Information Sciences 000 (2016) 1–17

Contents lists available at ScienceDirect

Information Sciences

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

Tell me the truth: Practically public authentication for outsourced databases with multi-user modification

Wei Song[a,b,c], Bing Wang[c], Qian Wang[a,*], Zhiyong Peng[a,b], Wenjing Lou[c]

[a] The State Key Laboratory of Software Engineering, Wuhan University, China
[b] School of Computer Science, Wuhan University, China
[c] Department of Computer Science, Virginia Polytechnic Institute and State University, USA

ARTICLE INFO

Article history:
Received 1 January 2016
Revised 12 May 2016
Accepted 11 July 2016
Available online xxx

Keywords:
Outsourced database
Public authentication
Homomorphic verifiable tags

ABSTRACT

With the advent of cloud computing, outsourcing databases to remote cloud servers provide the elastic, flexible and affordable data management services for the Internet users. The cloud users can create, store, access and update the remote outsourced databases just as they are using the database system locally. However, unlike storing data in a fully-controlled local database, storing data in a remote cloud server raises data privacy and security concerns, i.e., the correctness and completeness of the query results. Although some solutions have been proposed to address this problem, they do not scale well when multiple users update the remote outsourced database for two major reasons. First, the existing schemes mainly use the authenticated data structure (ADS) to provide the verification service which incurs expensive computation cost, especially when modifications are made to the database. Second, the data owner has to remain online all the time to participate in generating signatures for the modified data. Consider the fact that the outsourced databases involve lots of heavy multi-user modification operations, the existing solutions are not practical from the efficiency perspective. To address the above concerns, in this paper, we first propose a novel and efficient signature scheme which features additive homomorphic operations. On top of that, we further propose a new and practical mechanism for correctness and completeness verification with the support of multi-user modifications and without requiring an always-online data owner. Finally, we prove the security of our scheme under the well-known Computational Diffie–Hellman assumption and conduct extensive experiments to evaluate the performance of our scheme. The experimental results show that our scheme outperforms the existing solutions.

1. Introduction

In cloud computing, the users and the enterprises can lease the computing and storage resources provided by the powerful cloud service provider. Under this new IT Paradigm, database outsourcing [6] is considered as a prominent service model by providing an elastic, flexible and affordable solution for the organizations and enterprises to maintain their database services. As can be seen in recent years, more and more enterprises begin to move their data management services to the cloud for the easy management and low cost. The successful real-world examples such as Amazon Relational Database

* Corresponding author.
E-mail addresses: songwei@whu.edu.cn (W. Song), bingwang@vt.edu (B. Wang), qianwang@whu.edu.cn (Q. Wang), peng@whu.edu.cn (Z. Peng), vjlou@vt.edu (W. Lou).

http://dx.doi.org/10.1016/j.ins.2016.07.031
0020-0255/

(RDS)[1], Azure[2], and EnterpriseDB[3] enable users around the world to share and update their outsourced data anywhere anytime. While enjoying all the benefits, the users have to worry about their data security and privacy because the outsourced database service providers are often not in the same trusted domain with cloud users. As a result, for the correct utilization of the outsourced data at the cloud, the correctness and completeness of the query results over the outsourced data should be verified from the users' perspective.

To this end, a number of solutions have been proposed based on the various technologies [4,7,8,10–15,17,19,20,22,23,25,26,28] in recent years. However, most of them only consider the cases that the outsourced data is static or updated only by a single data owner. Others [8,17] require the data owner to sign the verification data structures for every modification. This mode requires the data owner to stay online all the time to support multi-user modification. On the other hand, the existing works commonly utilize the authenticated data structures (ADS), which incurs expensive computation cost for every data modification operation. Obviously, such verification approaches do not scale well when the frequency of the data modification operations increases. Given the fact that the outsourced databases involve a large amount of multi-user modification operations, the existing solutions are not practical from the efficiency perspective.

Consider the following application scenario. A supermarket such as Walmart has many branch stores. This supermarket outsources the management of its sales data to the cloud. Each branch store is able to upload and update the sales data stored on the cloud. At the same time, the clients such as the analysts, the managers and so on need to access the sales data contributed by multiple branch stores. However, the cloud is not fully trusted (i.e., it may be malicious or become prey to an external attack). So, the problem is how can the client efficiently and correctly verify the results generated and returned from the cloud, particularly when the outsourced data is modified by multiple users frequently? Therefore, there still lacks a practical correctness and completeness verification method for the outsourced data with multi-user modifications. From the users' perspective, a practical verification scheme should have the following properties:

- **Correctness verification**: The outsourced data stored at the remote server may be polluted by the attackers. But, the cloud server may hide the fact for the economic reasons and its reputation. So, the user should be able to detect the polluted data in the query results returned from the outsourced database server.
- **Completeness verification**: The outsourced database server is not fully trusted. To save the resources, it is possible that the outsourced database server only returns partial results or does not execute the query over the entirely outsourced data at all. The user should be able to detect such misbehavior by verifying the completeness of the query results.
- **The support of multi-user modification**: One of the main advantages of the outsourced database is to allow the multiple online users to share the data. Therefore, the practical integrity verification mechanism should allow the user to efficiently verify the integrity of the query results, even the outsourced data is contributed by the multiple users.
- **Public verification**: To improve the usability of the verification scheme, it should be able to allow anyone in the system to verify the integrity of the query results without storing some meta data locally or retrieving the entire data collection from cloud, even if some outsourced data have been modified and signed by multiple users. In addition, the system should not depend on any special entity, e.g., an always online data owner, to execute the verification.
- **High efficiency**: The user should be able to verify the query results of the outsourced databases with higher communication and computation efficiency than the approach of downloading the outsourced data and executing the query locally.

Keep the above goals in mind, in this work, we propose an efficient signature scheme based on the bilinear map to support the integrity verification of the outsourced databases with multi-user modification. Our solution also supports the public verifiability, in which the authorized user signs the data by its private key after the modification operation. The user then can use the public key to verify the query results even the outsourced data have been modified and signed by multiple users. To efficiently support multi-user modification, the proposed verification scheme is designed to enable the user to independently sign the data without an always-online data owner and/or the third party trusted entity. The main contributions in this paper can be summarized as follows:

- We for the first time propose an efficient and practical scheme to support the public verification of the outsourced databases with multiple writers. Compared with the existing solutions, our scheme is highly efficient, secure and scalable.
- We evaluate the performance of our scheme by numerical analysis, which illustrates that the proposed scheme is indeed an efficient integrity verification solution for outsourced databases. Moreover, we formally prove that the proposed scheme is secure.
- We fully implement a prototype of the proposed scheme and evaluate its performance through the extensive experiments. Our experimental results further validate its effectiveness and efficiency.

The rest of our paper is organized as follows. In the next section, we discuss the related work. We set up the system model in Section 3. Then we introduce several cryptographic primitives used in this paper in Section 4. Section 5 details our signature scheme, based on which the integrity verification mechanism is presented in Section 6. We analyze the perfor-

[1] http://aws.amazon.com/.
[2] http://azure.microsoft.com/.
[3] http://www.enterprisedb.com/.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

mance and prove the security of the proposed scheme in Section 7. In Section 8, we carry out some experiments to evaluate the performance of our scheme. Finally, our paper concludes in Section 9.

2. Related work

The outsourcing databases [6,18] provide the elastic, flexible and affordable solution for the enterprises and Internet users to manage their data. While enjoying all the benefits, the users have to worry about their data security and privacy. The searchable encryption schemes [5,24] have been proposed to protect the privacy of the outsourced data. However, these technologies can not ensure the integrity of the outsourced data services. The problem of verifying the integrity of the outsourced data at an untrusted server has been explored extensively in the existing literature [4,7,8,10–15,17,19,20,22,23,25,26,28]. In most of these works [8,17], a single writer who executes the data update operations is assumed. Other solutions [7,13,22,23] have been proposed to ensure the integrity of outsourced databases using Merkle Hash Tree (MHT) [9]. The main problem with the MHT structure is its expensive maintenance cost. As a result, these solutions cannot efficiently support frequent data modifications. Moreover, they require the data owner to keep online and participate signing the data for every modification operation. Li et al. [8] offered a solution for the single writer case, where an embedded Merkle tree (EMB tree) is designed for integrity verification. An EMB tree is an embedded B+ tree similar to the Merkle Hash tree. The root hash of the tree is made available to the clients. With the help of this root hash, the clients can verify the correctness and completeness of their query results. Updates are performed only by the data owner and the updated root label is then distributed to the clients. Sion [17] extended the "ringer" concept to provide the proofs of arbitrary query execution in the outsourced databases for the read-only queries. To support data updates, it requires additionally preliminary processing. Zheng et al. [29] designed the Authenticated Outsourced Ordered Data Set and the Homomorphic Linear Tag to verify the integrity of query results for outsourced dynamic databases. This method allows aggregate queries and flexible join queries, but the update operations are performed by both the data owner and the cloud server. Jain et al. [7] used the MHT to provide the assured provenance for all update transactions to force an untrusted server to provide the trustworthy data services. Other research efforts have been put on the data provenance and the tamper-proofing of the outsourced data [4,11,19,20,25,28], but most of them only focus on the data correctness verification.

To reduce the maintenance cost of the authenticated data structure at the outsourced database server, Papadopoulos et al. [15] proposed SAE to separate the authentication from the query execution. Attila [27] designed a signature scheme based on the immutable signatures to provide authentication and integrity for the outsourced database applications. Mykletun et al. [10] used the notion of signature aggregation to achieve the bandwidth-efficient integrity verification of query results. However, it only ensures the correctness of queries. Narasimha et al. [12] extended [10] to provide the completeness guarantee. It provides a solution for the multi-user model and allows the updates from multiple clients using the BGLS [3] signature scheme. However, this work needs an online data owner to sign the authenticated data structure. Wang et al. [21] proposed a verifiable auditing scheme for the outsourced database, which can simultaneously achieve the correctness and completeness of search results. In this scheme, the outsourced data only can be updated by the data owner, so it is still impractical for the case that the outsourced date can be modified by multiple users. Yang et al. [26] proposed the algorithms for the authenticated join processing. Papadopoulos et al. [14] investigated the authenticated multi-dimensional range queries over the outsourced databases. Although this solution introduced the bucket structure to improve the efficiency of rebuilding the authentication data structure for the update operations, it still requires an online data owner. To the best of our knowledge, all the existing solutions do not support practical integrity verification for the outsourced databases with multi-user modifications.

3. Problem statement

3.1. System model

In this paper, we consider a cloud-based database outsourcing system in Fig. 1. The system consists of two entities: the users and the outsourced database. The user outsources its data to the cloud outsourced database system which provides the outsourcing data management services. The outsourced data will be subject to a variety of security threats as defined in Section 3.3. To achieve the scalable and flexible cloud outsourced data services, we do not differentiate between users in the system model. It implies that all of them can publish the outsourced data and modify them, even the outsourced data are originally uploaded by others. The outsourced data stored at the cloud server maybe be polluted due to the external attacks or hardware failures. However, the cloud may hide the fact that some data have been tampered for economic reasons and its reputation. Moreover, the cloud-based outsourced database system may have some 'lazy behaviors', i.e., the cloud server does not correctly execute the queries over the entire outsourced data collection for saving its resources. Therefore, the user would like to constantly verify the integrity of the query results with the outsourced data contributed by multiple users.

To ensure the integrity of the outsourcing service, the attribute value is signed by the user who last modifies it.[4] Whenever the user attempts to verify the correctness and completeness of the query result, it generates a challenge message and

[4] Note that, we use a single searchable attribute as the example to present our scheme, but it is straight-forward to extend our scheme to the relation with multiple attributes.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

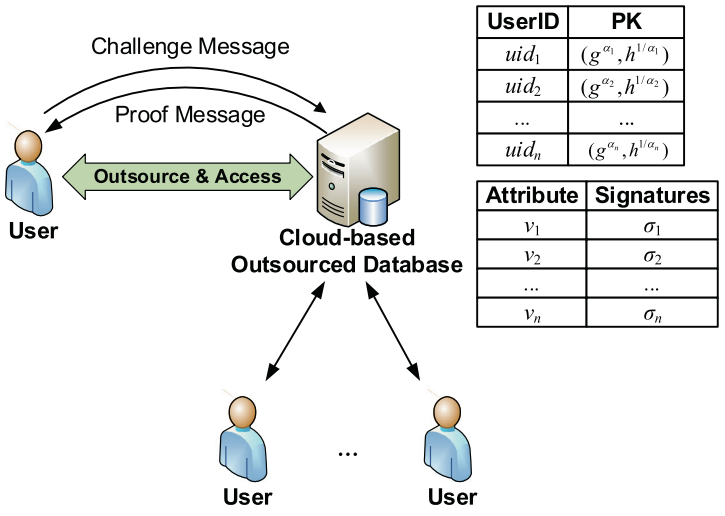


Fig. 1. The system model of a cloud-based database outsourcing system. Besides the outsourced data, the outsourced database server stores all the users' identifications and their public keys.

sends it to the cloud server. Then, while the cloud server executes the query, it also outputs the proof message to reply to the user's challenge. Note that, to enable the practical and efficient verification for the outsourced databases with multi-user modification, every user is able to independently sign the data modified by it without the participation of the data owner or others. Moreover, the cloud server is able to generate a proof message based on the homomorphic property of the signatures for multiple tuples even they are generated by the different users.

Our verifiable signature scheme for the outsourced database consists of the polynomial algorithms: SetUp, KeyGen, Sign, Verify, ChalGen, ProofGen, and VerifyProof. In what follows, we present these algorithms.

- $\mathsf{SetUp}(1^{\lambda}) \rightarrow (e, p, \mathbb{G}_1, \mathbb{G}_2, g, h, \omega, H)$ is run by the cloud server to initialize the outsourced database system. It takes a security parameter λ as input and returns the global security parameters for the system.
- $\mathsf{KeyGen}(1^{\kappa}) \rightarrow (pk, sk)$ is run by a new user. It takes a security parameter κ as input and returns the public key pk and secret key sk for the new user.
- $\mathsf{Sign}(v, tid, p_{tid}, s_{tid}, sk) \rightarrow \sigma_v$ is run by the user to generate the signature for the new data v. Given the new data v, the tuple identifier tid of v, and the tuple identifiers p_{tid}, s_{tid} of v's previous and successive tuples. While the user creates the new data or modifies the outsourced data, the Sign algorithm takes v, tid, p_{tid}, s_{tid}, and the user's private key sk as input and outputs the signature σ_v for the new data v.
- $\mathsf{Verify}(v, \sigma_v, pk) \rightarrow (True, False)$ is run by the user or the cloud server to verify the integrity of the outsourced data by checking the signature. It takes the outsourced data value v, its signature σ_v, and signer's public key pk as input. It returns $True$ if the checking passed the integrity verification. Otherwise, it returns $False$.
- $\mathsf{ChalGen}(|R|) \rightarrow chal$ is run by the database querier. After receiving the query results R, the querier generates the challenge message $chal$ to verify the correctness and completeness of R. The input is the total number of tuples in the query results R and the output is the challenge message $chal$.
- $\mathsf{ProofGen}(chal, R, \sigma_R) \rightarrow P$ is run by the cloud server. It takes the challenge $chal$, the query results R, and the signatures of all data in R as input. It outputs a proof P to allow the querier to verify the correctness and completeness of the results R.
- $\mathsf{VerifyProof}(P) \rightarrow (True, False)$ is executed by the querier. After receiving the proof P, the querier verify the correctness and completeness of the results R by checking P. It outputs $True$ if the checking result passes the verification. Otherwise, it returns $False$.

3.2. Security model

In this paper, we define $\langle P(R, \sigma_R), V\rangle(PK)$ to be a public proof for the query results R, where the prover (cloud server) P takes the query results R and R's signatures σ_R, and the public keys PK of R's signers as input, where $P(x)$ denotes the prover P holds the secret x and $\langle P, V\rangle(x)$ denotes the prover P, and the verifier (user) V share x in the protocol execution. In the following discussion, we give the definition of the security model of the proposed integrity verification scheme for the cloud-based outsourced databases.

Definition 1 (Security model)**.** A pair of interactive machines (P, V) [16] is called an available proof of verification for the correctness and completeness of the outsourced databases if P is an unbounded probabilistic algorithm, V is a deterministic

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

polynomial-time algorithm, and the following conditions hold for some polynomials $p_1(\cdot)$, $p_2(\cdot)$, and all $\kappa \in \mathbb{N}$:

- *Correctness*: Given a query results R, which is correctly executed over the entire outsourced data, $\sigma_R = \{\sigma | \sigma \in \mathsf{Sign}(v, tid, p_{tid}, s_{tid}, sk), v \in R\}$,

$$Pr[\langle P(R, \sigma_R), V\rangle(PK) = 1] \geq 1 - 1/p_1(\kappa). \quad (1)$$

- *Soundness*: Given a query results R and its signature collection σ_R^*, if $\exists \sigma^* \in \sigma_R^*, v \in R, \sigma^*$ is v's signature, and $\sigma^* \notin \mathsf{Sign}(v, tid, p_{tid}, s_{tid}, sk)$. For every interactive machine P^*

$$Pr[\langle P^*(R, \sigma_R^*), V\rangle(PK) = 1] \leq 1/p_2(\kappa), \quad (2)$$

where $p_1(\cdot)$ and $p_2(\cdot)$ are two polynomials, and κ is the security parameter used in $\mathsf{KeyGen}(1^\kappa)$.

In this definition, the function $1/p_1(\kappa)$ is called correctness error, and the function $1/p_2(\kappa)$ is called soundness error. The correctness means that if the cloud server correctly executes the query over the outsourced data and the proof is generated by the valid signatures, then it can pass the verification with a nonnegligible probability. And the soundness means that if the proof is generated by any invalid or forged signature or missing any tuple, the probability that the proof passes the verification is negligible.

3.3. Threat model

In this work, we assume that the cloud-based outsourced database is "semi-honest-but-curious" server. That means the cloud server may not correctly follow our proposed protocol but return a fraction of search result and execute only a fraction of searching operations honestly [21]. Two types of attackers are considered: 1) An external attacker refers to an entity who attempts to tamper the integrity of the outsourced database via the public channels; 2) An internal attacker refers to the cloud server, who may cheat in search process for benefits. In this paper, we consider the following potential security threats:

1. **Data corruption**: The adversaries aim at corrupting the outsourced data without being detected by the verification. The adversaries involved in this type of attack could be the semi-trusted cloud server or the external attackers.
2. **Forgery attack**: The adversaries may try to tamper the outsourced data and forge a valid signature without knowing the user's private key. The adversary aims at using the corrupt data and forged signature to successfully pass the integrity verification.
3. **Lazy behavior**: The untrusted server has 'lazy behaviors' for saving its computation resources, i.e., the cloud server avoids the consumption of CPU or the storage resources associated with the query execution. Specifically, the cloud server omits some valid tuples or replies with the random even the incorrect results to the client.

3.4. Design goals

In this paper, we propose to achieve the following design goals: 1) **Correctness verification**: The user is able to correctly verify the correctness of the query results returned from the outsourced databases, even if the outsourced data is modified and signed by the multiple users; 2) **Completeness verification**: After receiving the query results returned from the cloud-based outsourced database system, the database querier is able to verify whether any valid tuple has been omitted by checking the proof message returned from cloud; 3) **The support of efficient multi-user modification**: During the verification, the outsourced database does not need to maintain the complex authenticated data structure after every data modification operation, while the user is able to independently sign the data modified by himself. The verification mechanism does not depend on a special entity such as an online data owner to participate in the signature generation processes for every data modification.

4. Preliminaries

In this section, we briefly review the cryptographic tools used to construct our protocol in this paper, including the bilinear maps and the Homomorphic Verifiable Tags (HTVs).

4.1. Bilinear maps

We say a map $e : \mathbb{G}_1 \times \hat{\mathbb{G}}_1 \to \mathbb{G}_2$ is a bilinear map if e has the following properties:

1. $\mathbb{G}_1$, $\hat{\mathbb{G}}_1$ and $\mathbb{G}_2$ are the groups of the same prime order p;
2. For all $a, b \in Z_p, g \in \mathbb{G}_1$, and $h \in \hat{\mathbb{G}}_1$, then $e(g^a, h^b) = e(g, h)^{ab}$ is efficiently computable;
3. The map is non-degenerate, i.e., $e(g, g) \neq 1$;
4. There exists a computable isomorphism from $\hat{\mathbb{G}}_1$ to $\mathbb{G}_1$ (In this paper, $\mathbb{G}_1$ equals to $\hat{\mathbb{G}}_1$).

The security of the bilinear maps is based on the computational Diffie-Hellman assumption, which is the security basis of our scheme.

Computational Diffie–Hellman (CDH) assumption. For $x, y \in Z_p$, given $g, g^x, g^y \in \mathbb{G}_1$, it is hard to compute g^{xy}.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

Table 1
Notations used in this paper.

Symbols	Descriptions
tid_i	The identifier of the ith tuple
p_tid_i	The identifier of the ith tuple's previous tuple
s_tid_i	The identifier of the ith tuple's successive tuple
$\mathbb{G}_1, \mathbb{G}_2$	Two groups of the same prime order p
g, h	The generators of $\mathbb{G}_1$
ω	A random element of $\mathbb{G}_1$
e	A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$
$H(\cdot)$	A one-way hash function from the arbitrary strings to the elements in Z_p
$\sigma(r, s, t)$	The signature
δ	A system parameter which decides the maximal number of tuples allowed to be inserted into two adjacent original data v_i and v_{i+1}
R	The set of the query results
R_i	A subset of R, and the data in R_i is signed by the same user
$Range_i$	A continuous subrange in R
σ_R	The signatures of all data in R
m	The number of the all continuous subranges in R
d	The number of the signers for the tuples in R

4.2. Homomorphic verifiable tags

The Homomorphic Verifiable Tags (HVTs) [1] allow the cloud to generate the aggregated signatures for multiple data verification and reply to the verifier in one proof message. Given a message m and its homomorphic verifiable tag T_m, both of which are stored on the untrusted server, HVTs act as the verification metadata. The HVTs are homomorphic in the sense that given two HVTs T_{v_i} and T_{v_j}, anyone can combine them into another HVT $T_{v_i+v_j}$ for the sum of the messages $v_i + v_j$ without knowing v_i and v_j.

5. Signature scheme

In this paper, we propose a new public key based signature scheme, which serves as HVTs. On top of the signature design, we propose our integrity verification mechanism for the outsourced databases. Table 1 summarizes the notations used in this paper.

5.1. Signature scheme description

We first present the signature algorithms in our scheme, including the algorithms $\mathsf{SetUp}(1^\lambda)$, $\mathsf{KeyGen}(1^\kappa)$, $\mathsf{Sign}(v, tid, p_{tid}, s_{tid}, sk)$ and $\mathsf{Verify}(v, \sigma_v, pk)$.

SetUp $(1^\lambda) \to (e, p, \mathbb{G}_1, \mathbb{G}_2, g, h, \omega, H)$: The cloud server sets up the global security parameters to initialize the outsourced database system at the cloud. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of prime order p. The global security parameters of our scheme are $(e, p, \mathbb{G}_1, \mathbb{G}_2, g, h, \omega, H)$, where g, h be the generators of $\mathbb{G}_1$, ω be a random element of $\mathbb{G}_1$, $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map, and H: $\{0, 1\}^* \to Z_p$ is a one-way hash function from the arbitrary strings to the elements in Z_p.

KeyGen $(1^\kappa) \to (pk, sk)$: When a new user joins the outsourced database system, it calls the algorithm KeyGen to generate its public key and private key. The user first selects a random $\alpha \in Z_p$ as its private key sk. Then, the KeyGen algorithm outputs the public key pk as the Eq. (3) for the new user. After that, the user uploads its public key pk to the cloud outsourced database server and stores its private key sk itself.

$$sk = \alpha, \quad pk = (g^\alpha, h^{1/\alpha}). \tag{3}$$

After the cloud server receives the user's public key pk, it verifies the public key by the equation $e(g^\alpha, h^{1/\alpha}) \stackrel{?}{=} e(g, h)$. If the verification result is true, the cloud server inserts the user's information including the user identification and the public key into the user information table which is shown in Fig. 1. Otherwise, the cloud server outputs $\perp$.

Sign $(v, tid, p_{tid}, s_{tid}, sk) \to \sigma_v$: To ensure the integrity of outsourced database services, the cloud user uses the Sign algorithm to sign the outsourced data after it creates or modifies it. Let v be the attribute value inserted or updated by the user whose public key is $pk = (g^\alpha, h^{1/\alpha})$. In our scheme, every tuple has a tuple identifier which is unique in the outsourced database system. To generate the signature for the data v, the user first selects a random $k \in Z_p$, then sets $r = (h\omega)^k, s = \alpha(H(r||tid) + k(p\_tid + s\_tid)) (\mathrm{mod}\ p)$, and $t = \omega^{\alpha(v+k(p\_tid+s\_tid))}$ in which tid denotes the tuple identifier for the data v, and $\{p\_tid, s\_tid\}$ denotes the tuple identifiers of v's previous and successive tuples respectively. At last, the Sign algorithm outputs the signature σ_v for data v as the Eq. (4) and uploads σ_v to the cloud server.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

$$\begin{aligned}\sigma_v &= (r, s, t)\\ r &= (h\omega)^k\\ s &= \alpha(H(r||tid) + k(p\_tid + s\_tid))(\text{mod } p)\\ t &= \omega^{\alpha(v+k(p\_tid+s\_tid))}.\end{aligned} \tag{4}$$

Verify $(v, \sigma_v, pk) \rightarrow (True, False)$: Our scheme features the public verification of the integrity of the outsourced data. Given a tuple which has the attribute value v with the signature $\sigma_v = (r, s, t)$, the verifier, who can be any one of the cloud users, is able to verify the integrity of data v by checking the Eq. (5).

$$e(g, h^s \cdot t) \stackrel{?}{=} e(pk^{(1)}, r^{(p\_tid+s\_tid)} h^{H(r||tid)} \omega^v). \tag{5}$$

The signer of v, i.e., the user who last modifies v, has the public key $pk = (g^\alpha, h^{1/\alpha})$, where $pk^{(1)} = g^\alpha$ is the first component of the public key pk. Based on the properties of the bilinear maps, the correctness of the above equation can be proved as:

$$\begin{aligned}e(g, h^s \cdot t) &= e(g, h^{\alpha(H(r||tid)+k(p\_tid+s\_tid))}\\ &\quad \times \omega^{\alpha(v+k(p\_tid+s\_tid))})\\ &= e(g^\alpha, (hw)^{k(p\_tid+s\_tid)} \times h^{H(r||tid)} \times \omega^v)\\ &= e(pk^{(1)}, r^{(p\_tid+s\_tid)} h^{H(r||tid)} \omega^v).\end{aligned} \tag{6}$$

The proposed signature scheme is a type of *Homomorphic Verifiable Tags*. This feature enables the cloud outsourced database server to aggregate the signatures of the multiple tuples into one proof message to prove the correctness and completeness of a query result. We proof the proposed signature scheme satisfies the properties of Homomorphic Verifiable Tags mentioned in Section 4.2. Given two data v_1, v_2 is signed by a user whose public key is pk, two random numbers $\lambda_1, \lambda_2 \in Z_p$, and two signatures $\sigma_1(r_1, s_1, t_1)$, $\sigma_2(r_2, s_2, t_2)$ for v_1, v_2, the verifier is able to check the integrity of data $v' = \lambda_1 v_1 + \lambda_2 v_2$ by the Eq. (7) without knowing data v_1 and v_2.

$$e(g, h^{\lambda_1 s_1 + \lambda_2 s_2} t_1^{\lambda_1} t_2^{\lambda_2}) \stackrel{?}{=} e\left(pk^{(1)}, \prod_{i=1}^{2} \left(r_i^{(p\_tid_i+s\_tid_i)} h^{H(r_i||tid_i)}\right)^{\lambda_i} \times \omega^{v'}\right). \tag{7}$$

Based on the proposed signature scheme, we design the integrity verification mechanism for the queries over the outsourced data at cloud with multi-user modification.

6. Integrity verification of the outsourced databases

6.1. Sign the outsourced data

The main contribution of this work is to develop an integrity verification scheme for the outsourced database with multi-user modification. So we have to consider how to efficiently maintain the verification data structure to support the dynamic outsourced data before introducing the entire integrity verification mechanism. The existing solutions [7,13,22,23] use various types of Authenticated Data Structures (ADS) to verify data integrity. However, they are not efficient for dynamic data due to the huge maintenance overheads of the ADS. Specifically, if a tuple is inserted or deleted, all the identifiers of tuples which are after the modified data are changed. The change requires the user to re-generate the signatures for these data, even though these data have not been changed at all.

We design a virtual tuple identifier mechanism for the outsourced data as showed in Fig. 2, by which we allow a user to modify a tuple without changing the identifiers of others. More specifically, the tuple identifier is unique in the outsourced database for an attribute to identify the data. While the user (the original data owner) initially outsourced its data to the cloud database server, i.e., the branch store of a supermarket uploads the newest sales data to the outsourced database server, it ordered the tuples by the attribute values[5]. The initial tuple identifier of v_i is computed as $tid_i = i \times \delta$, where $\delta \in Z_p$ is a system parameter decided by the data owner. If a new tuple v' is inserted between v_i and v_{i+1}, then the tuple identifier of v' is computed as $tid' = \lfloor (tid_i + tid_{i+1})/2 \rfloor$. Clearly, if tuples v_i and v_{i+1} are both initially created by the original data owner, the maximal number of the inserted tuples that is allowed to be inserted between v_i and v_{i+1} is $\delta - 1$. So, the original data owner should choose a proper value of δ based on the requirements of outsourced data application. Moreover, to enable the completeness verification, the data owner inserts two boundary virtual tuples for an attribute, for example $+\infty$ and $-\infty$, whose tuple identifiers are 0 and $(n+1)\delta$ respectively.

Using the virtual tuple identifier mechanism, when a user inserts a new data, it only needs to compute the tuple identifier for the new data and does not need to re-compute the identifiers for other tuples. Moreover, the virtual tuple identifier ensures that all the tuples are in a right order, for instance, if $v_i > v_j$, then $tid_i > tid_j$.

Based on the virtual tuple identifier (tid), the user calls the Sign algorithm to sign the data v which is created or modified by it. The data modification in our scheme has two-round communications. In the first round, the user executes

[5] If the attribute values of two tuples are the same, it is necessary to use an additional mechanism to break the tie.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

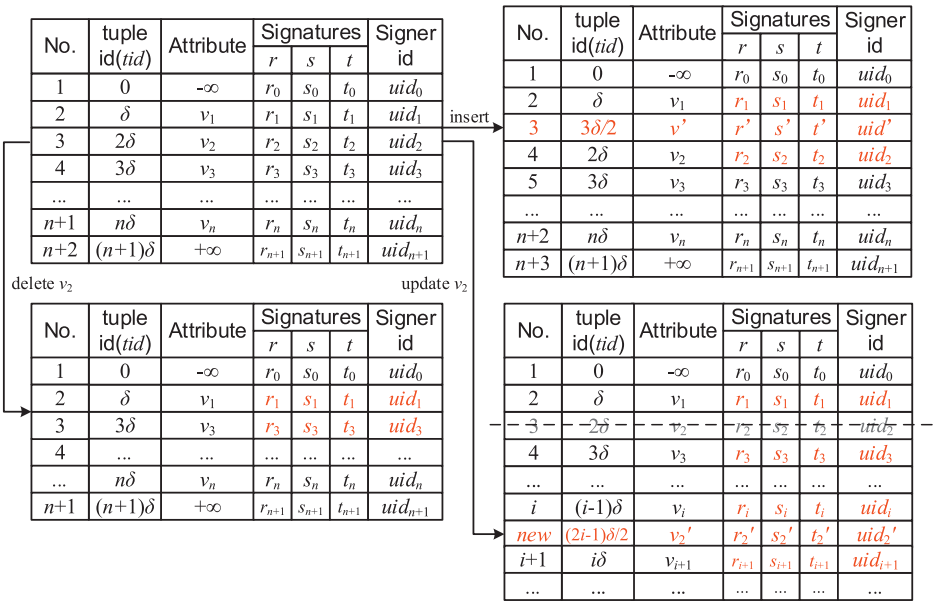


Fig. 2. Insert v', delete v_2 and update v_2 in the outsourced database. All the changed data have been marked in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

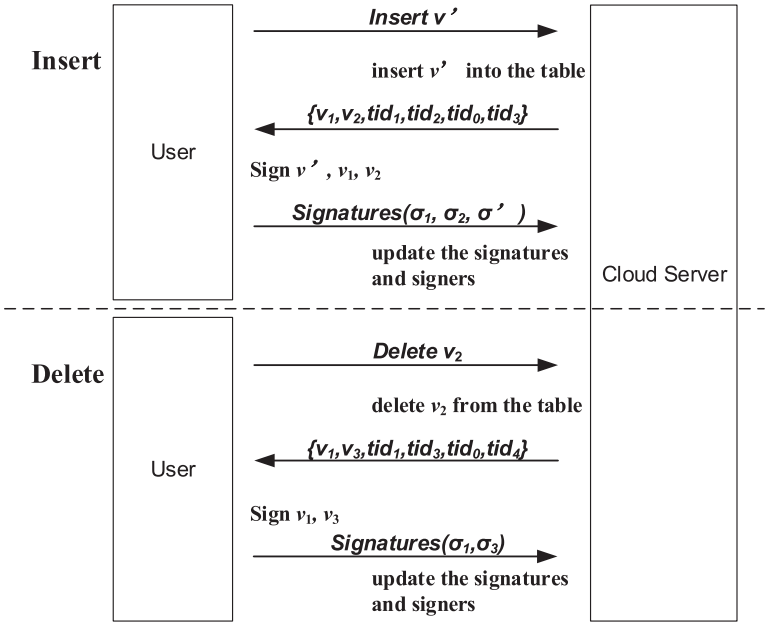


Fig. 3. The data modification processes in our scheme.

the SQL operation over the outsourced data and gets the corresponding tuple identifiers for the changed data. In the second round, the user outputs the signatures and submits them to the cloud server to complete the data modification. Using the example in Fig. 2, the data modification processes can be illustrated by Fig. 3. Below, we will take Fig. 3 as an example to present the data modification operations over the outsourced database.

Insert : While the user inserts v' between v_1 and v_2 (v_1 is the previous tuple of v', v_2 is the successive tuple of v'), it asks the outsourced database server to execute the insert operation in the first round communication. To enable the completeness verification, the user needs to sign the new data (v') and its adjacent tuples (v_1, v_2) by the $\mathsf{Sign}(v, tid, p_{tid}, s_{tid}, sk)$ algorithm. The cloud server returns v_1, v_2, the tuple identifiers (tid_1, tid_2) of v_1, v_2, v_1's *p_tid* (tid_0), and v_2's *s_tid* (tid_3) to the user. The user computes v''s tuple identifier as $tid' = \lfloor (tid_1 + tid_2)/2 \rfloor$. Then, it selects three random numbers k', k_1, k_2 $\in Z_p$ and outputs the signatures σ', σ_1, σ_2 for v', v_1, v_2 by the $\mathsf{Sign}(v, tid, p_{tid}, s_{tid}, sk) \rightarrow \sigma_v$ algorithm as the Eq. (8). In the

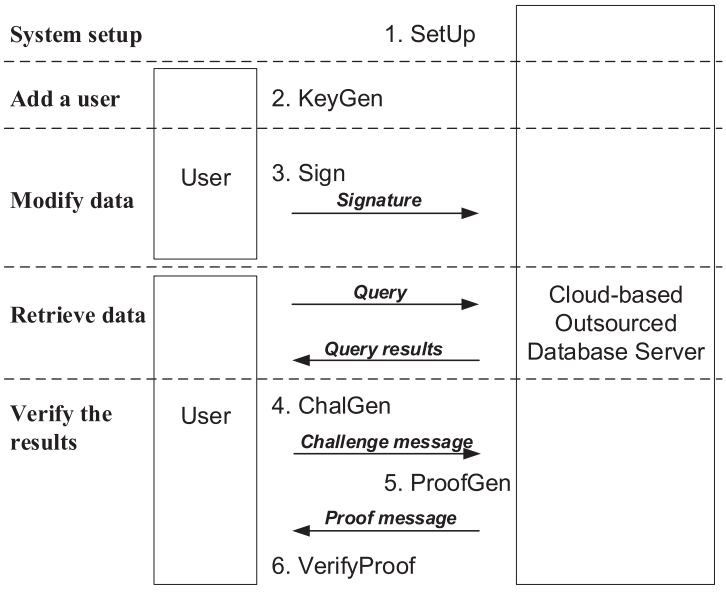


Fig. 4. The verifying processes in this paper.

second round communication, the user uploads the generated signatures $(\sigma', \sigma_1, \sigma_2)$ to the cloud server. Finally, the cloud server stores these signatures and updates the signers of the data v', v_1, v_2 as the user to complete the insert operation.

$$\begin{aligned}
\sigma' &= (r', s', t') = ((h\omega)^{k'}, \\
&\quad \alpha(H(r'||tid') + k'(tid_1 + tid_2)), \omega^{\alpha(v' + k'(tid_1 + tid_2))}) \\
\sigma_1 &= (r_1, s_1, t_1) = ((h\omega)^{k_1}, \\
&\quad \alpha(H(r_1||tid_1) + k_1(tid_0 + tid')), \omega^{\alpha(v_1 + k_1(tid_0 + tid'))}) \\
\sigma_2 &= (r_2, s_2, t_2) = ((h\omega)^{k_2}, \\
&\quad \alpha(H(r_2||tid_2) + k_2(tid' + tid_3)), \omega^{\alpha(v_2 + k_2(tid' + tid_3))}).
\end{aligned} \tag{8}$$

Delete : While the user deletes v_2 from the relation (v_2's previous tuple is v_1, v_2's successive tuple is v_3), besides the delete operation, the user needs to generate the signatures for the previous tuple (v_1) and the successive tuple (v_3). To generate the signatures, the cloud server returns v_1, v_3, the tuple identifiers of $v_1, v_3 (tid_1, tid_3)$, v_1's $p\_tid(tid_0)$, and v_3's $s\_tid(tid_4)$ to the user. Then, the user selects two random numbers $k_1, k_3 \in Z_p$ and outputs the signatures σ_1, σ_3 for the data v_1, v_3 by the $\mathsf{Sign}(v, tid, p_{tid}, s_{tid}, sk) \rightarrow \sigma_v$ algorithm as the Eq. (9). After receiving the signatures, the cloud server stores these signatures and updates the signers of v_1 and v_3 as the user to complete the delete operation.

$$\begin{aligned}
\sigma_1 &= (r_1, s_1, t_1) = ((h\omega)^{k_1}, \\
&\quad \alpha(H(r_1||tid_1) + k_1(tid_0 + tid_3)), \omega^{\alpha(v_1 + k_1(tid_0 + tid_3))}) \\
\sigma_3 &= (r_3, s_3, t_3) = ((h\omega)^{k_3}, \\
&\quad \alpha(H(r_3||tid_3) + k_3(tid_1 + tid_4)), \omega^{\alpha(v_3 + k_3(tid_1 + tid_4))}).
\end{aligned} \tag{9}$$

Update : In our scheme, an update operation is equal to an insert operation and a delete operation. As can be seen in the example in Fig. 2, while the user uploads v_2 to v_2', besides signs the new data (v_2') and its previous tuple and successive tuple, the user also needs to sign the previous tuple (v_1) and the successive tuple (v_3) of the old data (v_2).

In our scheme, we design the virtual tuple identifier mechanism to generate the unique tuple identifier for every tuple. By the virtual tuple identifier, the user only needs to generate the tuple identifier for the new data without changing the identifiers of others. Also take Fig. 2 as the example, the user updates the data v_2 to v_2' which is inserted between the data v_i and v_{i+1}, the tuple identifiers of which are tid_i and tid_{i+1}. Then, the user computes the tuple identifier for the new data v_2' as $tid_2' = \lfloor (tid_i + tid_{i+1})/2 \rfloor$ and generate the corresponding signatures.

6.2. Integrity verification for the outsourced database

We design the ChalGen, ProofGen, and VerifyProof algorithms, by which the verifier, i.e., the inquirer, is able to verify the integrity of the query results from the remote outsourced database server. The integrity verification processes are described in Fig. 4.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

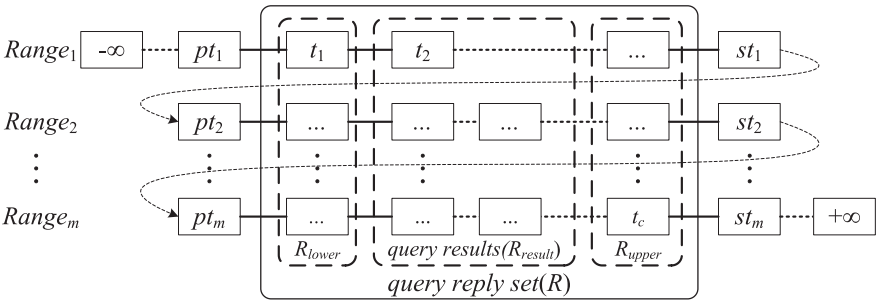


Fig. 5. The cloud server returns the query results R to the user. The set of the query results R can be divided into m subranges $Range_i$.

To guarantee the completeness of the query results, besides returning the tuples satisfied the user's query, the cloud server also returns the previous tuple and the successive tuple for every subrange in the query results. As shown in Fig. 5, we assume that the query result is R, which is composed of several subranges as follows:

1. R is composed of several subsets $Range_i (1 \le i \le m)$, in which $Range_i$ is a continuous subrange of R. If the tuples $tid_{i-1}, tid_{i+1} \in Range_j$, then we must have the tuple $tid_i \in Range_j$.
2. If $i \neq j$, $Range_i \cap Range_j = \varnothing$, and $\bigcup_{i=1}^{m} Range_i = R$.
3. The set R_{result} consists of all the tuples which satisfy the user's query.
4. R_{lower} includes all the lower boundary tuples for every subrange $Range_i$. And R_{upper} includes all the upper boundary tuples for every subrange $Range_i$. $R = R_{result} \cup R_{lower} \cup R_{upper}$, and $R_{result} \cap R_{lower} = R_{result} \cap R_{upper} = \varnothing$.
5. pt_i and st_i respectively represent the previous tuple and the successive tuple of the set $Range_i$.

ChalGen $(|R|) \rightarrow chal$: After receiving the query results R from the cloud server, the user generates the challenge $chal$ to verify the correctness and completeness of R as follows:

1. For the set of query results R which has $|R|$ tuples $v_l (1 \le l \le |R|)$, the verifier chooses $|R|$ random numbers $\lambda_l \in Z_p$ for the $|R|$ tuples $v_l \in R$.
2. The verifier outputs the challenge $chal$ as $chal = (l, \lambda_l)_{v_l \in R, 1 \le l \le |R|}$ and submits it to the cloud server.

ProofGen $(chal, R, \sigma_R) \rightarrow P$: After receiving the challenge $chal = (l, \lambda_l)_{v_l \in R, 1 \le l \le |R|}$ from the user, the cloud server generates the proof P to reply the user's challenge as follows:

1. The cloud server divides R into d subsets $\{R_1, R_2, \ldots, R_d\}$ based on the different signers. $\forall v \in R_i$ and $v' \in R_j$, if $i = j$, then v and v' must be signed by the same user, and if $i \neq j$, then v and v' are signed by the different users.
2. The cloud server collects the public keys $\boldsymbol{PK} = \{pk_i^{(1)}\}_{i \in [1,d]}$, where $pk_i^{(1)} = g^{\alpha_i}$ is the public key's first component of the signer of R_i.
3. For every subset R_i, the cloud server computes μ_i as in the Eq. (10), where s_k and t_k are the second and the third components of v_k's signature σ_k.

$$\mu_i = \prod_{v_k \in R_i} (h^{s_k} t_k)^{\lambda_k} \in \mathbb{G}_1. \tag{10}$$

4. The cloud server computes φ_i for each verifying tuple $v_i \in R$ as in the Eq. (11), where $\sigma_i(r_i, s_i, t_i)$ is v_i's signature.

$$\varphi_i = \begin{cases} r_i^{(p\_tid_i + s\_tid_i)} h^{H(r_i || tid_i)} & \text{if } v_i \in R_{result} \\ r_i^{s\_tid_i} h^{H(r_i || tid_i)} & \text{if } v_i \in R_{lower} \\ r_i^{p\_tid_i} h^{H(r_i || tid_i)} & \text{if } v_i \in R_{upper}. \end{cases} \tag{11}$$

5. We assume that the query results R has m continuous subranges $Range_i (1 \le i \le m)$. The cloud server collects all the tuple identifiers of pt_i and st_i as $\boldsymbol{PT} = \{pt\_tid_i\}_{i \in [1,m]}$ and $\boldsymbol{ST} = \{st\_tid_i\}_{i \in [1,m]}$, in which pt_i is the previous tuple of $Range_i$ and st_i is the successive tuple of $Range_i$.
6. The cloud server outputs the proof message as $P = \{\boldsymbol{\mu}, \boldsymbol{\varphi}, \boldsymbol{PT}, \boldsymbol{ST}, \boldsymbol{PK}\}$, where $\boldsymbol{\mu} = \{\mu_i\}_{i \in [1,d]}$, $\boldsymbol{\varphi} = \{\varphi_i\}_{v_i \in R}$, and $\boldsymbol{PK} = \{pk_i^{(1)}\}_{i \in [1,d]}$. Then, the cloud server returns P to the user to prove the integrity of R.

VerifyProof $(P) \rightarrow (True, False)$: After the user receives the proof $P = \{\boldsymbol{\mu}, \boldsymbol{\varphi}, \boldsymbol{PT}, \boldsymbol{ST}, \boldsymbol{PK}\}$ from the cloud server, it verifies the correctness and completeness of the query results R as follows:

1. The cloud server computes $\tau_i = \sum_{v_k \in R_i} (v_k \times \lambda_k)$ for each $R_i (1 \le i \le d)$.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

2. With τ_i and the challenge message $chal = (l, \lambda_l)_{v_l \in R}$, the user verifies the received proof message P as the Eq. (12).

$$e\left(g, \prod_{i=1}^{d} \mu_i\right) \stackrel{?}{=} \prod_{i=1}^{d} e\left(pk_i^{(1)}, \omega^{\tau_i} \times \prod_{\substack{v_j \in R_i \\ v_j \in R_{lower}}} r_j^{pt\_tid_j \times \lambda_j} \times \prod_{\substack{v_j \in R_i \\ v_j \in R_{upper}}} r_j^{st\_tid_j \times \lambda_j} \times \prod_{v_j \in R_i} \varphi_j^{\lambda_j}\right). \tag{12}$$

If the Eq. 12 is satisfied, based on the chain of tuple identifiers in the signatures, it proves that all the tuples in the set R_{result} are in the continuous subranges. Moreover, all the previous tuples in R_{lower} and the successive tuples in R_{upper} for each subrange $Range_i$ do not satisfy the user's query. So, the user is able to believe that the integrity of the query result R is guaranteed. Otherwise, the user believes that the received query results are not correctly executed over the original and entire outsourced data at the cloud.

7. Performance and security analysis

7.1. Performance analysis

7.1.1. Communication cost

For the proposed scheme, the main communication overhead of the verification for the outsourced database includes the challenge message and the proof message. Therefore, we focus the analysis of the communication cost on two cases: the challenge message sent by the user (the querier and the verifier) and the proof message returned from the outsourced database server.

The size of the challenge message $(l, \lambda_l)_{v_l \in R}$ is $|R| \times (|S_M| + |S_\lambda|)$, where $|R|$ is the total number of the reply tuples, S_M is an element of the set $[1, M]$ (M is the data scale of the outsourced database), $|S_\lambda|$ is the size of λ and equals to an element of Z_p.

The size of the proof message $\{\boldsymbol{\mu}, \boldsymbol{\varphi}, \boldsymbol{PT}, \boldsymbol{ST}, \boldsymbol{PK}\}$ is $d(|S_\mu| + |S_{pk}|) + |R| \times |S_\varphi| + 2m|S_{tid}|$, where d is the different signers in R, m is the total number of the subranges in the set of the query results R, $|S_\mu|$, $|S_\varphi|$, and $|S_{pk}|$ are the sizes of μ, φ, and $pk^{(1)}$ respectively, all of which equal to the size of an element of $\mathbb{G}_1$, and $|S_{tid}|$ is the size of the tuple identifier which equals to an element of Z_p. Therefore, the total communication cost for a verification task is $|R| \times (S_M + S_P + S_G) + 2d \times S_G + 2m \times S_P$, in which S_G is the size of an element of $\mathbb{G}_1$, and S_P is the size of an element of Z_p.

7.1.2. Computation cost

For the signing process, the computation cost of signing a data is $5T_{mul} + 2T_{exp} + 4T_{add} + T_{hash}$, where T_{mul}, T_{exp}, T_{add}, and T_{hash} represent one multiplication operation, one additive operation, one exponentiation operation, and one hash operation respectively.

The computation cost for the cloud server to output the proof message is $T_{group} + 3|R|T_{mul} + |R_{result}|T_{add} + 4|R|T_{exp} + |R|T_{hash}$, where T_{group} represents the computation cost for grouping the set R based on the signer, $|R|$ and $|R_{result}|$ represent the total number of the tuples in the set R and the set R_{result}. $|R| = |R_{result}| + 2m$, where m represents the total number of subranges in the set of the query results R.

While the user verifies the correctness and the completeness of the query results R, the computation cost for the user to check the proof message returned from the cloud server is $(4m + 5d + |R|)T_{mul} + (2m + d + |R|)T_{exp} + (d + 1)T_{pair}$, where T_{pair} represents one paring operation as $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

7.2. Security analysis

Based on the security model defined in Section 3.2, we prove that our scheme is secure under the attacks given in Section 3.3. That is, our scheme can ensure the correctness and completeness of the integrity verification.

7.2.1. Correctness

Theorem 1. ***Correctness:*** *Given the query result R which is correctly executed over the entire outsourced data, the challenge chal, and the proof* $P \leftarrow ProofGen(chal, R, \sigma_R)$ *where* σ_R *are valid signatures for R. Our scheme can ensure the correctness of integrity verification, which means that P can pass the verification with a nonnegligible probability.*

Proof. The correctness of the integrity verification in our scheme is equivalent to the correctness of Eq. (12). Based on the properties of bilinear maps, the correctness of our scheme can be elaborated as Eq. (13).

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

$$
\begin{aligned}
e\left(g, \prod_{i=1}^{d} \mu_i\right) &= \prod_{i=1}^{d} e\left(g, \prod_{v_j \in R_i} (h^{\alpha_j (H(r_j||tid_j)+k_j(pt\_tid_j+st\_tid_j))} \times \omega^{\alpha_j (v_j+k_j(pt\_tid_j+st\_tid_j))})^{\lambda_j}\right) \\
&= \prod_{i=1}^{d} e\left(pk_i^{(1)}, \prod_{v_j \in R_i} (h^{H(r_j||tid_j)} r_j^{(pt\_tid_j+st\_tid_j)})^{\lambda_j} \omega^{v_j \times \lambda_j}\right) \\
&= \prod_{i=1}^{d} e\left(pk_i^{(1)}, \prod_{v_j \in R_i} \left(\omega^{v_j \times \lambda_j} \times \varphi_j^{\lambda_j} \times \prod_{v_j \in R_{lower}} r_j^{pt\_tid_j \times \lambda_j} \times \prod_{v_j \in R_{upper}} r_j^{st\_tid_j \times \lambda_j}\right)\right) \\
&= \prod_{i=1}^{d} e\left(pk_i^{(1)}, \omega^{\tau_i} \prod_{v_j \in R_i} \left(\varphi_j^{\lambda_j} \times \prod_{v_j \in R_{lower}} r_j^{pt\_tid_j \times \lambda_j} \times \prod_{v_j \in R_{upper}} r_j^{st\_tid_j \times \lambda_j}\right)\right). \qquad (13)
\end{aligned}
$$

□

7.2.2. Soundness

Theorem 2. ***Soundness:*** *Given the query result R which is correctly executed over the entire outsourced data, the challenge chal, σ_R are valid signatures for R, and the proof P. Our scheme is sound, which means if $P \neq ProofGen(chal, R, \sigma_R)$ the probability that P can pass the verification is negligible.*

Proof. In this paper, we assume that the security of the proposed integrity verification scheme is threatened by three security threats defined in the Section 3.3. We prove our scheme is sound under the three security threats.

The threats of '*data corruption*' and '*forgery attack*': While only the '*data corruption*' threat exists, because the query result R has been tempered and the proof P is generated by the valid signatures, based on the properties of the bilinear map the proof P is difficult to pass the integrity verification, i.e., the VerifyProof$(P) \rightarrow (True, False)$ algorithm. While both the threats of '*data corruption*' and '*forgery attack*' exist, the attackers have tampered the outsourced data and forged the signature without knowing the user's private key. The adversary attempts to use the corrupt data and forged signature to successfully pass the integrity verification. Following the standard security model [1], we prove that our scheme is sound by proving neither the external attacker and nor the internal adversary (the cloud server) can build an adversary A with a non-negligible probability ϵ that solves the CDH problem in the bilinear map.

Suppose the system global parameters are $(e, p, \mathbb{G}_1, \mathbb{G}_2, g, h, \omega, H)$, in which $h = g^a, a \in Z_p$. Given the user's private key $sk = \alpha$, set $\alpha = a \times b, b \in Z_p$, and then the user's public key $pk = (g^{\alpha}, h^{1/\alpha}) = (g^{ab}, g^{\frac{a}{\alpha}}) = (g^{ab}, g^{1/b})$. In this attack scenario, the adversary A outputs the CDH challenge (g^x, g^y) as $(g^{ab}, g^a) = (pk^{(1)}, h)$. Therefore, the adversary A should not be able to compute and output $g^{a^2 b}$ under the security assumption CDH in the bilinear map.

Let q_H be the total number of the queries on H. Thus, the collision in the hash function H occurs with the probability as most $q_H/2^k$, where k is the length of H's output. Applying the Reset Lemma [2], A can produce two valid signatures $\sigma_1(r, c_1, s_1, t)$ and $\sigma_2(r, c_2, s_2, t)$ signed by the user for the value v with the probability at least $(\epsilon - (\epsilon q_H + 1)/2^k)^2$, where $s_1 = \alpha(c_1 + k(p\_tid + s\_tid)) \bmod p$ and $s_2 = \alpha(c_2 + k(p\_tid + s\_tid)) \bmod p$, and c_1, c_2 are two different random responses for the hash function $H(r||tid)$. Based on these assumptions, A can solve the CDH problem by computing and outputting:

$$
\begin{aligned}
\left(\frac{h^{s_1}}{h^{s_2}}\right)^{1/(c_1-c_2)} &= \left(\frac{h^{\alpha(c_1+k(p\_tid+s\_tid))}}{h^{\alpha(c_2+k(p\_tid+s\_tid))}}\right)^{1/(c_1-c_2)} \\
&= h^{\alpha} = g^{a^2 b} = g^{ab \times a}. \qquad (14)
\end{aligned}
$$

Clearly, if the adversary can build a forged signature, then we can solve the CDH problem in the bilinear map $\mathbb{G}_1$, which is computationally infeasible. Therefore, the adversary is impossible to forge the signature for a value v in our scheme. Then, our scheme is secure under the threats of '*data corruption*' and '*forgery attack*'.

The threat of *lazy behavior*: The cloud server maybe omit some valid tuples for saving its computation resources. The signature of a data has the information of its previous and successive tuple identifiers. Therefore, any omitted or added tuples will break the chain of the tuple identifiers in the signatures. Based on the property of bilinear map, this security threat cannot pass the integrity verification. □

8. Experiments

8.1. Experimental setup

In this section, we carry out the experiments to evaluate the performance of our verification scheme. We performed our experiments on a 64-bits machine with Intel 2.5 GHz, running Ubuntu. We utilized Java Pairing-Based Cryptography Library (jPBC)[6] to implement the bilinear pairing computation. We set the security parameter p as 160 bits. In addition,

[6] jPBC, http://gas.dia.unisa.it/projects/jpbc/.

Table 2
The parameters in the experiments.

Symbols	Descriptions	Values
$\|R\|$	The number of the tuples returned from the outsourced database server	$10^3 - 10^6$
d	The number of the signers	1, 5, 10, 15, 20, 25, 30
m	The number of continuous subranges in the set of the query results	1, 3, 5, 7, 9

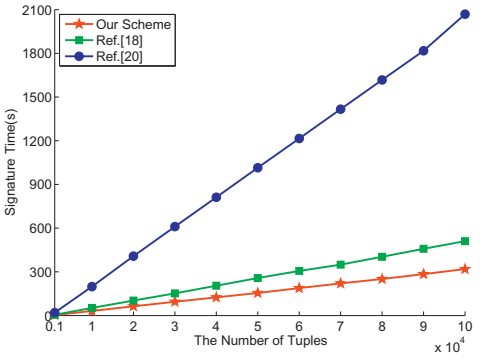


Fig. 6. Time cost for signing different number of the tuples.

all the experiments are performed over a table with 1, 000, 000 tuples. The additional parameters of our scheme in the experiments are shown in Table 2.

8.2. Experimental results

8.2.1. Time cost for signing data

To evaluate the performance of signing data, we measure the time for signing data with respect to the number of tuples. We vary the size of the signing tuples from 1000 to 100, 000 by the Sign algorithm. We compare the time cost of signing data in our scheme with those in the reference [12] and the reference [14], where [14] uses the update-efficient scheme with Merkle trees [9]. As shown in Fig. 6, the computation time of signing data in our scheme increases linearly with the increase of the number of the signing tuples. The time cost which is from 3.178 s to 318.94 s is consistent with our analysis and better than those in the references [12,14]. Moreover, it is worth noting that both [12] and [14] require an online data owner to participate in the signing processes. But, our scheme does not have this constraint. So, our scheme is able to efficiently support the integrity verification for the dynamic outsourced data with multi-user modification by allowing the user to independently sign the tuples modified by itself.

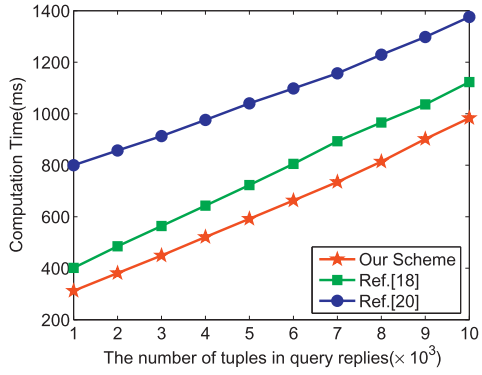
8.2.2. Computation overhead for the integrity verification

We compare the computation overhead of the integrity verification in our scheme with those in the references [12,14]. We respectively examine the computation time at the outsourced database server and the client.

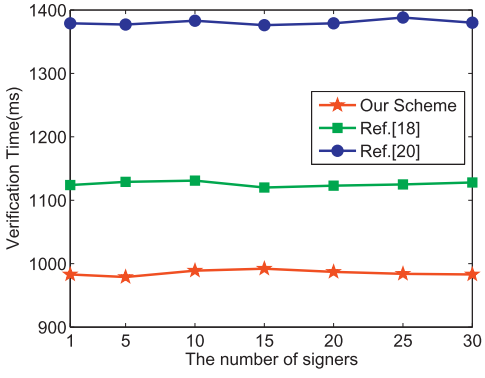
For the outsourced database server, the main computation overhead for a verification task is to output the proof message by the ProofGen algorithm. We evaluate the time costs of generating the proof message at the outsourced database server with various $|R|$, d and m values, where $|R|$ is the number of tuples in set of the query results, d is the number of different signers, and m is the total number of the continuous subranges in the query results. Fig. 7a illustrates the computation time at the server versus $|R|$ with $m = 1$ and $d = 1$. The CPU time is from 312 milliseconds to 983 milliseconds which are more efficient than those in the reference [12] and the reference [14]. Fig. 7b illustrates the time cost at the server versus d with $|R| = 10,000$ and $m = 1$. Clearly, we can find that the CPU time of our scheme at the outsourced database server with various signers is also more efficient than those in the reference [12] and the reference [14]. Fig. 7c illustrates the time cost at the server versus m with $|R| = 10,000$ and $d = 1$. Because our scheme uses the virtual tuple identifier structure and the aggregated signature mechanism to manage the chain of the tuple identifiers, our scheme achieves the higher efficiency of the completeness verification when the set of the query results has multiple continuous subranges (m).

Through the experimental results, we can find that our scheme is efficient and practical for the integrity verification of the outsourced databases with multi-user modification. Moreover, the efficiency of generating the proof message can be further improved in a real cloud database system due to the large computing power of the cloud servers.

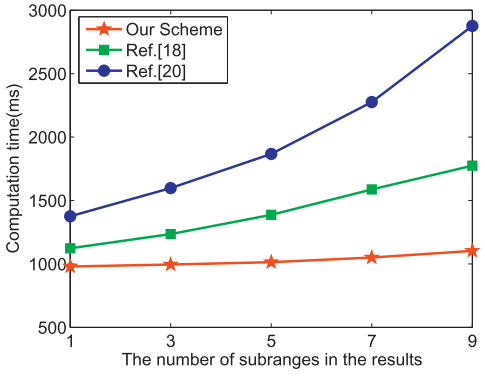
For the client, the main computation overhead for a verification task is to verify the proof message returned from the remote server by the VerifyProof algorithm. We evaluate the time cost at the client of the verification with various $|R|$, d, and m values. Fig. 8a illustrates the verification time versus $|R|$ with $m = 1$ and $d = 1$. The verifying time at the client in the proposed scheme is from 212 milliseconds to 725 milliseconds which are comparable with that in the reference [12] and



(a) Time vs. $|R|(d = 1, m = 1)$



(b) Time vs. $d(|R| = 10,000, m = 1)$



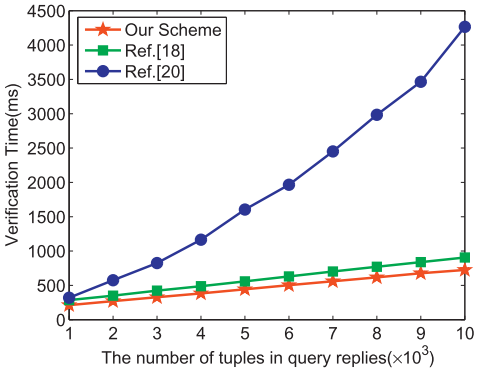
(c) Time vs. $m(|R| = 10,000, d = 1)$

Fig. 7. Time cost at the cloud server for integrity verification.

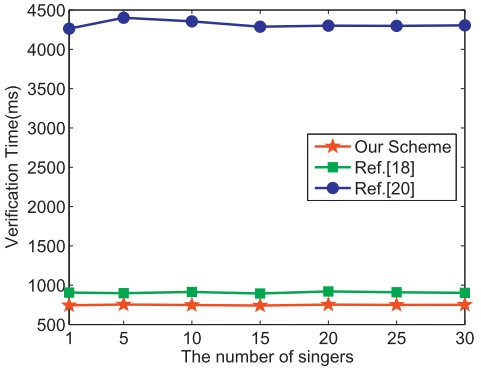
more efficient than that in the reference [14]. Fig. 8b illustrates the verification time versus d with $|R| = 10,000$ and $m = 1$. As shown in the experimental results, the verifying time at the client in the proposed scheme with various signers (d) is comparable with that in the reference [12] and more efficient than that in the reference [14]. Fig. 8c illustrates the verifying time cost versus m with $|R| = 10,000$ and $d = 1$. The experimental results show that our scheme enables the client to verify the integrity of the query results more efficient especially when the query results have multiple subranges.

Therefore, through these experiments we verify that the proposed verification mechanism achieves a practical and ideal computation overhead for the various scale of the outsourced data even they have been contributed and signed by multiple users.

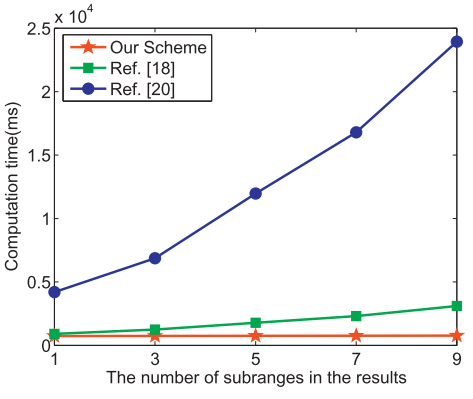
Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031



(a) Time vs. $|R|(d = 1, m = 1)$



(b) Time vs. $d(|R| = 10,000, m = 1)$



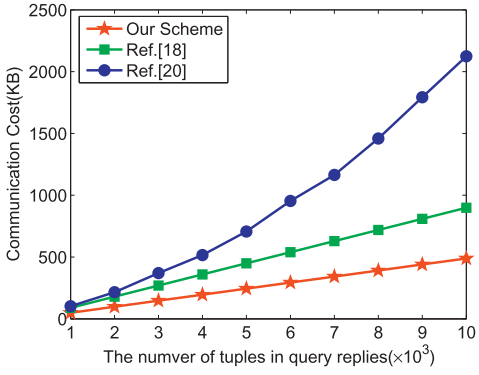
(c) Time vs. $m(|R| = 10,000, d = 1)$

Fig. 8. Time cost at the client for the integrity verification.

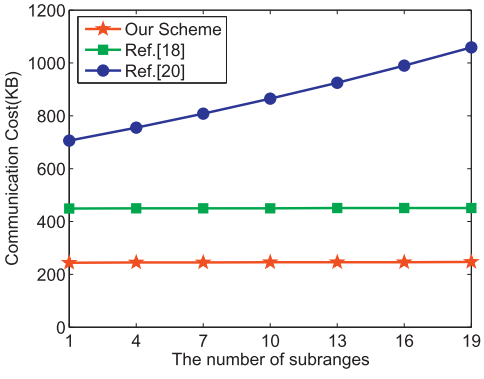
8.2.3. Communication overhead for integrity verification

As analyzed in Section 7.1, the communication cost for a verification task in our scheme is $|R| \times |S_M| + (2d + |R|)|\mathbb{G}_1| + (|R| + 2m)|Z_p|$ bits in total, where $|S_M| = 80$ bits is the size of an element in the set [1, M], $|\mathbb{G}_1| = 160$ bits is the size of an element in $\mathbb{G}_1$, and $|Z_p| = 160$ bits is the size of an element in Z_p. To verify the correctness of the analysis, we carry out the experiments to evaluate the communication overheads of the verification with various $|R|$ and m values, where m is the subranges in the query results. Fig. 9a illustrates the communication overheads versus $|R|$ with $m = 1$ and $d = 1$. The communication costs at the client for the integrity verification is from 48.9KB to 488.4KB, which is consistent with our analysis. Fig. 9b illustrates the communication overheads of the integrity verifications with various subranges(m) with $|R| = 5,000$ and $d = 1$. As the experimental results shown in Fig. 9b, the communication costs are less than those in the

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031



(a) Communication vs. $|R|(m{=}1,d=1)$



(b) Communication vs. $m(|R|=5000,d=1)$

Fig. 9. Communication overhead of the integrity verification.

references [12,14]. Moreover, through the experiments, we can find that the communication costs of our scheme are better when the parameters $|R|$ and m are bigger. Therefore, our scheme is a more practical integrity verification method for the outsourced database applications.

9. Conclusion

In this paper, we explored the problem of the integrity verification of outsourced database with multi-user modification. To address this problem, we proposed a novel signature scheme which allows the user to sign the modified data independently and is homomorphically verifiable. Based on the proposed signature scheme, we designed the integrity verifying mechanism to verify the correctness and the completeness of the query results returned from the remote outsourced database server. Because the proposed mechanism does not depend on an online data owner and not need to resign the authenticated data structure for every data modification, we achieve the efficient and the practical integrity verification. Based on the thorough security analysis, we proved that the verification scheme is secure against the security threats of the data corruption and the lazy behavior of the remote server. Moreover, we proved that the signature is unforgeable under the CDH assumption. We also compared our approach to the state-of-the-art schemes. The analysis and experiments both demonstrate that our approach is indeed an efficient and practical solution for the integrity verification of the outsourced databases.

Acknowledgments

The work in this paper is partially supported by National Natural Science Foundation of China under Grants No. 61373167, 61202034, 61572378 and 61232002, National Basic Research Program of China (973 Program) under Grant No. 2014CB340600, CCF Opening Project of Chinese Information Processing under Grant No. CCF2014-01-02, the Program for Innovative Research Team of Wuhan under Grant No. 2014070504020237, Wuhan Science and Technology Bureau under Grant No. 2015010101010020, and Fundamental Research Funds for the Central Universities under Grant No. 2042016kf0137.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031

References

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, Provable data possession at untrusted stores, in: Proceedings of ACM CCS, 2007, pp. 598–610.
[2] M. Bellare, A. Palacio, GQ and schnorr identification schemes: proofs of security against impersonation under active and concurrent attacks, in: Proceedings of CRYPTO, 2002, pp. 162–177.
[3] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and variafiable encrypted signatures from bolinear maps, in: Proceedings of EUROCRYPT, 2003, pp. 416–432.
[4] A.P. Champman, H.V. Jagadish, P. Ramanan, Efficient provenance storage, in: Proceedings of SIGMOD, 2008, pp. 993–1006.
[5] Z. Fu, K. Ren, J. Shu, X. Sun, F. Huang, Enabling personalized search over encrypted outsourced data with efficiency improvement, IEEE Trans. Parallel Distrib. Syst. PP (99) (2015) 1, doi:10.1109/TPDS.2015.2506573.
[6] H. Hacigümüş, B. Iyer, S. Mehrotra, Providing database as a service, in: Proceedings of ICDE, 2002, pp. 29–38.
[7] R. Jain, S. Prabhakar, Trustworthy data from untrusted databases, in: Proceedings of ICDE, 2013, pp. 529–540.
[8] F. Li, M. Hadjieftheriou, G. Kollios, L. Reyzin, Dynamic authenticated index structures for outsourced databases, in: Proccedings of SIGMOD, 2006, pp. 121–132.
[9] R.C. Merkle, A certified digital signature, in: Proccedings of CRYPTO, 1989, pp. 218–238.
[10] E. Mykletun, M. Narasimha, G. Tsudik, Authentication and integrity in outsourced databases, in: Proceedings of NDSS, 2004, pp. 107–138.
[11] E. Mykletun, M. Narasimha, G. Tsudik, Authentication and integrity in outsourced databases, ACM Trans. Storage 2 (2) (2006) 107–138.
[12] M. Narasimha, G. Tsudik, Athentication of outsourced databases using signature aggregation and chaining, in: Proceedings of DASFAA, 2006, pp. 420–436.
[13] H. Pang, K. Tan, Authenticating query in edge computing, in: Proceedings of ICDE, 2004, pp. 560–571.
[14] D. Papadopoulos, S. Papadopoulos, N. Triandopoulos, Taking authenticated range queries to arbitrary dimensions, in: Proceedings of CCS, 2014, pp. 819–830.
[15] S. Papadopoulos, D. Papadias, W. Cheng, K. Tan, Sepatating authentication from query execution in outsourced databases, in: Proceedings of ICDE, 2009, pp. 1148–1151.
[16] Z. Ren, L. Wang, Q. Wang, M. Xu, Dynamic proofs of retrievability for coded cloud storage systems, IEEE Trans. Serv. Comput. PP (99) (2015) 1, doi:10.1109/TSC.2015.2481880.
[17] R. Sion, Query execution assuarance for outsourced databases, in: Proceedings of VLDB, 2005, pp. 601–612.
[18] W. Song, Z. Peng, Q. Wang, F. Cheng, X. Wu, Y. Cui, Efficient privacy-preserved data query over ciphertext in cloud computing, Secur. Commun. Netw. Vol.7 (2014) 1049–1065.
[19] B. Wang, B. Li, H. Li, Public auditing for shared data with efficient user revocation in the cloud, in: Proceedings of INFOCOM, 2013. 2914–2902
[20] C. Wang, S.M. Chow, Q. Wang, K. Ren, W. Lou, Privacy-preserving public auditing for secure cloud storage, IEEE Trans. Comput. 62 (2) (2013) 362–375.
[21] J. Wang, X. Chen, X. Huang, I. You, Y. Xiang, Verifiable auditing for outsourced database in cloud computing, IEEE Trans. Comput. 64 (11) (2015) 3293–3303.
[22] Q. Wang, C. Wang, J. Li, K. Ren, W. Lou, Enabling public verifiability and data dynamics for storage security in cloud computing, in: Proceedings of ESORICS, 2009, pp. 355–370.
[23] Q. Wang, C. Wang, K. Ren, W. Lou, J. Li, Enabling public auditability and data dynamics for storage security in cloud computing, IEEE Trans. Parallel Distrib. Syst. 22 (5) (2011) 847–859.
[24] Z. Xia, X. Wang, X. Sun, Q. Wang, A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data, IEEE Trans. Parallel Distrib. Syst. 27 (2) (2015) 340–352.
[25] M. Xie, H. Wang, J. Yin, X. Meng, Integrity auditing of outsourced data, in: Proceedings of VLDB, 2007, pp. 782–793.
[26] Y. Yang, D. Papadias, S. Papadopoulos, P. Kalnis, Authenticated join processing in outsourced databases, in: Proceedings of SIGMOD, 2009, pp. 5–18.
[27] A.A. Yavuz, Immutable authentication and integrity schemes for outsourced databases, IEEE Trans. Dependable Secure Comput. (2016).
[28] J. Yuan, S. Yu, Efficient public integrity cheching for cloud data sharing with multi-user modification, in: Proceedings of INFOCOM, 2014, pp. 2121–2129.
[29] Q. Zheng, S. Xu, G. Ateniese, Efficient query integrity for outsourced dynamic databases, in: Proceedings of the ACM Cloud Computing Security Workshop (CCSW), 2012, pp. 71–82.

Please cite this article as: W. Song et al., Tell me the truth: Practically public authentication for outsourced databases with multi-user modification, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.07.031