time) will cause the above conflict. Fortunately, OWL adopts state channel as a buffer to gather these transactions (with same **key**) into one transaction off-chain, explained in Section 6.2, avoiding the **key collisions** effectively.

9. Discussion

This section discusses the limitations of OWL. It is inefficient on the dynamic side and not good at the denial of service (DoS) attacks. We also give possible future work directions.

9.1. Dynamics

OWL utilizes the traceable ring signature to achieve controllable anonymity and adopts the state channel to alleviate conflicts among providers. However, these technologies suffer from poor dynamics. The TRS we used does not support member joining and revocation. As a result, whenever a member joins or leaves, the other members in this group must perform the setup procedure together and resign the shared data. To fix that, we are researching a TRS variation that enables dynamic members. On the other hand, the state channel is also hard to update the member. Although we use the ring signature to loosen the restriction of updating the channel state in the blockchain, the open and close procedure still demands the involvement of all members in this group. It is easy to set an administrator to manage the change of members, but it introduces a centralized trust. A dynamic state channel is also a potential research direction.

9.2. DoS attacks

In OWL, the DSP is vulnerable to DoS attacks from consumers. The malicious provider might continually upload illegal data to pollute the data sharing pool. OWL is able to catch the evildoer and reorganize the group to exclude him to halt this attack. However, consumers are free to request the DSP indefinitely, and OWL has no methods to hold it. The work [8] uses anonymous payments and proof of work to alleviate it. These methods can be implemented into our scheme. Furthermore, we can also study the traceable anonymous payment scheme suitable for data sharing.

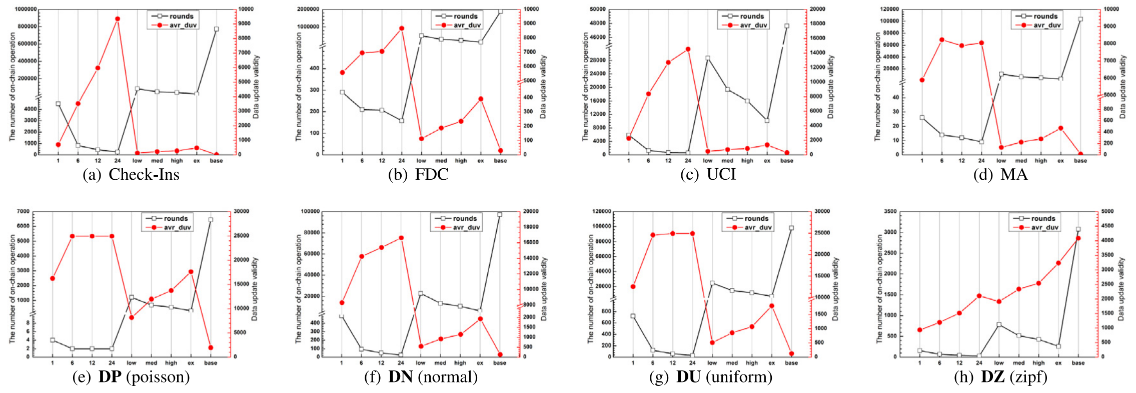


Fig. 11. The comparison between event-driven mechanism (Eved), epoch-based mechanism (Epob) and baseline mechanism (Base) in the number of on-chain operation and average data update validity. The first four number in X axis represent time interval, 1 h, 6 h, 12 h and 24 h, respectively. The next four words represent the threshold T, $low(1000)$, $medium(5000)$, $high(10,000)$ and $extreme(50,000)$, respectively, where the impact factor $\alpha = 0.01$. The words "base" denotes scheme Base, this mechanism means any update will trigger the on-chain operation.

10. Conclusion

To address the privacy and integrity concerns in the existing data-sharing, we proposed OWL, a data-sharing scheme with integrity and controllable anonymity for group users. We designed traceable data structures along with traceable ring signatures to achieve controllable anonymity that can catch the malicious provider while protecting the honest providers. Vector commitment is used to ensure data integrity and prevent privacy leakage. OWL employs the blockchain to store auxiliary information and uses the state channel to alleviate the contradiction between the poor throughput of blockchain and the high-frequency data update. Methods are proposed to resolve the concurrent conflicts among group users in a single state channel. Our evaluation demonstrates the utility of OWL. We also discuss the limitations of OWL in terms of dynamics and DoS attack, which would be addressed in future work.

CRediT authorship contribution statement

Yongxin Zhang: Conceptualization, Methodology, Formal analysis, Software, Writing – original draft. **Zijian Bao:** Conceptualization, Methodology, Formal analysis, Writing – review & editing. **Qinghao Wang:** Investigation. **Ning Lu:** Writing – review & editing. **Wenbo Shi:** Writing – review & editing. **Bangdao Chen:** Supervision. **Hong Lei:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the Finance Science and Technology Project of Hainan Province (No. ZDKJ2020009); in part by the National Key R&D Program of China (No. 2021YFB2700601); in part by the National Natural Science Foundation of China (Nos. 62163011, 62072092, 62072093 and U1708262); in part by the Fundamental Research Funds for the Central Universities, China (No. N2023020); in part by the Natural Science Foundation of Hebei Province, China (No. F2020501013); in part by the China Postdoctoral Science Foundation (No. 2019M653568); and in part by the Key Research and Development Project of Hebei Province (No. 20310702D).

Appendix. Building blocks

The original definition of vector commitment [47] consists of algorithms (VC.KeyGen, VC.Com, VC.Open, VC.Ver, VC.Update, VC.ProofUpdate) such that:

- VC.KGen(1^k, ℓ,v). *Given the security parameter k and the size v of the committed vector with v = poly(k), the key generation outputs some public parameters* pp.
- VC.Com(pp, $m_1, m_2, \dots, m_v$). *On input a sequence of v messages $m_1, \dots, m_v \in \mathcal{M}$ ($\mathcal{M}$ is the message space) and the public parameters* pp*, the algorithm outputs a commitment string C and an auxiliary information* aux.
- VC.Open(m, i, aux). *This algorithm is run by the committer to produce a proof i that m is the ith committed message.*
- VC.Verf(C, m, i, Λ_i). *The verification algorithm accepts (i.e., it outputs 1) only if Λ_i is a valid proof that C was created to a sequence $m_1, \dots, m_q$ such that $m = m_i$.*
- VC.Update(C, m, m', i). *This algorithm is run by the committer, takes as input the old message m, the new message m' and the position i. It outputs a new commitment C' together with an update information U.*
- VC.ProofUpdate(C, Λ_i, m', i, U). *This algorithm can be run by any user who holds a proof Λ_i for some message at positionj w.r.t. C, and it allows the user to compute an updated proof Λ_j (and the updated commitment C') such that Λ'_j will be valid with regard to C' which contains m' as the new message at position i. Basically, the value U contains the update information.*

The original definition of traceable ring signature [32] consists of algorithms (TRS.KGen, TRS.Sign, TRS.Ver, TRS.Trace) such that:

- TRS.KGen(1^{k}, n). *Given the security parameter k and the number of the group members n, the algorithm outputs ring public/private key pairs $(tpk_1, tsk_1), (tpk_2, tsk_2), \dots, (tpk_n, tsk_n)$.*
- TRS.Sign(m, L, $\mathsf{sk_i}$). *Given the message $m \in \{0,1\}^*$ with respect to tag $L = (issue, tpks)$, where $tpks = \{tpk_1, tpk_2, \dots, tpk_n\}$, and the private key tsk_i, the algorithm outputs the signature σ.*
- TRS.Verf(m, L, σ). *Given the message m, tag L and signature σ. If all checks are successfully completed, accept it and return 1, otherwise reject it and return $\perp$.*
- TRS.Trace(L, σ, σ'). *Given the tag L, two different signatureσ and σ' for different message m and m', respectively, the algorithm outputs the tracing result. If only one $user_i$ meets, output the tpk_i, if all users meet, output "linked", if more than one but not the whole, output "indep".*

We have modified state channel [23] into an OWL-customized multi-party state channel scheme, including (SC.Setup, SC.Open, SC.Update, SC.Close) such that:

- $\mathsf{SC.Setup}(1^k, n)$. *Given the security parameter k and the number of the group members n, the algorithm outputs public/private key pairs (pk_i, sk_i).*
- $\mathsf{SC.Open}(\mathsf{pks}, \mathsf{tpks}, \mathsf{signs}, \mathsf{rsign}, \mathsf{info})$. *On input the $signs$, pks and $data$, the algorithm first checks the validity of $signs$ for $data$ with pks. Then set the owner of Channel via pks, set the public information with $info$, set the ring public key with $tpks$ and return 1. If not pass, return $\perp$. ($pks = \{pk_1, pk_2, \dots, pk_n\}$, $signs = \{sign_1, sign_2, \dots, sign_n\}$)*
- $\mathsf{SC.Update}(\mathsf{rsign}, \mathsf{info})$. *On input the $rsign$ and $data$, the algorithm first checks the validity of $rsign$ for $data$ with $tpks$. Then reset the public information with $info$ and return 1. If not pass, return $\perp$.*
- $\mathsf{SC.Close}(\mathsf{signs}, \mathsf{rsign}, \mathsf{info})$. *On input the $signs$, $rsign$ and $info$, the algorithm first checks the validity of $rsign$ for $data$ with $tpks$ and check the validity of $signs$ with pks Then reset the public information with $info$ and return 1. If not pass, return $\perp$.*

References

[1] Y. Lu, X. Huang, Y. Dai, S. Maharjan, Y. Zhang, Blockchain and federated learning for privacy-preserved data sharing in industrial IoT, IEEE Trans. Ind. Inform. 16 (6) (2020) 4177–4186, http://dx.doi.org/10.1109/TII.2019.2942190.

[2] J. Shen, D. Liu, Q. Liu, X. Sun, Y. Zhang, Secure authentication in cloud big data with hierarchical attribute authorization structure, IEEE Trans. Big Data 7 (4) (2021) 668–677, http://dx.doi.org/10.1109/TBDATA.2017.2705048.

[3] H. Wang, D. He, J. Yu, Z. Wang, Incentive and unconditionally anonymous identity-based public provable data possession, IEEE Trans. Serv. Comput. 12 (5) (2019) 824–835, http://dx.doi.org/10.1109/TSC.2016.2633260.

[4] S.K. Nayak, S. Tripathy, SEPDP: Secure and efficient privacy preserving provable data possession in cloud storage, IEEE Trans. Serv. Comput. 14 (3) (2021) 876–888, http://dx.doi.org/10.1109/TSC.2018.2820713.

[5] K. He, J. Chen, Q. Yuan, S. Ji, D. He, R. Du, Dynamic group-oriented provable data possession in the cloud, IEEE Trans. Dependable Secur. Comput. 18 (3) (2021) 1394–1408, http://dx.doi.org/10.1109/TDSC.2019.2925800.

[6] G. Ateniese, R.C. Burns, R. Curtmola, J. Herring, L. Kissner, Z.N.J. Peterson, D.X. Song, Provable data possession at untrusted stores, in: Proceedings of the 2007 ACM Conference on Computer and Communications Security, (CCS), ACM, 2007, pp. 598–609, http://dx.doi.org/10.1145/1315245.1315318.

[7] Y. He, Y. Xu, X. Jia, S. Zhang, P. Liu, S. Chang, EnclavePDP: A general framework to verify data integrity in cloud using intel SGX, in: Proceeding of the 2020 International Symposium on Research in Attacks, Intrusions and Defenses, (RAID), USENIX, 2020, pp. 195–208.

[8] Y. Hu, S. Kumar, R.A. Popa, Ghostor: Toward a secure data-sharing system from decentralized trust, in: Proceeding of the 2020 USENIX Symposium on Networked Systems Design and Implementation,(NSDI), USENIX Association, 2020, pp. 851–877.

[9] C.C. Erway, A. Küpçü, C. Papamanthou, R. Tamassia, Dynamic provable data possession, in: Proceedings of the 2009 ACM Conference on Computer and Communications Security, (CCS), ACM, 2009, pp. 213–222, http://dx.doi.org/10.1145/1653662.1653688.

[10] Q. Wang, C. Wang, J. Li, K. Ren, W. Lou, Enabling public verifiability and data dynamics for storage security in cloud computing, in: Proceeding of the 2009 European Symposium on Research in Computer Security, Vol. 5789, 2009, pp. 355–370, http://dx.doi.org/10.1007/978-3-642-04444-1_22.

[11] W. Shen, J. Qin, J. Yu, R. Hao, J. Hu, Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage, IEEE Trans. Inf. Forensics Secur. 14 (2) (2019) 331–346, http://dx.doi.org/10.1109/TIFS.2018.2850312.

[12] A. K., M.B. Moses, An efficient SG-DACM framework for data integrity with user revocation in role based multiuser cloud environment, Comput. Commun. 155 (2020) 84–92, http://dx.doi.org/10.1016/j.comcom.2020.03.006.

[13] W. Song, Y. Wu, Y. Cui, Q. Liu, Y. Shen, Z. Qiu, J. Yao, Z. Peng, Public integrity verification for data sharing in cloud with asynchronous revocation, Digit. Commun. Netw. 8 (1) (2022) 33–43, http://dx.doi.org/10.1016/j.dcan.2021.02.002.

[14] Y. Yan, L. Wu, W. Xu, H. Wang, Z.M. Liu, Integrity audit of shared cloud data with identity tracking, Secur. Commun. Netw. 2019 (2019) 1354346:1–1354346:11, http://dx.doi.org/10.1155/2019/1354346.

[15] H. Huang, X. Chen, J. Wang, Blockchain-based multiple groups data sharing with anonymity and traceability, Sci. China Inf. Sci. 63 (3) (2020) http://dx.doi.org/10.1007/s11432-018-9781-0.

[16] F. Wu, X. Li, L. Xu, S. Kumari, A privacy-preserving scheme with identity traceable property for smart grid, Comput. Commun. 157 (2020) 38–44, http://dx.doi.org/10.1016/j.comcom.2020.03.047.

[17] E.B. Sifah, Q. Xia, K.O.O. Agyekum, H. Xia, A. Smahi, J. Gao, A blockchain approach to ensuring provenance to outsourced cloud data in a sharing ecosystem, IEEE Syst. J. 16 (1) (2022) 1673–1684, http://dx.doi.org/10.1109/JSYST.2021.3068224.

[18] K.O.O. Agyekum, Q. Xia, E.B. Sifah, C.N.A. Cobblah, H. Xia, J. Gao, A proxy re-encryption approach to secure data sharing in the internet of things based on blockchain, IEEE Syst. J. 16 (1) (2022) 1685–1696, http://dx.doi.org/10.1109/JSYST.2021.3076759.

[19] N. Lu, Y. Zhang, W. Shi, S. Kumari, K.R. Choo, A secure and scalable data integrity auditing scheme based on hyperledger fabric, Comput. Secur. 92 (2020) 101741, http://dx.doi.org/10.1016/j.cose.2020.101741.

[20] K. Fan, Z. Bao, M. Liu, A.V. Vasilakos, W. Shi, Dredas: Decentralized, reliable and efficient remote outsourced data auditing scheme with blockchain smart contract for industrial IoT, Future Gener. Comput. Syst. 110 (2020) 665–674, http://dx.doi.org/10.1016/j.future.2019.10.014.

[21] A.R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, I. Miers, Fairness in an unfair world: Fair multiparty computation from public bulletin boards, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, (CCS), ACM, 2017, pp. 719–728, http://dx.doi.org/10.1145/3133956.3134092.

[22] A. Gervais, G.O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, S. Capkun, On the security and performance of proof of work blockchains, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, (CCS), ACM, 2016, pp. 3–16, http://dx.doi.org/10.1145/2976749.2978341.

[23] S. Dziembowski, S. Faust, K. Hostáková, General state channel networks, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, (CCS), 2018, pp. 949–966, http://dx.doi.org/10.1145/3243734.3243856.

[24] M. Zamani, M. Movahedi, M. Raykova, RapidChain: Scaling blockchain via full sharding, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, (CCS), ACM, 2018, pp. 931–948, http://dx.doi.org/10.1145/3243734.3243853.

[25] V. Rajkumar, M. Prakash, V. Vennila, Secure data sharing with confidentiality, integrity and access control in cloud environment, Comput. Syst. Sci. Eng. 40 (2) (2022) 779–793, http://dx.doi.org/10.32604/csse.2022.019622.

[26] J. Domingo-Ferrer, O. Farràs, J. Ribes-González, D. Sánchez, Privacy-preserving cloud computing on sensitive data: A survey of methods, products and challenges, Comput. Commun. 140–141 (2019) 38–60, http://dx.doi.org/10.1016/j.comcom.2019.04.011.

[27] Z. Chen, W. Xu, B. Wang, H. Yu, A blockchain-based preserving and sharing system for medical data privacy, Future Gener. Comput. Syst. 124 (2021) 338–350, http://dx.doi.org/10.1016/j.future.2021.05.023.

[28] K.M. Hossein, M.E. Esmaeili, T. Dargahi, A. Khonsari, M. Conti, BCHealth: A novel blockchain-based privacy-preserving architecture for IoT healthcare applications, Comput. Commun. 180 (2021) 31–47, http://dx.doi.org/10.1016/j.comcom.2021.08.011.

[29] M. Backes, A. Herzberg, A. Kate, I. Pryvalov, Anonymous RAM, in: Proceeding of the 2016 European Symposium on Research in Computer Security, (ESORICS), Vol. 9878, Springer, 2016, pp. 344–362, http://dx.doi.org/10.1007/978-3-319-45744-4_17.

[30] A. Hamlin, R. Ostrovsky, M. Weiss, D. Wichs, Private anonymous data access, in: Proceeding of the 2019 Annual International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT), Vol. 11477, Springer, 2019, pp. 244–273, http://dx.doi.org/10.1007/978-3-030-17656-3_9.

[31] D. Chaum, E. van Heyst, Group signatures, in: Proceeding of the 1991 Workshop on the Theory and Application of of Cryptographic Techniques, Vol. 547, Springer, 1991, pp. 257–265, http://dx.doi.org/10.1007/3-540-46416-6_22.

[32] E. Fujisaki, K. Suzuki, Traceable ring signature, in: Proceedings of the 2007 International Conference on Practice and Theory in Public-Key Cryptography, Vol. 4450, PKC, Springer, 2007, pp. 181–200, http://dx.doi.org/10.1007/978-3-540-71677-8_13.

[33] A. Scafuro, B. Zhang, One-time traceable ring signatures, in: Proceeding of the 2021 European Symposium on Research in Computer Security, Vol. 12973, ESORICS, Springer, 2021, pp. 481–500, http://dx.doi.org/10.1007/978-3-030-88428-4_24.

[34] H. Tan, An efficient IoT group association and data sharing mechanism in edge computing paradigm, Cyber Secur. Appl. 1 (2023) 100003, http://dx.doi.org/10.1016/j.csa.2022.100003.

[35] Z. Xu, M. Luo, C. Peng, Q. Feng, Sanitizable signature scheme with privacy protection for electronic medical data sharing, Cyber Secur. Appl. 1 (2023) 100018, http://dx.doi.org/10.1016/j.csa.2023.100018.

[36] Y. Xu, L. Ding, J. Cui, H. Zhong, J. Yu, PP-CSA: A privacy-preserving cloud storage auditing scheme for data sharing, IEEE Syst. J. 15 (3) (2021) 3730–3739, http://dx.doi.org/10.1109/JSYST.2020.3018692.

[37] C. Priebe, D. Muthukumaran, D. O'Keeffe, D.M. Eyers, B. Shand, R. Kapitza, P.R. Pietzuch, CloudSafetyNet: Detecting data leakage between cloud tenants, in: Proceedings of the 2014 Edition of the ACM Workshop on Cloud Computing Security, (CCSW), ACM, 2014, pp. 117–128, http://dx.doi.org/10.1145/2664168.2664174.

[38] Q. Feng, D. He, S. Zeadally, M.K. Khan, N. Kumar, A survey on privacy protection in blockchain system, J. Netw. Comput. Appl. 126 (2019) 45–58, http://dx.doi.org/10.1016/j.jnca.2018.10.020.

[39] A.M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, G. Danezis, The loopix anonymity system, in: Proceeding of the 2017 USENIX Conference on Security Symposium, USENIX, 2017, pp. 1199–1216.

[40] R. Dingledine, N. Mathewson, P.F. Syverson, Tor: The second-generation onion router, in: Proceedings of the 2004 USENIX Conference on Security Symposium, USENIX, 2004, pp. 303–320.

[41] H. Dang, T.T.A. Dinh, D. Loghin, E. Chang, Q. Lin, B.C. Ooi, Towards scaling blockchain systems via sharding, in: Proceedings of the 2019 International Conference on Management of Data, (SIGMOD), ACM, 2019, pp. 123–140, http://dx.doi.org/10.1145/3299869.3319889.

[42] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, B. Ford, OmniLedger: A secure, scale-out, decentralized ledger via sharding, in: Proceedings of the 2018 IEEE Symposium on Security and Privacy, (S&P), IEEE Computer Society, 2018, pp. 583–598, http://dx.doi.org/10.1109/SP.2018.000-5.

[43] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, K. Hostáková, Multi-party virtual state channels, in: Proceeding of the 2019 Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vol. 11476, EUROCRYPT, 2019, pp. 625–656, http://dx.doi.org/10.1007/978-3-030-17653-2_21.

[44] S. Dziembowski, L. Eckey, S. Faust, D. Malinowski, Perun: Virtual payment hubs over cryptocurrencies, in: Proceeding of the 2019 IEEE Symposium on Security and Privacy, (S&P), 2019, pp. 106–123, http://dx.doi.org/10.1109/SP.2019.00020.

[45] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, P. McCorry, Sprites and state channels: Payment networks that go faster than lightning, in: Proceeding of the 2019 Financial Cryptography and Data Security International Conference, (FC), Vol. 11598, Springer, 2019, pp. 508–526, http://dx.doi.org/10.1007/978-3-030-32101-7_30.

[46] X. Bonnetain, M. Naya-Plasencia, A. Schrottenloher, Quantum security analysis of AES, IACR Trans. Symmetric Cryptol. 2019 (2) (2019) 55–93, http://dx.doi.org/10.13154/tosc.v2019.i2.55-93.

[47] D. Catalano, D. Fiore, Vector commitments and their applications, in: Proceeding of the 2013 International Conference on Practice and Theory in Public-Key Cryptography, (PKC), Vol. 7778, 2013, pp. 55–72, http://dx.doi.org/10.1007/978-3-642-36362-7_5.

[48] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, L. Nizzardo, Incrementally aggregatable vector commitments and applications to verifiable decentralized storage, in: Proceeding of the 2020 International Conference on the Theory and Application of Cryptology and Information Security,(ASIACRYPT), Vol. 12492, Springer, 2020, pp. 3–35, http://dx.doi.org/10.1007/978-3-030-64834-3_1.

[49] D. Yang, D. Zhang, V.W. Zheng, Z. Yu, Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs, IEEE Trans. Syst. Man Cybern. Syst. 45 (1) (2015) 129–142, http://dx.doi.org/10.1109/TSMC.2014.2327053.

[50] K.S. Carignan, L.A. Taylor, B.W. Eakins, R.J. Caldwell, D.Z. Friday, P.R. Grothe, E. Lim, Digital elevation models of central California and San Francisco Bay: procedures, data sources, and analysis, 2011, URL https://repository.library.noaa.gov/view/noaa/1189/noaa_1189_DS1.pdf.

[51] L. Ale, N. Zhang, H. Wu, D. Chen, T. Han, Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network, IEEE Internet Things J. 6 (3) (2019) 5520–5530, http://dx.doi.org/10.1109/JIOT.2019.2903245.