Journal of Ambient Intelligence and Humanized Computing (2019) 10:3045–3057
https://doi.org/10.1007/s12652-018-0757-8

ORIGINAL RESEARCH

HVDB: a hierarchical verifiable database scheme with scalable updates

Zhiwei Zhang[1] · Xiaofeng Chen[1] · Jin Li[2] · Xiaoling Tao[3] · Jianfeng Ma[1]

Received: 5 January 2018 / Accepted: 11 March 2018 / Published online: 17 March 2018

Abstract
The notion of verifiable database (VDB) plays an important role in secure outsourcing of data storage, which allows a client, typically a resource-constraint one, to outsource the large-scale databases to an untrusted server and make retrieval and update queries. For each query request, the server responds with a query result and a proof which is used to verify the result. Plenty of research work has been done on designing efficient VDB schemes in the past years. However, it seems that all of the existing VDB schemes are presented in the amortized model. That is, all existing VDB schemes require a prohibitively expensive pre-processing stage. As a result, the overhead for the initialization algorithm (e.g., the key generation algorithm) is too heavy to afford by the client. Furthermore, VDB schemes can only ensure the integrality of the whole database. However, it is incapable of verifying a segment of the database and localizing the tampered record efficiently when the database is tampered with. In this paper, we firstly propose a new primitive called Vector Commitment Tree (VCT), in which each node is a vector commitment (VC) of its q children. Then, we utilize VCT as a building block to propose a hierarchical verifiable database scheme (HVDB) with scalable updates, which supports the hierarchical verification and the tampered record localization. Besides, HVDB can also greatly reduce the burden of initialization algorithm of VDB schemes. Finally, the analysis and experimental results show that the proposed HVDB scheme can achieve the desired security requirements and improve the efficiency for practical application.

Keywords Verifiable database · Vector commitment tree · Outsourced Storage · Cloud computing · Tampered data localization

1 Introduction

Cloud computing (Buyya et al. 2009) integrates massive amounts of computing, networking and storage resources, and provides them in the form of corresponding on-demand and scalable services over the internet (Armbrust et al. 2010). However, in order to take advantage of these services, the traditional tasks executed in local devices have to be delegated to the cloud service providers (Chen et al. 2015a). In this case, the cloud users lose control of their own data (Rittinghouse and Ransome 2016). As a result, the separation of data ownership and management raises many security and privacy problems (Takabi et al. 2010; Xiao and Xiao 2013; Stojmenovic et al. 2016).

Consequently, it is common to encrypt the outsourced data (Vimercati et al. 2007; Chow et al. 2009), which ensures that only the authorized users are able to access and decrypt the outsourced data. Nonetheless, it is infeasible for the users to verify the data integrality and storage order only with encryption technologies, because these technologies are

✉ Jin Li
lijin@gzhu.edu.cn

Zhiwei Zhang
zwzhang@xidian.edu.cn

Xiaofeng Chen
xfchen@xidian.edu.cn

Xiaoling Tao
txl@guet.edu.cn

Jianfeng Ma
jfma@mail.xidian.edu.cn

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

[2] School of Computer Science, Guangzhou University, Guangzhou, China

[3] 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

mainly used to protect the confidentiality of the outsourced data. The primitive of Verifiable Database (VDB) (Benabbas et al. 2011; Catalano and Fiore 2013), which can be seen as a special kind of verifiable computing, are proposed to provide the cloud storage with security guarantees including authenticity, integrality, (public or private) verifiability, etc. Although plenty of researchers have well studied VDB and presented many VDB schemes, there are still various problems and challenges in processing the large volume of the outsourced data.

Firstly, all the existing VDB schemes employ the amortized computational model, which assumes that a one-time expensive computational effort in the **Setup** stage can be accepted to generate the necessary auxiliary parameters like the private keys and public keys. However, even the one-time computational effort could be omitted, the long-term storage cost for these auxiliary parameters has to be considered, as to be shown in the following example, the storage cost of these VDB schemes may be unacceptable for practical applications, especially when the number of data records grows too large. Take the first VDB scheme based on vector commitments (VC) (Catalano and Fiore 2013) as an example, if a database protected by the VDB scheme possesses q records, the size of its private key parameters is over q, while the size of its public key parameters exceeds q^2. That is to say, if there are one million (10^6) records in the database, which is verifiable under the mentioned VDB scheme constructed with the bilinear pairings, its private and public key size is one million (10^6) and one trillion (10^{12}) respectively, and they occupy about 128 MB and 128 TB storage space of the data owner and cloud service provider. The cost of 128 TB storage space in Amazon S3 is about $3000 per month, which does not yet include the request, transfer, management and other costs. More importantly, the massive public key size introduces higher computation overhead and communication overhead into the VDB's initialization and query stages, and makes the key distribution and other management related processes more complicated. Therefore, it is desired to keep the key size as small as possible, no matter how massive the outsourced database is.

Secondly, the existing VDB schemes are competent in verifying whether the outsourced database is tampered, but none of them can localize the tampered record, i.e., they cannot specifically determine which data record is tampered when the verification result indicates that the outsourced database has been modified without authorization of the data owner. However, it should not be the ultimate goal of a VDB scheme to just detect the fact that a database is tampered, the VDB schemes are expected to be capable of finding out the specific tampered record. Without the ability of tampered record localization, the verifier, a user who detects the unauthorized modification to the outsourced database, should check all the records one by one so as to find out the tampered one. Obviously, this approach is infeasible for practical applications, especially when the database size turns larger. Hence, the requirement of tampered record localizability should be introduced into VDB schemes.

Finally, the data records belonging to a logically entire database are usually stored at different geographic location, such as cloud storage and content delivery network (CDN) (Pallis and Vakali 2006; Spagna et al. 2013). To verify this kind of database with the existing VDB schemes, extra communication overheads are introduced by transmitting the distributed records to the requested server. Besides, companies are usually organized with a rigid hierarchy, so group-oriented applications and hierarchical data management structures are widely employed, and the data access is controlled with users' privileges. For example, an ordinary staff can only access to her own data, a manager can access to the data belonging to her department, while the chief executive officer (CEO) who has the supreme seniority can access to all data in the company (Liu et al. 2013; Wen et al. 2015; Tang et al. 2016). However, all the traditional VDB schemes can and only can verify the entirety of the outsourced database, which cannot be directly applied to these distributed data scenarios and hierarchical data scenarios.

1.1 Our contributions

In this paper, we focus on the three problems in the existing VDB schemes as mentioned above: the key parameters explosion, the incompetence of tampered record localization, and the lack of distributive and hierarchical verification. Our contributions are given as follows:

- We propose the notion of vector commitment tree (VCT), which is an extension of the original VC definition in a recursive way. The VCT framework divides the elements into fixed size cells, these cells share one set of key parameters, so as to control over the number of key parameters. Furthermore, in VCT framework, the cells could be related to access permissions, thus the elements in different cells are relatively independent, they can be dealt with respectively and concurrently, which make the distributive and scalable (or hierarchical) verification possible. Besides, we construct a concrete VCT scheme based on CDH assumption in bilinear groups.
- We propose a new VDB framework called HVDB whose private key and public key numbers are decided by a constant parameter rather than the size of the outsourced database. Unlike the existing VDB framework, the proposed HVDB framework allows a data user to verify part of the outsourced database according to her privilege. Especially, if a record is tampered, HVDB could give a range of location including the index of the tampered record. Then a concrete HVDB scheme is constructed

with our VCT scheme, and we prove that the HVDB scheme is secure under the CDH assumption. Moreover, the comparison and performance evaluation show that our HVDB scheme is superior to the existing ones in the setup and query stages.

1.2 Related work

Cloud computing facilitates outsourcing computation, or delegating computation in some papers (Goldwasser et al. 2016; Zhang and Safavi-Naini 2014), which allows the clients, usually with limited resources (Li et al. 2014a; Stergiou et al. 2018), to outsource the heavy computation tasks to the cloud servers (Chen et al. 2014a; Chang and Yang 2017). e.g., very recently, a new data outsouring and sharing scheme is proposed in (Shen et al. 2018) which supports both anonymity and traceability of access control. At the same time, outsourcing computation also suffers from some new security challenges (Li et al. 2017b). Consequently, plenty of researchers have devoted considerable attention to the problem of how to securely outsource different kinds of expensive computations (Li et al. 2015), and many secure outsourcing computation and delegating computation schemes have been proposed in recent years (Motahari-Nezhad et al. 2009; Gennaro et al. 2010; Gentry and Halevi 2011; Vu et al. 2013; Chen et al. 2014b; Li et al. 2014b; Reingold et al. 2016).

Cloud storage (outsourcing storage or database), which is the specific instance of the outsourcing computation, allows the clients to store their data to cloud servers through the Internet, and then they can retrieve or share their outsourced data (Zhang et al. 2017; Li et al. 2017a). Many cloud storage applications are available in real world, such as DropBox, iCloud, Google Drive, Amazon Drive, and so on. Similar to cloud computing (outsourcing computation or delegating computation), in cloud storage, the data, which is corresponding to the computation tasks in cloud computing, has to be moved to the cloud. Therefore, the verifiability and other security requirements are also desired in the realm of cloud storage, and they are still attracting attention (Wang et al. 2013, 2010; Wu et al. 2010; Chen et al. 2015b).

Benabbas et al. firstly introduced the primitive of verifiable database (VDB) and then presented the first practical VDB scheme (Benabbas et al. 2011) from outsourcing the high degree polynomial functions (Gennaro et al. 2010). However, as pointed out in (Catalano and Fiore 2013), the VDB scheme in (Benabbas et al. 2011) only supports the private verifiability. Catalano and Fiore proposed a new publicly verifiable database scheme based on vector commitment (VC) (Catalano and Fiore 2013). However, these two schemes are vulnerable to the forward automatic update (FAU) attack (Chen et al. 2015b). In order to solve this issue, Chen et al. proposed a new VDB framework with VC based on the binding commitment (Chen et al. 2015b) technique. Chen et al. also introduced the notion of verifiable database with incremental updates (Inc-VDB) in (Chen et al. 2016), which is suitable for the case when the database undergoes frequent but small modifications. Later, Miao et al. proposed a new VDB scheme supporting all the updating operations (replacement, deletion and insertion) based on the hierarchical commitment techniques (Miao et al. 2017). However, it seems that all of the existing VC-based VDB schemes are presented in the amortized model, which makes them infeasible in the practical applications.

1.3 Organization

The rest of this paper is organized as follows. We present some preliminaries in Sect. 2. In Sect. 3, we propose a new notion called Vector Commitment Tree (VCT), which is a main building block for HVDB constructions. In Sect. 4, we present a framework and a concrete scheme for HVDB, together with the security and efficiency analysis. The experimental performance evaluation for the proposed HVDB scheme is given in Sect. 5. Finally, Sect. 6 concludes this paper and discusses the future work.

2 Preliminaries

In this section, we introduce some preliminaries for this paper.

2.1 Bilinear pairings

Let $\mathbb{G}_1$ and $\mathbb{G}_T$ be two cyclic multiplicative groups of prime order p, and let g be a randomly chosen generator of $\mathbb{G}_1$. A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ with the following properties:

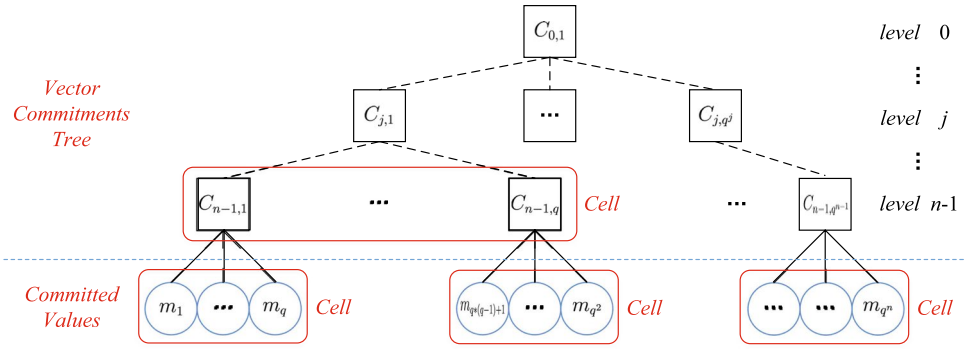
- *Bilinear* For all $u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p^*$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
- *Non-degenerate* $e(g, g) \neq 1$.
- *Computable* For all $u, v \in \mathbb{G}_1$, there is an efficient algorithm to compute $e(u, v)$.

The CDH problem in $\mathbb{G}_1$ is defined as follows:

Definition 1 (*CDH Problem*) Let $\mathbb{G}_1$ be a cyclic group of prime order p, and g is a generate of $\mathbb{G}_1$. Given a triple (g, g^a, g^b), for any random $a, b \in_R \mathbb{Z}_p$, it is a computational intractable problem to compute g^{ab}.

That is, we say the CDH assumption holds in $\mathbb{G}_1$ if for every PPT algorithm $\mathcal{A}$, there exists a negligible function

Fig. 1 Vector commitment tree



negl$(\cdot)$ such that $\mathbf{Pr}[\mathcal{A}(1^k, g, g^a, g^b) = g^{ab}] \leq \mathbf{negl}(\cdot)$ for all security parameter k.

2.2 Vector commitment

Catalano and Fiore proposed a new primitive called *Vector Commitment* (VC) in (Catalano and Fiore, 2013), which allows one to commit to an sequence of q messages denoted as $(m_1, \cdots, m_q)$. A formal definition of vector commitment is presented as follows.

Definition 2 Vector Commitment is defined as a non-interactive primitive, which consists of the following polynomial time algorithms:

- **VC.KeyGen**$(1^k, q)$ Take as inputs the security parameter k and the size q of the committed vector (where $q = poly(k)$), this key generation algorithm outputs some public parameters PP, which implicitly indicate the available message space $\mathcal{M}$ can be committed.
- **VC.Com**$_{\mathbf{PP}}(m_1, \ldots, m_q)$ Take as inputs a sequence of q messages $M_q = (m_1, \ldots, m_q)$, where $m_i \in \mathcal{M}$, and the public parameters PP, this committing algorithm outputs a committed string denoted by C and some auxiliary information denoted by aux.
- **VC.Open**$_{\mathbf{PP}}(m, i, \text{aux})$ Take as inputs a message $m \in M_q$ as well as its sequence number i in M_q, and the auxiliary information aux, this opening algorithm outputs a proof Λ_i demonstrating that m is the i-th committed message, and it is run by the original committer.
- **VC.Ver**$_{\mathbf{PP}}(C, m, i, \Lambda_i)$ Take as inputs the commitment value C, a message m with sequence number i, and their proof Λ_i, this verification algorithm outputs 1 or an error $\perp$. This algorithm accepts the proof and outputs 1 only when Λ_i can be verified as a valid proof that C is the commitment to a sequence $(m_1, \ldots, m_q)$ where $m = m_i$; otherwise, it rejects the proof and outputs $\perp$.
- **VC.Update**$_{\mathbf{PP}}(C, m, i, m')$ Take as inputs the original commitment value C, the old message m with sequence number i, and the new message $m' \in \mathcal{M}$ to substitute m, this commitment update algorithm outputs a new commitment value C' and some update information denoted by U. This algorithm is run by the original committer to update commitment C when the i-th message is changed to m'.
- **VC.ProofUpdate**$_{\mathbf{PP}}(C, \Lambda_j, U, m', i)$ Take as inputs the original commitment value C, the old proof Λ_j, the update information U, and the updated message m' at the position i in M_q, this proof update algorithm outputs a new proof Λ'_j and the updated commitment C'. This algorithm can be run by any user who holds an original proof Λ_j for the message at the position j related to the original commitment C when the i-th message is updated to m'.

3 Vector commitment tree

In this section, we introduce a new notion of vector commitment tree (VCT, for short) and present a concrete VCT scheme based on the CDH assumption in bilinear groups.

3.1 Definition of VCT

Informally speaking, a VCT is a q-ary[1] tree which is used to commit to an ordered sequence of Q messages in a recursive way and allows to open the commitment only with respect to a specific position. An example of VCT is illustrated in Fig. 1. The leaf nodes are the Q messages to be committed, the deepest non-leaf nodes are the commitments of its q children messages, and other nodes are the commitments of its q children commitments. The hight of the sample VCT is n, which is related to Q the size of the message sequence and q the size of the cells. In a VCT scheme, a cell is the basic unit to be committed with an original vector commitment scheme, it could be a sub-sequence of the messages or a group of the commitments. The relationship of n, q and Q

[1] For the sake of clarity, in the rest of this paper, we will use Q instead of q to denote the total number of a outsourced database records.

should satisfy this equation: $n = \lceil \log_q Q \rceil$. Note that given a database with Q records, there could be two different ways to satisfy the above mentioned equation: (1) Let n be fixed, then set a appropriate value for q. (2) Let q be fixed, then set a appropriate value for n.

Definition 3 The vector commitment tree is a non-interactive primitive that consists of the following polynomial-time algorithms:

- **VCT.KeyGen**$(1^k, q, Q)$ Take as inputs the security parameter k, the cell size q and the total message number Q, where $Q = poly(k)$, this key generation algorithm outputs the public parameter set PP and the VCT hight n (from the root to the deepest non-leaf node). Besides, this algorithm implicitly stipulates the message space denoted as $\mathcal{M}$ and the commitment space denoted as $\mathcal{C}$.
- **VCT.Com**$_{\mathbf{PP}}(m_1, \ldots, m_Q, f)$ Take as inputs a ordered sequence of Q messages $(m_1, \ldots, m_Q) \in Q \times \mathcal{M}$ and a projection function $f : \mathcal{C} \to \mathcal{M}$, depending on the public parameters PP, this committing algorithm outputs the auxiliary information aux and a set of commitments denoted as C, in which each value is a vector commitment to a cell of messages or commitment.
- **VCT.Open**$_{\mathbf{PP}}(m, i, aux, AC, f)$ Take as inputs the message m with index i in the database, the auxiliary information aux, the access control pair $AC = (\mu, \nu) \in \mathbb{Z}_n \times \mathbb{Z}_{q \times \nu}$, and the projection function $f : \mathcal{C} \to \mathcal{M}$, only if AC satisfies the access control policies, this opening algorithm outputs a set of proofs $\Lambda_i \in \mathcal{C}^n$ demonstrating that m is the i-th committed message in the total Q ones. And this algorithm is run by the original committer or other authorised ones.
- **VCT.Ver**$_{\mathbf{PP}}(C, m, i, \Lambda_i, AC)$ Take as inputs the commitment set C, the message m with index i, the proof set Λ_i, and the access control pair AC, this verification algorithm outputs $(0, \emptyset)$, $(1, \emptyset)$ or $(\perp, \Omega)$. The output pair $(0, \emptyset)$ means that AC does not meet the access control policies. Otherwise, this algorithm outputs $(1, \emptyset)$ only if Λ_i is a valid proof set that C was created to the sequence $(m_1, \ldots, m_Q)$ such that $m = m_i$; or, it outputs $(\perp, \Omega)$, where Ω is a range including the tampered record.
- **VCT.Update**$_{\mathbf{PP}}(C, m_i, m_i', i, f)$ Take as inputs the commitment set C, the old message m_i and new message m_i' at the i-th position, and the projection function $f : \mathcal{C} \to \mathcal{M}$, this commitment update algorithm outputs a new commitment set C' together with an update information set U. And this algorithm is run by the original committer who produced C and wants to update it by changing the i-th message m_i to m_i'.
- **VCT.ProofUpdate**$_{\mathbf{PP}}(C, \Lambda_j, m_i', i, U, f)$ Take as inputs the commitment set C, the proof set Λ_j, the new message m_i' which will be used to replace the i-th record in the outsourced database, the update information U, and the projection function $f : \mathcal{C} \to \mathcal{M}$, this proof update algorithm outputs a new proof set Λ_j'. This algorithm can be run by any user who holds a proof set Λ_j for some message at position j with respect to C, and it allows the user to compute an updated proof set Λ_j' which should be valid with respect to the updated commitments set C'.

3.2 A concrete VCT scheme based on CDH assumption

Depending on the above mentioned definition of VCT framework, we propose a concrete VCT scheme based on the CDH assumption in bilinear groups, and the CDH-based VC scheme defined in (Catalano and Fiore 2013) is employed to commit to the cells.

The construction of the VCT scheme is as follows:

- **VCT.KeyGen**$(1^k, q, Q)$ Known from the definition of VCT, q is the size of each cell and Q is the total number of messages to be committed, for the sake of clarity, we assume that $Q = q^n$.[2] Then, this key generation algorithm can be concretely constructed as follows: let $\mathbb{G}, \mathbb{G}_T$ be two bilinear groups of prime order p equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $g \in \mathbb{G}$ be a random generator. Randomly choose $z_1, \ldots, z_q \xleftarrow{R} \mathbb{Z}_p$. For all $i = 1, \ldots, q$, set $h_i = g^{z_i}$. For all $i, j = 1, \ldots, q$ and $i \neq j$, set $h_{ij} = g^{z_i z_j}$. Set $\mathrm{PP} = (g, \{h_i\}_{i \in [1,q]}, \{h_{i,j}\}_{i,j \in [1,q], i \neq j})$. Besides, the message space is $\mathcal{M} = \mathbb{Z}_p$[3] and the commitment space is $\mathcal{C} = \mathbb{G}_t$.
- **VCT.Com**$_{\mathbf{PP}}(m_1, \ldots, m_Q, f)$ Let $C_{x,y}$ denote the y-th commitment value at x-th level in the VCT, then the commitment set C is the collection of these $C_{x,y}$. As shown in Fig. 1, the root node is denoted as $C_{0,1}$, and the deepest nodes are denoted as $C_{n-1,1}$, $C_{n-1,2}$ and so on. Consequently, the deepest nodes are computed by:

$$C_{n-1,y} = \prod_{t=1}^{q} h_t^{m_{(y-1)q+t}}, \tag{1}$$

where $y = 1, \ldots, q^{n-1}$. However, for the commitment space $\mathbb{G}_T$ is different from $\mathbb{Z}_p$, the Eq. 1 cannot be directly used to compute other nodes. To fix this problem, we employ a collision-resistant hash function $f : \mathbb{G}_T \to \mathbb{Z}_p$ to project an element in bilinear group $\mathbb{G}_T$ to an element

[2] This assumption is reasonable, because Q is typically the maximum size of a database, and the unused position can be set to some special value like *null* or 0 so as to be treated as other messages.

[3] As pointed out in (Catalano and Fiore 2013), it is easy to extend the scheme to support arbitrary messages in $\{0,1\}^*$ by employing a collision-resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_p$.

in $\mathbb{Z}_p$, and the other nodes ($C_{x,y}$, where $x = n-2, \ldots, 0$ and $y = 1, \ldots, q^x$) are computed by:

$$C_{x,y} = \prod_{t=1}^{q} h_t^{f(C_{x+1,(y-1)q+t})}. \tag{2}$$

Then, this algorithm outputs the commitment set $C = \{C_{x,y} | x \in [n-1, 0], y \in [1, q^x]\}$ and the auxiliary information $aux = (m_1, \ldots, m_Q)$.

– **VCT.Open$_{\mathbf{PP}}$**(m, i, aux, AC, f) Parse the access control pair $AC = (\mu, \nu)$ which means only the elements belonging to the ν-th commitment at the μ-th level are accessible, and verify the validity of the AC parameters as described below: If $\mu \neq \lceil i/q^{n-\nu} \rceil$, this algorithm is terminated and outputs $\perp$; otherwise, it computes the proof set Λ_i. Firstly, the deepest proof $\Lambda_{i,n-1}$ is computed as follows:

$$\begin{aligned} \Lambda_{i,n-1} &= \prod_{j=1, j\neq\delta(i,n-1)}^{q} h_{\delta(i,n-1),j}^{m_{\theta(i,n-1)q+j}} \\ &= \left(\prod_{j=1, j\neq\delta(i,n-1)}^{q} h_j^{m_{\theta(i,n-1)q+j}} \right)^{z_{\delta(i,n-1)}}, \end{aligned} \tag{3}$$

where $\theta(i,x) = \lceil i/q^{n-x} \rceil - 1$ and $\delta(i,x) = \lceil i/q^{n-x-1} \rceil - \theta(i,x)q$ are two functions used to find the cell sequence number and the related element's sequence number in the cell of the i-th message at x-th level. Furthermore, for $x = n-2, \ldots, \nu$, this algorithm computes

$$\begin{aligned} \Lambda_{i,x} &= \prod_{j=1, j\neq\delta(i,x)}^{q} h_{\delta(i,x),j}^{f(C_{x+1,\theta(i,x)q+j})} \\ &= \left(\prod_{j=1, j\neq\delta(i,x)}^{q} h_j^{f(C_{x+1,\theta(i,x)q+j})} \right)^{z_{\delta(i,x)}}, \end{aligned} \tag{4}$$

where $\theta(i,\cdot)$ and $\delta(\cdot)$ are the same functions defined in Eq. 3. In addition, this algorithm computes $M_{i,x}$ corresponding to $\Lambda_{i,x}$:

$$M_{i,n-1} = m_{\theta(i,n-1)q+\delta(i,n-1)}.$$

And, for $x = n-2, \ldots, \nu$,

$$M_{i,x} = f(C_{x+1,\theta(i,x)q+\delta(i,x)}).$$

where the value of $C_{x,y}$ can be computed with Eqs. 1 and 2. Finally, this algorithm outputs $\Lambda_i = \{(\Lambda_{i,x}, M_{i,x}) | x \in [\nu, n-1]\}$. Note that the input parameters μ and ν are used to control the scale of verifiable records.

– **VCT.Ver$_{\mathbf{PP}}$**(C, m, i, Λ_i, AC) Parse and verify the access control pair $AC = (\mu, \nu)$ like this: if $\mu \neq \lceil i/q^{n-\nu} \rceil$, this algorithm terminates and outputs 0 with an empty set $\emptyset$. Otherwise, set $Rst = (\alpha, \beta) = (-1, -1)$. For $x = n-1, \ldots, \nu$, set $\beta = x$, if $e(C_{x,\theta(i,x)+1} / h_{\delta(i,x)}^{M_{i,x}}, h_{\delta(i,x)}) = e(\Lambda_{i,x}, g)$, set $\alpha = 1$ and go on; else set $\alpha = \perp$ and break. And the end of the loop on x, if $\alpha = 1$ and $\beta = \nu$, the verification passes, this algorithm outputs $(1, \emptyset)$; else the verification fails which means some message must be tampered without authorization, and this algorithm can give a scope including the tampered message. Let $\Phi(C_{x,y})$ denote the set of messages that committed by $C_{x,y}$, then the tampered message is within

$$\Omega = \Phi(C_{\beta,\theta(i,\beta)+1}) - \Phi(C_{\beta+1,\theta(i,\beta+1)+1}), \tag{5}$$

and this algorithm outputs $(\perp, \Omega)$.

– **VCT.Update$_{\mathbf{PP}}$**(C, m_i, m_i', i) This algorithm updates all the commitments on the path from the leaf node (the i-th message) to the root node. The same as the above two algorithms, the processing of deepest non-leaf node and other nodes is slightly different. Firstly, this algorithm updates the commitment at the $(n-1)$-th level,

$$C'_{n-1,\delta(i,n-1)} = C_{n-1,\delta(i,n-1)} \cdot h_{\delta(i,n-1)}^{m'_{\theta(i,x)q+\delta(i,n-1)} - m_{\theta(i,x)q+\delta(i,n-1)}}.$$

Then, for $x = n-2, \ldots, 0$, other commitments are updated as described below:

$$C'_{x,\delta(i,x)} = C_{x,\delta(i,x)} \cdot h_{\delta(i,x)}^{f(C'_{x+1,\theta(i,x)q+\delta(i,x)}) - f(C_{x+1,\theta(i,x)q+\delta(i,x)})}.$$

Finally, this algorithm outputs C' and U, where C' is the updated commitment set $\{C'_{x,y} | x \in [0, n-1], y \in [1, q^x]\}$ and U is the update information (m_i, m_i', i).

– **VCT.ProofUpdate$_{\mathbf{PP}}$**$(C, \Lambda_j, m_i', i, U)$ Given U, a user who owns the original commitment set C and the proof set Λ_j for some message at position j can use this algorithm to update her proof set and commitment set. The update includes the following two stages:

  – Update the commitment set. This update is the same as the above constructed function VCT.Update$_{\mathrm{PP}}(\cdot)$, therefore, we omit the detail statement.
  – Update the proof set. For all $x = n-1, \ldots, 0$, if $\delta(j,x) = \delta(i,x)$, then $\Lambda_{j,x}$ stays the same; else for $x = n-1$, compute

$$\begin{aligned} \Lambda'_{j,n-1} &= \Lambda_{j,n-1} \cdot \left(h_{\delta(j,n-1)}^{m'_{\theta(j,x)q+\delta(j,x)} - m_{\theta(j,x)q+\delta(j,x)}} \right)^{z_{\delta(j,x)}} \\ &= \Lambda_{j,n-1} \cdot h_{\delta(j,x),\delta(i,x)}^{m'_{\theta(j,x)q+\delta(j,x)} - m_{\theta(j,x)q+\delta(j,x)}}, \end{aligned}$$

and for $x = n-2, \ldots, 0$, compute

$$\begin{aligned} \Lambda'_{j,x} &= \Lambda_{j,x} \cdot \left(h_{\delta(j,x)}^{f(C'_{x+1,\theta(j,x)q+\delta(j,x)}) - f(C_{x+1,\theta(j,x)q+\delta(j,x)})} \right)^{z_{\delta(j,x)}} \\ &= \Lambda_{j,x} \cdot h_{\delta(j,x),\delta(i,x)}^{f(C'_{x+1,\theta(j,x)q+\delta(j,x)}) - f(C_{x+1,\theta(j,x)q+\delta(j,x)})}. \end{aligned}$$

4 HVDB: hierarchical verifiable database with scalable updates

In this section, we extend the previous VDB framework to make it more practical by improving the computation and storage efficiencies of the setup phase, as well as by providing the abilities of the hierarchical verifiability and tampered record localizability.

4.1 Security requirements of HVDB

Generally, a VDB scheme should satisfy the properties of *security*, *correctness*, *efficiency*, and *accountability*. Among these properties, the first three ones have been brought out with the definitions of VDB in Benabbas et al. (2011) and Catalano and Fiore (2013), and the last one is introduced by Chen et al. quite recently in (Chen et al. 2015b; Chen et al. 2016). In the following, we extend the property of efficiency and introduce another new requirement termed as *localizability*.

In the previous works, the requirement of efficiency is used to emphasize that the VDB client should not be involved in plenty of expensive computation and storage, except for the initialization phase. However, in this paper, we take into account the efficiency of initialization, which may cause unacceptable computational and storage overheads for very large databases. The formal definition of the extended efficiency is as follows:

Definition 4 (*Efficiency (extended)*) A VDB scheme is efficient if for any database $DB \in [Q] \times \{0,1\}^*$, where $Q = poly(k)$, the computation and storage resources invested by the client, the server and other users are no more than $\mathcal{O}(Q)$.

In this paper, informally, we use the term localizability to emphasize the requirement that a verifier should be able to not only detect that the outsourced database is tampered, but also tell which record sub-sequence is tampered. In some way, this localizability requirement can be seen as the extensions and merging of the properties of security and accountability. The formal definition of localizability is as follows:

Definition 5 (*Localizability*) A VDB scheme is localizable if for any database $DB \in [Q] \times \{0,1\}^*$, where $Q = poly(k)$, when the outsourced database has been modified without authorization from the data owner, the verifiers can discover this exception and should be provided with some clues to find the sequence number of the tampered record.

4.2 HVDB Framework

According to the definition of VDB in Catalano and Fiore (2013), the new proposed HVDB is an extension of VDB to imply the new requirements of efficiency and localizability. The formal definition of HVDB is as follows:

Definition 6 A hierarchical verifiable database with scalable updates scheme (HVDB, for short) consists of the following five polynomial time algorithms, i.e., HVDB = (Setup, Query, Verify, ClientUpdate, ServerUpdate).

- **HVDB.Setup**$(1^k, q, DB, Q)$ Take as inputs the security parameter k, the cell size q and a database DB whose size is Q, this setup algorithm outputs the secret key SK to be distributed to the client (database owner) and kept private by it, the public key PK to be distributed to all users who want to verify the proofs, and the database encoded into S to be uploaded to the sever. And this algorithm is usually run by the client. Moreover, to overcome the computation and storage problems of setup in VDB, the size of the public key PK in this proposed HVDB should be only related to the parameter q which is much smaller than the database size Q.
- **HVDB.Query**(PK/SK, S, x, AC) Take as inputs the public key PK or the secret key SK, the encoded database S, the query index x and the access control parameter AC from a query requester (the client or an authorised user), this query algorithm firstly checks whether the access control parameter AC is valid, only if AC satisfies the access control policies, it outputs a pair of query results $\tau = (v, \pi)$; otherwise, it outputs a special error result $\perp$. In the proposed HVDB, AC is said valid if its owner has the permission to access the x-th record in the encoded database S, besides, this new introduced access control parameter AC is one of the key parameters used to achieve the hierarchical and scalable verifications as well as the localizability. And this algorithm is requested by the client or the authorised users and executed by the server.
- **HVDB.Verify**(PK, x, τ, AC) Take as inputs the public key PK, the query index x, the query result pair τ and the verifier's access control parameter AC, similar with the query algorithm, this verification algorithm also firstly check the validity of AC, and it outputs a value v with an empty set when both AC is valid and τ is verified correctly with respect to x; otherwise, it outputs an error result $\perp$ and an empty set or a non-empty range Ω indicating the access control AC is invalid or the position scope of the tampered record respectively.
- **HVDB.ClientUpdate**(SK, x, v') Take as inputs the private key SK, the update record index x and the new value

v' of the x-th record, if the record is updated successfully, this client update algorithm outputs a update token t'_x and an updated public key PK'; otherwise, it outputs an special error $\perp$. And this algorithm is run by the client to change the value of the x-th database record to v'.

– **HVDB.ServerUpdate**(PK, S, x, t'_x) Take as inputs the public key PK, the encoded database S, the update record index x and the update token t'_x, this server update algorithm outputs the new value v' of the x-th record in S. And this algorithm is run by the server to update the encoded database S according to the token t'_x from the client.

4.3 A concrete HVDB scheme from vector commitment tree

With the proposed HVDB framework, we construct a concrete HVDB scheme based on the VCT scheme presented in Sect. 3. The construction of HVDB is given in detail as follows:

– **HVDB.Setup**($1^k, q, \mathrm{DB}, Q$) Suppose that the original input database is organized in *key-value* pairs, and denote it as $\mathrm{DB} = \{(i, v_i)|i = 1, \ldots, Q\}$. Other database models can be treated as key-value through some transformations, and all the existing related schemes have been proposed based on the key-value model, therefore, we also choose this model to describe the construction of HVDB scheme. Firstly, acquire the public parameter set PP by invoking the KeyGen($\cdot$) algorithm in the previous concrete VCT scheme:

$$\mathrm{PP} \leftarrow \mathrm{VCT.KeyGen}(1^k, q, Q),$$

where the input parameters are the corresponding ones in this setup algorithm. Then parse the key-value pair records in database DB to form a tuple of Q values denoted as $(v_1, \ldots, v_Q)$, and, with the public parameter set PP, we acquire a pair of (C, aux) by invoking the $\mathrm{Com}_{\mathrm{PP}}(\cdot)$ algorithm in VCT scheme:

$$(C, aux) \leftarrow \mathrm{VCT.Com}_{\mathrm{PP}}(v_1, \ldots, v_Q).$$

Finally, we construct the three output results with the couple of these intermediate parameters:

$$\begin{cases} \mathrm{PK} &= (\mathrm{PP}, C), \\ \mathrm{S} &= (\mathrm{PP}, aux, DB), \\ \mathrm{SK} &= \perp. \end{cases}$$

The empty secret key problem has been discovered and solved in Chen et al. (2015b), for simplicity, we still follow the expression of (Catalano and Fiore, 2013). Furthermore, our scheme can be updated to resist the FAU attack with the idea from Chen et al. (2015b), and the specific details are omitted here.

– **HVDB.Query**(PK, S, x, AC) Let $v_x = \mathrm{DB}(x)$ denote the x-th record in database DB, and parse the access control parameter as $AC = (\mu, \nu)$ which represents that the query requestor can access to all the elements belonging to the ν-th commitment at the μ-th level. We acquire the proof set of the x-th record by invoking the $\mathrm{Open}_{\mathrm{PP}}(\cdot)$ algorithm in VCT scheme:

$$\Lambda_x \leftarrow \mathrm{VCT.Open}_{\mathrm{PP}}(v_x, x, aux, (\mu, \nu)),$$

where all the inputs are fetched from this query algorithm directly or indirectly. Then, the output result τ is constructed with the following equation:

$$\tau = (v_x, \Lambda_x).$$

– **HVDB.Verify**(PK, x, τ, AC) This database query result verification is completed by invoking the $\mathrm{Ver}_{\mathrm{PP}}(\cdot)$ algorithm in VCT scheme:

$$(RstValue, RstSet) \leftarrow \mathrm{VCT.Ver}_{\mathrm{PP}}(C, x, v_x, \Lambda_x).$$

And then process the temporary result pair (*RstValue*, *RstSet*) as follows:

  – Case 1: If $RstValue = 0$ and $RstSet = \emptyset$, this algorithm outputs $(\perp, \emptyset)$ which indicates that the verifier has no permission to access the x-th record.
  – Case 2: If $RstValue = 1$ and $RstSet = \emptyset$, this algorithm outputs $(v_x, \emptyset)$ where the v_x is the x-th record value in the database parsed from τ, and this output result indicates that the database is unmodified.
  – Case 3: If $RstValue = \perp$ and $RstSet \neq \emptyset$, this algorithm outputs $(\perp, \Omega)$, which means that some database located in Ω is tampered without authorization. As mentioned above, the symbol $\perp$ stands for a special error and the interval Ω is from Eq. 5.

Note that our proposed HVDB scheme is publicly verifiable, for this verification algorithm takes the public key PK as input.

– **HVDB.ClientUpdate**(SK, x, v'_x)The client update in HVDB is done by invoking the $\mathrm{Update}_{\mathrm{PP}}(\cdot)$ algorithm in VCT scheme. However, before executing the update, the integrality of the current database must be confirmed, that can be achieved with the above constructed Query($\cdot$) and Verify($\cdot$) algorithms. This update is done as follows: Firstly, the client retrieves the x-th record in the database from the server by invoking the Query($\cdot$) algorithm in HVDB:

$$\tau \leftarrow \mathrm{HVDB.Query}(\mathrm{PK}, \mathrm{S}, x).$$

With the query result τ, the client confirms that the outsourced database is unmodified by invoking the

integrality verification algorithm Verify(·) in HVDB and checking whether its output satisfies the Case 2 listed in the above description of Verify(·):

$$\begin{cases} (v_t, s_t) & \leftarrow \text{HVDB.Verify}(\text{PK}, x, \tau), \\ (v_t, s_t) & = (v_x, \emptyset). \end{cases}$$

If the validation succeeds, this update algorithm invokes the $\text{Update}_{\text{PP}}(\cdot)$ algorithm in VCT scheme:

$$(C', U) \leftarrow \text{VCT.Update}_{\text{PP}}(C, v_x, v'_x, x),$$

and outputs the new public key PK' = (PP, C') as well as the update token t'_x = (PK', v'_x, U); otherwise, this algorithm is disrupted and return the special error $\perp$. However, the existing works do not explain how other HVDB participators to update their old public keys and commitments, therefore, we present some advices here. In practice, the public key holder(s) can update PK by employing the ProofUpdate(·) algorithm in VCT scheme or replace the old PK with the new PK$'$ from the client update algorithm. And other users who own the old commitment set and proof sets of some records can also employ the ProofUpdate(·) algorithm to update their verification materials, or they can update these materials by executing the Query(·) algorithm again.

– **HVDB.ServerUpdate**(PK, S, x, t'_x) After the client finishes updating the x-th record, once the database storage server receives the update token t'_x,[4] it can start updating the related values stored in it: the server firstly parse the update token t'_x as (PK', v'_x, U), then replaces the x-th database record with v'_x, and adds the update information U to *aux* in S.

4.4 Security analysis

We prove that the proposed HVDB satisfies the requirements of security, correctness, efficiency, accountability and localizability as follows.

Theorem 1 T*he proposed HVDB scheme is secure*.

Proof As proved in Catalano and Fiore (2013) and Chen et al. (2015b), for any polynomial-time adversary $\mathcal{A}$, his success probability in the experiment $\text{Exp}_{\mathcal{A}}^{VDB}[DB, k]$ is ϵ which is negligible. Actually, our HVDB scheme can be seen as a series of independently repeated operation with the existing VDB scheme, therefore, in the same environment, $\mathcal{A}$'s success probability will be no more than ϵ^h which is also negligible. □

Theorem 2 *The proposed HVDB scheme is correct*.

Proof Assume that the server is honest, according to the construction of our VCT-based HVDB scheme, if the output $\tau = (v_x, \Lambda_x)$ of the query algorithm and the access control parameter AC can pass the examination of the verification algorithm, the output of the verification algorithm is always the desired correct x-th record v_x in the outsourced database. □

Theorem 3 *The proposed HVDB scheme is efficient*.

Proof It is obvious that the computation and storage resources invested in every HVDB algorithm by the client and the server are no more than $\mathcal{O}(Q)$. Especially, the computation and storage efficiencies of the Setup(·) algorithm are only dependent on the cell size q instead of the database size Q, furthermore, the client's computation complexities are constant in the Verify(·) and Update(·) algorithms. □

Theorem 4 *The proposed HVDB scheme is accountable*.

Proof According to the construction of HVDB scheme, if the query result $\tau = (v_x, \Lambda_x)$ is verified to be valid, the verification algorithm in HVDB gives out $(v_x, \emptyset)$ which indicating the query result is correct and the database is integral; otherwise, both the outputs $(\perp, \emptyset)$ and $(\perp, \Omega)$ mean that the verification is failed, the first one is permission error, and the second one is database error. □

Theorem 5 *The proposed HVDB scheme is localizable*.

Proof When the verification of τ is failed, given the range Ω containing the tampered record, the verifier can easily narrow the range to localize it within a cell by checking records in Ω rather than traversing the entire database. □

4.5 Efficiency analysis and comparison

In this subsection, we compare the proposed HVDB scheme with two other typical VC-based VDB schemes denoted as *CF* and *CLHML* respectively. The CF scheme is proposed by Catalano and Fiore (2013) with the primitive of vector commitment , it is the first VC-based VDB scheme. After that, the CLHML scheme is proposed by Chen et al. (2016), it finds and fixes the FUA security weakness of CF scheme, furthermore, it introduces the new requirement of accountability into the design of VDB

[4] For security, we suggest that a verification of t'_x should be executed here, although few of the previous works have considered this verification.

Table 1 Comparison among Three VC-based Schemes

Schemes	CF	CLHML	HVDB
Security assumption	CDH	CDH	CDH
Verifiability	Public	Public	Public/hierarchical
Accountability	No	Yes	Yes
Localizability	No	No	Yes
Scalable update	No	No	Yes
Public key size	$\mathcal{O}(Q^2)$	$\mathcal{O}(Q^2)$	$\mathcal{O}(q^2)$
Setup complexity	$\mathcal{O}(Q^2)$	$\mathcal{O}(Q^2)$	$\mathcal{O}(Q)$
Query complexity	$(Q-1)(M+E)$	$(Q-1)(M+E)$	$(q(Q-1)/(q-1)-n)(M+E)$
Verify complexity	$M+E+I+2P$	$2M+E+I+4P$	$n(M+E+I+2P)$
Update complexity (Full)	$(Q+1)(M+E)+I+2P$	$(Q+2)(M+E)+M+2I+4P$	$(q(Q-1)/(q-1)+n)(M+E)+n(I+2P)$

framework and the contribution of VDB scheme. Both of them are classic VDB schemes, they have played important roles in the development of VDB and promoted the study of VDB (including ours), therefore, here we analyze the abilities and performances of these schemes from different aspects in theory, and present the comparison among the three schemes in Table 1.

From Table 1, we have the following findings about the functions of the three schemes. Firstly, all of these three schemes support public verification based on CDH security assumption and vector commitment schemes. Secondly, our scheme can localize the tampered record when the verification algorithm is failed, while the other two can only tell the verifier that the outsourced database is modified without owner's authorization. Thirdly, only our scheme allows hierarchical verification and scalable update which means the accessible scopes can be controlled depending on user authorities.

Furthermore, we quantify the theoretical performances of the three schemes, and the results are listed as time complexities in Table 1, where the symbols of M, E, I and P are respectively used to denote the multiplication in $\mathbb{G}_1$ (or $\mathbb{G}_2$), exponentiation in $\mathbb{G}_1$, inverse in $\mathbb{G}_1$ and computation of the pairing, besides, other operations, like addition in $\mathbb{G}_1$, are omitted. In addition, Q and q respectively represent the size of outsourced database and the size of each divided database records cell, and n equals to $\lceil \log_q Q \rceil$. It can be seen in Table 1 that the public key size of our scheme is sharply reduced (see also Fig. 2, which is only related to the cell size q, rather than the database size Q like the other two schemes, and accordingly, the efficiency of **Setup** algorithm is also significantly improved. Moreover, we find that except **Query** algorithm, the complexities of our **Verify** and **Update** algorithms are related to the VCT height n, but that would not lower the practical efficiency of our scheme, the reasons are as follows: the height n can be kept small enough by choosing q properly in practice, and our tree structure based scheme can be naturally speeded up by employing the parallel processing technologies.

5 Performance evaluation

In this section, we evaluate the performance of the above theoretically analyzed CF, CLHML and HVDB schemes in a simulation environment. These schemes are implemented based on the Stanford PBC library (The Paring-Based Cryptography Library), and all the experiments were executed on the same personal computer whose operation system is Ubuntu 14.04 Desktop (64-bit), processor is Intel Xeon CPU E5-1620 v3 3.50 GHz and main memory size is 16 GB . The results of performance evaluation experiments are illustrated in Figs. 2, 3 and 4.

Firstly, we show the relationships between the public key sizes of the tree schemes and the database size Q in Fig. 2, and we want to emphasize the following two aspects: (1) As

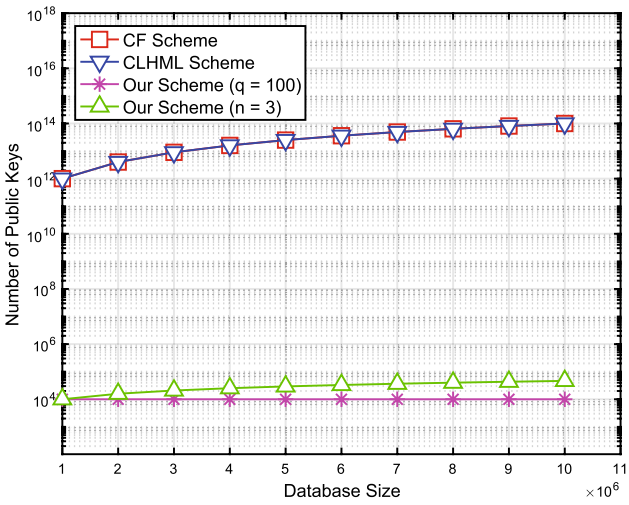


Fig. 2 Comparison of pubic key sizes

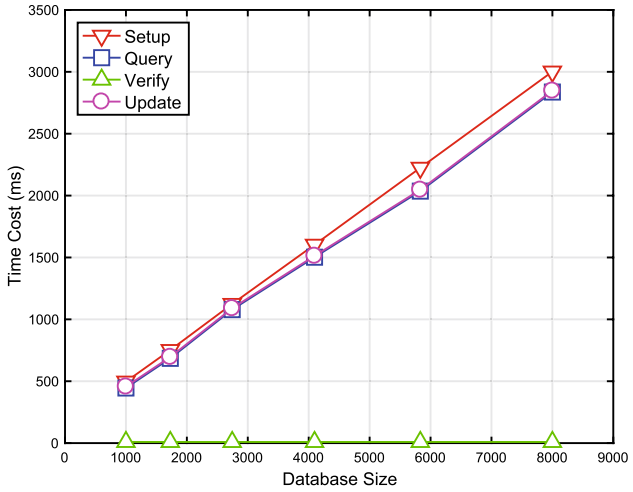


Fig. 3 Performance of our scheme ($n = 3$)

analyzed in theory, it can be seen that the public key sizes of CF scheme and CLHML scheme increase sharply when the database size gets larger. (2) According to Sect. 3, there are two way to satisfy the relationship among Q, q and n, thus there are two curves for our scheme. They illustrate that the public key size of our HVDB scheme is invariable if the cell size q is fixed (here we set it to be 100); otherwise, if we fix the height n (here we set it to be 3), the public key size also increases with database size, but it is much smaller than the CF and CLHML schemes for the same database size.

Then, we test our HVDB scheme by respectively measuring its time costs in **Setup**, **Query**, **Verify** and **Update** algorithms for different database size Q . For conciseness, we fix $n = 3$ in the rest of the experiments. The test result is illustrated in Fig. 3, which shows that the time costs of all algorithms except **Verify** grow with Q. For the practical advantage of our scheme over other VDB schemes cannot be demonstrated obviously by only testing our scheme singly, we consequently test and compare the time costs of the three schemes.

In Fig. 4, we use four separate sub-figures to respectively illustrate the time costs of **Setup**, **Query**, **Verify** and **Update** algorithms in the three schemes for different database size Q. From these sub-figures in Fig. 4, it can be clearly seen that our scheme is much more acceptable in practice than the other two schemes. Especially, as shown in Fig. 4a, compared with CF and CLHML schemes, the time costs of **Setup** algorithm in our scheme is dramatically reduced. From Fig. 4b, d, we find that the time costs of **Query** and **Update** algorithms are both decreased over 50 percent relative to the other two schemes. Note that, for the tree structure of our HVDB scheme, it is naturally to speed the query up with parallel processing technologies, and the multi-threading method is used in our scheme. Besides, different from the other three sub-figures, Fig. 4c shows three curves of **Verify** algorithm in our scheme corresponding to the situations for $n = 2, 3, 4$. Because the efficiency of verification algorithm is quite high and independent of Q, even though the time costs can be larger than the others two schemes when n turns bigger, we think our scheme is still acceptable in practice.

6 Conclusion

The primitive of VDB can be used to achieve the aim of securely outsourcing a very large and dynamic database. However, it seems that all previous VDB schemes are presented in the amorted computation model, which requires a prohibitively expensive setup phase (though it is done only one time). In this paper, we focus on the verifiable database schemes, especially the VC-based ones. Many VDB schemes were proposed in the past few years, however, these existing VDB schemes suffer the inefficiency and public key explosion problems, and they do not support hierarchical verification and tamped record localization. Therefore, to overcome these problems, we redefine the original conception of efficiency in VDB and introduce a new security requirement of localizability into it. Consequently, we extend the primitive of VC to VCT in a recursive way, propose a new HVDB framework based on VCT, and construct a concrete HVDB scheme. Our analysis and performance evaluation show that, compared with the previous schemes, the proposed HVDB scheme's public keys number is independent of the outsourced database size, its initialization efficiency is improved remarkably, and it realizes the functions of hierarchical verification and tampered record localization. In the future work, we will focus on the secure outsourcing of the data stream.

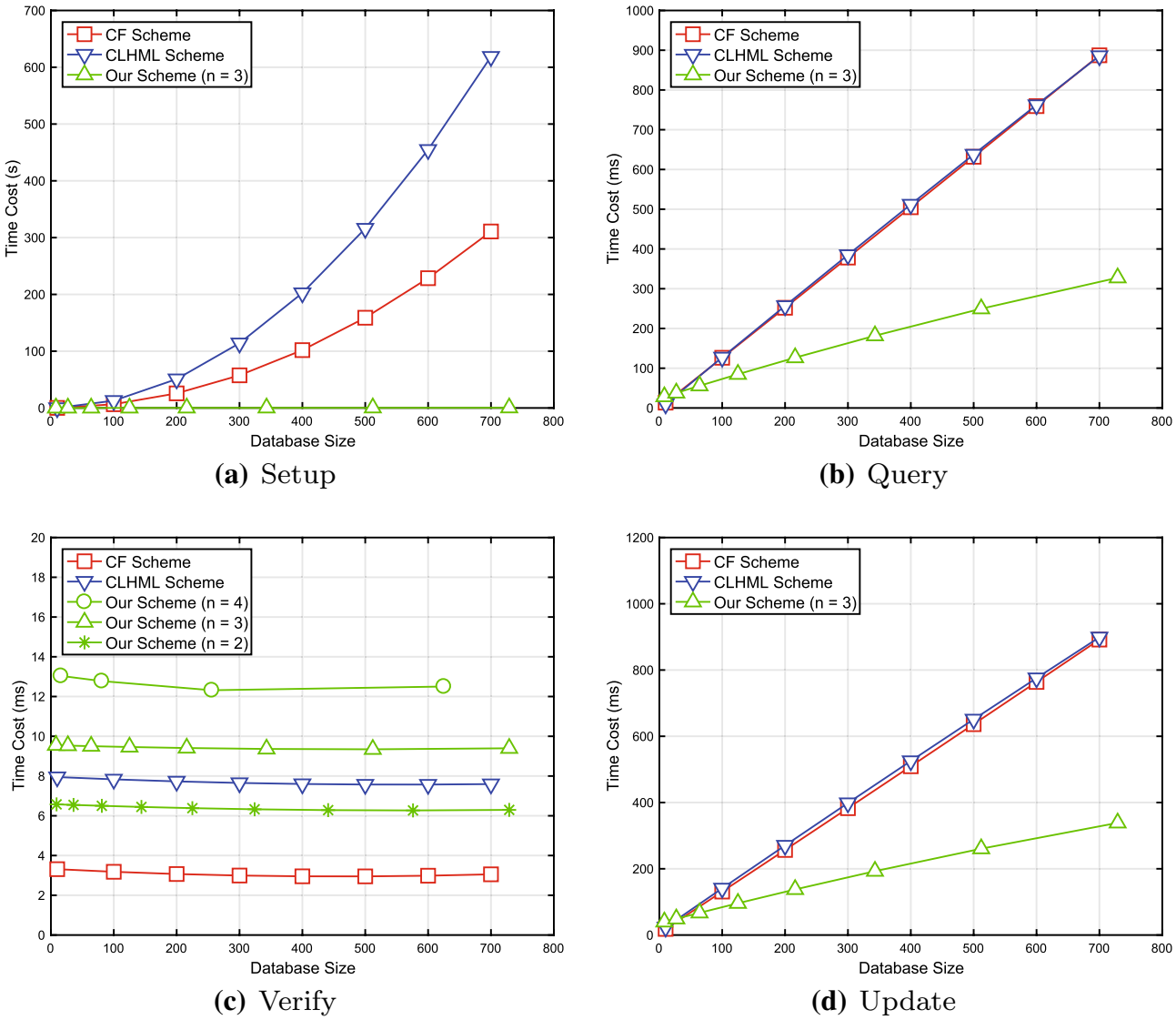


Fig. 4 Performance comparison among three schemes

Acknowledgements This work is supported by 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

Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I et al (2010) A view of cloud computing. Commun ACM 53(4):50–58

Benabbas S, Gennaro R, Vahlis Y (2011) Verifiable delegation of computation over large datasets. In: Annual Cryptology Conference, Springer, pp 111–131

Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616

Catalano D, Fiore D (2013) Vector commitments and their applications. In: Public Key Cryptography, Springer, pp 55–72

Chang X, Yang Y (2017) Semisupervised feature analysis by mining correlations among multiple tasks. IEEE Trans Neural Netw Learn Syst 28(10):2294–2305

Chen X, Li J, Huang X, Li J, Xiang Y, Wong DS (2014a) Secure outsourced attribute-based signatures. IEEE Trans Parallel Distrib Syst 25(12):3285–3294

Chen X, Li J, Ma J, Tang Q, Lou W (2014b) New algorithms for secure outsourcing of modular exponentiations. IEEE Trans Parallel Distrib Syst 25(9):2386–2396

Chen X, Huang X, Li J, Ma J, Lou W, Wong DS (2015a) New algorithms for secure outsourcing of large-scale systems of linear equations. IEEE Trans Inf Forensics Secur 10(1):69–78

Chen X, Li J, Huang X, Ma J, Lou W (2015b) New publicly verifiable databases with efficient updates. IEEE Trans Dependable Secure Comput 12(5):546–556

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

Chow R, Golle P, Jakobsson M, Shi E, Staddon J, Masuoka R, Molina J (2009) Controlling data in the cloud: outsourcing computation without outsourcing control. In: Proceedings of the 2009 ACM workshop on Cloud computing security, pp 85–90

Gennaro R, Gentry C, Parno B (2010) Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Annual Cryptology Conference, Springer, pp 465–482

Gentry C, Halevi S (2011) Implementing gentry's fully-homomorphic encryption scheme. EUROCRYPT, Springer 6632:129–148

Goldwasser S, Kalai YT, Rothblum GN (2016) Delegating computation: interactive proofs for muggles. J ACM 62(4):1–64

Li H, Lin X, Yang H, Liang X, Lu R, Shen X (2014a) Eppdr: an efficient privacy-preserving demand response scheme with adaptive key evolution in smart grid. IEEE Trans Parallel Distrib Syst 25(8):2053–2064

Li H, Yang Y, Dai Y, Bai J, Yu S, Xiang Y (2017a) Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data. IEEE Transactions on Cloud Computing PP (99):1–1. https://doi.org/10.1109/TCC.2017.2769645

Li J, Huang X, Li J, Chen X, Xiang Y (2014b) Securely outsourcing attribute-based encryption with checkability. IEEE Trans Parallel Distrib Syst 25(8):2201–2210

Li J, Li J, Chen X, Jia C, Lou W (2015) Identity-based encryption with outsourced revocation in cloud computing. IEEE Trans Comput 64(2):425–437

Li P, Li J, Huang Z, Gao CZ, Chen WB, Chen K (2017b) Privacy-preserving outsourced classification in cloud computing. Cluster Computing pp 1–10. https://doi.org/10.1007/s10586-017-0849-9

Liu X, Zhang Y, Wang B, Yan J (2013) Mona: secure multi-owner data sharing for dynamic groups in the cloud. IEEE Trans Parallel Distrib Syst 24(6):1182–1191

Miao M, Wang J, Ma J, Susilo W (2017) Publicly verifiable databases with efficient insertion/deletion operations. J Comput Syst Sci 86:49–58

Motahari-Nezhad HR, Stephenson B, Singhal S (2009) Outsourcing business to cloud computing services: opportunities and challenges. IEEE Internet Comput 10(4):1–17

Pallis G, Vakali A (2006) Insight and perspectives for content delivery networks. Commun ACM 49(1):101–106

Reingold O, Rothblum GN, Rothblum RD (2016) Constant-round interactive proofs for delegating computation. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, pp 49–62

Rittinghouse JW, Ransome JF (2016) Cloud computing: implementation, management, and security. CRC Press, Inc., Boca Raton

Shen J, Zhou T, Chen X, Li J, Susilo W (2018) Anonymous and traceable group data sharing in cloud computing. IEEE Trans Inf Forensics Secur 13(4):912–925

Spagna S, Liebsch M, Baldessari R, Niccolini S, Schmid S, Garroppo R, Ozawa K, Awano J (2013) Design principles of an operator-owned highly distributed content delivery network. IEEE Commun Mag 51(4):132–140

Stergiou C, Psannis KE, Kim BG, Gupta B (2018) Secure integration of iot and cloud computing. Future Gen Comput Syst 78:964–975

Stojmenovic I, Wen S, Huang X, Luan H (2016) An overview of fog computing and its security issues. Concurr Comput: Pract Exp 28(10):2991–3005

Takabi H, Joshi JB, Ahn GJ (2010) Security and privacy challenges in cloud computing environments. IEEE Secur Priv 8(6):24–31

Tang S, Li X, Huang X, Xiang Y, Xu L (2016) Achieving simple, secure and efficient hierarchical access control in cloud computing. IEEE Trans Comput 65(7):2325–2331

Vimercati SDCD, Foresti S, Jajodia S, Paraboschi S, Samarati P (2007) A data outsourcing architecture combining cryptography and access control. In: ACM Workshop on Computer Security Architecture, pp 63–69

Vu V, Setty S, Blumberg AJ, Walfish M (2013) A hybrid architecture for interactive verifiable computation. In: IEEE Symposium on Security and Privacy (SP), pp 223–237

Wang C, Ren K, Lou W, Li J (2010) Toward publicly auditable secure cloud data storage services. IEEE Netw 24(4):19–24

Wang C, Chow SS, Wang Q, Ren K, Lou W (2013) Privacy-preserving public auditing for secure cloud storage. IEEE Trans Comput 62(2):362–375

Wen S, Haghighi MS, Chen C, Xiang Y, Zhou W, Jia W (2015) A sword with two edges: propagation studies on both positive and negative information in online social networks. IEEE Trans Comput 64(3):640–653

Wu J, Ping L, Ge X, Wang Y, Fu J (2010) Cloud storage as the infrastructure of cloud computing. In: International Conference on Intelligent Computing and Cognitive Informatics (ICICCI), IEEE, pp 380–383

Xiao Z, Xiao Y (2013) Security and privacy in cloud computing. IEEE Commun Surv Tutor 15(2):843–859

Zhang L, Safavi-Naini R (2014) Verifiable delegation of computations with storage-verification trade-off. In: European Symposium on Research in Computer Security, Springer, pp 112–129

Zhang Y, Chen X, Li J, Wong DS, Li H, You I (2017) Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing. Inf Sci 379:42–61