Annals of Telecommunications (2019) 74:483–499
https://doi.org/10.1007/s12243-018-0687-7

New efficient constructions of verifiable data streaming with accountability

Zhiwei Zhang[1] · **Xiaofeng Chen**[1] · **Jianfeng Ma**[1] · **Xiaoling Tao**[2]

Received: 6 June 2018 / Accepted: 22 November 2018 / Published online: 8 January 2019

Abstract

Data streaming is widely used in various environments. Resource-limited devices outsource the processing and storage of massive numbers of sequential elements to cloud-based servers, and security protection is of primary importance for the outsourced streams. The streaming authenticated data structure schemes and verifiable data streaming schemes are introduced to provide data owners and verifiers with the ability to verify streaming elements. However, due to their enormous numbers of key parameters, expensive updating overheads, signature revocation, and other security and application problems, few of the existing schemes are feasible when massive numbers of streaming elements are involved and allowed to be updated. In this paper, we define and construct a new primitive, namely, dimension-increasing vector commitment (DIVC). Then, we present the definition of constant verifiable data streaming (CVDS), which is an extension of the original verifiable data streaming (VDS) scheme. Moreover, with the proposed DIVC scheme, which is based on the CDH assumption in bilinear pairings, we construct two concrete CVDS schemes, namely, the probabilistic verifiability CVDS (P-CVDS) scheme and the deterministic verifiability CVDS (D-CVDS) scheme, by respectively employing the counting Bloom filter and a dynamic accumulator, which is based on the q-SDH assumption in bilinear pairings. The analyses prove that both the P-CVDS and D-CVDS schemes satisfy the security requirements that are formulated in the CVDS definition. Finally, the efficiency and performance evaluation demonstrate that the proposed schemes are feasible in practical applications.

Keywords Verifiable data streaming · Vector commitment · Counting Bloom filter · Dynamic accumulator · Cloud computing

1 Introduction

Nowadays, various types of data are growing explosively, among which a new class of data models, namely, *data streaming* [1], is particularly interesting due to its different characteristics from traditional data models. In a data-streaming model, the individual elements (or named data items) are generally assembled and can be treated as special relational tuples, e.g., $\langle timestamp, value\rangle$ and $\langle counter, value\rangle$, and these elements continuously arrive in multiple, rapid, time-varying, possibly unpredictable, and unbounded streams. There are many applications of data streaming in the fields of the Internet of Things, network traffic monitoring, call records, electronic commerce, online video, stock market, and so on. Consequently, many data-streaming management models, architectures, algorithms, and applications have been well studied in the past years [2–4].

In these previous works, high-performance computational, storing, and networking powers are necessary for handling the massive elements in data streaming, which goes beyond the capacities of the ubiquitous resource-limited devices and increases the cost of data center installation and maintenance. Then, cloud computing, which is a new conception of the computation model, was introduced. It overcomes the above-mentioned problems. Hence, increasingly, many data and related applications, including data streaming, are outsourced to cloud servers, which can offer unbounded resources as services in the pay-as-you-go model.

However, since the streaming elements are stored outside the owners' management scopes, the cloud servers are not all trusted by the users. The servers may tamper with (or even lose) the outsourced data records or return incorrect

✉ Zhiwei Zhang
zwzhang@xidian.edu.cn

Extended author information available on the last page of the article.

results of outsourced computational tasks. Therefore, methods that verify data correctness, completeness, and integrity are desired by cloud users (the data owner or verifiers) to ensure the security of the outsourced data streaming from the beginning. Since data storage is the main precondition for other research and applications, scholars have proposed the streaming authenticated data structures (SADS) [5, 6] and verifiable data streaming (VDS) [7–9] schemes for providing the users verifiability of outsourced data streaming in recent years. However, there are still various problems and challenges in the efficient and secure storage of the growing amount of streaming data.

First, the schemes that are based on tree structures usually originate from the Merkle hash tree (MHT) [10], which is intended to authenticate a set of n values $x_1, x_2, \cdots, x_n$. An MHT is constructed as a binary tree in which the leaves correspond to the hashes of the n values of the elements in the set. The value of each non-leaf node corresponds to the hash of the concatenation of its two children (maintaining their order). The tree root is signed using a public key signature scheme (e.g., RSA or DSA). An MHT can be used to securely and efficiently prove that an element (leaf) is in the set with the help of a verification object (VO). A VO is a collection of $\log(n)$ internal tree nodes that allow for the verifier to re-compute the root of the MHT, the signature of which can be verified. Therefore, these tree-based data streaming schemes suffer from heavy overheads (in proportion to the logarithm of the number of elements) of computation and communication in element appending, query, and verification.

The existing streaming authenticated data structure schemes and other symmetric cryptography-based secure data streaming schemes do not take element updating into account. Schröder and Schröder introduced element updating in the definition of verifiable data streaming [7] and the subsequent VDS schemes employ the chameleon hash as their building block, following the original VDS concrete scheme. However, as Chen et al. studied in [11], the traditional chameleon hash schemes suffer from the key exposure problem or the inefficiency of the certificate-based system model, which makes them insecure or infeasible to apply in practice. Hence, a data streaming verification scheme that supports efficient and secure updating of the outsourced elements is desired.

Furthermore, in addition to the chameleon hash, the primitive of vector commitment is a building block for the verifiable database and data-streaming schemes, which can commit to a fixed-size sequence of messages and later verify the correctness of an element's index and its related element according to the original committed sequence. However, as we have analyzed in [12], the number of public parameters of the vector commitment increases rapidly with the number of committed elements, which makes these VC-based schemes infeasible for use in practical storage applications, especially in data streaming environments. This is because, for a data streaming application, the number of elements is typically enormous, increases insistently, and cannot be predetermined. Therefore, the vector commitment should be modified appropriately when it is employed to construct verifiable data streaming schemes.

Last, there is no definition of accountability in the security requirements of the available verifiable data streaming studies. The accountability of an outsourced database is introduced in [13], which emphasizes that the client can provide proof if the dishonest server has tampered with the database. However, the existing verifiable data streaming schemes can only offer the verifiers the proof that is provided by the server or client according to the streaming elements at the current state; neither the server nor the client can deny or prove the verification result when they disagree with it because the faults can come from both of them. Consequently, verifiable data streaming schemes should enable the verifiers to judge the server or client who is responsible for an unsuccessful verification result.

1.1 Our contributions

In this paper, we focus on the following problems in the existing secure data-streaming schemes: the complexity of tree structure, the public parameter explosion of vector commitment, and the lack of accountability. Our main contributions are listed as follows:

- We modify the original definition of vector commitment and propose a new primitive: dimension-increasing vector commitment (DIVC for short). Moreover, we present a concrete DIVC scheme that is based on the CDH assumption in bilinear pairings. In DIVC, each newly arrived streaming element is committed chronologically along with the previous ones within its cell, and the commitment of a cell cannot be determined until its number of elements reaches the upper bound of the cell capacity. In addition, these cells share the same set of key parameters, which directly reduce the number of parameters of the entire data streaming process and avoids the use of tree-based structures.
- We extend the definition of verifiable data streaming and present the notion of constant verifiable data streaming with updates (CVDS for short) by introducing two additional security requirements: *efficiency* and *accountability*. The efficiency requirement emphasizes that the operational complexities of the client and the verifier should be independent of the number of outsourced elements, and the accountability requirement provides the verifier with the ability to ascertain responsibility for a failing verification result.

– We construct two concrete CVDS schemes, namely, P-CVDS and D-CVDS, based on DIVC by employing the counting Bloom filter and the dynamic accumulator in bilinear pairings, respectively. The DIVC can be used to verify the integrity of a completed cell, while the Bloom filter and the accumulator ensure that all the cell commitments are correct. We analyze both of our schemes and prove that they satisfy the security requirements in the definition of CVDS, and the results of the experimental evaluation demonstrate that both schemes are efficient and applicable.

1.2 Related work

Cloud computing delivers scalable and elastic computation, storage and networking resources as services through the Internet in a pay-as-you-go model. It enables users with resource-limited devices to outsource heavy data processing tasks to cloud-based servers without purchasing and building their own IT infrastructures. However, the price is that the users must give their data to the cloud, which may result in various security and privacy problems [14–18]. Verifiable computing has been well studied by many researchers in the recent years and plenty of concrete schemes have been presented [19, 20], of which verifiable outsourcing storage is an interesting branch [21–23]. For example, the primitive definition and schemes of verifiable database (VDB) [13, 24, 25] provide the cloud storage with security guarantees, including authenticity, integrality, and public/private verifiability.

Data streaming can be treated as a special type of database, whose elements are typically massive and their number cannot be predetermined [1]. However, this causes the existing secure (verifiable) database technologies to be infeasible for streaming environments. Therefore, many secure data streaming schemes have been proposed.

Schröder and Schröder proposed the first definition of verifiable data string (VDS) for supporting publicly verifiability and element updating and presented an efficient instantiation that was based on a newly designed data structure, namely, chameleon authentication tree (CAT), in [7]. To reduce the size of the proof and the client state and improve the time/space complexities of the appending, updating, and verification algorithms, Schröder and Simkin presented another CAT-based framework for verifiable data streaming and two novel schemes: fully dynamic CAT and δ-bounded CAT in [8]. To make the client more efficient, Krupp et al. proposed two new VDS schemes in [9]: Their first scheme is constructed based on their newly presented definition of chameleon vector commitment (CVC), and the second scheme shifts the workload to the server by combining signature schemes with cryptographic accumulators. However, these schemes suffer from the following problems. First, like all the previous chameleon-hash-based schemes, the CVC-based VDS scheme introduces many additional random numbers, which increases the storage cost. Second, although the scheme based on a signature and an accumulator achieves nearly constant time and space complexities for the client, it cannot fully satisfy the requirements of VDS, e.g., it can only verify the signal element's correctness, instead of that of the whole data stream. Moreover, this signature-accumulator scheme appends a random number after each element, which increases the storage and communication costs.

Furthermore, since the hash tree scheme was introduced by Merkle [26, 27] for implementing a static authenticated dictionary, many other authenticated data structures [28, 29] have been proposed for enabling an untrusted *responder* to answer queries on a data structure on behalf of a trusted *source* and provide proof of the validity of the answer to the *user*. The model of streaming authenticated data structures (SADS) was first proposed by Papamanthou et al. in [5], whose building block is a new authentication tree, namely, the generalized hash tree. The generalized hash tree was instantiated with a hash function based on lattice assumptions. However, as pointed out by Qian et al. in [6], this SADS scheme is impractical to implement for real-world applications because of its use of Ajtai's collision–resistant hash function [30], which is based on the small integer problem and has a very large public key size. Then, they presented an abstract SADS construction that can use any hash function that satisfies the properties that are formally defined in [6], which leads to simpler exposition of the fundamental ideas of Papamanthou et al.'s work and a practical implementation of a streaming authenticated data structure that employs the efficient SWIFFT hash function. Nevertheless, these streaming authenticated data-structure-based schemes do not take element updating into account, and the streaming elements cannot be updated efficiently once they have been outsourced.

Many secure data streaming schemes have been proposed based on symmetric cryptographic algorithms. In [31], Do and Song designed an access privilege sharing management technique that enables a safe streaming media service by using AONT-based XOR threshold secret sharing on the basis of cloud computing. Puthal et al. studied how to ensure end-to-end security (integrity, authenticity, and so on) of the data stream for risk-critical applications in [32]. They proposed a Dynamic-Key-Length-Based Security Framework (DLSeF) that was based on a shared key that was derived from synchronized prime numbers. However, due to the characteristics of symmetric cryptography, one of the serious issues for the shared-key and symmetric-key schemes is large-scale key management among massive

numbers of users; additionally, it is difficult for these schemes to support public verifiability. There are also many new secure data streaming schemes that are constructed based on public key algorithms. For example, in [33], Chen et al. proposed a novel data structure, namely, the arithmetic Merkle tree (AMT), and integrated it with partially homomorphic encryption and fully homomorphic encryption algorithms to address the problem of integrity protection for data streaming in a cloud-assisted healthcare sensor system. In [34], Sun et al. presented AC-MTAT, which is a dynamic authenticated data structure with access control on an outsourced data stream by employing the trapdoor hash and CP-ABE [35] technologies, based on which a novel authentication scheme was established, which supports addition to and updating of the data stream in real time and verification with fine-grained access control. However, these cryptographic algorithms are infeasible in practice because of their inefficiencies.

1.3 Organization

The remainder of this paper is organized as follows: Preliminaries are presented in Section 2. In Section 3, we describe the notion and construction of dimension-increasing vector commitment (DIVC) from the original vector commitment. Then, DIVC is employed as one of the building blocks for the newly proposed constant verifiable data streaming (CVDS) in Section 4. Moreover, two concrete schemes, together with their security and efficiency analyses, are presented in Section 4. The performance evaluations based on the experimental results of the proposed CVDS schemes are presented in Section 5. Finally, Section 6 presents the conclusions of the paper and discusses future work.

2 Preliminaries

2.1 Hardness assumptions in bilinear pairings

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic multiplicative groups of prime order p, and let g be a randomly chosen generator of $\mathbb{G}$. A bilinear pairing is a map $e : \mathbb{G}\times\mathbb{G} \rightarrow \mathbb{G}_T$ with the following properties:

- *Bilinear*: For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p^*$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
- *Non-degenerate*: $e(g, g) \neq 1$.
- *Computable*: For all $u, v \in \mathbb{G}$, there is an efficient algorithm for computing $e(u, v)$.

Before describing the related schemes, we present two main hardness assumptions on which they are based.

The CDH problem in $\mathbb{G}$ is defined as follows:

Definition 1 (Computational Diffie-Hellman (CDH) Problem) Let $\mathbb{G}$ be a cyclic group of prime order p and g a generator of $\mathbb{G}$. Given a triple (g, g^a, g^b) for any random $a, b \in_R \mathbb{Z}_p$ as inputs, compute and output g^{ab}.

The *CDH assumption* holds in $\mathbb{G}$ if for every PPT algorithm $\mathcal{A}$, there exists a negligible function $\mathbf{negl}(\cdot)$ such that

$$\mathbf{Pr}[\mathcal{A}(1^k, g, g^a, g^b) = g^{ab}] \leq \mathbf{negl}(k)$$

for every security parameter k.

The q-strong DH problem in $\mathbb{G}$ and $\mathbb{G}_T$ is defined as follows [36]:

Definition 2 (q-Strong Diffie-Hellman (q-SDH) Problem) Let $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a tuple that is generated by a bilinear instance generator, and let r be randomly chosen in $\mathbb{Z}_p^*$. Given a tuple $(p, G, G_T, e, g, g^r, g^{r^2}, \cdots, g^{r^q})$ for any $x \in_R \mathbb{Z}_p$ as inputs, compute and output a pair $(c, g^{\frac{1}{r+c}}) \in \mathbb{Z}_p^* \times \mathbb{G}$.

The *q-SDH assumption* holds in $(\mathbb{G}, \mathbb{G}_T)$ if for every PPT algorithm $\mathcal{A}$, there exists a negligible function $\mathbf{negl}(\cdot)$ such that

$$\mathbf{Pr}[\mathcal{A}(p, G, G_T, e, g, g^r, g^{r^2}, \cdots, g^{r^q}) = (c, g^{\frac{1}{r+c}})] \leq \mathbf{negl}(k),$$

where $(c, g^{\frac{1}{r+c}}) \in \mathbb{Z}_p^* \times \mathbb{G}$ for every security parameter k.

2.2 Verifiable data streaming

Schröder and Schröder formally defined the term verifiable data streaming (VDS) in [7].

Definition 3 A verifiable data streaming protocol $\mathcal{VDS} =$ (Setup, Append, Query, Verify, Update) is a protocol between two PPT algorithms: a client $\mathcal{C}$ and a server $\mathcal{S}$. The server can store an exponential number of elements in its database DB and the client maintains a small state $\mathcal{O}(\log n)$. The scheme consists of the following PPT algorithms:

- **Setup**(1^λ): The setup algorithm takes as input the security parameter 1^λ. It returns a verification key PK and a secret key SK. The verification key PK is given to the server $\mathcal{S}$, the secret key is given to the client $\mathcal{C}$, and SK contains PK.
- **Append**(SK, s): This algorithm appends the value s to the database DB that is maintained by the server. The client sends a single message to the server, which stores the element in DB. Adding elements to the database may change the private key to SK', but it does not change the verification key PK.

– **Query**(PK, DB, i): The interactive query protocol is defined as $\langle\ \mathcal{S}(PK, DB),\ \mathcal{C}(i)\rangle$ and is executed between $\mathcal{S}(PK, DB)$ and $\mathcal{C}(i)$. At the end of the protocol, the client either outputs the ith entry $s[i]$ of DB together with a proof $\pi_{s[i]}$, or $\perp$.
– **Verify**$(PK, i, s, \pi_{s[i]})$: The verification algorithm outputs $s[i]$ if $s[i]$ is the i-th element in the database DB; otherwise, it returns $\perp$.
– **Update**(PK, DB, SK, i, s'): The interactive update protocol $\langle\mathcal{S}\ (PK,\ DB),\ \mathcal{C}(SK,\ i,\ s')\rangle$ is implemented between the server $\mathcal{S}(PK,\ DB)$ and the client $\mathcal{C}(SK,\ i,\ s')$ who wishes to update the ith entry of the database DB to s'. At the end of the protocol, the server sets $s[i] \leftarrow s'$ and both parties update PK and PK'.

A verifiable data streaming protocol must fulfill the requirements of security, efficiency and completeness.

3 Dimension-increasing vector commitment

The original vector commitment scheme is designed to commit a fixed-length sequence of messages, all of which are available to be committed. However, in data stream environments, elements arrive ceaselessly and unpredictably, which means that the number of elements cannot be determined until the data stream terminates. Unfortunately, the streaming elements usually need to be processed immediately, and the length of a data stream is typically too large for its owner to buffer and process it in batch mode. Moreover, even if these problems could be overcome, as we have analyzed in [12], with the increase in the number of stream elements, the number of public key parameters in the VC scheme will be cumbersome and unacceptable for practical applications. The original vector commitment definition and schemes are infeasible for data streams. Therefore, in this section, to satisfy the characteristics and requirements of data streams, we modify the original definition of vector commitment and construct a concrete scheme that is based on the CDH assumption in bilinear parings.

3.1 Definition of dimension-increasing vector commitment

In this section, we propose a new notion, namely, dimension-increasing vector commitment (DIVC), by adjusting Catalano and Fiore's definition of VC in [25] to satisfy the data streaming application requirements. As shown in Fig. 1, the client $\mathcal{C}$ updates the local commitment when a new element is outsourced and appends two additional values (the cell commitment and the commitment signature) to the end of the element cell once the counter reaches a multiple of the cell size s. Accordingly, the server $\mathcal{S}$ stores these outsourced streaming elements and other messages from the client $\mathcal{C}$ in their order of arrival (or reorders them according to their indices), and they are split into fixed-sized cells by the cell commitments and the commitment signatures. The caching cell is the current cell whose number of elements is less than s, and it is called a full cell when the cell size reaches s but has not received the cell commitment and signature. Once the full cell is completed, it is moved from the caching storage to the complete storage. In every set of s elements, their vector commitment and signature is called a completed cell, and the collection of all completed cells in server $\mathcal{S}$ is called the completed storage. The formal definition of DIVC is as follows.

Definition 4 The dimension-increasing vector commitment is a non-interactive primitive and consists of the following polynomial-time algorithms:

– **DIVC.KeyGen**$(1^k, s, cnt)$. Taking as inputs the security parameter k, the cell size s (where $s = poly(k)$) and the global counter cnt, this key generation algorithm outputs public parameters PP and private secret parameters SP, which implicitly indicate the available message space $\mathcal{M}$ that can be committed, and initializes

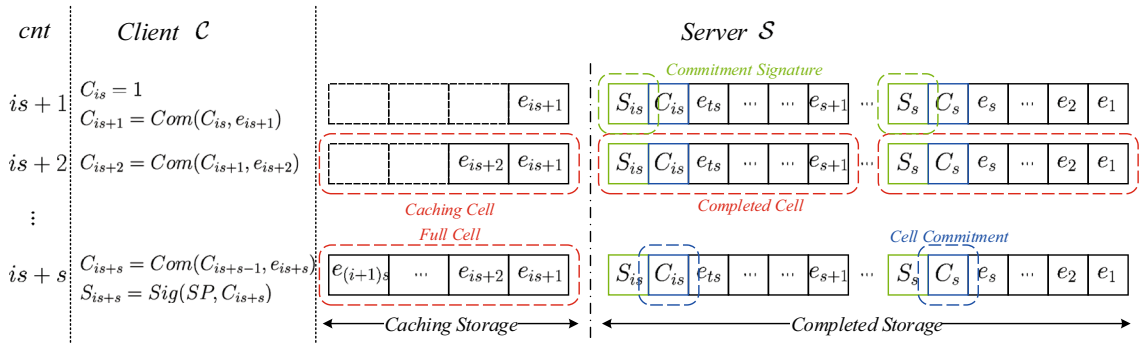


Fig. 1 Definition of DIVC

cnt. The parameters in PP are publicly accessible to the client (data owner) $\mathcal{C}$, the server $\mathcal{S}$ and the verifier $\mathcal{V}$, while the private SP is only accessible to the client $\mathcal{C}$.

- **DIVC.Com**$(PP, SP, cnt, e, s, C_{cnt-1})$. Taking as inputs the public and private parameters PP and SP, the global counter cnt, a new streaming element e (where $e \in \mathcal{M}$) and the previous state commitment value C_{cnt-1}, this commitment algorithm increases the counter by 1, appends the new element e to the data streaming storage server, and updates the caching window commitment from C_{cnt-1} to C_{cnt}. If the current caching cell is full, which means that $cnt \mod s = 0$, this algorithm appends C_{cnt} as the cell commitment to the server with its signature S_{cnt}.[1] C_{cnt} and S_{cnt} may be included in the auxiliary information aux. Then, C_{cnt} is set to 1 (the identity element). This algorithm finally outputs the current state commitment value C_{cnt} and the auxiliary information aux. Note that different from the original vector commitment definition, this commitment algorithm is designed to be executed in a loop-calling manner.
- **DIVC.Open**$(PP, e, i, s, \text{aux})$. Taking as inputs the public parameters PP, an element e and its index i in the stored stream, the cell size s, and the auxiliary information aux, with $t := i \mod s$ (if $t = 0$, set $t := s$), this opening algorithm outputs a proof Λ_i that demonstrates that e is the tth committed element within a completed cell. For simplicity, in the remainder of this paper, we only assume that the ith element belongs to a completed cell, and the verification of the caching storage, which can be accomplished separately, is omitted.
- **DIVC.Ver**$(PP, C_i, e, i, s, \Lambda_i)$. Taking as inputs the public parameters PP, the cell commitment value C_i of the element e with index i, the cell size s, and the corresponding proof Λ_i, this verification algorithm outputs 1 or an error $\perp$. Specifically, let $t := i \mod s$ (if $t = 0$, set $t := s$). This algorithm accepts the proof and outputs 1 only when Λ_i can be verified as a valid proof that the tth element is equal to e in a completed cell whose commitment value is C_i; otherwise, this algorithm rejects the proof and outputs an error $\perp$.
- **DIVC.Update**$(PP, SP, C_i, e, i, s, e')$. Taking as inputs the public and private parameters PP and SP, the original cell commitment value C_i, the original element e with index i, the cell size s, and the new element $e' \in \mathcal{M}$ for substituting e, this commitment-updating algorithm outputs a new commitment value $C_i{}'$ with its new signature $S_i{}'$ and some update information denoted by U. This algorithm is run by the original committer (e.g., the data streaming owner) to update the cell commitment C_i when the ith element e is changed to e'.
- **DIVC.ProofUpdate**$(PP, C_i, \Lambda_j, U, s)$. Taking as inputs the public parameters PP, the original cell commitment value C_i, the original proof Λ_j, the updated information U, the updated message e' at the position i in the data stream, and the cell size s, this proof-updating algorithm outputs a new proof Λ_j' and the updated commitment $C_i{}'$. This algorithm can be run by any user who holds an original proof Λ_j for the element at position j relative to the original commitment C_i, when the ith element is updated to e'.

3.2 Construction of DIVC based on the CDH assumption

According to the definition of DIVC, we propose a concrete DIVC scheme that is based on the CDH assumption in bilinear groups, as follows:

- **DIVC.KeyGen**$(1^k, s, cnt)$. Let $\mathbb{G}$, $\mathbb{G}_T$ be two bilinear groups of prime order p that are equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Randomly choose a generator g of $\mathbb{G}$ and $\{z_1, \cdots, z_s\} \in \{\mathbb{Z}_p\}^s$. For all $i \in [1, s]$, set $h_i := g^{z_i}$. For all $i, j \in [1, s]$ and $i \neq j$, set $h_{i,j} := g^{z_i z_j}$. Set the global counter to $cnt := 0$, the public parameter tuple to $PP := \{spk, g, \{h_i\}_{i\in[1,s]}, \{h_{i,j}\}_{i,j\in[1,s],i\neq j}\}$, and the private parameter tuple to $SP := \{ssk\}$, where spk and ssk are the verification key and signature key of a cryptographic digital signature scheme (e.g., the RSA scheme, which is employed here). Compute the initial previous state commitment value $C_0 := 1$, where 1 is the identity element in $\mathbb{G}$. The message space is $\mathcal{M} = \mathbb{Z}_p$.
- **DIVC.Com**$(PP, SP, cnt, e, s, C_{cnt-1})$. Set $cnt := cnt + 1$, let $t := cnt \mod s$ (if $t = 0$, set $t := s$), compute

$$C_{cnt} := C_{cnt-1} \cdot h_t^e = h_1^{e_{(cnt-t)+1}} \cdot h_2^{e_{(cnt-t)+2}} \cdot \cdots \cdot h_t^{e_{(cnt-t)+t}},$$

  and update the auxiliary information $aux := aux \cup \{e\}$. Additionally, if $t = s$, which means that this element is the last element in the current caching cell and the cell is full, then set $aux := aux \cup \{C_{cnt}, S_{cnt}\}$, where $S_{cnt} := RSA(ssk, C_{cnt})$, and reset $C_{cnt} := 1$ for the next caching cell. Finally, output C_{cnt}, S_{cnt} and aux. The commitment value is only related to the elements within its cell, not the entire stream.
- **DIVC.Open**$(PP, e, i, s, \text{aux})$. To find the indices m and n of the first element and the last element in the cell that includes the ith element, let $t := i \mod s$ (if

[1] When we want to refer to the cell commitment value of a completed cell, for simplicity, in this context, we may employ any index that falls into the same cell, rather than only using the last index. In other words, for a completed cell, only the cell commitment value and its signature are stored in the server.

$t = 0$, set $t := s$), and compute $m := i - t + 1$ and $n := i - t + s$. Then, compute the proof

$$\Lambda_i := \prod_{j=m, j\neq i}^{n} h_{i-m+1, j-m+1}^{e_j} = \left(\prod_{j=m, j\neq i}^{n} h_{j-m+1}^{e_j} \right)^{z_{i-m+1}}.$$

At the end of this algorithm, it outputs (Λ_i, C_n, S_n), where C_n and S_n can be found in the auxiliary information aux. Similar to the commitment algorithm, the proof of the ith element only depends on its own cell.

– **DIVC.Ver**$(PP, C_i, e, s, i, \Lambda_i)$. This verification algorithm first determines whether the signature is valid or not by checking whether the following equation holds:

$$C_n \stackrel{?}{=} RSA(spk, S_n). \quad (1)$$

If it does not hold, this verification algorithm is aborted and outputs 0. Otherwise, let $t := i \mod s$ (if $t = 0$, set $t := s$), and check whether the following equation holds:

$$e(C_i / h_t^e, h_t) \stackrel{?}{=} e(\Lambda_i, g). \quad (2)$$

If the above equation holds, this verification algorithm outputs 1; otherwise, it outputs 0. As described above, the commitment with its signature and proof value are based on the cell; the verification result only indicates whether the location within the cell is correct, which is a relative location, and this algorithm performs a local verification of the data stream.

– **DIVC.Update**$(PP, SP, C_i, e, i, s, e')$. To find the indices m and n of the first element and the last element in the cell that includes the ith element, let $t := i \mod s$ (if $t = 0$, set $t := s$), and compute $m := i - t + 1$ and $n := i - t + s$. Then, update the cell commitment and its signature as follows:

$$C_n' := C_i \cdot h_{i-m+1}^{e'-e},$$
$$S_n' := RSA(ssk, C_n').$$

Finally, this updating algorithm outputs the updated commitment C_n', the new signature S_n' of this cell, and the update information tuple $U = (e, e', i)$. The auxiliary information should be updated correspondingly, and it is recommended to verify the cell before it is updated.

– **DIVC.ProofUpdate**$(PP, C_i, \Lambda_j, U, s)$. A client or a verifier who has a proof Λ_j, which is valid with regard to C_i for an element at position j, can use the update information $U = (e, e', i)$ to compute the updated commitment C_i' and produce a new proof Λ_j', which will be valid with regard to C_i'. This proof-updating algorithm is executed under the following two cases:

  – $i = j$. The proof Λ_j remains the same.

  – $i \neq j$. Update the proof with the following equation:

$$\Lambda_j' := \Lambda_j \cdot \left(h_t^{e'-e} \right)^{z_t} = \Lambda_j \cdot h_{j-m+1, i-m+1}^{e'-e},$$

where m and t are computed in the same way as in the algorithms that are described above.

4 Constant verifiable data streaming with updates

The proposed DIVC can only provide the verification of one cell and cannot verify the entire data stream. Therefore, it is not sufficient to directly construct a secure data streaming scheme based on DIVC alone. Consequently, in this section, we present a new data streaming verification protocol, which is named constant verifiable data streaming (CVDS for short), by extending the original definition of VDS in [7]. Furthermore, based on the previous DIVC scheme, we propose two concrete CVDS schemes with the other two cryptographic primitives: the counting Bloom filter and the dynamic accumulator. The two CVDS schemes are named P-CVDS and D-CVDS for the characteristics of their different underlying technologies. Moreover, the security proofs and efficiency analyses of P-CVDS and D-CVDS are presented.

4.1 Formal definition of CVDS

The formal definition of CVDS is as follows:

Definition 5 A constant verifiable data streaming $\mathcal{SVDS}$ scheme is a protocol that consists of five PPT algorithms (Setup, Append, Query, Verify, and Update) and involves three participants: a client $\mathcal{C}$, a server $\mathcal{S}$, and a verifier $\mathcal{V}$.

– **CVDS.Setup**$(1^\lambda, s, cnt)$: Taking as inputs the security parameter 1^λ, the cell size s and the global counter cnt, this setup algorithm initializes the system parameters (e.g., the counter cnt) and outputs a public key tuple PK and a secret key tuple SK. The public key PK is available for every user in the system who is allowed to access the outsourced streaming elements, while the secret key SK should be only accessible to the client $\mathcal{C}$ who is the owner of these elements. For clarity, in this paper, we use the counter cnt to label an element in the stream, which is a common practice in many data streaming applications. The streaming elements can also be labeled with timestamps in other applications.

– **CVDS.Append**(SK, e): Taking as inputs the secret key SK and a new element e, this appending algorithm adds e to the server's coded streaming element record set DS with the index cnt after the necessary processing

and outputs the updated data streaming record set DS' and auxiliary information aux', which could include the total number of elements, static proof material, and other information. Note that adding elements should not change the public key PK.

- **CVDS.Query**(PK, DS, i, aux): Taking as inputs the public key PK, the coded record set DS of the outsourced data streaming elements, the index i of the requested element, and the auxiliary information aux, this query algorithm outputs the element e_i and a proof π_i that proves e_i is the ith streamed element in DS. The proof consists of two parts, which are provided by the server $\mathcal{S}$ and the client $\mathcal{C}$ independently and can be denoted as the server proof π_{si} and the client proof π_{ci},[2] or this algorithm outputs a special error symbol $\perp$.
- **CVDS.Verify**(PK, i, e, π_i, aux): Taking as inputs the public key PK, the index i of the streaming element e, the proof π_i, and the auxiliary information aux, this verification algorithm outputs the element value e directly if it can be verified that e is the ith element in the outsourced data stream DS according to the proof π_i; otherwise, it returns a special error symbol $\perp$ and could further clarify whether the server or the client should take the responsibility for this failure under the requirement of accountability.
- **CVDS.Update**$(PK, SK, DS, i, e, e', aux)$: Taking as inputs the public key PK and the secret key SK, the encoded streaming element record set DS, the index i of the element e, which is updated to e', and the auxiliary information aux, this updating algorithm modifies the value of the ith element of the outsourced data stream that is stored in the server from e to e'. It also updates the related parameters in the auxiliary information aux.

4.2 Security requirements of CVDS

A constant verifiable data streaming scheme should satisfy the following requirements of *security*, *correctness*, *efficiency*, and *accountability*.

The first requirement of a CVDS scheme is security. That is, without the secret key, the adversary (e.g., a malicious server) cannot forge an invalid tuple of the element index, value, and proof (i^*, e_i^*, π_i^*) that can be verified successfully.

[2]The purpose of splitting the proof is to support the security requirement of accountability; the core idea comes from the basic concept of verifiable outsourcing computation, which can be found in related works, such as [37, 38]. To reduce the client's workload, the client proof π_{ci} can be released by a trusted agent or proxy instead of the client. In addition, in data streaming environments, since the client is always online until the data streaming has finished, there is no need to distinguish between when the client is online or offline in related algorithms.

Definition 6 (Security) A constant verifiable data streaming scheme is said to be secure if for any data stream $DS \in [M] \times \{0,1\}^*$, where $M = poly(\lambda)$ is the upper bound on the number of streaming elements, and for any probabilistic polynomial time (PPT) algorithm adversary $\mathcal{A}$, there exists a negligible function **negl**$(\cdot)$ such that

$$\mathbf{Pr}[\mathcal{A}(1^\lambda, PK, DS, aux) \rightarrow (i^*, e_i^*, \pi_i^*) : \text{Verify}(PK, i^*, e_i^*, \pi_i^*, aux) \neq \perp, (i^*, e_i^*) \notin DS] \leq \text{negl}(\lambda)$$

for every security parameter λ.

The second requirement of a CVDS scheme is correctness. That is, the element value and the proof in a query result that is generated by an honest server should be always verified successfully.

Definition 7 (Correctness) A constant verifiable data streaming scheme is said to be correct if for any data stream $DS \in [M] \times \{0,1\}^*$, where $M = poly(\lambda)$ is the upper bound on the number of streaming elements, and for any valid query result (e_i, π_i) that is generated by an honest server, the output of the verification algorithm is always the element value e_i.

The above two requirements are almost the same as the definitions in previous works, such as [5, 7]. The third requirement of a CVDS scheme is efficiency, which differs from the existing schemes in the required performance level. That is, the client and the verifier in the CVDS scheme cannot be involved in expensive computation and storage, and all their operations' time and space complexities should be constant.

Definition 8 (Efficiency) A constant verifiable data streaming scheme is said to be efficient if for any data stream $DS \in [M] \times \{0,1\}^*$, where $M = poly(\lambda)$ is the upper bound on the number of streaming elements, the computation and storage resources that are invested by the client, and the verifier must be independent of M and N, where N is the number of streaming elements that have already been outsourced to the server, and their time and space complexities are $\mathcal{O}(1)$.

The final requirement of a CVDS scheme is accountability, which is newly introduced in this paper. That is, when a verification is unsuccessful, the verifier can tell who, namely, the client or the server, should take responsibility. Unlike other available secure data streaming and database schemes, we assume that both the server and client may make mistakes in their processing, rather than only taking the server into account.

Definition 9 (Accountability) A constant verifiable data streaming scheme is said to be accountable if for any data stream $DS \in [M] \times \{0,1\}^*$, where $M = poly(\lambda)$ is the upper bound on the number of streaming elements, when the query result is not verified successfully, the verifier can judge who is responsible for the failure.

4.3 Probabilistic verifiability CVDS scheme based on the Bloom filter

According to the definition of CVDS, in this subsection, we construct a concrete CVDS scheme that is based on the primitive of the counting Bloom filter (CBF). A Bloom filter (BF) [39] is a data structure optimized for fast, space-efficient set membership tests. Bloom filters have the unusual property of requiring constant time to add an element to the set or test for membership, regardless of the size of the elements or the number of elements already in the set. No other constant-space set data structure has this property. Furthermore, a counting Bloom filter (CBF) [40] generalizes a Bloom filter data structure so as to allow membership queries on a set that can be changing dynamically via insertions and deletions [41]. For the false positives of the CBF schemes, we name our first CVDS scheme probabilistic verifiability CVDS (P-CVDS for short). The details of the P-CVDS scheme are as follows.

– **CVDS.Setup**$(1^\lambda, s, cnt)$: Let the acceptable error rate be ϵ and set the number of hash functions that are needed to $k := \ln 2 \cdot (m/n)$, where $m := n \log_2 e \cdot \log_2(1/\epsilon)$ is the number of counters and n is the upper bound on the number of messages that can be filtered.[3] For $i = 1, \cdots, m$, set the ith counter to $Counter[i] := 0$. For $j = 1, \cdots, k$, initialize the jth independent hash function $\mathcal{H}_j(\cdot)$, and all k cryptographic hash functions take as input any message $u \in \{0,1\}^*$ and output a value $v \in \{1, \cdots, m\}$. Then, this algorithm invokes **DIVC.KeyGen**$(\cdot)$ to initialize the dimension-increasing vector commitment scheme. Finally, this algorithm sets the public key to

$$PK := \big\{s,\ PP = \{spk, g, \{h_i\}_{i\in[1,s]}, \{h_{i,j}\}_{i,j\in[1,s], i\neq j}\}, \{\mathcal{H}_i\}_{i\in[1,k]}\big\},$$

sets the secret key to

$$SK := \big\{SP = \{ssk\}, \{z_i\}_{i\in[1,s]}\big\},$$

and sets the auxiliary information to

$$aux := \{cnt, DIVC.aux\}.$$

[3] In this scheme, n is equal to the maximum number of cells in the data stream, which means there could be $n \times s$ stream elements.

– **CVDS.Append**(SK, e): When a new element e is streamed, the client invokes the **DIVC.Com**$(\cdot)$ algorithm to add it into the server's DS. If the appended element is the last one in its cell ($cnt - 1 \mod s = 0$), the client adds the cell commitment C_{cnt-1} into the CBF as follows. For $i = 1, \cdots, k$, compute the hash value v_i of $(cnt - 1 \| e)$ with the ith hash function $\mathcal{H}_i(\cdot)$, that is,

$$v_i := \mathcal{H}_i(cnt - 1 \| C_{cnt-1}),$$

and set the v_i-th counter to

$$Counter[v_i] := Counter[v_i] + 1.$$

– **CVDS.Query**(PK, DS, i, aux): This query algorithm first invokes **DIVC.Open**$(\cdot)$, which is the opening algorithm in the DIVC scheme, to obtain the ith element's cell commitment proof pair (Λ_i, C_i, S_i). Then, the server recomputes the Bloom filter value with the cell commitments as follows. For $x \in [1, \lfloor cnt/s \rfloor]$ and $y \in [1, k]$,

$$\begin{aligned} v_i &:= \mathcal{H}_y(x \| C_{xs}), \\ SCounter[v_i] &:= SCounter[v_i] + 1, \end{aligned}$$

where definitions of the server counters $\{SCounter[x]\}_{x\in[1,m]}$ are the same as those of the client counters $\{Counter[x]\}_{x\in[1,m]}$, which are initialized to 0. Next, let $\alpha := \lceil i/s \rceil$. The server employs another collision-resistant cryptographic hash function, namely, $Hash(\cdot)$, to compute

$$H_{\mathcal{S}} := Hash\left(\{SCounter[\mathcal{H}_x(\alpha s \| C_{\alpha s})]\}_{x\in[1,k]}\right).$$

Then, the server constructs the server proof

$$\pi s_i := \{\Lambda_i, C_i, S_i, H_{\mathcal{S}}\}.$$

Meanwhile, the client provides the current counter values for constructing the client proof

$$\pi c_i := \{V_{\mathcal{C}}, H_{\mathcal{C}}\},$$

where $H_{\mathcal{C}} = Hash(V_{\mathcal{C}})$ and $V_{\mathcal{C}} = \{Counter[H_x(\alpha s \| C_i)]\}_{x\in[1,k]}$. Finally, this query algorithm outputs the query result (e, π_i), where the proof

$$\pi_i := \{\pi s_i, \pi c_i\}.$$

– **CVDS.Verify**(PK, i, e, π_i, aux): The verifier parses the proof π_i as $\{\pi s_i, \pi c_i\}$, which can be further treated as $\{\{\Lambda_i, C_i, S_i, H_{\mathcal{S}}\}, \{V_{\mathcal{C}}, H_{\mathcal{C}}\}\}$. Then, the verification is performed via the following three steps: First, verify that C_i is valid for the current CBF: for $x = 1, \cdots, k$, if all $V_{\mathcal{C}}[x] \neq 0$, this step of verification of C_i is passed; otherwise, this algorithm is aborted and returns an error

$\perp$.[4] Second, check whether $H_{\mathcal{S}}$ is equal to $H_{\mathcal{C}}$. If they are equal, the verification process continues; otherwise, this algorithm is aborted and returns an error $\perp$. Third, verify that the signature S_i is correct according to the given cell commitment C_i and Λ_i is valid according to the cell commitment C_i by invoking **DIVC.Ver**$(\cdot)$, which is the verification algorithm in the DIVC scheme. If the result of **DIVC.Ver**$(\cdot)$ is 1, this algorithm returns the element value e as its result; otherwise, it returns an error $\perp$. When this verification algorithm outputs $\perp$, to realize the requirement of accountability, the verifier could perform other operations, which we will describe in the following proof.

– **CVDS.Update**$(PK, SK, DS, i, e, e', aux)$: To update the ith element e to e', two relevant parts should be recomputed: the $\lceil i/s \rceil$th cell commitment, which is stored by the server (whose value can be denoted by any element within it, e.g., C_i, as described in the DIVC scheme construction), and the CBF counters, which are maintained by the client (the original data owner). The cell commitment value can be updated by invoking the **DIVC.Update**$(\cdot)$ algorithm, which updates the commitment value from C_i to C_i' and updates its signature from S_i to S_i'. The CBF counters can be modified via the following steps: Set $\alpha := \lceil i/s \rceil$, remove the value $(\alpha s \| C_{\alpha s})$ from CBF, and add the value $(\alpha s \| C'_{\alpha s})$ to CBF, where $C'_{\alpha s}$ is equal to C_i', which is one of the output parameters from the **DIVC.Update**$(\cdot)$ algorithm. For $x = 1, \cdots, k$, update the counters as follows.

$$Counter[\mathcal{H}_x(\alpha s \| C_{\alpha s})] := Counter[\mathcal{H}_x(\alpha s \| C_{\alpha s})] - 1,$$
$$Counter[\mathcal{H}_x(\alpha s \| C'_{\alpha s})] := Counter[\mathcal{H}_x(\alpha s \| C'_{\alpha s})] + 1.$$

4.3.1 Security analysis of the P-CVDS scheme

We prove that the proposed probabilistic verifiability CVDS scheme (P-CVDS) satisfies the requirements of security, correctness, efficiency and accountability, as follows.

Theorem 1 *The proposed P-CVDS scheme is secure.*

Proof If a PPT adversary $\mathcal{A}$ wants to pass the verification algorithm of the P-CVDS scheme with an invalid tuple (i^*, e_i^*, π_i^*), he should be able to pass all three verification steps, which means that $\mathcal{A}$ could succeed in the following tasks. The first one is to provide an invalid C_i^* that satisfies the CBF; the success probability for $\mathcal{A}$ is δ, which is the non-negligible false-positive probability of the CBF. The second one is to provide k invalid counter values that match the CBF; the success probability for $\mathcal{A}$ is δ^M, where M is the number of streaming elements that have already been outsourced to the server. The last one is to succeed in providing a forgery of RSA signature S_i^* (Eq. 1) and a forgery of vector commitment C_i^* in bilinear groups (Eq. 2); the success probability ϵ for $\mathcal{A}$ is negligible [13, 25]. Therefore, the probability of $\mathcal{A}$ successfully forging a tuple (i^*, e_i^*, π_i^*) in the P-CVDS scheme should be negligible (it is equal to $\delta \times \delta^M \times \epsilon$), and we say the proposed P-CVDS scheme is secure. □

Theorem 2 *The proposed P-CVDS scheme is correct.*

Proof Assume the server $\mathcal{S}$ and the client $\mathcal{C}$ have executed their algorithms without any mistakes. Then, the query proof $\pi_i = \{\pi s_i, \pi c_i\}$, where $\pi s_i = \{\Lambda_i, C_i, S_i, H_{\mathcal{S}}\}$ and $\pi c_i = \{V_{\mathcal{C}}, H_{\mathcal{C}}\}$. Let $\alpha := \lceil i/s \rceil$. Then, $V_{\mathcal{C}} = \{Counter[H_x(\alpha s \| C_i)]\}_{x \in [1,k]}$. Therefore, the k counter values in $V_{\mathcal{C}}$ must be no less than 1 and, the first verification step always passes. Next, $H_{\mathcal{S}}$ and $H_{\mathcal{C}}$ are computed by $\mathcal{S}$ and $\mathcal{C}$ with their own CBF counters, and these counters, which have the same index, should be equal to each other. Thus, the second verification step can be passed successfully. Finally, the signature and commitment verifications are correct because $C_i = RSA(spk, S_i) = RSA(spk, RSA(ssk, C_i)) = C_i$.[5] Therefore, the output of the verification algorithm is always the streaming element value e_i, and the proposed P-CVDS scheme can be said to be correct. □

Theorem 3 *The proposed P-CVDS scheme is efficient.*

Proof From the description of the P-CVDS scheme, all the computation and storage complexities of the client's operations are constant and independent of the upper bound on the number of streamed elements and the number of elements that have been streamed so far. In the DIVC scheme, the most frequently executed algorithm by the client is the append algorithm. The client's complexities are $\mathcal{O}(1)$, which is the same as those of the setup and updating algorithms. When cooperating with the counting Bloom filter, the client only holds and operates a constant number of hash functions and counters, which makes the client's complexities $\mathcal{O}(1)$. The verification time and space complexities of a verifier are also $\mathcal{O}(1)$. Therefore, according to the definition of efficiency in CVDS, the proposed P-CVDS scheme is efficient. □

[4] According to the characteristics of the CBF scheme, if this step of verification is not passed, the final result of the **CVDS.Verify**$(\cdot)$ algorithm cannot be passed; however, if this verification step is passed, the final result could be correct or not. That is the reason why we say that this SVDS scheme is probabilistically verifiable.

[5] The proof of commitment verification can be found in [13, 25] and other related works.

Theorem 4 *The proposed P-CVDS scheme is accountable.*

Proof When the verification of a proof π_i fails in the P-CVDS scheme, the verifier could ask the server to provide the cell commitments and their signatures, while asking the client to provide the corresponding counter values of the counting Bloom filter. If any signature is invalid, the server should take responsibility for the unsuccessful verification; otherwise, if all the signatures are valid and the proof $\pi s_i \in \pi_i$ from the server is correct according to the cell commitments, which means that the client has made mistakes in preparing the proof πc_i or in outsourcing the elements, then the client should be responsible for the verification failure. Therefore, the proposed P-CVDS scheme is accountable. □

4.4 Deterministic verifiability CVDS scheme based on accumulator

The primitive of accumulator was first introduced by Benaloh and Mare in [42]; it allows one to hash a large set of inputs into one short value, such that there is a short proof that a given input was incorporated into this value. Like the Bloom filter, the cryptographic accumulator can also be used to perform set membership tests, while the results of accumulator can be proved secure and accurate. However, the early accumulator schemes did not take the set member insertion and deletion into account; thus, the update of the set was hard. Consequently, a dynamic accumulator, which was first introduced by Camenisch and Lysyanskaya in [43], allows one to dynamically add and delete a value, such that the cost of an add or delete is independent of the number of accumulated values.

In this subsection, we employ the dynamic accumulator scheme that is based on bilinear pairing, which was presented by Nguyen in [44], to construct a deterministic verifiability CVDS scheme (D-CVDS for short).

– **CVDS.Setup**$(1^\lambda, s, cnt)$: As a concrete DIVC scheme that is based on bilinear pairing has been proposed, to decrease the number of redundant key parameters, we share some DIVC parameters in this SVDS scheme construction. First, invoke **DIVC.KeyGen**$(\cdot)$ and obtain the following parameters: two bilinear groups of prime order p that are equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and a randomly chosen generator g of $\mathbb{G}$. Next, the client chooses a random integer r from $\mathbb{Z}_p^*$ as a parameter of the secret key SK and initializes the auxiliary material as $PS := \{g\} = \{g^{r^0}\}$ and the client accumulator value as $A_\mathcal{C} := 1$. Finally, this algorithm sets the public key as

$$PK := \{s, PP = \{g, \{h_i\}_{i\in[1,s]}, \{h_{i,j}\}_{i,j\in[1,s],i\neq j}\}\},$$

sets the secret key as

$$SK := \{r, \{z_i\}_{i\in[1,s]}, A_\mathcal{C}\},$$

and sets the auxiliary information as

$$aux := \{cnt, PS, DIVC.aux\}.$$

– **CVDS.Append**(SK, e): Similar to the construction of the P-CVDS scheme, the client invokes **DIVC.Com**$(\cdot)$ to add the new streaming element e into the server's DS. Then, if this appended element is the last one in its cell ($cnt-1 \mod s = 0$), the client adds the completed cell commitment C_{cnt-1} into the accumulator $A_\mathcal{C}$ as follows. Let $\alpha := \lceil (cnt-1)/s \rceil$. The client sets

$$A_\mathcal{C} := A_\mathcal{C} \times (f((cnt-1)\|C_{cnt-1})+r) = \prod_{i=1}^{\alpha}(f(is\|C_{is})+r),$$

where $f : \{0,1\}^* \to \mathbb{Z}_p^*$, and sets

$$PS := PS \cup \{g^{r^\alpha}\}.$$

That makes it possible to verify the outsourced cell commitment values deterministically, and relative to the previous P-CVDS scheme, we denote this scheme as deterministic verifiability CDVS (D-CVDS).

– **CVDS.Query**(PK, DS, i, aux): This query algorithm invokes the first algorithm in the DIVC scheme to obtain the cell proof tuple (Λ_i, C_i, S_i). Let $\beta := \lfloor cnt/s \rfloor$. The server computes the current server accumulator $A_\mathcal{S}$ as follows.

$$A_\mathcal{S} := g^{\prod_{j=1}^{\beta}(f(js\|C_{js})+r)} = \prod_{j=1}^{\beta} g^{c_j r^j} = \prod_{j=1}^{\beta}(g^{r^j})^{c_j},$$

where c_j is the coefficient of r^j in the polynomial $\mathcal{F}_\mathbb{C}(r) = \prod_{j=1}^{\beta}(f(js\|C_{js}) + r)$ and the set $\mathbb{C} = \{C_x\}$ consists of all the cell commitment values ($|\mathbb{C}| = \beta$). Then, the server constructs the server-side query proof

$$\pi s_i := \{\Lambda_i, C_i, S_i, A_\mathcal{S}\}.$$

Furthermore, the client computes the client-side query proof

$$\pi c_i := g^{A_\mathcal{C}/(f(\alpha s\|C_{\alpha s})+r)},$$

where $\alpha = \lceil i/s \rceil$. Finally, this query algorithm outputs the query result (e, π_i), where the query proof

$$\pi_i := \{\pi s_i, \pi c_i\}.$$

– **CVDS.Verify**(PK, i, e, π_i, aux): The verifier parses the proof π_i as $\{\pi s_i, \pi c_i\}$, which can be further parsed as $\{\{\Lambda_i, C_i, S_i, A_\mathcal{S}\}, g^{A_\mathcal{C}/(f(\alpha s\|C_{\alpha s})+r)}\}$. Then, the verification is carried out via the following two steps. In the first step, this verification algorithm determines whether all the cell commitment values are consistent with the original ones. That can be done by

checking whether the membership verification of C_i can succeed or not. That is, let $\alpha := \lceil i/s \rceil$ and test

$$e(\pi c_i, g^{f(\alpha s \| C_i)} \cdot g^r) \stackrel{?}{=} e(A_{\mathcal{S}}, g). \quad (3)$$

If these two pairs are equal, this algorithm proceeds to the next step; otherwise, it is aborted and returns an error $\perp$. In the second step, the verifier continues to determine whether the signature and the cell proof Λ_i are valid for the cell commitment C_i. That can be done by invoking the verification algorithm **DIVC.Ver**$(\cdot)$ in the DIVC scheme. If **DIVC.Ver**$(\cdot)$ outputs 1, this algorithm returns the element value e as its result; otherwise, it outputs an error $\perp$.

– **CVDS.Update**$(PK, SK, DS, i, e, e', aux)$: The updating of the ith element from e to e' can be performed by recomputing the following two relevant parameter parts. Let $\alpha := \lceil i/s \rceil$. The two parts of parameters are the αth cell commitment $C_{\alpha s}$ with its signature $S_{\alpha s}$, which is stored by the server, and the accumulated value $A_{\mathcal{C}}$, which is held by the original data owner (the client). The cell commitment value and its signature can be updated by the updating algorithm **DIVC.Update$_{\mathbf{PP}}$**$(\cdot)$ in the DIVC scheme, which updates the commitment value from C_i to C_i' and the signature value from S_i to S_i'. In contrast, the client modifies the local accumulated value

$$A_{\mathcal{C}}' := A_{\mathcal{C}}/(f(\alpha s \| C_i) + r) \times (f(\alpha s \| C_i') + r).$$

4.4.1 Security analysis of the D-CVDS scheme

We prove that the proposed deterministic verifiability CVDS scheme (D-CVDS) satisfies the requirements of security, correctness, efficiency, and accountability as follows.

Theorem 5 *The proposed D-CVDS scheme is secure.*

Proof If a PPT adversary $\mathcal{A}$ wants to pass the verification algorithm of the D-CVDS scheme with an invalid tuple (i^*, e_i^*, π_i^*), he should be able to pass both verification steps, which means that $\mathcal{A}$ could succeed in the following tasks. The first one is to provide an invalid C_i^* that satisfies Eq. 3; the success probability for $\mathcal{A}$ is ε, which has been proven to be negligible in [13, 25] and other related works. The second one is to succeed in providing a forgery of RSA signature S_i^* (Eq. 1) and a forgery of vector commitment C_i^* in bilinear groups (Eq. 2); the success probability ϵ for $\mathcal{A}$ is also negligible. Therefore, the probability for $\mathcal{A}$ to successfully forge a tuple (i^*, e_i^*, π_i^*) in the D-CVDS scheme should be negligible (which equals $\varepsilon \times \epsilon$) and we say that the proposed D-CVDS scheme is secure. □

Theorem 6 *The proposed D-CVDS scheme is correct.*

Proof Assume the server $\mathcal{S}$ and the client $\mathcal{C}$ have executed their algorithms without any mistakes. Then, the query proof $\pi_i = \{\pi s_i, \pi c_i\}$. Let $\alpha := \lceil i/s \rceil$. Then, $\pi s_i := \{\Lambda_i, C_i, S_i, A_{\mathcal{S}}\}$ and $\pi c_i := g^{A_{\mathcal{C}}/(f(\alpha s \| C_{\alpha s})+r)}$. Thus,

$$\begin{aligned} e(\pi c_i, g^{f(\alpha s \| C_i)} \cdot g^r) &= e(g^{A_{\mathcal{C}}/(f(\alpha s \| C_{\alpha s})+r)}, g^{f(\alpha s \| C_i)+r}) \\ &= e(g^{A_{\mathcal{C}}}, g) = e(A_{\mathcal{S}}, g), \end{aligned}$$

which means that the first step of the verification algorithm in the D-CVDS scheme can always succeed. Next, by the same proofs as for P-CVDS, the signature and commitment verifications are also correct. Therefore, the output of the verification algorithm is always the streaming element value e_i, and the proposed D-CVDS scheme can be said to be correct. □

Theorem 7 *The proposed D-CVDS scheme is efficient.*

Proof From the construction of the D-CVDS scheme, all the computational and storage complexities of the client's operations are constant and independent of the upper bound on the number of streamed elements and the number of elements that have been streamed so far. In the DIVC scheme, the most frequently executed algorithm by the client is the append algorithm. The client's complexities are $\mathcal{O}(1)$, which is the same as those of the setup and updating algorithms. When cooperating with the dynamic accumulator scheme in bilinear pairings, for each element or algorithm execution, the client only holds an accumulated value $A_{\mathcal{C}}$ and executes few multiplications on integers and exponentiations in bilinear groups. Thus, the client's complexities are $\mathcal{O}(1)$. The verification time and space complexities of a verifier are also $\mathcal{O}(1)$. Therefore, according to the definition of efficiency in CVDS, the proposed D-CVDS scheme is efficient. □

Theorem 8 *The proposed D-CVDS scheme is accountable.*

Proof When the verification of a proof π_i fails in the D-CVDS scheme, the verifier could ask the server to provide the cell commitments and their signatures while asking the client to provide the corresponding accumulator value $g^{A_{\mathcal{C}}}$. If any signature is invalid, the server should take responsibility for the unsuccessful verification; otherwise, if all the signatures are valid and the other parameters in the server proof $\pi s_i \in \pi_i$ are correct according to these cell commitments that have been further provided by the server, that means the client has made mistakes in preparing the proof πc_i or in outsourcing the elements; in this case, the client should be responsible for the

Table 1 Comparison of the proposed CVDS schemes and existing VDS schemes

Schemes	P-CVDS	D-CVDS	CAT-VDS	CVC-VDS	ACC-VDS
Security assumption	CDH	q-SDH	Dlog	CDH	q-SDH
Unbounded elements	No	No	No	Yes	Yes
Append (Client)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Query (Server)	$\mathcal{O}(N/s)$	$\mathcal{O}(N/s)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_q N)$	$\mathcal{O}(U)$
Verify (Verifier)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_q N)$	$\mathcal{O}(1)$
Update (Client)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_q N)$	$\mathcal{O}(1)$
Proof size	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_q N)$	$\mathcal{O}(1)$
Public key size	s^2	s^2	$\mathcal{O}(1)$	q^2	$\mathcal{O}(1)$
Verified scale	Entire	Entire	Entire	Entire	Element

verification failure. Therefore, the proposed D-CVDS scheme is accountable. □

4.5 Efficiency analysis and comparison

In this subsection, we compare the two proposed CVDS schemes (P-CVDS and D-CVDS) with the other three existing VDS schemes: the CAT-VDS scheme, which was proposed in [7] and presented the first formal definition of verifiable data streaming with updates, and the CVC-VDS and ACC-VDS schemes, which were proposed in [9]. CVC-VDS was the most efficient VDS scheme until the ACC-VDS scheme was presented as the first VDS scheme with nearly constant complexities. All these schemes are notable milestones in the development of verifiable data streaming, and they have promoted the study of VDS (including ours). Hence, we analyze these five schemes from different theoretical aspects and present the comparison results in Table 1, where the symbols M, N, U, s, and q are respectively used to denote the upper bound on the number of data streaming elements, the number of data streaming elements that have been outsourced, the number of data streaming elements that have been updated, the cell size of the dimension-increasing vector commitment, and the vector size of the chameleon vector commitment.

From Table 1, we draw the following conclusions based on the comparison results of the five schemes: First, all these schemes support public verification and outsourced element updating. Second, the ACC-VDS scheme and our proposed P-CVDS and D-CVDS schemes realize constant time and space complexities of all the appending, verification, and updating algorithms (on the client and the verifier sides); however, the ACC-VDS scheme can only verify a single element at a time, and the client and verifier complexities in the remainder of the existing schemes are related to either the upper bound on the number of elements M or the number of outsourced elements N. Third, since the P-CVDS and CVC-VDS schemes are constructed based on the primitive of vector commitment, their public key sizes are represented as s^2 and q^2, where the s and q are constant system parameters, which means their public key sizes are still $\mathcal{O}(1)$. For the D-CVDS and ACC-VDS schemes, which employ the dynamic accumulator based on the q-Strong Diffie-Hellman assumption in bilinear pairings, the public key sizes for the accumulator scheme increase with the number of elements that have been accumulated. However, these key parameters are included in the auxiliary information. Thus, the public key size of these schemes is also $\mathcal{O}(1)$. Finally, the numbers of outsourced data streaming elements in the CVC-VDS and ACC-VDS schemes are unbounded, while the other three schemes, including ours, require upper bounds on the number of streaming elements that are either polynomial or exponential numbers, according to the security parameters[6].

5 Performance evaluation

In this section, we evaluate the performances of the proposed P-CVDS and D-CVDS schemes in a simulation environment. We implement both the schemes in the C/C++ programming language by employing two open-source toolkits: OpenSSL and the Stanford PBC library. All the following experiments were executed on the same personal computer with an Intel Xeon CPU E5-1620 v3 3.50-GHz processor, 16-GB main memory size, and the Ubuntu 14.04 Desktop (64-bit) operating system, and the performance evaluation results of these experiments are illustrated in Figs. 2, 3, 4, and 5.

First, we evaluated the initialization performances of the P-CVDS and D-CVDS schemes and mainly concerned the

[6]These parameters are mainly the security-related parameters, such as the security parameter in the setup algorithm of a verifiable data streaming scheme, the security parameter of an accumulator scheme, and the upper bound of element number within one Bloom filter.

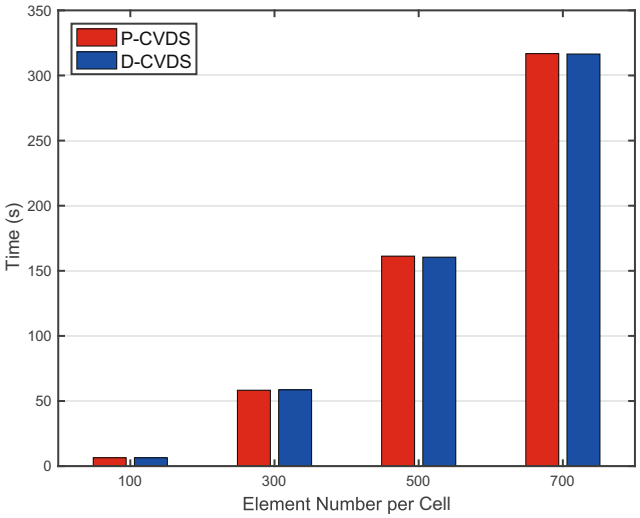


Fig. 2 Performance evaluation of initialization

time overhead of their **Setup**(·) algorithms. As the time costs are mainly related to the cell size in DIVC, we evaluated both schemes for various cell sizes, and the results of the initialization performance experiments on these two schemes are compared and shown in Fig. 2.

Note that, for clarity, in the remainder of our evaluations of the P-CVDS and D-CVDS schemes, we fixed various parameters in our experiments, e.g., the DIVC parameter was fixed to $s := 500$ and the counting Bloom filter parameters to $\epsilon := 10^{-2}$, $n := 2^{20}$, and $m := 2^{23}$.

Next, we tested the average appending and updating time overheads of the clients (data streaming owners) for one element in the P-CVDS and D-CVDS schemes with various numbers of outsourced streaming elements. We simulated six streaming data sets, for which the numbers of elements were 10^3, 10^4, 10^5, 10^6, 10^7, and 10^8, and the appending and updating operations in the P-CVDS and D-CVDS schemes were executed 100 times over each of them. The time consumptions were recorded, and we display their average time consumptions in Fig. 3. Figure 3a illustrates the appending time for one element in P-CVDS and D-CVDS, and Fig. 3b shows the element updating times. According to these figures, in our two concrete CVDS schemes, the client complexities of appending and updating are constant and independent of the upper bound on the number of elements and the number of outsourced elements. Moreover, based on the construction of P-CVDS and D-CVDS, the servers in appending and updating only utilize the elementary input/output operations. Thus, we omitted the related tests.

Furthermore, we carried out experiments to evaluate the time overheads of the clients and the servers in the **Query**(·) algorithms of the P-CVDS and D-CVDS schemes. Like the preceding tests of appending and updating, for each element scale, the query algorithms are executed 100 times, and the average results are illustrated in Fig. 4. According to the definition of CVDS and the constructions of P-CVDS and D-CVDS, the clients are supposed to avoid expensive computation and storage, and the left side of Fig. 4a shows that the client complexities are also constant and independent of the upper bound on the number of elements and the number of outsourced elements. Thus, both our schemes satisfy this definitional requirement. In contrast, since most computational tasks are transferred to the servers, their complexities inevitably depend on the number of elements have been outsourced, which can be observed on the right side of Fig. 4b. In addition, as shown in Fig. 4b, the server time cost of the D-CVDS scheme increases more quickly than that of the P-CVDS scheme, which is mainly because the D-CVDS scheme involves many computations

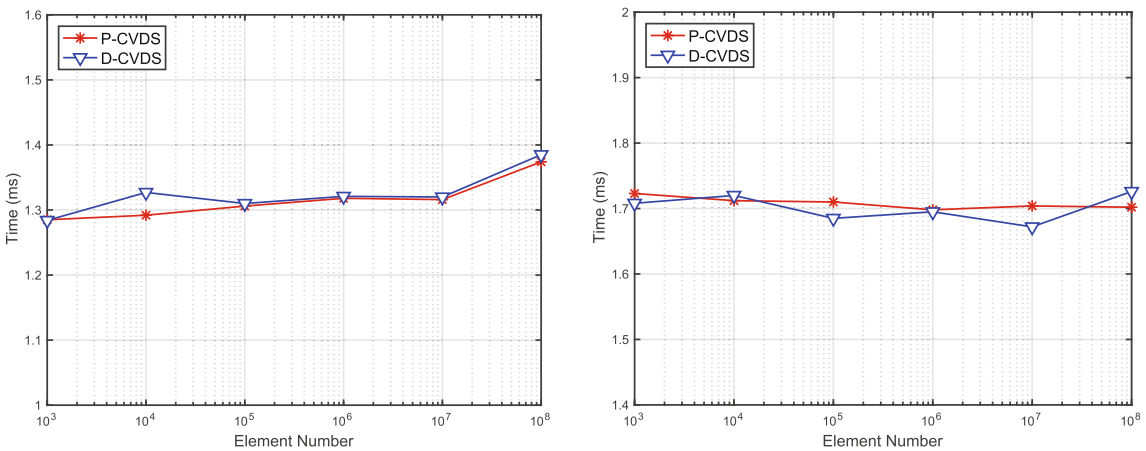
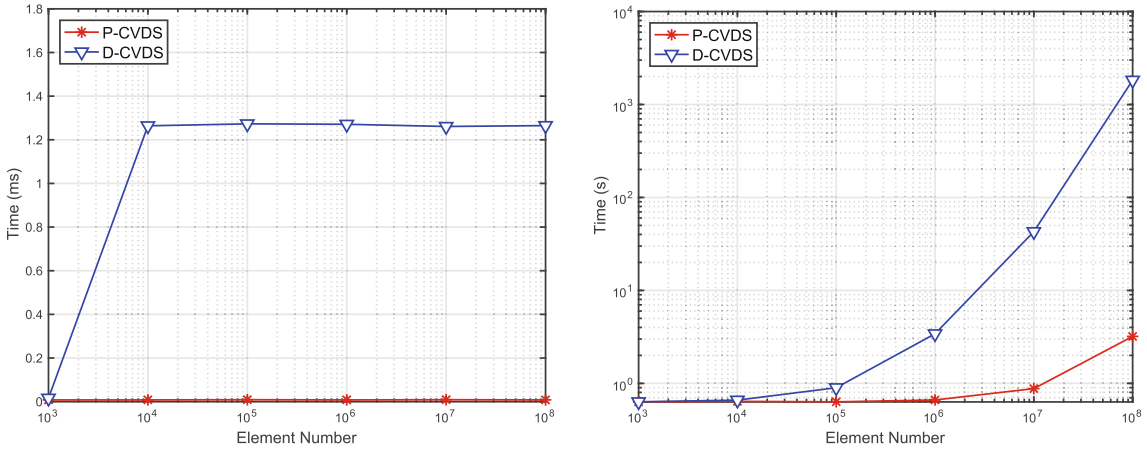


Fig. 3 Performance evaluation of appending(**a**)and updating (**b**)



(a) Client Query Time (b) Server Query Time

Fig. 4 **a**, **b** Performance evaluation of querying

in bilinear groups; however, the server can be accelerated by employing parallel computing. Therefore, we argue that both the P-CVDS and D-CVDS schemes are feasible in practice.

Finally, we evaluated the verification performances of the P-CVDS and D-CVDS schemes, and the average test results of their time overheads for the **Verify**$(\cdot)$ algorithms are presented in Fig. 5. The verifiers in both P-CVDS and D-CVDS are designed to execute minimal numbers of operations during verification. As expected, according to Fig. 5, the verifier complexities are also constant and independent of the upper bound on the number of elements and the number of outsourced elements. Although the verification time costs of the D-CVDS scheme are higher than those of the P-CVDS scheme, both are sufficiently efficient and acceptable for most application environments.

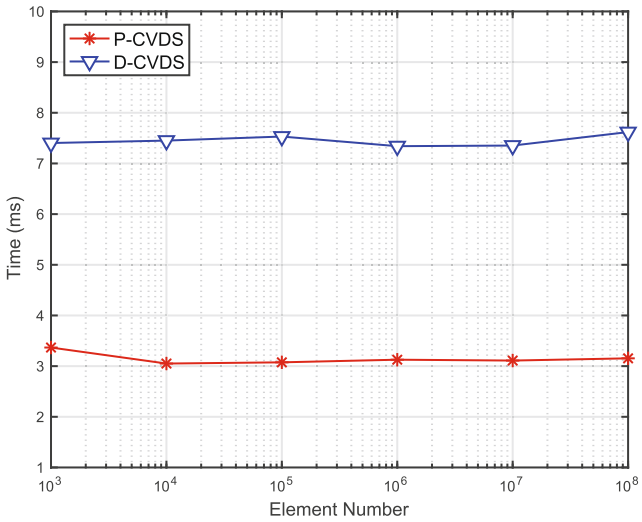


Fig. 5 Performance evaluation of verification

6 Conclusion

Data streaming has been utilized for many applications in the Internet of Things and other related environments, and streaming data are increasingly outsourced to cloud-based servers to be processed and stored. To improve the efficiency of existing verifiable data streaming schemes with updating, in this paper, we first present a new primitive, namely, dimension-increasing vector commitment, which is extended from the original vector commitment. Then, the concrete bilinear DIVC scheme is used as a building block in constructing the newly proposed constant verifiable data streaming schemes. Our analyses prove that the two concrete CVDS schemes, namely, the P-CVDS and D-CVDS schemes, which were constructed based on the counting Bloom filter and the bilinear-map dynamic accumulator, respectively, satisfy all the security requirements in the CVDS definition. Moreover, the efficiency and performance evaluations demonstrate that both the P-CVDS and D-CVDS schemes are efficient for the client and verifier in theory and practice. In these aspects, CVDS exceeds the abilities and efficiencies of the existing streaming authenticated data structures and verifiable data streaming schemes. In the future, we will study the consistency verification of streaming data from the stream recipient side, i.e., how to enable a stream recipient to verify the orders and values of streaming elements on the fly.

Funding information This work is supported by the National Natural Science Foundation of China (no. 61572382), Key Project of Natural Science Basic Research Plan in Shaanxi Province of China (no. 2016JZ021), China 111 Project (no. B16037), Guangxi Cooperative Innovation Center of cloud computing and Big Data (no. YD17X07), and Guangxi Colleges and Universities Key Laboratory of cloud computing and complex systems (no. YF17103).

References

1. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: ACM Sigmod-Sigact-Sigart symposium on principles of database systems, pp 1–16
2. Abadi DJ, Carney D, Çetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S (2003) Aurora: a new model and architecture for data stream management. VLDB J 12(2):120–139
3. Golab L, Tamer Özsu M (2003) Issues in data stream management. Acm Sigmod Record 32(2):5–14
4. Krishnaswamy S (2005) Mining data streams: a review. Acm Sigmod Record 34(2):18–26
5. Papamanthou C, Shi E, Tamassia R, Yi K (2013) Streaming authenticated data structures. In: Advances in cryptology – EUROCRYPT 2013, Springer, Berlin, pp 353–370.
6. Yi Q, Zhang Y, Xi C, Papamanthou C (2014) Streaming authenticated data structures: abstraction and implementation. In: Edition of the ACM workshop on cloud computing security, pp 129–139
7. Schröder D, Schröder H (2012) Verifiable data streaming. In: Proceedings of the ACM conference on computer and communications security, ACM, pp 953–964
8. Schöder Dominique, Simkin Mark (2015) Veristream – a framework for verifiable data streaming. In: International conference on financial cryptography and data security. pp 548–566
9. Krupp J, Schröder D, Simkin M, Fiore D, Ateniese G, Nuernberger S (2016) Nearly optimal verifiable data streaming. In: Proceedings, Part I, of the 19th IACR international conference on public-key cryptography – PKC 2016, vol 9614. Springer, New york inc., pp 417–445
10. Merkle RC (1980) Protocols for public key cryptosystems. In: IEEE symposium on security and privacy(SP). pp 122–134
11. Chen X, Zhang F, Susilo W, Tian H, Li J, Kim K (2014) Identity-based chameleon hashing and signatures without key exposure. Information Sciences An International Journal 265(5):198–210
12. Zhang Z, Chen X, Li J, Tao X, Ma J (2018) Hvdb: a hierarchical verifiable database scheme with scalable updates. Journal of Ambient Intelligence and Humanized Computing
13. Chen X, Li J, Huang X, Ma J, Lou W (2015) New publicly verifiable databases with efficient updates. IEEE Trans Dependable Secure Comput 12(5):546–556
14. Li J, Liu Z, Chen X, Xhafa F, Tan X, Wong DS (2015) L-encdb: a lightweight framework for privacy-preserving data queries in cloud computing. Knowl-Based Syst 79:18–26
15. Wang J, Chen X, Huang X, You I, Xiang Y (2015) Verifiable auditing for outsourced database in cloud computing. IEEE Trans Comput 64(11):3293–3303
16. Li T, Liu Z, Li J, Jia C, Li KC (2017) CDPS: a cryptographic data publishing system. J Comput Syst Sci 89:80–91
17. Liu Q, Guo Y, Wu J, Wang G (2017) Effective query grouping strategy in clouds. J Comput Sci Technol 32(6):1231–1249
18. Li P, Li J, Huang Z, Gao CZ, Chen WB, Chen K (2017) Privacy-preserving outsourced classification in cloud computing. Cluster Computing
19. Li J, Huang X, Li J, Chen X, Xiang Y (2014) Securely outsourcing attribute-based encryption with checkability. IEEE Trans Parallel Distrib Syst 25(8):2201–2210
20. Chen X, Huang X, Li J, Ma J, Lou W, Wong DS (2015) New algorithms for secure outsourcing of large-scale systems of linear equations. IEEE Trans Inf Forensics Secur 10(1):69–78
21. Chen X, Li J, Weng J, Ma J, Lou W (2016) Verifiable computation over large database with incremental updates. IEEE Trans Comput 65(10):3184–3195
22. Li J, Li J, Xie D, Cai Z (2016) Secure auditing and deduplicating data in cloud. IEEE Trans Comput 65(8):2386–2396
23. Wang J, Chen X, Li J, Zhao J, Shen J (2016) Towards achieving flexible and verifiable search for outsourced database in cloud computing. Futur Gener Comput Syst 67
24. Benabbas S, Gennaro R, Vahlis Y (2011) Verifiable delegation of computation over large datasets. In: Annual cryptology conference, Springer, pp 111–131
25. Catalano D, Fiore D (2013) Vector commitments and their applications. In: Public key cryptography, Springer, pp 55–72
26. Merkle RC (1980) Protocols for public key cryptosystems. ieee symposium on security and privacy, pp 122–122
27. Merkle RC (1990) A certified digital signature. In Advances in cryptology — CRYPTO' 89 proceedings, Springer, New York, pp 218–238
28. Tamassia R (2003) Authenticated data structures. In: European symposium on algorithms, pp 2–5
29. Miller A, Hicks M, Katz J, Shi E (2014) Authenticated data structures, generically. In: ACM Sigplan-sigact symposium on principles of programming languages, pp 411–423
30. Ajtai M (1996) Generating hard instances of lattice problems. In: Twenty-Eighth ACM symposium on theory of computing, pp 99–108
31. Do J-M, Song Y-J (2014) Secure streaming media data management protocol. International Journal of Security and Its Applications 8(2):193–202
32. Puthal D, Nepal S, Ranjan R, Chen J (2015) A dynamic key length based approach for real-time security verification of big sensing data stream, In: International conference on web information systems engineering, pp 93–108
33. Chen C-Y, Wu H-M, Wang L, Yu C-M (2017) Practical integrity preservation for data streaming in cloud-assisted healthcare sensor systems. Comput Netw 129:472–480. Special Issue on 5G Wireless Networks for IoT and Body Sensors
34. Yi S, Chen X, Du X, Xu J (2017) Dynamic authenticated data structures with access control for outsourcing data stream. IET Inf Secur 11(5):235–242
35. Bethencourt J, Sahai A, Waters B (2007) Ciphertext-policy attribute-based encryption. In: IEEE symposium on security and privacy, pp 321–334
36. Boneh D, Boyen X (2004) Short signatures without random oracles. In: International conference on the theory and applications of cryptographic techniques, Springer, pp 56–73
37. Chen X, Li J, Ma J, Tang Q, Lou W (2014) New algorithms for secure outsourcing of modular exponentiations. IEEE Trans Parallel Distrib Syst 25(9):2386–2396
38. Chen X, Li J, Huang X, Li J, Xiang Y, Wong DS (2014) Secure outsourced attribute-based signatures. IEEE Trans Parallel Distrib Syst 25(12):3285–3294
39. Bloom BH (1970) Space/time trade-offs in hash coding with allowable errors. Commun ACM 13(7):422–426
40. Li F, Cao P, Almeida J, Broder AZ (2000) Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Trans Networking 8(3):281–293
41. Rottenstreich O, Kanizo Y, Keslassy I (2014) The variable-increment counting bloom filter. IEEE/ACM Transactions on Networking (TON) 22(4):1092–1105
42. Benaloh J, de Mare M (1994) One-way accumulators: A decentralized alternative to digital signatures. In: Advances in cryptology — EUROCRYPT '93, Springer, Berlin, pp 274–285

43. Camenisch J, Lysyanskaya A (2002) Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Annual international cryptology conference, Springer, pp 61–76

44. Nguyen L (2005) Accumulators from bilinear pairings and applications. In: Cryptographers' track at the RSA conference, Springer, pp 275–292

Affiliations

Zhiwei Zhang[1] · Xiaofeng Chen[1] · Jianfeng Ma[1] · Xiaoling Tao[2]

Xiaofeng Chen
xfchen@xidian.edu.cn

Jianfeng Ma
jfma@mail.xidian.edu.cn

Xiaoling Tao
txl@guet.edu.cn

[1] State Key Laboratory of Integrated Service Networks (ISN), Xidian University, Xi'an, 710071, Shaanxi, China

[2] Guangxi Cooperative Innovation Center of Cloud Computing and Big Data, Guangxi Colleges and Universities Key Laboratory of Cloud Computing and Complex Systems, Guilin University of Electronic Technology, Guilin, 541004, China