Distributedly and Provably Continuous Storage Scheme with Forward Secure for IoT Healthcare Data

Yuan Su, Jiahui Li, Jiliang Li, Wei Tang, Chengyi Dong, Jiawei Zhang, Kefeng Ding, Shouling Ji, *Member, IEEE*

***Abstract*—The proliferation of healthcare devices enabled by consumer electronics has led to the generation of massive volumes of sensitive data, posing significant challenges for secure and continuous storage. While distributed edge storage offers scalability and accessibility, existing provable continuous storage schemes must predefine the entire storage period and precompute all audit proofs by repeatedly evaluating trapdoor-based Verifiable Delay Functions (VDF), incurring preprocessing costs that scale linearly with storage time and audit frequency. Moreover, once these proofs are generated, these schemes become inherently static - any data update invalidates all precomputed proofs and necessitates the regeneration of the entire preprocessing phase, which renders real-time updates impractical. In this paper, we propose a Distributedly and Provably Continuous Storage Scheme (DPCSS) tailored for healthcare data. DPCSS enables periodic integrity verification without trusted auditors by combining VDF with homomorphic authenticators. To eliminate the inefficiency of local pre-verification, DPCSS introduces a cross-validation method among edge servers that allows sequentially proving and then verifying. Furthermore, a structural interaction protocol is designed to reduce communication overhead, and puncturable key wrapping is leveraged to enable efficient, real-time data and key updates with forward security. Formal security analysis demonstrates that DPCSS ensures confidentiality, forward security, and secure against forgery attack, and experimental results show that our DPCSS achieves low verification cost and constant-time preprocessing that is independent of storage time and audit frequencies, where DPCSS improves the preprocessing time by about 1.4 times compared to the state-of-the-art scheme.**

***Index Terms*—Distributed and continuous storage, cross-validation, real-time updates, data forward security.**

The work was supported in part by the China Postdoctoral Science Foundation (No.2025M781523 and No. 2025M780763), State Key Laboratory of Cryptography and Digital Economy Security (No. KFYB2504), Zhejiang Provincial Natural Science Foundation Exploration of China (No. LMS26F020003), the Fundamental Research Funds for the Central Universities (No. ZYTS25194), the Natural Science Foundation of China (No. 62472347). (Yuan Su and Jiahui Li are co-first authors. Shouling Ji and Jiliang Li are corresponding authors.)

Yuan Su is with the Cancer Institute (Key Laboratory of Cancer Prevention and Intervention, Ministry of Education), The Second Affiliated Hospital, School of Medicine, Zhejiang University, Hangzhou 310027, with State Key Laboratory of Cryptography and Digital Economy Security, Qingdao 266237, and with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: yuansu@zju.edu.cn).

Jiahui Li is with School of Mathematics and Statistics, Xidian University, Xi'an 710071, China (e-mail: lijiahui@xidian.edu.cn).

Jiliang Li, Wei Tang and Chengyi Dong are with the School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: tangwei2023@stu.xjtu.edu.cn; jiliang.li@xjtu.edu.cn; dcy920609@gmail.com).

Jiawei Zhang and Kefeng Ding are with the Cancer Institute (Key Laboratory of Cancer Prevention and Intervention, Ministry of Education), The Second Affiliated Hospital, School of Medicine, Zhejiang University, Hangzhou 310027, Zhejiang, China (e-mail: jwzhang@zju.edu.cn; dingkefeng@zju.edu.cn.

Shouling Ji is with the Cancer Institute (Key Laboratory of Cancer Prevention and Intervention, Ministry of Education), The Second Affiliated Hospital, School of Medicine, Zhejiang University, Hangzhou 310027, Zhejiang, China, with Sanya Traditional Chinese Medicine Hospital, Sanya 572000, China, and with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: sji@zju.edu.cn).

Manuscript received XX XX, 2025; revised XX XX, 2025.

I. Introduction

The rapid proliferation of the Internet of Things (IoT) has reshaped the digital economy and introduced unprecedented opportunities for data-driven services [1]. Among the numerous applications, healthcare has been one of the most profoundly transformed by IoT technology. Wearable devices such as smartwatches, fitness trackers, and remote health monitoring systems enable real-time collection and transmission of physiological metrics, including heart rate, blood pressure, and blood oxygen levels [2, 3]. These advancements provide unprecedented opportunities for personalized healthcare and proactive medical management [4, 5]. However, the explosive growth of IoT healthcare devices inevitably leads to an exponential increase in data volume, and thus storing large-scale healthcare data on-device not only imposes significant resource burdens but also obstructs timely data sharing and collaborative analytics.

Distributed edge storage emerges as a promising paradigm by alleviating local storage burdens while offering flexible data accessibility [6–8]. Nevertheless, healthcare data is highly sensitive and requires stringent guarantees of confidentiality, integrity, and availability [9]. Healthcare data must be preserved continuously and securely over long time to ensure availability, fault tolerance, and compliance with medical regulations [10]. Existing schemes [11–20] focus on continuous guarantee for outsourced data, but these schemes rely on computationally intensive preprocessing phase, or heavy computation tools such as zero-knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) to ensure continuous storage. Specifically, these schemes predefine the entire storage time and must precompute all challenge–proof pairs before data outsourcing. This preprocessing requires repeatedly invoking the Verifiable Delay Function (VDF) with the trapdoor to generate unpredictable randomness of each audit round for generating audit proof. Because the number of precomputed challenge–proof pairs grows linearly with both the storage time and the audit frequency, the preprocessing overhead becomes substantial for long-term or high-frequency auditing. Thus, this raises the first issue - *how to guarantee continuous storage with lightweight preprocessing that is independent of the storage time for resource-constrained clients*. Once healthcare data is outsourced to distributed edge servers, clients inevitably lose direct physical control over their data. The data must remain available for secure on-demand access, update, and revocation throughout its storage period. Therefore, healthcare data requires dynamic updates for data control during continuous storage. The existing schemes [11–20] follow the verify-after-sequentially-prove mechanism to check data integrity, where clients first perform local checks using a trapdoor function to obtain local results. At the end of the storage period, the verification results are compared with local computations to check data integrity. Once the proofs are precomputed, these schemes are inherently static, and any data modification or deletion invalidates the precomputed proofs and forces the entire preprocessing phase to be repeated. Such inefficiency not only compromises the real-time accessibility requirements of healthcare data but also imposes high bandwidth and processing costs on resource-constrained clients. Thus, this raises the second issue - *how to enable efficient and real-time updates for the continuous storage of healthcare data*.

To summarize, the existing provable continuous storage schemes rely on heavy trapdoor-based preprocessing which linearly increases with the storage time and audit frequency, and inherently static verification chains. Thus, the outsourced healthcare data must be proven to be continuously stored with lightweight preprocessing and verification, real-time updates, and forward security. To tackle this challenge, this paper proposes a Distributedly and Provably Continuous Storage Scheme (DPCSS) for securing healthcare data. By the VDF and Homomorphic Verifiable Tags (HVT), DPCSS performs data integrity checks periodically without any assistance. DPCSS also implements a cross-validation mechanism among edge servers, allowing sequentially-prove-and-verify audit proofs without relying on post-proving verification, thereby eliminating the need for locally verifying computation at the client side. By removing the barriers of the static verification chain, our DPCSS facilitates dynamic updates. To enable cross-validation, all servers need to interact to exchange audit proofs, which incurs quadratic communication overhead. To reduce the communication overhead, DPCSS designs a structural interaction method that achieves $O(n^{\frac{1}{2}})$ communication overhead, where n is the number of servers. The main contributions are summarized as follows.

- We propose a secure continuous storage scheme for healthcare data that supports distributed edge storage and cross-validation. The cross-validation method enables sequentially-prove-and-verify audit proofs, eliminating local computation for pre-verification. Moreover, in the presence of corruptions, DPCSS is also capable of promptly detecting malicious cheaters.
- We design a structural interaction method to reduce communication overhead, making cross-validation more efficient. Based on the Puncturable Key Wrapping (PKW), DPCSS achieves efficient and real-time updates, and provides forward security.
- We formally prove our DPCSS scheme is confidentiality, forward secure, and secure against forgery attacks. The experiments demonstrate that our DPCSS scheme is verification efficient, and has constant preprocessing time that is independent of storage time, check frequencies.

The rest of this paper is organized as follows. Section II presents the related work. Section III reviews the preliminaries involved in our schemes. Section IV gives the specific system model and threat model, scheme definition, and design goals of the work. Section V describes the DPCSS scheme in detail. Sections VI and VII analyze the security and performance of our proposed scheme, respectively. Section VIII concludes the whole paper.

II. Related Work

Proof of storage has gained much more attention, enabling clients to outsource their storage and periodically check the integrity of storage.

Classic proof of storage. The seminal work called Provable Data Possession (PDP) initiates the study of data integrity check of outsourced storage [21]. The PDP allows clients to outsource data associated with proofs to a server. The verifier samples a few challenges and requires the server to respond with data and corresponding proofs according to these challenges. The integrity of these random challenged proofs shows that the data has been properly stored at the remote server. The proofs of the retrievability scheme have extended PDP to support data retrievability [22]. Along the way, PDP-style schemes have evolved significantly, incorporating various functionalities such as public auditability and data dynamics [23–28]. Moreover, PDP-style schemes are not limited to single cloud storage but have also been applied to emerging paradigms such as multi-cloud storage [29, 30] and edge computing, offering more robust and convenient storage services [31].

Proof of storage time. The PDP-style schemes rely on interactive challenge–response protocols between the verifier and servers, which incur frequent communication and require both parties to remain online. To address these issues, Filecoin [11] introduced the notion of proof of storage time. In this scheme, servers sequentially compute audit challenges derived deterministically from the proof of the previous round, and then use zk-SNARKs [32] to demonstrate the correctness of this sequential computation. Although this enables offline verification, zk-SNARKs impose significant computational overhead on servers, creating economic disincentives for storage providers. Ateniese *et al.* [12] proposed an efficient and compact provable continuous storage scheme that ensures data availability over time. In their scheme, clients use the trapdoor of the VDF to execute a preprocessing phase to compute local verification results. During the storage period, the server produces a chain of challenge-proof pairs, where each challenge is derived from the VDF output of the preceding proof, and compresses all pairs into a single hash value for verification. However, the restricted interaction model prevents any data updates after outsourcing. To enable secure sharing of healthcare data, Mehla *et al.* [13] designed a blockchain-integrated architecture that encrypts attributes, enforces privacy-preserving access controls, and leverages the homomorphic property of ElGamal encryption to verify authorized users without compromising sensitive information. To address the limitations of restricted audit interactions, PoRt is proposed using the RSA-based encoding technique to ensure continuous storage [14]. By encoding data with random values and generating corresponding tags, PoRt decouples data from tag information, enabling efficient updating of outsourced data after the storage period. Furthermore, Zhang *et al.* [15] introduced a stateless and publicly verifiable provable continuous storage scheme using zk-SNARK and Vector Commitment (VC) to aggregate and prove audit proof, thereby achieving stateless public verifiability. Rabaninejad *et al.* [16] further employed Directed Acyclic Graph (DAG)-based commitment graph and probabilistic sampling to achieve stateless storage with public verifiability. Additionally, Yu *et al.* [17] proposed an efficient method for continuous integrity checking in decentralized storage systems by adopting a data–time sampling strategy that probabilistically verifies multiple files in each time slot.

Blockchain-enabled proof of storage time. The blockchain has been employed to check data integrity, allowing public verification and eliminating the trusted third party [33, 34]. Fang *et al.* [18] proposed a distributed storage architecture for metaverse healthcare systems, mitigating the limitations of centralized decision-making and single-point access while ensuring secure data storage. By designing a VC-based transmission scheme, this protocol offloads critical-path computations to storage nodes, enhancing inter-node verification efficiency. Erukala *et al.* [35] proposed a secure solution for IoT–cloud collaborative data storage that supports smart sensor authentication, key management, data confidentiality and integrity, secure storage, and fine-grained access control. Wang *et al.* [19] proposed an improved proof-of-storage time algorithm to protect data privacy on the blockchain, and used a general state channel to scale the chain to reduce the token and resource consumption. Huang *et al.* [20] propose a blockchain-based protocol for continuous data integrity verification with zero-knowledge privacy protection. This scheme leverages a high-efficiency VDF combined with proof of retrievability.

In Table III, we outline the detailed comparison between the previous and our schemes. In summary, existing secure storage schemes offer continuous storage guarantees along with additional features such as stateless and public verifiability, access control, and distributed storage. Nevertheless, these schemes incur substantial computational overhead during the preprocessing phase, rendering them unsuitable for resource-constrained settings. Furthermore, most schemes rely on a verify-after-sequentially-prove paradigm or a trusted third party, such as blockchain, to ensure data integrity, and thus failing to support dynamic updates or forward security without depending on such trusted entities.

III. Preliminaries

In this section, we introduce the building blocks involved in our DPCSS scheme.

A. Authenticated Encryption with Associated Data

The Authenticated Encryption with Associated Data (AEAD) provides both encryption and authentication, and also associates additional data with the encrypted message

TABLE I: Comparison of provable continuous storage schemes

Scheme	Model	Main idea	real-time updates	forward security
[12]	single-cloud	TDF	✗	✗
[15]	single-cloud	VC+zk-SNARK	✗	✗
[16]	single-cloud	DAG based commitment graph	✓	✗
[20]	single-cloud	blockchain-assisted	✓	✗
[14]	multi-cloud	VDF+RSA	✗	✗
Ours	multi-cloud	HVT+cross-validation	✓	✓

[36]. The AEAD guarantees confidentiality and integrity for ciphertexts. Here we recall the syntax and security of a nonce-based AEAD scheme below.

Definition 1. *The nonce-based AEAD scheme is a tuple consisting of the following algorithms:*

- $\mathsf{Enc}(sk, nonce, ad, M) \rightarrow C$: it computes ciphertext C for a given message M using the secret key sk, a nonce $nonce$ and associated data ad.
- $\mathsf{Dec}(sk, nonce, ad, C) \rightarrow M$: given the secret key sk, a nonce $nonce$ and associated data ad, the Dec outputs a message M.

The correctness for the nonce-based AEAD scheme stipulates that $\mathsf{Dec}(sk, nonce, ad, \mathsf{Enc}(sk, nonce, ad, M)) = M$. For security, a nonce-based AEAD scheme satisfies confidentiality (Indistinguishability under Chosen-Plaintext Attack (IND-CPA) secure) and ciphertext integrity [37].

B. Puncturable Key Wrapping

PKW enables parties to wrap and unwrap data encryption keys using a master secret key. Additionally, it allows for updating the master secret key such that specific wrapped data encryption keys are rendered irrecoverable [37].

Definition 2. *A puncturable key wrapping scheme* PKW=(KeyGen, Wrap, Unwarp, Punc) *defines as follows:*

- $\mathsf{KeyGen}(1^\kappa, n) \rightarrow sk$: it takes security parameter 1^κ as input, and outputs the secret key sk.
- $\mathsf{Warp}(sk, T, H, K) \rightarrow C$: given the secret key sk, tag T, header H and encryption key K, it outputs ciphertext C.
- $\mathsf{Unwarp}(sk, T, H, C) \rightarrow K/\perp$: it computes the plaintext K for a given ciphertext C using the secret key sk, tag T and header H.
- $\mathsf{Punc}(sk, T) \rightarrow sk'$: given the secret key sk and a tag T, it produces an updated secret key sk'.

The correctness of the PKW scheme requires that a wrapped key can be recovered from its wrapping ciphertext unless the secret key has been punctured on the tag used for the wrapping step. For security, a PKW scheme must satisfy confidentiality and integrity as defined in [37].

C. Verifiable Delay Function

A VDF is a function whose evaluation involves lengthy sequential operations and takes a prescribed time to compute, even on a parallel computer [38]. The computing time Δ is measured as an amount of sequential work. Here we recall the definition of VDF.

Definition 3. *A VDF $F : \mathcal{X} \rightarrow \mathcal{Y}$ is a scheme consisting of the following three algorithms:*

- $\mathsf{KeyGen}(1^\kappa, s) \rightarrow (pk, sk)$: it takes security parameter κ and a delay parameter s as input, and outputs the public key pk and secret key sk.
- $\mathsf{Eval}(pk, x, \Delta) \rightarrow (y, \pi)$: on input public key pk, $x \in \mathcal{X}$ and time parameter Δ, it outputs $y \in \mathcal{Y}$ and a proof π.
- $\mathsf{Verify}(pk, x, y, \pi) \rightarrow \{0, 1\}$: given public key pk and proof π, it checks that y is correctly computed from x at least Δ time.

The correctness of VDF requires that Eval must produce the same result on the same input, and satisfy δ-evaluation time and sequentiality. The soundness requires that if y is not computed from x, the Verify algorithm outputs 0. Then, without secret key sk, it is computationally infeasible to generate the correct output in less than Δ time.

D. Symmetric balanced incomplete block design

The Symmetric Balanced Incomplete Block Design (SBIBD) [6, 39] is a structure $\mathcal{D} = \{[0, n-1], \mathcal{BD}\}$, where $\mathcal{BD} = \{\mathcal{BD}_0, \cdots, \mathcal{BD}_{v-1}\}$ is a collection of subsets of V, each of size v. A SBIBD is called a (n, v, λ)-design ($\mathcal{DS}_{(n,v,\lambda)}$), indicating that every element of V appears in exactly v blocks of B, and every pair of elements appears together in exactly λ blocks. In this work, we employ a $\mathcal{DS}_{(n,v+1,1)}$ where $n = v^2 + v + 1$. For illustration, we present the construction of a $\mathcal{DS}_{(7,3,1)}$: $\{\mathcal{BD}_0 = \{0, 1, 2\}, \mathcal{BD}_1 = \{1, 3, 5\}, \mathcal{BD}_2 = \{2, 3, 6\}, \mathcal{BD}_3 = \{0, 3, 4\}, \mathcal{BD}_4 = \{1, 4, 6\}, \mathcal{BD}_5 = \{2, 4, 5\}, \mathcal{BD}_6 = \{0, 5, 6\}\}$.

E. Secure problem

Definition 4 (Computational Diffie–Hellman (CDH) problem)**.** *Let $(\mathbb{G}, g, p)$ be a cyclic group of prime order p, where g is a generator of $\mathbb{G}$. Given elements $g^x, g^y \in \mathbb{G}$ for unknown exponents $x, y \in \mathbb{Z}_p$, the CDH problem is hard if, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the probability of computing g^{xy} is negligible. Formally,*

$$\Pr\left[\mathcal{A}(g^x, g^y) = g^{xy} \;\middle|\; \begin{array}{l} (\mathbb{G}, g, p) \leftarrow \mathit{GroupGen}(1^\kappa), \\ x, y \leftarrow \mathbb{Z}_p \end{array}\right] \le \epsilon(\kappa),$$

where $\epsilon(\kappa)$ is a negligible function.

Definition 5 (t-Rivest, Shamir and Wagner (RSW) problem)**.** *In the RSA group (N, p, q), the t-RSW problem is $(t_p.t_o, \epsilon)$-hard if for any PPT adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ where $\mathcal{A}_1$ runs in time t_p and $\mathcal{A}_2$ runs in time t_o such that $t_o < t$, it holds that*

$$\Pr\left[y = x^{2^t} \;\middle|\; \begin{array}{l} (N, p, q) \leftarrow \mathit{GroupGen}(1^\kappa), \\ \mathit{state} \leftarrow \mathcal{A}_1(1^\kappa, N), \\ x \leftarrow \{2, \cdots, N-1\}, \\ y \leftarrow \mathcal{A}_2(1^\kappa, x, \mathit{state}) \end{array}\right] \le \epsilon(\kappa).$$

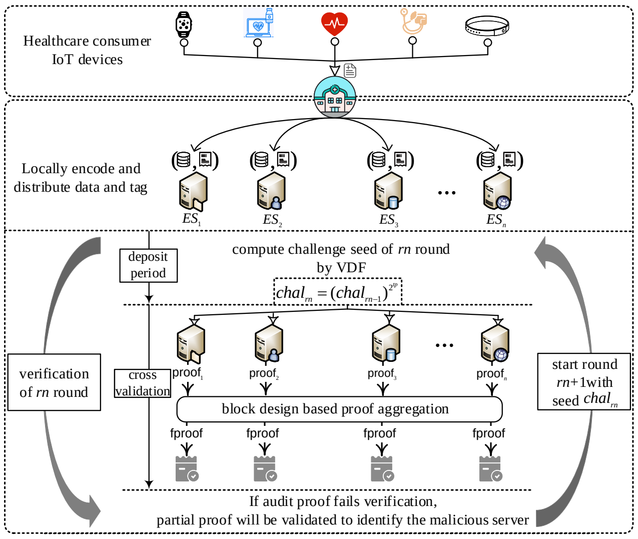
Healthcare consumer IoT devices
Locally encode and distribute data and tag
ES_1
ES_2
ES_3
ES_n
deposit period
compute challenge seed of rn round by VDF
$chal_{rn} = (chal_{rn-1})^{2^{tp}}$
verification of rn round
cross validation
proof$_1$
proof$_2$
proof$_3$
proof$_n$
block design based proof aggregation
fproof
start round rn+1with seed $chal_{rn}$
If audit proof fails verification, partial proof will be validated to identify the malicious server

Fig. 1: System model for our distributedly and provably continuous storage scheme.

IV. Problem Formulation

In this section, we will give the system model, scheme definition, and design goals of DPCSS. Some notations are shown in Table II.

TABLE II: Notations

Notation	Description
pp, sp	public and secret parameter
n	number of edge servers
$[N]$	$\{1, \cdots, N\}$
$F = \{f_i\}_{i\in[N]}$	data file
id_F	identifier of file F
nc	file nonce
$chal^{(rn)}$	challenge seed for round rn
$\mathbf{d}^i_{jk}$	k-th data block of data d^i_j
$\mathbf{d}^i_{jk,l}$	l-th entry of block $\mathbf{d}^i_{jk}$
K_i	encrypted key for data c_i
wk_i	wrapped key of K_i
h_i	file header
σ^i_{jk}	tag for data block $\mathbf{d}^i_{jk}$
π	proof of sequential squarings
$\mathsf{data}^*_j, \mathsf{tg}_j$	encoded data and tag for ES_j
fproof	final audit proof
$\mathsf{aux\_info}$	auxiliary information

A. System Model

As shown in Figure 1, the underlying architecture of our DPCSS scheme for e-health involves two types of entities.

- The healthcare clients will receive data from consumer devices. Clients encounter substantial storage challenges, particularly in the context of hospitals, where healthcare records must be stored for a long time while maintaining confidentiality and ensuring timely access.

- Edge servers (ESs) are responsible for providing computation and storage services, providing essential resources to support a wide range of applications, such as provable and continuous storage and data access services.

In our proposed DPCSS scheme, clients generate authenticated tags for stored healthcare data and upload them to the ESs. Each ES periodically generates integrity proofs with the challenge seeds computed by VDF, and their validity is cross-checked through a structural interaction method designed to minimize overhead.

B. Scheme Definition

Let $F = \{f_1, \cdots, f_N\}$ be the healthcare data outsourced by the healthcare client. To achieve data integrity check, the client computes tags for data F, allowing the ESs to generate audit proofs according to the challenge request initialized by the client, who then checks the correctness of audit proofs. To enable continuous storage for healthcare data and reduce the computational burden on the client, we propose a distributedly and provably continuous storage scheme, called DPCSS. The DPCSS scheme consists of a tuple of five algorithms:

- **Setup**$(1^\kappa, T, n, w, N) \rightarrow (pp, sp)$: on input security and other necessary parameters $1^\kappa, T, n, w, N$, this algorithm outputs public and secret parameters (pp, sp).
- **Store**$(pp, sp, F) \rightarrow (\mathsf{data}_j^*, \mathsf{tg}_j)_{j\in[n]}$: on input public and secret parameters pp and sp and data F, this algorithm generates encoded data $\{\mathsf{data}_j^*\}_{j\in[n]}$ and tag $\{\mathsf{tg}_j\}_{j\in[n]}$.
- **Prove**$(pp, tp, chal^{(rn-1)}, (\mathsf{data}_j^*, \mathsf{tg}_j)_{j\in[n]}) \rightarrow (\mathsf{fproof}, \mathsf{aux\_info})$: inputs public parameter pp, squarings parameter tp, and challenge $chal^{(rn-1)}$, data and tags $(\mathsf{data}_j^*, \mathsf{tg}_j)_{j\in[n]}$, outputs final audit proof fproof and auxiliary information $\mathsf{aux\_info}$.
- **Verify**$(pp, \mathsf{fproof}, \mathsf{aux\_info}) \rightarrow \{0, 1\}$: inputs final audit proof fproof and auxiliary information $\mathsf{aux\_info}$, this algorithm verifies its validation and outputs 1 if verification succeeds, otherwise it returns 0.
- **Update**$(pp, sp, \mathsf{state}) \rightarrow (sk'_{punc}, (h'_1, \cdots, h'_n))$: inputs public parameter pp, secret parameter sp, update state state, this algorithm performs update operation depending on the state, and outputs the updated key sk'_{punc} and file headers $(h'_1, \cdots, h'_n)$.

Security. Informally, similar to the existing provable continuous storage schemes [11–20], the DPCSS assumes that the client is fully trusted, while the ESs are semi-honest. That is each ES follows the prescribed protocol but may attempt to infer private information or forge audit proofs to deceive the client. The proposed DPCSS scheme must satisfy the following completeness and soundness properties.

Completeness. For a client and a set of batch servers that follow the protocol honestly, the proof generated by the honest servers using the Prove algorithm will be accepted by the Verify algorithm. More specifically, for parameters (pp, sp) outputted by Setup$(1^\kappa, T, n)$, all files $F \in \{0, 1\}^*$, and $(\mathsf{data}_j^*, \mathsf{tg}_j)_{j\in[n]}$ computed by Store(pp, sp, F), the proofs $(\mathsf{fproof}, \mathsf{aux\_info})$ generated by servers using Prove$(pp, tp, chal^{(rn-1)}, (\mathsf{data}_j^*, \mathsf{tg}_j)_{j\in[n]})$ will be always accepted by Verify$(pp, \mathsf{fproof}, \mathsf{aux_info})$.

Soundness. Informally, the soundness property ensures that an adversary cannot successfully construct a valid proof without possessing all the data corresponding to a given challenge, except by guessing all the challenged blocks. The soundness game is defined by the following game between a PPT adversary $\mathcal{A}$ who controls a set of provers and a challenger $\mathcal{C}$.

- *Setup*: the challenger generates (pp, sp) by calling Setup$(1^\kappa, T, n, w, N)$, and provides pp to $\mathcal{A}$.
- *Store query*: the adversary $\mathcal{A}$ can make queries of some files F to a *store* oracle. The challenger $\mathcal{C}$ invokes Store(pp, sp, F) computes $(\mathsf{data}_j^*, \mathsf{tg}_j)_{j\in[n]}$, and sends both $(\mathsf{data}_j^*, \mathsf{tg}_j)_{j\in[n]}$ to $\mathcal{A}$ for further verification.
- *Audit*: for any file F on which $\mathcal{A}$ has made a *Store* query, $\mathcal{A}$ starts an audit procedure with $\mathcal{C}$ with a tag tg. Upon completing the audit procedure, the adversary $\mathcal{A}$ receives the output from $\mathcal{C}$, which is a binary indicator of whether the verification succeeds or not. Even if $\mathcal{A}$ is involved in an ongoing audit procedure, it can initiate other instances of the audit with respect to a tg.
- *Output*: $\mathcal{A}$ outputs a challenged tag tg^* returned from some *Store* query. If the *Audit* procedure between $\mathcal{C}$ and $\mathcal{A}$ with input challenged tag tg^* succeeds with probability at least ϵ, then $\mathcal{A}$ is ϵ-admissible.

Definition 6 (Soundness)**.** *A distributed and continuous provable storage scheme is ϵ-sound if there exists an extractor* Extr *such that, for every adversary $\mathcal{A}$ playing the above game, succeeds with probability at least ϵ,* Extr *can successfully extract the corresponding file f with overwhelming probability.*

C. Design Goals

To achieve efficient and continuous storage for healthcare data, our scheme needs to realize the following requirements:

- Continuous storage guarantee: the proposed scheme must ensure that healthcare data is continuously stored on edge servers, and allow periodic checks of the data integrity without a trusted third party. Additionally, each server can efficiently localize and identify the misbehaving server in the event of data corruption.
- Real-time update: during continuous storage, the proposed scheme must provide data update and key update for remote healthcare data, ensuring forward security. The proposed scheme is designed to facilitate the dynamic management of remote healthcare data, including provisions for both data and key updates. Moreover, our scheme provides forward secrecy, ensuring that the confidentiality of historical data is preserved regardless of any future key compromise.
- Efficient preprocessing: the proposed scheme is designed for efficient preprocessing, aiming for constant overhead that is independent of deposit time and audit frequency.

V. Continuous Provable Storage Scheme with Optimized Communication

This section presents the overview of our DPCSS based on the cross-validation with cheater identification method, followed by detailed construction.

A. Overview

The proposed DPCSS consists of five algorithms to support secure continuous storage, verifiable auditing, and flexible key update. The *Setup* algorithm initializes all cryptographic primitives and structural parameters required by the system. The *Store* algorithm prepares the data and corresponding tags for remote storage. The *Prove* and *Verify* algorithms jointly enable continuous integrity checking of the outsourced data. Finally, the *Update* algorithm handles data shredding and key updating, ensuring forward security.

More concretely, the *Setup* phase establishes RSA group parameters for the time-lock puzzle, generates public and private keys for tag generation and key wrapping, and defines the hash and pseudo-random functions used in the scheme. It also constructs a $\mathcal{DS}_{(n,v+1,1)}$ structure to partition the ESs into groups, enabling efficient cross-validation during the proof exchange phase. In the *Store* algorithm, the client encrypts the data to ensure confidentiality and wraps the encryption key using the puncturable secret key sk_{punc}. The encrypted data and wrapped keys are then encoded for fault tolerance and distributed to the ESs together with auxiliary metadata such as identifiers, headers, and nonces.

The *Prove* algorithm is periodically executed to verify data integrity. The client determines the audit interval, which is enforced through the time-lock puzzle (sequential squarings). Each ES derives a fresh challenge seed from the puzzle's output and generates an audit proof. A corresponding challenge proof allows other ESs to verify the correctness of the seed. The $\mathcal{DS}_{(n,v+1,1)}$ structure is then used to exchange and aggregate all per-server proofs into a single final audit proof fproof, which is validated in the *Verify* algorithm. When verification fails, each ES can locally check the received proofs to identify the misbehaving server. Finally, the *Update* algorithm allows client to shred single data or whole data, and supports key updates to ensure data forward security, which is essential for compliance with regulatory requirements.

B. Concrete construction

Setup$(1^\kappa, n, w, N)$: This algorithm is run by the client and takes as input security parameter κ, the total storage time T, and the number of servers and blocks w, N. $N_{pub} = p' \cdot q'$ are computed where p' and q' are two large primes. Two multiplicative cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with order p are selected to build a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, and generator $g \in \mathbb{G}_1$ is sampled. $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$, $H_2 : \{0,1\}^* \to \mathbb{G}_1$ and $H_{\mathsf{prime}} : \{0,1\}^* \to \mathbb{Z}_p^*$ are defined to generate associate data, along with two pseudo-random functions $\phi_1 : \{1,\cdots,wN\} \times \mathbb{Z}_{N_{pub}} \to \{1,\cdots,wN\}$ and $\phi_2 : \{1,\cdots,wN\} \times \mathbb{Z}_{N_{pub}} \to \mathbb{Z}_p^*$. A pair of public/secret keys (pk, sk) is computed for generating authenticated tags where $pk = g^\alpha$ and $sk = \alpha$ is uniformly sampled from $\mathbb{Z}_p$. Besides, $\mathsf{PKW.KeyGen}(1^\kappa)$ is run to produce a puncturable key sk_{punc} for key wrapping and updating. A SBIBD $\mathcal{DS}_{(n,v+1,1)}$ is initialized for data exchange. The public parameter is set as $pp = \{N_{pub}, p, e, \mathbb{G}_1, \mathbb{G}_2, H_1, H_2, H_{\mathsf{prime}}, g, pk, \mathcal{DS}_{(n,v+1,1)}\}$ and the secret parameter is $sp = \{sk, sk_{punc}\}$.

Algorithm 1 Store algorithm ▷ code for client

```
Input: public parameter $pp$, secret parameter $sp$, file $F = \{f_1, f_2, \cdots, f_N\}$
Output: encoded data $\{\mathsf{data}_j^*\}_{j\in[n]}$ and tag $\{\mathsf{tg}_j\}_{j\in[n]}$
1: compute file identifier $id_F = H_1(F)$
2: sample file nonce $nc \in \mathbb{Z}_p^*$
3: for $i = 1$ to $N$
4:     sample encryption key $K_i \in \mathbb{Z}_p^*$
5:     compute wrapped key $wk_i \leftarrow \mathsf{PKW.Warp}(sk_{punc}, H_1(i||id_F), H_1(id_F), K_i)$
6:     $c_i = \mathsf{AEAD.Enc}(K_i, nc, H_1(i||id_F), f_i)$
7:     encode $c_i$ as $(d_1^i, \cdots, d_n^i)$ using $(v+1, n)$ erasure code
8:     for $j = 1$ to $n$
9:         $d_j^i$ is split to $w$ blocks $(\mathbf{d}_{j1}^i, \cdots, \mathbf{d}_{jw}^i)$, where $\mathbf{d}_{jk}^i$ consists of $\theta$ sectors
10:        for $k = 1, \cdots, w$, compute tag $\sigma_{jk}^i = (H_2(id_F||nc||i||j||k) \prod_{l=1}^{\theta} \nu_l^{\mathbf{d}_{jk,l}^i})^\alpha$ for $\mathbf{d}_{jk}^i$, where $\nu_l = H_2(id_F||l)$
11: encode wrapped key $wk_1||wk_2||\cdots||wk_N$ as $(h_1, \cdots, h_n)$ using $(v+1, n)$ erasure code
12: set $\mathsf{data}_j^* = (id_F, nc, h_j, \{(\mathbf{d}_{j1}^i, \cdots, \mathbf{d}_{jw}^i)\}_{i\in[N]})$ and tag $\mathsf{tg} = \{(\sigma_{j1}^i, \cdots, \sigma_{jw}^i)\}_{i\in[N]}$ for each $j \in [n]$
13: return encoded data $\{\mathsf{data}_j^*\}_{j\in[n]}$ and tag $\{\mathsf{tg}_j\}_{j\in[n]}$
```

Store(pp, sp, F): As illustrated by Algorithm 1, the store algorithm taking $F = \{f_1, f_2, \cdots, f_N\}$ proceeds as follows:

1) *Encrypt data with forward security.* The client computes file identifier id_F, and samples a data encryption key K_i and computes wrapped key wk_i for each data f_i, where wrapped key wk_i is a wrapper of encryption key using Puncturable Pseudo-Random Function (PPRF) and AEAD and provides data forward security. By locally updating the file header and replacing the corresponding one at ES, the client can update the encryption key. Moreover, the client can locally update its secret key sk_{punc} or puncture it at specific data to make the remote data irrecoverable.
2) *Achieve fault-tolerant using erasure code.* For each $i \in [N]$, the client encodes encrypted data c_i as $(d_1^i, \cdots, d_n^i)$, where each d_j^i is split into w blocks $(\mathbf{d}_{j1}^i, \cdots, \mathbf{d}_{jw}^i)$ and $\mathbf{d}_{jk}^i$ consists of θ sectors. The client computes tag σ_{jk}^i for each data block $\mathbf{d}_{jk}^i$ consisting of θ sectors. The wrapped key $wk_1||wk_2||\cdots||wk_N$ is also encoded by erasure code to obtain file header $(h_1, \cdots, h_n)$.
3) *Disperse data.* For each ES_j $(j \in [n])$, send $\mathsf{data}_j^* = (id_F, nc, h_j, \{(\mathbf{d}_{j1}^i, \cdots, \mathbf{d}_{jw}^i)\}_{i\in[N]})$ and tag $\mathsf{tg} = \{(\sigma_{j1}^i, \cdots, \sigma_{jw}^i)\}_{i\in[N]}$.

The initial challenge $chal^{(0)}$ and audit frequency $N_{freq} = \lceil \frac{T}{\Delta} \rceil$ are computed by the client and then sent to each ES, where T represents the total storage time and $\Delta = tp \cdot ts$ is the targeted evaluation time with tp as the timing parameter and ts representing the time-cost of computing a squaring in the RSA group. [1]

[1] ts can be determined based on a reasonable estimation of hardware speed of individual servers [40].

Algorithm 2 Prove algorithm ▷ code for ES_j

```
Input: public parameter $pp$, squarings parameter $tp$, challenge $chal^{(rn-1)}$, data $\mathsf{data}_j^*$ and tags $\mathsf{tg}_j$
Output: fproof and aux_info
1: $chal^{(rn)} \leftarrow (chal^{(rn-1)})^{2^{tp}}$
   // compute challenge seed
2: $r_j \leftarrow H_{\mathsf{prime}}(rn||chal^{(rn-1)}||ID_j)$
3: $b_j \leftarrow 2^{tp} \bmod r_j$
4: $\pi_j \leftarrow (chal^{(rn-1)})^{b_j}$
   // compute proof of $tp$ sequential squarings
5: for $k = 1$ to $s$
6:     $i_k \leftarrow \lfloor \frac{\phi_1(k, chal^{(rn)})}{l} \rfloor$, $o_k \leftarrow \phi_1(k, chal^{(rn)}) \mod w$
7:     $\beta_k \leftarrow \phi_2(k, chal^{(rn)})$
   // compute short integrity proof of deposit data
8: $\sigma_j \leftarrow \prod_{k=1}^{s} (\sigma_{j(o_k)}^{i_k})^{\beta_k}$
9: $\mu_{jl} = \sum_{k=1}^{s} \beta_k \mathbf{d}_{j(o_k),l}^{i_k}$ for each $l \in [\theta]$
10: set $\mathsf{proof}_j = (\sigma_j, \mu_j = (\mu_{j1}, \cdots, \mu_{j\theta}), \pi_j)$
   // round 1: audit proof exchange
11: for $i \in [n]$
12:     compute $\alpha_j \leftarrow H_1(chal^{(rn)}||i)$
13:     send $\mathsf{proof}_j$ to $\mathsf{ES}_i$ if $j \in \mathcal{BD}_i$
14: wait for $\mathsf{proof}_i$ from $\mathsf{ES}_i$ $(i \in \mathcal{BD}_j)$
15: for $i \in \mathcal{BD}_j$
16:     $P_i^j \leftarrow \prod_{k \in \mathcal{BD}_j \setminus \{i\}} (\sigma_k)^{\alpha_k}$
17:     $\Lambda_{il}^j = \sum_{k \in \mathcal{BD}_j \setminus \{i\}} \alpha_k \mu_{kl}$ for each $l \in [\theta]$
18: set $\Lambda_i^j = (\Lambda_{i1}^j, \cdots, \Lambda_{i\theta}^j)$
   // round 2: partial audit proof exchange
19: for $i \in [n]$
20:     send $(\{\pi_k\}_{k \in \mathcal{BD}_j \setminus \{i\}}, P_i^j, \Lambda_i^j)$ to $\mathsf{ES}_i$ if $i \in \mathcal{BD}_j$
   // compute final proof
21: $P \leftarrow (\sigma_j)^{\alpha_j} \prod_{j \in \mathcal{BD}_i} P_j^i$
22: $\Lambda = (\alpha_j \mu_{j1} \prod_{j \in \mathcal{BD}_i} \Lambda_{j1}^i, \cdots, \alpha_j \mu_{j\theta} \prod_{j \in \mathcal{BD}_i} \Lambda_{j\theta}^i)$
23: set final proof $\mathsf{fproof} = (\{\pi_i\}_{i \in n}, P, \Lambda)$, $\mathsf{aux\_info} = (rn, id_F, nc, chal^{(rn-1)}, chal^{(rn)}, \{i_k, h_k, \beta_k\}_{k=1}^{s}, \{\alpha_j\}_{j=1}^{n})$
24: return fproof, aux_info
```

Prove$(pp, tp, chal^{(rn-1)})$: during round $0 < rn < N_{freq}$, each ES proceeds on the following three steps to generate integrity proof, shown in Algorithm 2.

1) *Sequentially compute challenge along with verification proof.* Given the challenge $chal^{(rn-1)}$, each ES_j computes the challenge $chal^{(rn)}$ and its proof π_j, where $chal^{(rn)}$ is computed through tp sequential squarings. The challenge proof π_j is used to verify that the challenge $chal^{(rn)}$ is indeed computed by performing tp sequential squarings over challenge $chal^{(rn-1)}$.
2) *Generate short audit proof.* The ES_j uses the challenge $chal^{(rn)}$ as seed to compute the indices of challenged data $\{i_k, o_k\}_{k \in [s]}$, and randomness $\{\beta_k\}_{k \in [s]}$ for aggregating data to keep data confidentiality. The ES_j generates short audit proof (σ_j, μ_j, R_j) through exponentially combining the tag at index (i_k, o_k) with randomness β_k, and linearly combining the data at index (i_k, o_k) and corresponding tag with randomness β_k.

Algorithm 3 Verify algorithm ▷ code for ES_j

```
Input: public parameter $pp$, auxiliary information aux_info and final audit proof fproof
Output: bool or corrupted ES
   // verify challenge proof
1: parse $\mathsf{fproof} = (\{\pi_i\}_{i \in n}, P, \Lambda)$, $\mathsf{aux\_info} = (rn, id_F, nc, chal^{(rn-1)}, chal^{(rn)}, \{i_k, h_k, \beta_k\}_{k=1}^{s}, \{\alpha_j\}_{j=1}^{n})$
2: for $i \in [n] \setminus \{j\}$
3:     $r_i \leftarrow H_{\mathsf{prime}}(rn||chal^{(rn-1)}||ID_i)$
4:     $b_i \leftarrow 2^{tp} \bmod r_i$
5:     if not $\pi_i^{r_i} \cdot (chal^{(rn-1)})^{b_i} = chal^{rn}$
6:         return false
   // validate integrity proof
7: if not $e(P, g) = e(\prod_{j=1}^{n} (\prod_{k=1}^{s} (H_2(id_F||nc||i_k||o_k))^{\beta_k})^{\alpha_j} \cdot \prod_{l=1}^{\theta} \nu_l^{\Lambda_l}, pk)$
8:     activate Identify corrupts
9: return true
```

3) *Two round proof exchange to obtain final audit proof.* A naive approach is for each ES to send its audit proof to every other ES and verify all received proofs locally, which incurs $O(n^2)$ communication. To reduce this cost, we partition the ESs into groups according to $\mathcal{DS}_{(n,v+1,1)}$ and restrict communication to group-based neighborhoods. With this structure, each ES interacts with an overhead of $O(n^{\frac{1}{2}})$ peers while still obtaining the audit proofs of all ESs, as shown in lines 10–16 of Algorithm 2. Specifically, in the first round, each ES_j sends proof_j to every ES_i if $j \in \mathcal{BD}_i$, and then aggregates the received set $\{\mathsf{proof}_i\}_{i \in \mathcal{BD}_j}$ into a partial audit proof. In the second round, each ES_j sends its corresponding proof_j to every ES_i if $i \in \mathcal{BD}_j$. After two-round proof exchange, each server derives the final audit proof fproof together with the auxiliary information aux_info.

Verify$(pp, rn, \mathsf{aux\_info}, \mathsf{proof}_j)$: The proof verification phase of ES is locally validating the final audit proof. The concrete steps are shown in Algorithm 3, including verifying the challenge and integrity proof, and identifying corrupted ES.

1) *Locally validate proof.* Each ES_j leverages Verify algorithm with aux_info and fproof to validate the proof.
2) *Identify cheaters.* If any challenge proof π_i fails verification, broadcast $(\mathsf{Implicate}, \pi_i, i)$ to all ES. Each ES_j also locally verifies the proofs $\{\mathsf{proof}_i\}_{i \in \mathcal{BD}_j}$ received during Round 1. If any verification fails, ES_j broadcasts $(\mathsf{Implicate}, \mathsf{proof}_i, i)$ to all ES. If the majority of the received implications indicate ES_i, then ES_j considers ES_i as a corrupted server.

Update(pp, state): The update algorithm involves three types of data updates, as detailed in Algorithm 4. The core idea is that the client maintains a puncturable key that precisely determines which files remain decryptable. Puncturing the key at the index of a specific file irrevocably removes the ability to derive that file's encryption key, even if an adversary

Algorithm 4 Update algorithm ▷ code for client

```
Input: public parameter $pp$, secret parameter $sp$, update state state
Output: updated key $sk'_{punc}$ and file headers $(h'_1, \cdots, h'_n)$
    // shredding data $f_i$
 1: if state = $(id_F, i, none)$
 2:     $sk'_{punc} \leftarrow$ PKW.Punc$(sk_{punc}, H_1(i||id_F))$
 3:     return $sk'_{punc}$
    // shredding whole data $F$
 4: if state = $(id_F, *, none)$
 5:     $sk'_{punc} \leftarrow$ PKW.KeyGen$(1^\kappa)$
 6:     return $sk'_{punc}$
    // key update
 7: if state = $(id_F, *, (h_1, \cdots, h_n))$
 8:     $sk'_{punc} \leftarrow$ PKW.KeyGen$(1^\kappa)$
 9:     decode $(h_1, \cdots, h_n)$ as wrapped key $wk_1||wk_2||\cdots||wk_m$
10:     for $i \in [w]$
11:         $K_i \leftarrow$ PKW.Unwarp$(sk_{punc}, id_F, H_1(id_F), wk_i)$
12:         $wk'_i \leftarrow$ PKW.Warp$(sk'_{punc}, id_F, H_1(id_F), K_i)$
13: encode $wk'_1||\cdots||wk'_w$ as new wrapped key $(h'_1, \cdots, h'_n)$ as using $(t, n)$ erasure code
14:         return $sk'_{punc}, (h'_1, \cdots, h'_n)$
```

later gains access to all wrapped keys stored on the servers. For single-data shredding, the client punctures the key at the target file index, making the corresponding encryption key permanently unrecoverable. For whole-data shredding, the client replaces the secret key entirely, invalidating all previously derivable encryption keys. For regular updates, the client refreshes the puncturable key and re-wraps the data keys, preventing long-term key exposure and maintaining fine-grained, up-to-date access control.

1) *Shred single data*. Given file identifier id_F and shred index i, the client uses the secret key sk_{punc} to puncture at index $H_1(i||id_F)$ to generate a new key sk^{new}_{punc}. This guarantees that the encryption key for data f_i cannot be extracted from the i-th wrapped key wk_i, since the updated key sk^{new}_{punc} has been explicitly punctured at f_i.
2) *Shred whole data*. The client discards the old secret key and samples the new secret key sk^{new}_{punc}, ensuring that the remote data cannot be decrypted correctly, and rendering it irrecoverable.
3) *Regularly key update*. With the secret key sk_{punc}, the client first extracts the encryption key from the current wrapped key (decoded from the file headers). The new wrapped key is then computed using the newly generated key sk^{new}_{punc}, and the file headers are updated accordingly.

VI. Security Proof

In this section, we analyze the security properties of the construction in relation to the defined security model.

Theorem 1 (Security)**.** *Assuming that the CDH problem is hard in bilinear groups and the RSW problem is computationally hard, the proposed DPCSS scheme is secure in the random oracle model, that is, ensures completeness and soundness.*

Proof. **Completeness**. This is trivial, as integrity proofs can always pass verification if they are correctly computed.

Soundness. We prove the soundness using a sequence of games adapted from [22]. The key distinction is the additional requirement that the challenge of round rn is unpredictable before the provers solve the time lock puzzle, preventing them from precomputing an audit proof for a specified challenge. In the following, we first prove that in each verification round $0 < rn < N_{freq}$, the adversary is unable to forge a valid proof to deceive the honest verifier. Subsequently, we prove that the adversary cannot obtain the challenge seed for round rn until the completion of round $rn - 1$.

G_0. The initial game is the original experiment that defines the soundness of the proposed scheme in Definition 6.

G_1. Same as G_0 except that the challenger keeps a list of all messages within the game. The challenger $\mathcal{C}$ aborts if $\mathcal{A}$ is successful in any of the audit instances, which falls into the following two cases:

1) The aggregated tag σ'_j is not equal to expected tag $\sigma_j = \prod_{k=1}^{s}(\sigma^{i_k}_{j(o_k)})^{\beta_k}$, where the adversary's responses are $(\sigma'_j, \mu'_j = (\mu'_{j1}, \cdots, \mu'_{j\theta}))$ and the expected responses from honest provers are $(\sigma_j, \mu_j = (\mu_{j1}, \cdots, \mu_{j\theta}))$. According to verification equation, for $l \in \{1, \cdots, \theta\}$, we define $\Delta\mu_{jl} = \mu_{jl} - \mu'_{jl}$, where at least one $\Delta\mu_j$ is nonzero; otherwise $\sigma_j = \sigma'_j$. As a result, one can construct a simulator to compute h^a given the tuple $(g, g^a, h) \in \mathbb{G}^3$. More specifically, the public key pk is set to g^a, where a is unknown. The simulator programs the random oracle through the following two steps: (i) for $l \in [\theta]$, set $\nu_k \leftarrow g^{\xi_l}h^{\gamma_l}$; (ii) for $i \in [N], k \in [w]$, sample $\eta^i_{jk} \in \mathbb{Z}^*_p$ and program $(H_2(id_F||nc||i||j||k) = g^{\eta^i_{jk}}/(\prod_{l=1}^{\theta}\nu_k^{\mathbf{d}^i_{jk,l}})$ to derive $\sigma^i_{jk} = (g^a)^{\eta^i_{jk}}$. Then, dividing the verification equation for σ_j by that for the σ'_j, one obtains the relation $e(\sigma'_j/\sigma_j, g) = e(\prod_{l=1}^{\theta}(g^{\xi_l}h^{\gamma_l})^{\Delta\mu_{jl}}, pk)$, and rearranges terms yield $e(\sigma'_j \cdot \sigma_j^{-1} \cdot pk^{\sum_{l=1}^{\theta}\xi_l\Delta\mu_{jl}}, g) = e(h, pk)^{\sum_{l=1}^{\theta}\gamma_l\Delta\mu_{jl}}$. Recall that $pk = g^a$, $h^a = (\sigma'_j \cdot \sigma_j^{-1} \cdot pk^{\sum_{l=1}^{\theta}\xi_l\Delta\mu_{jl}})^{\frac{1}{\sum_{l=1}^{\theta}\gamma_l\Delta\mu_{jl}}}$. Given that the CDH problem is intractable in group $\mathbb{G}$, this case occurs with negligible probability.
2) There are at least one of the aggregated messages $\{\mu_{jl}\}_{l=1}^{\theta}$ is not equal to expected $\mu_{jl} = \sum_{k=1}^{s}\beta_k\mathbf{d}^{i_k}_{j(o_k),l}$, where aggregated tag is same ($\sigma_j = \sigma'_j$). Thus, we can define $\Delta\mu_{jl} = \mu_{jl} - \mu'_{jl}$ where at least one $\Delta\mu_{jl}$ is nonzero; otherwise $\sigma_j = \sigma'_j$. Given a tuple $(g, h) \in \mathbb{G}^2$, one can construct a simulator that programs the random oracle for the adversary to compute $h = g^a$. More precisely, the simulator proceeds the same steps as in case 1), without programming the tags (step (ii)). According to verification equation, one can obtain $e(\prod_{k=1}^{s}(H_2(id_F||nc||i_k||j||o_k)) \cdot \prod_{l=1}^{\theta}\nu_j^{\mu_{jl}}, pk) = e(\prod_{k=1}^{s}(H_2(id_F||nc||i_k||j||o_k)) \cdot \prod_{l=1}^{\theta}\nu_j^{\mu'_{jl}}, pk)$, further concludes that $\prod_{l=1}^{\theta}\nu_j^{\mu_{jl}} = \prod_{l=1}^{\theta}\nu_j^{\mu'_{jl}}$. Consequently, $\prod_{l=1}^{\theta}(g^{\xi_l}h^{\gamma_l})^{\Delta\mu_{jl}} = 1$. The solution to the discrete

logarithm problem then yields $h = g^{-\frac{\sum_{l=1}^{\theta} \xi_l \Delta\mu_{jl}}{\sum_{l=1}^{\theta} \xi_l \Delta\mu_{jl}}}$. Thus, this case happens with negligible probability assuming that the discrete logarithm problem is hard in $\mathbb{G}$.

Therefore, the difference between G_1 and G_0 is negligible. Otherwise, we can construct a simulator capable of solving the CDH and discrete logarithm problem. Moreover, if the adversary $\mathcal{A}$ can solve at least one instance of the time lock puzzle using significantly fewer operations than the required number, it would be able to generate the proofs for the specified challenge in advance. In such a case, we can construct a simulator using $\mathcal{A}$ as a subroutine to solve the RSW problem with a non-negligible probability. Thus, the adversary cannot obtain the challenge seed for round rn until the completion of round $rn-1$. □

Theorem 2 (Confidentiality and forward security)**.** *Given that the underlying PKW scheme is forward indistinguishability and the AEAD scheme is IND-CPA, the proposed DPCSS scheme achieves confidentiality and forward security.*

Proof. The confidentiality and forward security properties of the scheme are formally defined through the following security game, which enables the adversary to distinguish a real ciphertext from a random string. In this game, the adversary is allowed to make encryption and challenge queries both before and after an update query, and key query. The update query models the secure deletion of data and key updates. The key query provides access to the current punctured key if all challenge files associated with the key have been securely shredded, and no further challenge queries are made under that key. Thus, we effectively capture the confidentiality and forward security of our proposed DPCSS scheme.

Setup: the challenger $\mathcal{C}$ generates (pp, sp) by Setup$(1^\kappa, T, n, w, N)$ and initializes an empty list L, and provides pp to $\mathcal{A}$.

Query: $\mathcal{A}$ can make the following queries.

(1) Encryption query (at most q_e times): upon receiving a query (id_F, nc, i, f_i), $\mathcal{C}$ computes $wk_i \leftarrow \mathsf{PKW.Warp}(sk_{punc}, H_1(i||id_F), H_1(id_F), K_i)$ with random $K_i \in \mathbb{Z}_p^*$, $d_i \leftarrow \mathsf{AEAD.Enc}(K_i, nc, H_1(i||id_F), f_i)$. The challenger $\mathcal{C}$ responds with $(\mathrm{id}_i = H_1(i||id_F), wk_i)$ and updates the list $L \leftarrow (\mathrm{id}_i, wk_i, b = 1)$.

(2) Challenge query (at most q_c times): upon receiving a query (id_F, nc, i, f_i), $\mathcal{C}$ flips a coin $b \in \{0, 1\}$. If $b = 1$, proceed as in the encryption query (1). Otherwise, $\mathcal{C}$ responds with randomly sampled id_i, d_i and wk_i, and updates the list $L \leftarrow (\mathrm{id}_i, wk_i, b = 0)$. [2]

(3) Update query: the update query includes three queries of shred single data, shred whole data and key update.

- Shred single data (i-th data) (at most q_s times): upon receiving a query (id_F, nc, i), $\mathcal{C}$ executes $sk_{punc} \leftarrow \mathsf{PKW.Punc}\ (sk_{punc}, \mathrm{id}_i)$, and removes the item $(\mathrm{id}_i, *, *)$ from L.
- Shred whole data: upon receiving a query $(id_F, nc, *)$, $\mathcal{C}$ generates a new punctured key $sk_{punc} \leftarrow \mathsf{PKW.KeyGen}\ (1^\kappa)$, and removes all item related to id_F from L.
- Key update (at most q_k): given a key update query associated with id_F, $\mathcal{C}$ updates all file headers of data with identifier id_F. Specifically, for each i-th item L_i in L, $\mathcal{C}$ randomly samples wk_i' if $L_i[2]{=}0$ (b in L_i). Otherwise, $\mathcal{C}$ computes K_i by $\mathsf{PKW.Unwarp}(sk_{punc}, id_F, \mathrm{id}_i, wk_i)$ and wk_i' through $\mathsf{PKW.Warp}(sk_{punc}', id_F, \mathrm{id}_i, K_i)$. Finally, $\mathcal{C}$ returns $\{\mathrm{id}_i, wk_i'\}_{i=1}^{|L|}$.

(4) Key query: the adversary $\mathcal{A}$ is allowed to request the punctured key sk_{punc} if it has shredded all previously challenged data and does not issue any further challenge queries related to that key. In this case, the challenger $\mathcal{C}$ responds by providing the corresponding sk_{punc}.

Guess: $\mathcal{A}$ outputs a bit b^* of a specified data.

The advantage of $\mathcal{A}$ against the Confidentiality and Forward Security (CFS) of our proposed DPCSS scheme as $Adv^{CFS}(\mathcal{A}) = |Pr(b^* == b) - \frac{1}{2}|$. In the following, we will prove the advantage of $\mathcal{A}$ is negligible via a sequence of games. Let $\{\mathsf{G}_i\}_{i=0}^{6}$ denote game i.

G_0. This game is same as the above game.

G_1. Same as G_0 except that the data identifiers for challenge encryptions are restricted to prevent overlap with any identifiers that have already been shredded. Let E_1 denote the event that during any of the q_c challenge queries, the data identifier in either the ideal or real-world setting is drawn from the set of at most q_s shredded data identifiers under the current punctured key. The two data identifiers are the same in at most q_s queries with probability $\frac{q_s}{2^p}$. Thus, $|Pr[\mathsf{G}_0] - Pr[\mathsf{G}_1]| \leq Pr[\mathsf{E}_1] \leq 2q_c \cdot \frac{q_s}{2^p}$, which implies that $Adv_0(\mathcal{A}) \leq 4q_c \cdot \frac{q_s}{2^p} + Adv_1(\mathcal{A})$.

G_2. Same as G_1 except that the adversary is restricted to making at most one query to the challenge oracle. Using the hybrid argument, we can analyze the adversary's advantage by transitioning from a game where all q_c challenge queries are responded to with random bits to a game where all challenge queries are answered with real ciphertexts. By modifying the response to one challenge query at a time, the difference between any two adjacent hybrid games is negligible. Thus, $Adv_1(\mathcal{A}) {=} q_c Adv_2(\mathcal{A})$ for all adversary $\mathcal{A}$.

G_3. This game is defined similarly to G_2 except that the identifier used for challenged data must not be equal to the encryption query. Let E_2 be the event that the identifier for challenged data is the same as that for the encryption query. We have $|Pr[\mathsf{G}_2] - Pr[\mathsf{G}_3]| \leq Pr[\mathsf{E}_2] \leq \frac{q_e+q_c-1}{2^p}$, and $Adv_2(\mathcal{A}) \leq q_c \cdot \frac{q_e+q_c-1}{2^p} + Adv_3(\mathcal{A})$. The result ensures that the probability of a challenge identifier colliding with an encryption query identifier remains negligible.

G_4. The challenger guesses the specific key (out of at most q_k key queries) during which the adversary will request its challenge query. Let event_3 be the event that the guess is correct, then $Pr[\mathsf{E}_2] = \frac{1}{q_k}$. Thus, we have $Pr[\mathsf{G}_4] = Pr[\mathsf{G}_4|\mathsf{E}_3] \cdot Pr[\mathsf{E}_3] + Pr[\mathsf{G}_4|\neg\mathsf{E}_3] \cdot Pr[\neg\mathsf{E}_3] = \frac{1}{q_k}(Pr[\mathsf{G}_3|\mathsf{E}_3] + \frac{q_k-1}{2})$ where $Pr[\mathsf{G}_4|\mathsf{E}_3] = Pr[\mathsf{G}_3|\mathsf{E}_3]$ when E_3 occurs, and $Pr[\mathsf{G}_4|\neg\mathsf{E}_3] = \frac{1}{2}$. Thus, $Adv_3(\mathcal{A}) = q_k \cdot Adv_4(\mathcal{A})$.

G_5. This game is identical to G_4, except that in the challenge query, the wrapped key is replaced by a random string. As a result, the difference in the probability distributions between G_4 and G_5 is bounded by the adversary's advantage in

[2]The adversary is allowed to query with a new challenge after successful key update.

TABLE III: Theoretical computation cost analysis

Scheme	Store	Prove (single time)	Verify
cPoSt [12]	$(2N_b + 3N_{freq} + 3)T_H + N_{freq}T_{VDF^*} + T_{Enc}$	$3N_bT_H + T_{VDF}$	T_H
CBDIC [17]	$(2N_b + 3sN_{freq} + 4)T_H + sN_{freq}$ $(T_{PRF} + T_{PRP} + T_{Exp} + T_{VDF^*}) + 4T_{Enc}$	$(s+2)T_H + T_{VDF}$ $+s(T_{PRF} + T_{PRP})$	T_H
PoRt [14]	$N_bT_{PRF} + N_{freq}T_{VDF^*}$	$sT_H + T_{Exp} + T_{VDF}$	$nT_{Exp} + T_{Dec}$
DPCSS	$NT_{Enc} + N_bT_{Exp}$	$(s+\theta+3)T_{Exp} + T_{VDF}$	$(s+n+\theta+3)T_{Exp} + 2T_{pair}$

N_b denotes the total number of data blocks, T_{PRF}, T_{PRP}, T_{Exp}, T_H, T_{pair}, T_{Enc} and T_{Dec} denote the computation computation time of PRF, PRP, exponentiation over group, hash function, pairing operation, symmetric encryption and decryption.

breaking the PKW scheme's confidentiality ($Adv(\mathcal{A}_{PKW})$). Therefore, we have $|Pr[\mathsf{G}_4] - Pr[\mathsf{G}_5]| \leq Adv(\mathcal{A}_{PKW})$.

G_6. Same as G_5 except that the ciphertext is replaced by a random string in the challenge query. The difference in probability distributions between G_5 and G_6 is bounded by $\mathcal{A}$'s advantage in breaking the confidentiality of AEAD scheme ($\mathcal{A}_{AEAD}$). Thus, we have $|Pr[\mathsf{G}_5] - Pr[\mathsf{G}_6]| \leq Adv(\mathcal{A}_{AEAD})$.

In summary, the adversary's advantage in breaking the confidentiality and forward security of the scheme is as follows: $Adv^{CFS}(\mathcal{A}) \leq 2q_e(\frac{2q_s+q_e+q_c-1}{2^p} + q_k(Adv(\mathcal{A}_{PKW}) + Adv(\mathcal{A}_{AEAD})))$, which is negligible. □

VII. Performance Evaluation

In this section, we analyze the theoretical performance of DPCSS , cPoSt [12], CBDIC [17] and PoRt [14] schemes. For the experiment test, the preprocessing and online cross-validation of our proposed DPCSS scheme are implemented to show the efficiency. The PoRt leverages the VDF and RSA to achieve continuous storage guarantee, and its preprocessing phase is implemented for comparison.

A. Theoretical Analysis

We analyze the computation cost of the proposed DPCSS scheme and the comparison schemes cPoSt [12], CBDIC [17] and PoRt [14] in Table III. We primarily consider the main operations, specifically, hash, PRF, Pseudo-Random Permutation (PRP) and exponentiation operations, involved in the store, prove and verify phase of each schemes. In the store phase, the cPoSt, CBDIC, and PoRt schemes must predefine the storage duration and consequently precompute all challenge–proof pairs in advance. The final audit proof is then derived through the trapdoor of the VDF. As a result, the overall computation overhead of these three schemes scales linearly with the audit frequency N_{freq}. In contrast, DPCSS only needs to generate a tag for each data block once, and its computation cost remains independent of the audit frequency.

To quantify the computation cost of a single prove round, we analyze each scheme separately. The cPoSt scheme must compute the digest over the entire dataset using an unpredictable randomness derived from the VDF, resulting in a cost of $3N_bT_H + T_{VDF}$. The CBDIC, PoRt, and DPCSS schemes adopt a random-sampling mechanism, where the computation overhead grows with the number of challenged blocks s. Among these three schemes, DPCSS incurs the highest cost due to its use of exponentiation operations for aggregating proofs, leading to a computation overhead of $(s+\theta+3)T_{Exp} + T_{VDF}$. In the verify phase, cPoSt and CBDIC only require a single hash operation to validate the audit result. In contrast, PoRt and DPCSS are designed for multi-cloud storage and therefore must aggregate proofs from all participating servers. In addition, since DPCSS employs homomorphic authenticators, it must further aggregate all metadata associated with the challenged blocks and corresponding sectors, whose computation cost is linear in $(s + \theta)$. Furthermore, the cPoSt, CBDIC, and PoRt schemes exhibit a static design, whereby any data update (e.g., modification or deletion) mandates a full re-execution of the store phase. This leads to significant additional computational overhead and significantly limits their practicality in dynamic storage environments.

B. Experimental Tests

Experimental Setup. The simulation experiments are conducted on a laptop equipped with a 2.90 GHz Intel Core i7-410700M CPU and 4 GB of memory, running the Ubuntu 18.04 LTS operating system. The programming language used is Python, and all evaluated protocols were implemented in Python 3 to ensure consistency in all evaluations. To allow interaction among different edge servers, we run our scheme on a single machine with multi-threaded or multi-processes emulation. The peer-to-peer communication channels were established using unauthenticated TCP sockets. Based on the Rust bilinear pairing library[3], the proposed DPCSS scheme adopts the pairing-friendly elliptic curve BLS12-381, with the RSA modulus set to 1024 bits. In the PoRt scheme, the block size was fixed at 128 bytes as this scheme involves RSA signature to compute the tag, whereas in DPCSS scheme, block sizes of 128 bytes, 512 bytes, and 1 KB are evaluated. The hash functions are instantiated by SHA-256.

While Table III provides a theoretical comparison of cPoSt, CBDIC, PoRt, and the proposed DPCSS, the experimental evaluation focuses solely on comparing with PoRt. The cPoSt and CBDIC schemes are hash-based static constructions designed for single-server environments and require precomputing all challenge–proof pairs for the entire storage time, which makes them incompatible with the dynamic and multi-cloud continuous storage setting considered in our experiments. Therefore, we compare our DPCSS scheme with PoRt, as it supports multi-server continuous auditing and only requires regenerating tags for updated data during the store phase.

[3]https://crates.io/crates/pairing

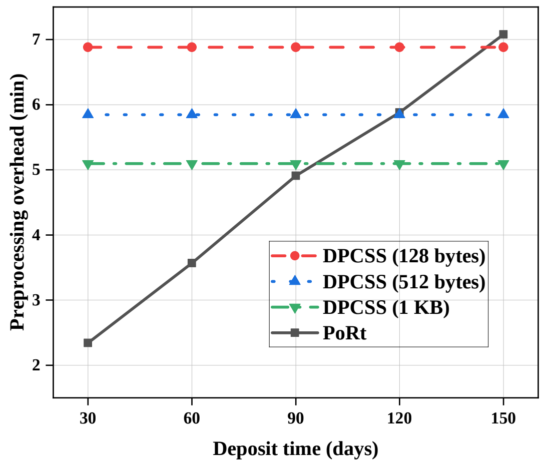

Fig. 2: Comparison of preprocessing overhead.

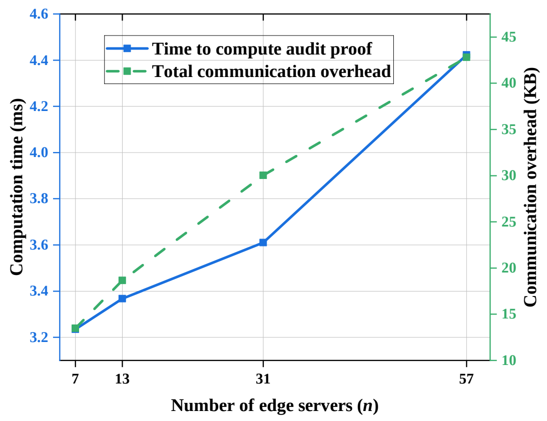

Fig. 3: Computation time and communication overhead of audit proof under different numbers of ES.

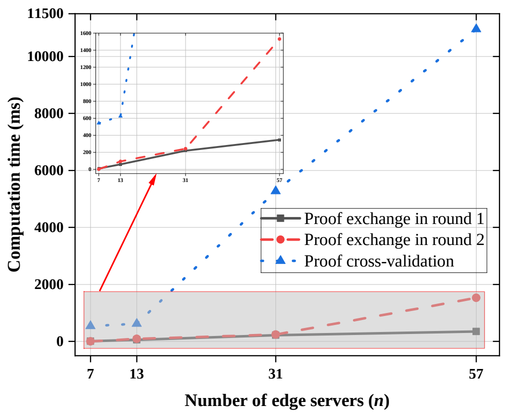

Fig. 4: Computation time for audit proof exchange and verification under different numbers of ES.

We evaluate the computational overhead for the preprocessing phase and verification phase, and the communication overhead. The preprocessing phase focuses on comparing the DPCSS scheme with the PoRt scheme. Since the PoRt scheme does not support data updates during the storage period, we only evaluate the overhead of the proposed DPCSS scheme. The data size is set to 10 MB, and the number of challenged data blocks during the verification phase is set to 460 to ensure a 99% error detection rate.

TABLE IV: Preprocessing time of the PoRt scheme (seconds)

N_{freq}	1440	720	480	360	288
Time	203.05	140.55	113.72	103.83	96.35

Preprocessing overhead. The preprocessing phase primarily processes data and generates tags to guarantee continuous storage. Fig. 2 reports the preprocessing overhead under different deposit times. In the PoRt scheme, the preprocessing process must precompute all challenge–proof pairs, resulting in a linear growth with the deposit time. When the deposit time extends from 30 to 150 days with audit frequency of one time per hour, the preprocessing cost rises from approximately 2.5 minutes to 7 minutes. Consequently, its preprocessing cost is tightly coupled with the audit frequency and becomes increasingly burdensome in long-term storage scenarios. In contrast, the proposed DPCSS remains independent of storage time and audit frequency because it only requires generating a tag for each data block. As shown in Fig. 2, with a block size of 1 KB, the preprocessing time of DPCSS stays constant at approximately 5 minutes. Table IV further shows the preprocessing overhead for 10 MB of data stored for 30 days under different audit frequencies. The PoRt exhibits decreasing preprocessing time at lower audit frequencies. When the audit frequency is 480 (90 min/per), the PoRt scheme's preprocessing time is 113.72 seconds.

Verification overhead. The verification phase of DPCSS leverages the $\mathcal{DS}_{(n,v+1,1)}$ structure to support efficient multi-server communication. Fig. 3 shows that the communication overhead grows moderately with the number of servers, the total communication is only 45 KB with 57 servers. The computation overhead is decomposed into proof generation, two rounds of proof exchange, and final proof verification, as shown in Fig. 4. Proof generation scales slightly with the number of servers, increasing from 3.2 ms ($n = 7$) to 4.2 ms ($n = 57$), demonstrating efficient proof computation. The two rounds of proof exchange incur 346.82 ms and 1530 ms at 57 servers, respectively. This overhead arises from the aggregation structure, where the second round requires processing aggregated proofs from a larger number of nodes, leading to a higher latency. As shown in Fig. 4, each ES requires 10.95 s to complete the verification of audit proof for 57 servers. Therefore, proof verification constitutes the primary computation overhead in the verification phase. This is expected because verification must process all aggregated metadata and perform homomorphic authenticator checks.

Update overhead. The proposed DPCSS supports efficient dynamic operations, including both key updates and secure data shredding. For 10 MB of data with a 1 KB block size, the key update time averages only 1.6 ms. Data shredding via puncturing the corresponding identifier requires 1.5 ms. Compared to static schemes such as PoRt, which must re-execute the entire storage procedure upon updates, DPCSS is much more practical in dynamic multi-cloud storage. By selecting a new puncturable key and discarding the previous one, all remote data can be rendered unrecoverable, with negligible computational overhead.

VIII. Conclusion

In this paper, we proposed DPCSS, a distributed and provably continuous storage scheme for IoT healthcare data. By combining VDF with homomorphic authenticators, DPCSS enables stateless and periodic integrity verification without relying on trusted third parties. To overcome the static nature of existing verification chains, we introduced a cross-validation mechanism among edge servers, which supports a sequential prove-and-verify process and reduces local verification overhead. Furthermore, DPCSS leverages puncturable key techniques to provide efficient and forward-secure dynamic updates. We rigorously analyzed the security of our DPCSS scheme. Experimental results show that DPCSS achieves constant preprocessing time independent of storage time and audit frequency, while supporting millisecond-level key updates and data shredding. In future work, we will focus on combining continuous storage with secure semantic search to enable efficient and privacy-preserving data retrieval over long-term outsourced healthcare data.

References

[1] G. K. Walia, M. Kumar, and S. S. Gill, "Ai-empowered fog/edge resource management for iot applications: A comprehensive review, research challenges, and future perspectives," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 1, pp. 619–669, 2024.

[2] A. A. Khan, R. K. Mahendran, F. Ullah, F. Ali, N. S. Alghamdi, A. A. AlZubi, and D. Kwak, "Fed-IoMT-Block: A privacy-preserving framework for secure federated learning in consumer-centric internet of medical things," *IEEE Transactions on Consumer Electronics*, 2025, early access.

[3] Y. Zuo, L. Xu, J. Li, J. Li, X. Wang, and M. J. Piran, "Secure and efficient blockchain-based access control scheme with attribute update," *IEEE Transactions on Consumer Electronics*, vol. 71, no. 1, pp. 1539–1550, 2025.

[4] Z. Wang, C. Xiang, J. Wei, D. Zhang, S. Su, and L. Liu, "A threshold multi-ca authentication model for privacy-preserving federated semi-supervised learning in healthcare iot," *IEEE Transactions on Consumer Electronics*, 2025, early access.

[5] C. Ding, L. Shen, X. Zhang, Y. Lu, Y. Li, and Q. L. n, "A privacy protection system for consumer electronics data storage devices based on a cloud computing–edge computing collaborative mechanism," *IEEE Transactions on Consumer Electronics*, 2025, early access.

[6] Y. Su, Y. Li, B. Yang, and Y. Ding, "Decentralized self-auditing scheme with errors localization for multi-cloud storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2838–2850, 2022.

[7] W. Tang, J. Li, X. Zhang, Y. Miao, Z. Su, and R. H. Deng, "Efficient mobile-cloud collaborative aggregation for federated learning with latency resilience," *IEEE Transactions on Mobile Computing*, 2025, early access.

[8] F. Yang, Z. Ding, L. Jia, Y. Sun, and Q. Zhu, "Blockchain-based file replication for data availability of ipfs consumers," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 1191–1204, 2024.

[9] Y. Su, Y. Li, K. Zhang, and B. Yang, "A privacy-preserving public integrity check scheme for outsourced ehrs," *Information Sciences*, vol. 542, pp. 112–130, 2021.

[10] Y. Hong, L. Yang, W. Liang, and A. Xie, "Secure access control for electronic health records in blockchain-enabled consumer internet of medical things," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 4574–4584, 2024.

[11] P. Labs, "Filecoin: A decentralized storage network," https://filecoin.io/filecoin.pdf, 2018.

[12] G. Ateniese, L. Chen, M. Etemad, and Q. Tang, "Proof of storage-time: Efficiently checking continuous data availability," in *Proceedings of Network and Distributed System Security Symposium (NDSS '20)*, 2020, pp. 1–15.

[13] R. Mehla, R. Garg, and M. A. Khan, "Privacy-preserving solution for data sharing in iot-based smart consumer electronic devices for healthcare," *IEEE Transactions on Consumer Electronics*, vol. 71, no. 2, pp. 4586–4595, 2025.

[14] R. Rabaninejad, B. Liu, and A. Michalas, "Port: Non-interactive continuous availability proof of replicated storage," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23)*, 2023, pp. 270–279.

[15] C. Zhang, X. Li, and M. H. Au, "epost: Practical and client-friendly proof of storage-time," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1052–1063, 2023.

[16] R. Rabaninejad, B. Abdolmaleki, G. Malavolta, A. Michalas, and A. Nabizadeh, "storna: Stateless transparent proofs of storage-time," in *Proceedings of the 28th European Symposium on Research in Computer Security (ESORICS '23)*, 2023, p. 389–410.

[17] H. Yu, Q. Hu, Z. Yang, and H. Liu, "Efficient continuous big data integrity checking for decentralized storage," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1658–1673, 2021.

[18] G. Fang, Y. Sun, M. Almutiq, W. Zhou, Y. Zhao, and Y. Ren, "Distributed medical data storage mechanism based on proof of retrievability and vector commitment for metaverse services," *IEEE Journal of Biomedical and Health Informatics*, vol. 28, no. 11, pp. 6298–6307, 2024.

[19] K. Wang, Q. Wu, T. Han, Y. Wang, Y. Zhang, and B. Qin, "Dcss: A smart contract-based data continuous storage scheme," in *Proceedings of the 5th ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI '23')*, 2023, p. 53–63.

[20] Y. Huang, Y. Yu, H. Li, Y. Li, and A. Tian, "Blockchain-based continuous data integrity checking protocol with zero-knowledge privacy protection," *Digital Communications and Networks*, vol. 8, no. 5, pp. 604–613, 2022.

[21] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kiss-

ner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, 2007, pp. 598–609.

[22] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '08)*, 2008, pp. 90–107.

[23] J. Li, H. Yan, and Y. Zhang, "Identity-based privacy preserving remote data integrity checking for cloud storage," *IEEE Systems Journal*, vol. 15, no. 1, pp. 577–585, 2021.

[24] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011.

[25] J. Yuan and S. Yu, "Public integrity auditing for dynamic data sharing with multiuser modification," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1717–1726, 2015.

[26] W. Susilo, Y. Li, F. Guo, J. Lai, and G. Wu, "Public cloud data auditing revisited: Removing the tradeoff between proof size and storage cost," in *Proceedings of the 27th European Symposium on Research in Computer Security (ESORICS '22)*, 2022, pp. 65–85.

[27] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious rams," *Journal of Cryptology*, vol. 30, pp. 22–57, 2017.

[28] J. Li, H. Yan, and Y. Zhang, "Certificateless public integrity checking of group shared data on cloud storage," *IEEE Transactions on Services Computing*, vol. 14, no. 1, pp. 71–81, 2021.

[29] H. Yan, J. Li, J. Han, and Y. Zhang, "A novel efficient remote data possession checking protocol in cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 78–88, 2017.

[30] J. Li, H. Yan, and Y. Zhang, "Efficient identity-based provable multi-copy data possession in multi-cloud storage," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 356–365, 2022.

[31] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing cache data integrity in the edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1210–1223, 2021.

[32] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '16)*, 2016, pp. 305–326.

[33] S. B. Erukala, B. Amogh, and K. Rajesh, "Sec-edge: Trusted blockchain system for enabling the identification and authentication of edge based 5g networks," *Computer Communications*, vol. 199, pp. 10–29, 2023.

[34] S. B. Erukala, D. Tokmakov, A. Perumalla, R. Kaluri, A. Bekyarova-Tokmakova, N. Mileva, and S. Lubomirov, "A secure end-to-end communication framework for cooperative iot networks using hybrid blockchain system," *Scientific Reports*, vol. 15, p. 11077, 2025.

[35] S. Babu Erukala, D. Tokmakov, A. Devi Aguru, R. Kaluri, A. Bekyarova-Tokmakova, and N. Mileva, "An end-to-end secure communication framework for smart homes environment using consortium blockchain system," *IEEE Access*, vol. 13, pp. 67 250–67 268, 2025.

[36] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)*, 2002, pp. 98–107.

[37] M. Backendal, F. Günther, and K. G. Paterson, "Puncturable key wrapping and its applications," in *Proceedings of International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '22)*, 2022, pp. 651–681.

[38] B. Wesolowski, "Efficient verifiable delay functions," in *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '19)*, 2019, pp. 379–407.

[39] I. Chung and Y. Bae, "The design of an efficient load balancing algorithm employing block design," *Journal of Applied Mathematics and Computing*, vol. 14, no. 1–2, p. 343–351, 1986.

[40] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Proceedings of Annual International Cryptology Conference (CRYPTO '18)*, 2018, pp. 757–788.