Optimal Verifiable Data Streaming Protocol with Data Auditing

Jianghong Wei[1,2,3], Guohua Tian[1], Jun Shen[1], Xiaofeng Chen[1,2(✉)], and Willy Susilo[4]

[1] State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, China
xfchen@xidian.edu.cn
[2] State Key Laboratory of Cryptology, P. O. Box 5159, Beijing 100878, China
[3] State Key Laboratory of Mathematical Engineering and Advanced Computing, PLA Strategic Support Force Information Engineering University, Zhengzhou, China
[4] School of Computing and IT, University of Wollongong, Wollongong 2522, NSW, Australia
wsusilo@uow.edu.au

Abstract. As smart devices connected to networks like Internet of Things and 5G become popular, the volume of data generated over time (i.e., stream data) by them is growing rapidly. As a consequence, for these resources-limited client-side devices, it becomes very challenging to store the continuously generated stream data locally. Although the cloud storage provides a perfect solution to this problem, the data owner still needs to ensure the integrity of the outsourced stream data, since various applications built upon stream data are sensitive of both its context and order. To this end, the notion of verifiable data streaming (VDS) was proposed to effectively append and update stream data outsourced to an untrusted cloud server, and has received significant attention. However, previous VDS constructions adopt Merkle hash tree to capture the integrity of outsourced data, and thus inevitably have logarithmic costs. In this paper, we further optimize the construction of VDS in terms of communication and computation costs. Specifically, we use the digital signature scheme to ensure the integrity of outsourced stream data, and employ a recently proposed RSA accumulator (v.s. Merkle hash tree) to invalidate the corresponding signature after each data update operation. Benefited from this approach, the resulted VDS construction achieves optimal, i.e., having constant costs. Furthermore, by specifying the underly signature scheme with the BLS short signature and carefully combining it with the RSA accumulator, we finally obtain an optimal verifiable data streaming protocol with data auditing. We prove the security of the proposed VDS construction in the random oracle model.

Keywords: Verifiable data streaming · Outsourced storage · Data auditing · Cryptographic accumulator

E. Bertino et al. (Eds.): ESORICS 2021, LNCS 12973, pp. 296–312, 2021.
https://doi.org/10.1007/978-3-030-88428-4_15

1 Introduction

With the extensive use of smart devices connected to networks like Internet of Things and 5G, both individuals and enterprises have been generating huge amounts of data. In particular, as a kind of important and common data type, stream data appears in various application scenarios, such as network intrusion detection, DNA sequence analysis, stock trading, and so on [2]. Although there has been a lot of research focusing on how to effectively store and manage stream data [1,12], the explosive and unpredicted growth of stream data scale still brings serious storage issues, especially for those resource-restricted client-side devices.

Fortunately, the emergence of cloud computing provides an economical and convenient storage mode for big stream data. That is, users can continuously stream the local data to cloud storage servers (e.g., Dropbox, Google Driver and Microsoft Azure), and then access the outsourced data via lightweight devices connected networks at anytime and anywhere. However, after the data is outsourced to cloud storage servers, the data owner completely loses its control right. This means that, to ensure the security of the outsourced data, the data owner has to trust these cloud storage servers. On the other hand, a large number of cloud data leakage incidents have shown that cloud storage servers are not completely trusted in fact [16]. To this end, a lot of active studies have been proposed to conquer the security issues of outsourced data in the context of cloud computing, e.g., proof of retrievability [26], secure duplication [3], verifiable update [10] and cryptographically enforced access control [28].

Among these security issues, the integrity of the outsourced data is especially important, since it is directly related to the data availability. In other words, a data user should be convinced that the data returned back by the cloud storage server is consistent with the data that was originally outsourced by the data owner. Otherwise, the analysis results based on the retrieved data may be biased or even maliciously misled. When focusing on stream data, due to its own features of unpredictable size and location-sensitive, this problem becomes more challenging.

As the first step towards solving the above problem, Schröder and Schröder [24] introduced the notion of verifiable data streaming (VDS) protocol. They put forward an efficient instantiation built upon a novel authenticated data structure named as chameleon authentication tree (CAT), which is essentially a generalized Merkle hash tree. Specifically, in their construction, the data owner assigns data items to leaf nodes of the CAT in a natural order, and thus binds the data content and the corresponding position. At the same time, the one-way nature of the underlying hash functions ensures that the outsourced data cannot be tampered with. In addition, by using a chameleon hash function and maintaining a dynamic verification key[1], the data owner can update previously outsourced data items in an efficient manner, and also invalidate them. On the other hand, any data user can independently verify the integrity of a requested data item by reconstructing the hash value at the root node of the CAT and further comparing it with that

[1] The verification key is updated after each data update operation.

contained in the verification key. On the whole, this VDS construction almost satisfies all of the above listed properties, except that it requires to fix the size of the CAT at the beginning. In other words, the number of data items that can be authenticated is bounded, which violates the first property. Moreover, both the computation cost and communication overhead of this construction are logarithmic in the bounded number of data items.

Towards optimizing the above VDS protocol, Schröder and Simkin [25] used the CAT in a black-box way, and proposed a more efficient VDS protocol. Particularly, in this construction, the number of data items is unbounded, and both the computation and communication costs are only logarithmic in the number of all authenticated data items so far. However, their construction was just proved secure in the random oracle model, at the cost of achieving these optimization goals. Furthermore, Krupp et al. [18] formally defined the notion of chameleon vector commitment (CVC), based on which they put forward the first unbounded VDS protocol in the standard model. Since they also followed the idea of Merkle hash tree used in previous constructions, the complexity of their VDS protocol built upon CVC is the same as Schröder and Simkin's [25] protocol. In addition, they also proposed another VDS construction by combining the digital signature scheme and the cryptographic accumulator. Notably, such a VDS construction is nearly optimal, that is, to achieve constant communication overhead and computation cost. But the computation cost of the cloud server responding to a query is linear in the number of update operations conducted so far.

Although a lot of valuable research efforts have been made, the state-of-the-art VDS construction is still not optimal, i.e., achieving constant computation and communication costs. In addition, the complexity of previous VDS protocols is evaluated under a single query. That is, for concurrent queries that retrieve multiple data items at once, the cost of these protocols is linear in the size of concurrent queries. In view of this state of affairs, a nature and important problem is how to design an optimal VDS protocol.

1.1 Our Contribution

In this paper, we focus on the problem of constructing more efficient VDS protocols, and improve upon the state-of-the-art. Roughly, to achieve this goal, we employ the approach of digital signature plus cryptographic accumulator to instantiate VDS, rather than the traditional method of Merkle hash tree. That is, the digital signature scheme is used to bind the content of each data item and its position, and also to ensure their integrity. Moreover, to invalidate the old signature after each data update operation, we add it to the cryptographic accumulator, and let the cloud server generate a non-membership witness for the current signature of each queried data item, so as to resist replay attacks.

In more detail, we adopt BLS signature [6] as the underlying digital signature scheme. It not only guarantees the integrity of the retrieved data, but also allows the data owner or other third party to audit the integrity of those previously outsourced data items without retrieving them. With respect to the cryptographic

accumulator, we employ Boneh et al.'s [5] RSA accumulator that can batch non-membership witnesses. By carefully combining these two primitives, we finally obtain a verifiable and auditable data streaming (VADS for short) protocol that has constant costs.

Specifically, we conduct the following contributions:

- We formalize the syntax and security notion of VADS protocol, and propose a concrete VADS construction built upon the Boneh-Lynn-Shacham (BLS) signature [6] and the RSA accumulator due to Boneh et al. [5].
- We prove the security of our proposal in the random oracle model, and reduce its security to the computational Diffie-Hellman assumption over bilinear groups as well as the security of the underlying RSA accumulator, which in turn depends on the adaptive root assumption over hidden order groups.
- We theoretically analyze the performance of the proposed VADS construction. Specifically, when considering our VADS construction as a VDS protocol, it outperforms the state-of-the-art, and achieves optimal. In addition, it also features the functionality of data auditing.

1.2 Related Work

A naive solution to verify the integrity of outsourced stream data is to use a simple Merkle hash tree [21]. Specifically, all data items are stored at the leaves of the tree, and each internal node is associated with the hash value of the concatenation of its children's values. Then, the hash value of the root node is used as the public verification key, and the integrity proof of each data item consists of all node values that are required for reconstructing root node's value from the corresponding leaf node. The shortcoming of this solution is that the verification key needs to be immediately updated after either updating an old data item or appending a new data item, which brings an infeasible burden of managing the verification key, especially for stream data with large scale and high transmission rate.

Towards overcoming the above shortcoming, Schröder and Schröder [24] introduced the notion of verifiable data streaming protocols, and proposed the first VDS construction supporting a bounded number of authenticated data items. In their protocol, appending a new data item no longer needs to update the verification key, but the total number of data items is initially fixed. Subsequently, Schröder and Simkin [25] removed such a limitation, and constructed the first unbounded VDS protocol in the random oracle model. Krupp et al. [18] further presented nearly optimal VDS constructions in terms of communication and computation costs.

In addition, there are several other related works that extend the original VDS protocol. For instance, Chen et al. [9] and Xu et al. [32] combined VDS with homomorphic encryption to realize verifiable computation on stream data. Zhang et al. [36] proposed a VDS construction with accountability. Xu et al. [31] presented a lightweight VDS construction supporting range queries. Recently,

Sun et al. [27] employed Schröder and Simkin's [25] approach[2] to propose a VDS protocol with public data auditing. However, Li et al. [20] demonstrated that their construction can be invalidated by any adversarial cloud server, and thus failed to achieve the goal of data auditing.

Another cryptographic primitive similar to VDS is verifiable databases (VDB) proposed by Benabbas et al. [4]. The main difference between the two is that the database size needs to be fixed in the setup phase of VDB schemes, while it is allowed to be unbounded in the setting of VDS. Therefore, VDS can be regarded as a generalization of VDB. Since its introduction, a lot of research around VDB has been conducted, such as new primitives for constructing VDB [8,19], VDB with efficient updates and deletions [11,22], VDB supporting various SQL queries [33,35] as well as public auditing [29]. In addition, Papamanthou et al. [23] also introduced a similar notion named as streaming authenticated data structures, which enable the data owner to efficiently compute a verification key, and then use it to verify the computation results of the stream data from the server. However, the verification key in their construction dynamically changes after streaming or uploading each data item.

Juels and Kaliski [17] put forward the notion of proof of retrievability (PoR, a.k.a data auditing), which is also closely related to VDS. This cryptographic primitive allows the data owner to verify the integrity of those outsourced data without retrieving them again, and has been extensively studied. Those early PoR constructions [17,26] are static, and cannot be used in the dynamic setting due to replay attacks. As discussed in [13], in order to resist replay attacks, authenticated data structures (e.g., Merkle hash trees and authenticated dictionaries) must be employed to invalidate those previously authenticated data items in the case of dynamic. However, dynamic PoR or data auditing schemes [14,15,30,34] following this manner usually incur logarithmic costs. Also note that in the setting of VDS, the data user first retrieves the data, and further checks its integrity for downstream processing. Thus, adding the functionality of data auditing to VDS makes it more flexible, and can also ensure the integrity of the outsourced stream data when we do not need to retrieve and use them.

1.3 Organization

The reset of this paper is organized as follows: In Sect. 2, we introduce necessary preliminaries. Section 3 formalizes the syntax and security notion of verifiable and auditiable data streaming protocol. We put forward a concrete VADS construction in Sect. 4, and prove its security in the random oracle model. The performance analysis of the proposed VADS construction is presented in Sect. 5. We conclude this paper in Sect. 6.

[2] In Sun et al.'s construction, they used the notion of adaptive trapdoor hash authentication tree. But we note that it is essentially the fully dynamic CATs constructed by Schröder and Simkin.

2 Preliminaries

2.1 Notations

Throughout this paper, we use the following notations:

- $\mathsf{Primes}(\lambda)$: The set of primes belonging to $[0, 2^\lambda)$.
- $\mathsf{EEA}(\cdot,\cdot)$: The extended euclidean algorithm.
- $\mathsf{H}_{\mathbb{G}}$, $\mathsf{H}_{\mathbb{G}'}$: Hash functions with domains $\mathbb{G}$, $\mathbb{G}'$.
- H_λ, $\mathsf{H}_{\mathsf{Prime}}$: Hash functions with domains $[0, 2^\lambda)$, $\mathsf{Primes}(\lambda)$.
- S: Unbounded stream data.
- $s[i]$: The i-th data item of the stream data S.
- $S[J]$: The data item set $\{s[j]|j \in J\}$.
- $[n]$: The integer set $\{1, 2, \ldots, n\}$.
- $x \xleftarrow{\$} R$: Randomly sampling x from the set R.
- $I \xleftarrow{\$} R$: Randomly sampling a subset I of the set R.

2.2 Bilinear Groups and CDH Assumption

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups with the prime order p in the size λ, and g be a random generator of the group $\mathbb{G}$. A map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is said to be bilinear if the following properties hold:

- *Bilinearity*: For any group elements $u, v \in \mathbb{G}$ and integers $\alpha, \beta \in \mathbb{Z}_p$, it holds that $e(u^\alpha, v^\beta) = e(u, v)^{\alpha\beta}$.
- *Non-degeneracy*: $e(g, g) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}_T}$ is the unit element of $\mathbb{G}_T$.
- *Computability*: For any group elements $u, v \in \mathbb{G}$, there exists a probabilistic polynomial-time (PPT) algorithm that can efficiently compute $e(u, v)$.

To simplify subsequent descriptions, we assume that there exists a PPT algorithm BilGen that takes the security parameter λ as input, and then generates bilinear groups, i.e., $(\mathbb{G}, \mathbb{G}_T, e, p, g) \leftarrow \mathsf{BilGen}(\lambda)$.

The security of our protocol is partially built upon the computational Diffie-Hellman (CDH) assumption over bilinear groups, which is formalized as follows:

Definition 1 (CDH Assumption [6]). *Let* $(\mathbb{G}, \mathbb{G}_T, e, p, g) \leftarrow \mathsf{BilGen}(\lambda)$ *be bilinear groups,* α *and* β *be two random integers sampled from* $\mathbb{Z}_p$*. We say the CDH assumption holds if for any PPT adversary given* (g, g^α, g^β)*, its probability of outputting* $g^{\alpha\beta}$ *is negligible in the security parameter* λ*.*

2.3 Groups of Unknown Order and RSA Accumulator

Our protocol employs a cryptographic accumulator based on a group $\mathbb{G}'$ of unknown order. Given a security parameter λ, we assume there exists a PPT algorithm $\mathsf{UGen}(\lambda) \to \mathbb{G}'$ that can generate such a group with order in a range $[a, b]$ such that $\frac{1}{a}$, $\frac{1}{b}$ and $\frac{1}{b-a}$ are all negligible in λ. As discussed by Boneh et

al. [5], available two concrete instantiations of $\mathbb{G}'$ include class groups and the quotient group $\mathbb{Z}_N^*/\{-1, 1\}$ of an RSA group, where N is an RSA modulus.

Specifically, we use Boneh et al.'s [5] RSA accumulator that supports batching non-membership witnesses. Since our protocol only involves adding elements to the accumulator and producing non-membership witnesses, here we just present a simplified version. In more detail, it consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm takes as input a security parameter λ. It first generates a hidden order group $\mathbb{G}' \leftarrow \mathsf{UGen}(\lambda)$, and randomly chooses an element $h \in \mathbb{G}'$. Furthermore, it initializes the element set as $R \leftarrow \emptyset$ and the corresponding accumulator as $\mathsf{Acc}(\emptyset) \leftarrow h$.
- $\mathsf{Add}(\mathsf{Acc}(R), R, z)$: The addition algorithm takes as input the current accumulator $\mathsf{Acc}(R)$ and element set R as well as a prime $z \in \mathsf{Primes}(\lambda)$. If $z \in R$, then it directly outputs $\mathsf{Acc}(R)$. Otherwise, it updates the element set as $R' \leftarrow R \cup \{z\}$ and the accumulator as $\mathsf{Acc}(R') \leftarrow (\mathsf{Acc}(R))^z$.
- $\mathsf{WitCreate}(\mathsf{Acc}(R), R, z)$: This non-membership witness creation algorithm takes as input the current accumulator $\mathsf{Acc}(R)$ and element set R as well as an element $z \notin R$. It first computes $z^* = \prod_{z' \in R} z'$, and employs the extended euclidean algorithm to compute coefficients $x, y \in \mathbb{Z}$ such that $x \cdot z^* + y \cdot z = 1$. Then, it assigns $Y = h^y$, and outputs $\pi = (x, Y)$ as the witnesses of $z \notin R$.
- $\mathsf{Verify}(\mathsf{Acc}(R), \pi, z)$: This verification algorithm takes as input the accumulator $\mathsf{Acc}(R)$, the non-membership witness π and the corresponding element z. It first parses the witness π as (x, Y). Then, it outputs 1 if $(\mathsf{Acc}(R))^x \cdot Y^z = h$ and 0 otherwise.

2.4 Hashing to Primes

In our VADS construction, we need a hash function $\mathsf{H}_{\mathsf{Prime}}$ that maps arbitrary strings to primes belonging to $[0, 2^\lambda)$ for the given security parameter λ. We make use of such a hash function introduced in [7].

Specifically, let $\mathsf{H}_\lambda : \{0,1\}^* \rightarrow [0, 2^\lambda)$ be a collision resistant hash function and $i \leftarrow 0$. Given an input $x \in \{0,1\}^*$, if $y \leftarrow \mathsf{H}_\lambda(x, i)$ is a prime then directly assign $y \leftarrow \mathsf{H}_{\mathsf{Prime}}(x)$, otherwise let $i \leftarrow i + 1$ and continue to evaluate $\mathsf{H}_\lambda(x, i)$, until a prime is found. If H_λ was a random function, then the running time of $\mathsf{H}_{\mathsf{Prime}}$ is $\mathcal{O}(\lambda)$, since the number of primes less than 2^λ is $\mathcal{O}(\frac{2^\lambda}{\lambda})$.

3 Verifiable and Auditable Data Streaming Protocol

In this section, we formalize the syntax of VADS protocol and its security requirement.

In the context of a VADS protocol, a client $\mathcal{C}$ can outsource some unbounded data $S = s[1], s[2] \ldots$ to an untrusted server $\mathcal{S}$ in a streaming manner. That is, $\mathcal{C}$ reads a data item $s[i] \in \{0,1\}^\lambda$ at a time and sends it to $\mathcal{S}$, who then stores it in a database DB. The outsourced stream data must be publicly verifiable and auditable in the sense that the server $\mathcal{S}$ can neither modify the position and

content of any data item nor append additional data items to the database. In addition, given some proof[3] generated and returned by the server, the client $\mathcal{C}$ (or a third party) can publicly verify that the requested data item $s[i]$ was indeed located at the position i of the outsourced stream data, and can also publicly judge that the server $\mathcal{S}$ has not deleted the data without retrieving it. Moreover, the client $\mathcal{C}$ is allowed to update any data item $s[i]$ with a new one $s'[i]$.

Formally, a verifiable and auditable data streaming protocol $\mathcal{VADS} = (\mathsf{Setup}, \mathsf{Append}, \mathsf{Query}, \mathsf{Audit}, \mathsf{Verify}, \mathsf{Judge}, \mathsf{Update})$ is seven-tuple of algorithms and protocols, and is interactively executed between a client $\mathcal{C}$ and a server $\mathcal{S}$, which are both modeled as PPT algorithms. The details of these algorithms are specified as follows:

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm is performed by the client $\mathcal{C}$, and takes a security parameter λ as input. It outputs a verification key vk and a secret key sk. The former is sent to the server $\mathcal{S}$, and the later is held by the client $\mathcal{C}$. To simplify descriptions, we assume that vk is part of sk.
- $\mathsf{Append}(\mathsf{sk}, s)$: The append protocol is initiated by the client $\mathcal{C}$, and takes the secret key sk and a data item s as input. It appends s to the database DB maintained by the server $\mathcal{S}$, and may output a new secret key sk'.
- $\mathsf{Query}(\mathsf{vk}, i, DB)$: The query protocol takes as input the verification key vk and the database DB provided by the server $\mathcal{S}$ as well as an index $i \in \mathbb{N}^*$ from the client $\mathcal{C}$. At the end of this protocol, the server $\mathcal{S}$ outputs the i-th entry $s[i]$ of the database DB along with a proof π_q, or an error symbol $\perp$.
- $\mathsf{Verify}(\mathsf{vk}, s[i], i, \pi_\mathsf{q})$: The verification algorithm is run by the client $\mathcal{C}$, and takes as input the verification key vk, an index i and a data item $s[i]$ as well as a proof π_q. It outputs the data item $s[i]$ if $s[i]$ is actually the i-th entry of the database DB. Otherwise, it outputs an symbol $\perp$.
- $\mathsf{Audit}(I)$: The audit protocol is initiated by the client $\mathcal{C}$, and takes as input an index set $I \subset \mathbb{N}^*$. At the end of this protocol, it outputs a proof π_a returned by the server $\mathcal{S}$.
- $\mathsf{Judge}(\mathsf{vk}, \pi_\mathsf{a})$: The judge algorithm is performed by the client $\mathcal{C}$, and takes the verification key vk and a proof π_a as input. It outputs 1 if the server passes the judge and 0 otherwise.
- $\mathsf{Update}(\mathsf{sk}, i, s', \mathsf{vk}, DB)$: The update protocol is conducted between the server $\mathcal{S}$ that provides the verification key vk as well as the database DB and the client $\mathcal{C}$ that provides the secret key sk, an index i as well as a data item s'. At the end of this protocol, the server $\mathcal{S}$ will replace the original i-th data item $s[i]$ of the database DB with s', and both $\mathcal{S}$ and $\mathcal{C}$ update the original verification key to vk'.

A VADS protocol must fulfill the usual requirement of correctness. That is, when all algorithms/protocols of a VDS protocol are honestly executed, it should work as expected. This is formally defined as follows.

[3] For different tasks (i.e. query and auditing), the corresponding proofs are also different.

Definition 2 (Correctness). *A VADS protocol $\mathcal{VADS}$ is said to be correct provided that for correctly generated verification and secret key* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(\lambda)$ *and any stream data* $S = s[1], s[2], \ldots$, *if* $\mathsf{sk}' \leftarrow \mathsf{Append}(\mathsf{sk}, s[i])$ *and* $(s[i], \pi_{\mathsf{q}}) \leftarrow \mathsf{Query}(\mathsf{vk}, DB, i)$ *as well as* $\pi_{\mathsf{a}} \leftarrow \mathsf{Audit}(I)$, *then* $\mathsf{Verify}(\mathsf{vk}, i, s[i], \pi_{\mathsf{q}}) \rightarrow s[i]$ *and* $\mathsf{Judge}(\mathsf{vk}, \pi_{\mathsf{a}}) \rightarrow 1$ *must hold with an overwhelming probability. Furthermore, the correctness must hold even after performing an arbitrary number of updates.*

The intuition behind the security of the VADS protocol is that an adversary $\mathcal{A}$ should not be able to modify those data items stored in the database, including their contents and positions, and nor should it be able to append new elements to the database. Moreover, for an updated data item $s'[i]$, its old value $s[i]$ should no longer pass verification. This can be modeled in a security game played between a challenger $\mathcal{B}$ that plays the role of the client and an adversary $\mathcal{A}$ that plays the role of the server. The security game comprises of the following three phases:

Setup Phase. In this phase, $\mathcal{B}$ runs the setup algorithm $\mathsf{Setup}(\lambda) \rightarrow (\mathsf{vk}, \mathsf{sk})$, and sends the verification key vk to the adversary $\mathcal{A}$. In addition, $\mathcal{B}$ creates a database DB and a set Q both initialized as empty.

Query Phase. The adversary $\mathcal{A}$ is allowed to adaptively add a new data item s to the database DB by streaming it to the challenger $\mathcal{B}$. As a response, $\mathcal{B}$ performs the algorithm $\mathsf{Append}(\mathsf{sk}, s)$, and returns the index i of s and corresponding proof π_{q} to the adversary $\mathcal{A}$. Furthermore, the adversary $\mathcal{A}$ can also update a data item $s[i] \in DB$ with a new one $s'[i]$ by providing a tuple $(i, s'[i])$ to the challenger $\mathcal{B}$, who then executes the algorithm $\mathsf{Update}(\mathsf{vk}, DB, i, s'[i])$ and returns a new proof $\hat{\pi}_{\mathsf{q}}$ to $\mathcal{A}$. Throughout this phase, the challenger $\mathcal{B}$ always immediately updates the verification/secret keys after each query, and adds new data items and corresponding indexes to the set Q, i.e., $Q = \{(1, s[1]), \ldots, (q(\lambda), s[q(\lambda)])\}$.

Challenge Phase. Finally, when the adversary $\mathcal{A}$ decides to end the game, it outputs a tuple $(i^*, I^*, s[i^*], \pi_{\mathsf{q}}^*, \pi_{\mathsf{a}}^*)$. Then, let $s^* \leftarrow \mathsf{Verify}(\mathsf{vk}, i^*, s[i^*], \pi_{\mathsf{q}}^*)$ and $b^* \leftarrow \mathsf{Judge}(\mathsf{vk}, \pi_{\mathsf{a}}^*)$. We say the adversary $\mathcal{A}$ wins this security game if it holds that:

$$\big(s^* \neq \perp \wedge (i^*, s^*) \notin Q\big) \vee \big(b^* = 1 \wedge \exists i' \in I^* \; s.t. \; (i', s[i']) \notin Q\big)$$

We denote by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{VADS}}(\lambda)$ the probability of $\mathcal{A}$ winning in the game.

Definition 3 (Security). *A VADS protocol is secure if for any PPT adversary $\mathcal{A}$, the probability* $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{VADS}}(\lambda)$ *is negligible in the security parameter λ.*

4 The Construction of VADS

In this section, we propose a concrete construction of verifiable and auditable data streaming protocol. To this end, we first outline our techniques, and provide an overview for ease of understanding. Then, we specify our VADS protocol that supports a single query, just like previous VDS constructions.

4.1 Overview

Our solution to the problem of verifying the integrity of the outsourced stream data starts from a trivial idea, i.e., signing each data item and its position in the stream data with a digital signature scheme. Obviously, the unforgeability of signatures makes the outsourced data items can neither be tampered with nor re-ordered. Here, the problem is how to update data items. Note that for a position i, simply signing i and the new data item $s'[i]$ is not sufficient, since the old signature σ_i for i and the original data item $s[i]$ would still remain valid. In other words, upon a query about retrieving the data item at position i, the cloud server can legitimately return $s[i]$ and σ_i, rather than $s'[i]$, although the data owner has conducted the update operation at the position i.

Observe that the key to making the above solution effective is to invalidate the old signature after each update operation. But here the verification key is the public key of the underlying signature scheme, and does not relate to the updated data item. Thus, to realize data update, the data owner has to generate a new pair of public key and secret key, and signs all data items again, which is clearly infeasible. This is also why previous VDS constructions [18,24,25] use the approach of Merkle hash tree to invalidate those old data items. But at the same time, this manner inevitably brings logarithmic costs.

Towards constructing optimal VDS protocol, we still follow the above trivial idea, but adopt the cryptographic accumulator to revoke those old signatures. More precisely, we use the state-of-the-art RSA accumulator proposed by Boneh et al. [5] to invalidate the old signature after each update operation. That is, we add the old signature into the current accumulator after the corresponding data item was updated. Then, upon a query for position i, the cloud server needs to create a non-membership proof π, so as to prove that the returned signature σ_i and the corresponding data item $s[i]$ are the newest ones. The data user employs the proof π to verify the membership of σ_i against the current accumulator, which is a short digest of all revoked signatures and assigned as part of the verification key. On the whole, by dynamically freshing the accumulator with each update operation, those old signatures can be effectively invalidated. In addition, since the underlying accumulator features of batching non-membership witnesses, the resulted VDS construction achieves constant computation and communication costs, and therefore is optimal.

Furthermore, since we use the digital signature scheme to ensure the integrity of the outsourced stream data, then it may be possible to enable the above VDS construction to capture the functionality of public data auditing as done in PoR schemes [26,30]. To this end, we instantiate the underlying signature scheme of the above VDS construction with the BLS signature [6], and achieve the goal of data auditing with the approach of Shacham and Waters [26]. Here we emphasize that we focus on dynamic stream data, but Shacham and Waters dealt with the case of static. Therefore, to resist replay attacks during the auditing procedure, we additionally require the cloud server to return a non-membership proof for those challenged data items along with a tag set that identifies them. This also

makes the communication cost of data auditing in our VDS construction is linear in the set of challenge set.

Finally, by carefully combing the BLS signature with Boneh et al.'s [5] RSA accumulator, we obtain a optimal VDS protocol with data auditing.

4.2 The Construction

The concrete construction consists of the following algorithms.

- $\mathsf{Setup}(1^\lambda)$: Given a security parameter λ, the client $\mathcal{C}$ first generates bilinear groups $(\mathbb{G}, \mathbb{G}_T, e, p, g) \leftarrow \mathsf{BilGen}(\lambda)$ and a hidden order group $\mathbb{G}' \leftarrow \mathsf{UGen}(\lambda)$. Then, $\mathcal{C}$ randomly picks an integer $\alpha \in \mathbb{Z}_p$ and two elements $u \in \mathbb{G}, h \in \mathbb{G}'$. The client $\mathcal{C}$ also selects four collision-resistant hash functions defined in the following way: $\mathsf{H}_\mathbb{G} : \{0,1\}^* \rightarrow \mathbb{G}$, $\mathsf{H}_{\mathbb{G}'} : \{0,1\}^* \rightarrow \mathbb{G}'$, $\mathsf{H}_{\mathsf{Prime}} : \{0,1\}^* \rightarrow \mathsf{Primes}(\lambda)$ and $\mathsf{H}_\lambda : \{0,1\}^* \rightarrow [0, 2^\lambda)$. In addition, the client $\mathcal{C}$ initializes a counter $\mathsf{cnt} \leftarrow 1$ and an accumulator $\mathsf{Acc}(\emptyset) \leftarrow h$. Finally, The client outputs the verification key $\mathsf{vk} = \{(\mathbb{G}, \mathbb{G}_T, e, p, g), \mathbb{G}', \mathsf{H}_\mathbb{G}, \mathsf{H}_{\mathbb{G}'}, \mathsf{H}_{\mathsf{Prime}}, \mathsf{H}_\lambda, \mathsf{Acc}(\emptyset), A = g^\alpha, u, h\}$ and the secret key $\mathsf{sk} = \{\alpha, \mathsf{cnt}, \mathsf{vk}\}$. After that, sk is kept privately by $\mathcal{C}$, and vk is sent publicly to the server $\mathcal{S}$, which further creates a database DB and a revocation list R that are both initialized as empty.
- $\mathsf{Append}(\mathsf{sk}, s, DB)$: To append a data item $s \in \mathbb{Z}_p$ to the database DB, the client $\mathcal{C}$ first retrieves the current counter cnt from the secret key sk, and assigns an index $i \leftarrow \mathsf{cnt}$ for s. Then, $\mathcal{C}$ generates a signature $\sigma_i \leftarrow \big(\mathsf{H}_\mathbb{G}(i||\mathsf{tag}_i) \cdot u^s\big)^\alpha$, and forwards the tuple $(i, s, \sigma_i, \mathsf{tag}_i)$ to $\mathcal{S}$, where tag_i[4] is a random string sampled from $\{0,1\}^\lambda$. Meanwhile, $\mathcal{C}$ updates the counter $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$. Upon receiving the tuple, the server $\mathcal{S}$ first verifies the validity of the signature as follows:

$$e(\sigma_i, g) \stackrel{?}{=} e\big(A, \mathsf{H}_\mathbb{G}(i||\mathsf{tag}_i) \cdot u^s\big) \tag{1}$$

  If it passes the verification, then $\mathcal{S}$ stores s at the position i of the database DB. Otherwise, $\mathcal{S}$ rejects $\mathcal{C}$'s append request.
- $\mathsf{Query}(\mathsf{vk}, i, DB)$: When the client $\mathcal{C}$ intends to query some data item that was stored in the database DB, it sends the corresponding index i to the server $\mathcal{S}$. To answer such a query, $\mathcal{S}$ first retrieves the tuple $(i, s[i], \sigma_i, \mathsf{tag}_i)$ from the database DB. Then, it needs to produce a non-membership proof with respect to the current accumulator $\mathsf{Acc}(R)$, so as to demonstrate that the signature σ_i has not been revoked by the client $\mathcal{C}$. To this end, $\mathcal{S}$ computes $z_i \leftarrow \mathsf{H}_{\mathsf{Prime}}(\mathsf{tag}_i)$ and directly retrieves[5] $z^* \leftarrow \prod_{\mathsf{tag} \in R} \mathsf{H}_{\mathsf{Prime}}(\mathsf{tag})$. It then lets $(x, y) \leftarrow \mathsf{EEA}(z^*, z_i)$, and assigns $\pi = (x, Y = h^y)$. Finally, the server $\mathcal{S}$ returns the requested data item $s[i]$ and the proof $\pi_\mathsf{q} = \{\sigma_i, \mathsf{tag}_i, \pi\}$ to the client $\mathcal{C}$.

[4] We assume that each signature has a unique tag. So we can use it to identify the corresponding signature as we do in the Query protocol.

[5] Note that the value z^* is computed with the revocation list R and is independent of i, and thus can be refreshed after each update operation.

- $\mathsf{Verify}(\mathsf{vk}, s[i], i, \pi_{\mathsf{q}})$: After getting the response from the server $\mathcal{S}$, the client $\mathcal{C}$ first parses the proof π_{q} as $\{\sigma_i, \mathsf{tag}_i, \pi\}$, where $\pi = (x, Y)$. After that, it recomputes $z_i \leftarrow \mathsf{H}_{\mathsf{Prime}}(\mathsf{tag}_i)$, and verifies if the following condition holds:

$$\big(\mathsf{Acc}(R)\big)^x \cdot Y^{z_i} \stackrel{?}{=} h \tag{2}$$

  If not, it outputs $\perp$ and terminates. Otherwise, it further checks if the following condition holds:

$$e(\sigma_i, g) \stackrel{?}{=} e\Big(A, \mathsf{H}_{\mathbb{G}}(i||\mathsf{tag}_i) \cdot u^{s[i]}\Big) \tag{3}$$

  If yes, it outputs $s[i]$. Otherwise, it outputs $\perp$.
- $\mathsf{Audit}(I)$: To check the retrievability of the outsourced stream data, the client $\mathcal{C}$ (or a third party) first retrieves the current counter cnt, and selects a random c-element subset $I \subseteq [\mathsf{cnt}]$. Then, for each $i \in I$, it chooses a random integer $\nu_i \in \mathbb{Z}_p$, and sends the set $\{(i, \nu_i)|i \in I\}$ to the server $\mathcal{S}$. After receiving this set, $\mathcal{S}$ retrieves $\{(i, s[i], \sigma_i, \mathsf{tag}_i)|i \in I\}$ from the database DB, and lets $\nu = \sum_{i \in I} \nu_i s[i] \mod p$ and $\sigma_I = \prod_{i \in I} \sigma_i^{\nu_i}$. In addition, it needs to produce an aggregated non-membership proof with respect to the current accumulator $\mathsf{Acc}(R)$ for the set $\{\sigma_i|i \in I\}$, i.e., proving each σ_i has not been revoked by $\mathcal{C}$. To this end, it assigns $Q_I = \{\mathsf{H}_{\mathsf{Prime}}(\mathsf{tag}_i)|i \in I\}$, and generates the aggregated non-membership proof as $\pi_I \leftarrow \mathsf{WitCreate}^*(\mathsf{Acc}(R), R, Q_I)$ (see Algorithm 1). Finally, the server $\mathcal{S}$ returns the proof $\pi_{\mathsf{a}} = \{\nu, \sigma_I, \pi_I, \{\mathsf{tag}_i|i \in I\}\}$ to $\mathcal{C}$.
- $\mathsf{Judge}(\mathsf{vk}, \pi_{\mathsf{a}})$: Given the challenge set $\{(i, \nu_i)|i \in I\}$ and the returned proof $\pi_{\mathsf{a}} = \{\nu, \sigma_I, \pi_I, \{\mathsf{tag}_i|i \in I\}\}$, the client $\mathcal{C}$ reconstructs $Q_I = \{\mathsf{H}_{\mathsf{Prime}}(\mathsf{tag}_i)|i \in I\}$, and judges the validity of the received proof with the following condition:

$$\mathsf{WitVerify}^*(\mathsf{Acc}(R), R, Q_I, \pi_I) \stackrel{?}{=} 1 \tag{4}$$

  If the proof fails to pass the above verification, then $\mathcal{C}$ outputs 0. Otherwise, $\mathcal{C}$ further checks the following condition:

$$e(\sigma_I, g) \stackrel{?}{=} e\Big(A, \textstyle\prod_{i \in I} \mathsf{H}_{\mathbb{G}}(i||\mathsf{tag}_i)^{\nu_i} \cdot u^{\nu}\Big) \tag{5}$$

  The client $\mathcal{C}$ outputs 1 if it holds, and 0 otherwise.
- $\mathsf{Update}(\mathsf{sk}, i, s', \mathsf{vk}, DB)$: To replace a previously outsourced data item $s[i]$ with a new one s', the client $\mathcal{C}$ first retrieves the current tuple $(i, s[i], \sigma_i, \mathsf{tag}_i)$ from the server $\mathcal{S}$, and then checks its validity via the equation (1). If it passes the verification, $\mathcal{C}$ creates $\sigma_i' \leftarrow \big(\mathsf{H}_{\mathbb{G}}(i||\mathsf{tag}_i') \cdot u^{s'}\big)^{\alpha}$, where tag_i' is a new random string uniformly sampled from $\{0,1\}^{\lambda}$, and updates the current accumulator $\mathsf{Acc}(R) \leftarrow (\mathsf{Acc}(R))^{\mathsf{H}_{\mathsf{Prime}}(\mathsf{tag}_i)}$. Furthermore, $\mathcal{C}$ sends $(s', \sigma_i', \mathsf{tag}_i')$ and $\mathsf{Acc}(R)$ to $\mathcal{S}$. After that, $\mathcal{S}$ checks the validity of s' according to the equation (1) again, and replaces $(i, s[i], \sigma_i, \mathsf{tag}_i)$ with $(i, s', \sigma_i', \mathsf{tag}_i')$ if it passes the verification. Finally, the client $\mathcal{C}$ updates $R \leftarrow R \cup \{\mathsf{tag}_i\}$. In addition, both $\mathcal{C}$ and $\mathcal{S}$ use the updated accumulator $\mathsf{Acc}(R)$ to update the verification key vk.

Algorithm 1. Batching Non-membership Witnesses of the RSA Accumulator

Require: $\mathsf{WitCreate}^*(\mathsf{Acc}(R), R, Q)$:

$z^* \leftarrow \prod_{z' \in R} z'$
$w' \leftarrow \prod_{w \in Q} w$
$(x, y) \leftarrow \mathsf{EEA}(z^*, w')$
$V \leftarrow (\mathsf{Acc}(R))^x$
$Y \leftarrow h^y$
$\ell_1 \leftarrow \mathsf{H}_{\mathsf{Prime}}(w', Y, h \cdot V^{-1})$
$t_1 \leftarrow \lfloor \frac{w'}{\ell_1} \rfloor$, $T_1 = Y^{t_1}$
$h' \leftarrow \mathsf{H}_{\mathbb{G}'}(\mathsf{Acc}(R), V)$
$X' = (h')^x$
$\ell_2 \leftarrow \mathsf{H}_{\mathsf{Primes}}(\mathsf{Acc}(R), V, X')$,
$\gamma \leftarrow \mathsf{H}_{\lambda}(\mathsf{Acc}(R), V, X', \ell_2)$
$t_2 \leftarrow \lfloor \frac{x}{\ell_2} \rfloor$, $r \leftarrow x \mod \ell_2$
$T_2 \leftarrow \left(\mathsf{Acc}(R) \cdot (h')^{\gamma}\right)^{t_2}$
return $\pi = \{V, Y, T_1, T_2, X', r\}$

Require: $\mathsf{WitVerify}^*(\mathsf{Acc}(R), R, Q, \pi)$:

$\{V, Y, T_1, T_2, X', r\} \leftarrow \pi$
$w' \leftarrow \prod_{w \in Q} w$
$\ell_1 \leftarrow \mathsf{H}_{\mathsf{Prime}}(w', Y, h \cdot Y^{-1})$
$r' \leftarrow w' \mod \ell_1$
if $T_1^{\ell_1} \cdot Y^{r'} \neq h \cdot V^{-1}$ **then**
  return 0
end if
$h' \leftarrow \mathsf{H}_{\mathsf{G}'}(\mathsf{Acc}(R), V)$
$\ell_2 \leftarrow \mathsf{H}_{\mathsf{Prime}}(\mathsf{Acc}(R), V, X')$
$\gamma \leftarrow \mathsf{H}_{\lambda}(\mathsf{Acc}(R), V, X', \ell_2)$
if $T_2^{\ell_2} \cdot (\mathsf{Acc}(R) \cdot (h')^{\gamma})^r \neq V \cdot (X')^{\gamma}$ **then**
  return 0
end if
return 1

Correctness. The correctness of the above VADS protocol is guaranteed by the Eqs. (1–5), which in turn depend on the correctness of the underlying BLS signature and RSA accumulator. Specifically, for a correctly generated signature σ_i for the i-th data item $s[i]$, if it has not been revoked by the client, then it can naturally pass through the verification of the Eqs. (1–4) due to the correctness of the BLS signature and RSA accumulator. Furthermore, for correctly generated and non-revoked signature set $\{\sigma_i | i \in I\}$, as shown below, the Eq. (5) also holds:

$$\begin{aligned} e(\sigma_I, g) &= e\Big(\prod_{i \in I} \sigma_i^{\nu_i}, g\Big) = e\Big(\prod_{i \in I} \mathsf{H}_{\mathbb{G}}(i||\mathsf{tag}_i)^{\nu_i} \cdot u^{s[i] \cdot \nu_i}, g^{\alpha}\Big) \\ &= e\Big(\prod_{i \in I} \mathsf{H}_{\mathbb{G}}(i||\mathsf{tag}_i)^{\nu_i} \cdot u^{\sum_{i \in I} s[i] \cdot \nu_i}, A\Big) \\ &= e\Big(A, \prod_{i \in I} \mathsf{H}_{\mathbb{G}}(i||\mathsf{tag}_i)^{\nu_i} \cdot u^{\nu}\Big). \end{aligned}$$

Security. The security of the above VADS construction is guaranteed by the following theorem.

Theorem 1. *If the hash function* $\mathsf{H}_{\mathsf{prime}}$ *is collision-resistant, the adaptive root assumption holds in the RSA group, the CDH assumption holds in the bilinear group, then the proposed VADS protocol is secure in the random oracle model.*

Due to the limit of space, we provide the proof in the full version of this paper.

5 Performance Analysis

In this section, we theoretically discuss the performance of our proposal by comparing it with previous VDS constructions in terms of computation and communication costs as well as security properties.

Table 1. Comparisons with previous works in terms of costs

VDS Protocols	Communication Cost		Computation Cost					
	$\lvert\pi_{\mathsf{a}}\rvert$	$\lvert\pi_{\mathsf{q}}\rvert$	Append	Query	Verify	Update	Audit	Judge
Schröder and Schröder [24]	N/A	$\mathcal{O}(\log M)$	$\mathcal{O}(\log M)$	$\mathcal{O}(\log M)$	$\mathcal{O}(\log M)$	$\mathcal{O}(\log M)$	N/A	N/A
Schröder and Simkin [25]	N/A	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	N/A	N/A
Krupp et al. CVC [18]	N/A	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	N/A	N/A
Krupp et al. ACC [18]	N/A	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(U)$	N/A	N/A
Sun et al. [27]	N/A	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	N/A	N/A
Ours	$\mathcal{O}(\lvert I\rvert)$	$\boldsymbol{\mathcal{O}(1)}$	$\boldsymbol{\mathcal{O}(1)}$	$\boldsymbol{\mathcal{O}(1)}$	$\boldsymbol{\mathcal{O}(1)}$	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\lvert I\rvert)$	$\mathcal{O}(\lvert I\rvert)$

* *$|I|$ is the size of challenge index set I used in the auditing protocol. $|\pi_{\mathsf{a}}|$ is the proof size in the audit protocol. $|\pi_{\mathsf{q}}|$ is the proof size in the query protocol. M is the maximum number of data entries allowed to store into the database. N is the number of data entries currently stored in the database. U denotes the number of previous update operations.* `N/A` *means this item is not applicable to the corresponding scheme. We only consider those dominant operations like pairing and exponentiation.*

As demonstrated in Table 1, for a single query, the communication and computation costs of our construction achieve constant, and is independent of either the size of outsourced stream data or the number of update operations. From this perspective, the proposed VDS protocol outperforms the state-of-the-art, and is optimal. What is more, for concurrent queries, our proposal also has constant costs, but that of other VDS constructions are linear in the size of concurrent queries[6]. On the other hand, for each data auditing, the computation and communication costs of our VADS construction grows linearly with the size of the challenge set. This is mainly because that the cloud server has to return those tags assigned to challenged data items, so as to enable the data owner to verify the returned non-membership proof against the current accumulator. In fact, for the pure data auditing scheme, only those schemes in the case of statics have constant cost, while the cost of those schemes in the case of dynamics is usually logarithmic in the size of authenticated data items. Therefore, the auditing cost of our VADS construction, which deals with dynamic data, is acceptable.

Table 2. Comparisons with previous works in terms of properties

VDS Protocols	Unbounded	Auditability	Security Model	Complexity Assumption
Schröder and Schröder [24]	✗	✗	Standard Model	Discrete Logarithm
Schröder and Simkin [25]	✓	✗	Random Oracle	Discrete Logairthm
Krupp et al. CVC [18]	✓	✗	Standard Model	CDH
Krupp et al. ACC [18]	✓	✗	Standard Model	q-strong DH
Sun et al. [27]	✓	✗	Random Oracle	Discrete Logairthm
Ours	✓	✓	Random Oracle	CDH and Adaptive Root

[6] Due to limited space, here we omit how to extend our VADS protocol to support concurrent queries. In fact, it is straightforward, and just needs to create an aggregated non-membership proof for those requested data items by invoking the Algorithm 1.

In Table 2, we compare the security properties of the proposed VADS construction with previous VDS protocols. Observe that, except for Schröder and Schröder's [24] VDS protocol, the other protocols are all unbounded, and meet the unbounded feature of stream data. Although Krupp et al.'s [18] CVC construction is proved secure in the standard model and also is more efficient than other VDS constructions [18,24], its security is built upon a non-standard assumption, i.e., q-strong Diffie-Hellman assumption. At the same time, other listed VDS schemes are proved secure under standard assumptions. In particular, our VADS construction uniquely enjoys the functionality of data auditing, and thus allows the data owner to check the integrity of outsourced stream data without retrieving them.

6 Conclusion

In this paper, we revisit the problem of verifiable data streaming, and focus on further optimizing the efficiency of previous constructions. Specifically, by carefully combing the BLS signature with a recently proposed RSA accumulator, we put forward an optimal verifiable data streaming protocol with data auditing. That is, it achieves constant communication and computation costs, and outperforms the state-of-the-art. Additionally, compared with the original VDS, it features of public data auditing, and thus enables the data owner to verify the integrity of outsourced stream data in the case of not having to retrieve them. We prove its security under standard assumptions in the random oracle model.

Acknowledgment. This work was supported by the National Nature Science Foundation of China under Grants 61960206014 and 62172434, and in part by the Project funded by China Postdoctoral Science Foundation No. 2020M673348 and No. 2021T140531.

References

1. Arasu, A., et al.: STREAM: the Stanford stream data manager. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, p. 665. ACM (2003)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 1–16. ACM (2002)
3. Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-locked encryption and secure deduplication. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 296–312. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_18
4. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_7
5. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 561–586. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_20

6. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
7. Campanelli, M., Fiore, D., Greco, N., Kolonelos, D., Nizzardo, L.: Incrementally aggregatable vector commitments and applications to verifiable decentralized storage. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 3–35. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_1
8. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_5
9. Chen, C., Wu, H., Wang, L., Yu, C.: Practical integrity preservation for data streaming in cloud-assisted healthcare sensor systems. Comput. Netw. **129**, 472–480 (2017)
10. Chen, X., et al.: Publicly verifiable databases with all efficient updating operations. IEEE Trans. Knowl. Data Eng. (2020). https://doi.org/10.1109/TKDE.2020.2975777
11. Chen, X., Li, J., Huang, X., Ma, J., Lou, W.: New publicly verifiable databases with efficient updates. IEEE Trans. Dependable Secure Comput. **12**(5), 546–556 (2015)
12. Cugola, G., Margara, A.: Processing flows of information: from data stream to complex event processing. ACM Comput. Surv. **44**(3), 15:1–15:62 (2012)
13. Erway, C.C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. ACM Trans. Inf. Syst. Secur. **17**(4), 15:1–15:29 (2015)
14. Esiner, E., Kachkeev, A., Braunfeld, S., Küpçü, A., Özkasap, Ö.: FlexDPDP: flexlist-based optimized dynamic provable data possession. ACM Trans. Storage **12**(4), 23:1–23:44 (2016)
15. Etemad, M., Küpçü, A.: Generic dynamic data outsourcing framework for integrity verification. ACM Comput. Surv. **53**(1), 8:1–8:32 (2020)
16. Grobauer, B., Walloschek, T., Stöcker, E.: Understanding cloud computing vulnerabilities. IEEE Secur. Priv. **9**(2), 50–57 (2011)
17. Juels, A., Kaliski Jr, B.S.: PORs: proofs of retrievability for large files. In: Proceedings of the 2007 ACM Conference on Computer and Communications Security, pp. 584–597. ACM (2007)
18. Krupp, J., Schröder, D., Simkin, M., Fiore, D., Ateniese, G., Nuernberger, S.: Nearly optimal verifiable data streaming. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 417–445. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_16
19. Lai, R.W.F., Malavolta, G.: Subvector commitments with application to succinct arguments. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 530–560. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_19
20. Li, S., Zhang, Y., Xu, C., Chen, K.: Cryptoanalysis of an authenticated data structure scheme with public privacy-preserving auditing. IEEE Trans. Inf. Forensics Secur. **16**, 2564–2565 (2021)
21. Merkle, R.C.: Protocols for public key cryptosystems. In: Proceedings of the 1980 IEEE Symposium on Security and Privacy, pp. 122–134. IEEE Computer Society (1980)
22. Miao, M., Ma, J., Huang, X., Wang, Q.: Efficient verifiable databases with insertion/deletion operations from delegating polynomial functions. IEEE Trans. Inf. Forensics Secur. **13**(2), 511–520 (2018)

23. Papamanthou, C., Shi, E., Tamassia, R., Yi, K.: Streaming authenticated data structures. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 353–370. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_22
24. Schröder, D., Schröder, H.: Verifiable data streaming. In: 19th ACM Conference on Computer and Communications Security (CCS'12), pp. 953–964. ACM (2012)
25. Schöder, D., Simkin, M.: VeriStream – a framework for verifiable data streaming. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 548–566. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_34
26. Shacham, H., Waters, B.: Compact proofs of retrievability. J. Cryptol. **26**(3), 442–483 (2013). https://doi.org/10.1007/s00145-012-9129-2
27. Sun, Y., Liu, Q., Chen, X., Du, X.: An adaptive authenticated data structure with privacy-preserving for big data stream in cloud. IEEE Trans. Inf. Forensics Secur. **15**, 3295–3310 (2020)
28. Wang, F., Mickens, J., Zeldovich, N., Vaikuntanathan, V.: Sieve: cryptographically enforced access control for user data in untrusted clouds. In: 13th USENIX Symposium on Networked Systems Design and Implementation, pp. 611–626. USENIX Association (2016)
29. Wang, J., Chen, X., Huang, X., You, I., Xiang, Y.: Verifiable auditing for outsourced database in cloud computing. IEEE Trans. Comput. **64**(11), 3293–3303 (2015)
30. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling public auditability and data dynamics for storage security in cloud computing. IEEE Trans. Parallel Distrib. Syst. **22**(5), 847–859 (2011)
31. Xu, J., Meng, Q., Wu, J., Zheng, J.X., Zhang, X., Sharma, S.: Efficient and lightweight data streaming authentication in industrial control and automation systems. IEEE Trans. Ind. Inf. **17**(6), 4279–4287 (2021)
32. Xu, J., Wei, L., Zhang, Y., Wang, A., Zhou, F., Gao, C.: Dynamic fully homomorphic encryption-based Merkle tree for lightweight streaming authenticated data structures. J. Netw. Comput. Appl. **107**, 113–124 (2018)
33. Xue, K., Li, S., Hong, J., Xue, Y., Yu, N., Hong, P.: Two-cloud secure database for numeric-related SQL range queries with privacy preserving. IEEE Trans. Inf. Forensics Secur. **12**(7), 1596–1608 (2017)
34. Zhang, Y., Blanton, M.: Efficient dynamic provable possession of remote data via update trees. ACM Trans. Storage **12**(2), 9:1–9:45 (2016)
35. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: VSQL: verifying arbitrary SQL queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy, pp. 863–880. IEEE Computer Society (2017)
36. Zhang, Z., Chen, X., Ma, J., Tao, X.: New efficient constructions of verifiable data streaming with accountability. Ann. Telecommun. **74**(7–8), 483–499 (2019). https://doi.org/10.1007/s12243-018-0687-7