Guaranteeing Correctness of Bulk Operations in Outsourced Databases

Luca Ferretti, Michele Colajanni, Mirco Marchetti

▶ **To cite this version:**

Luca Ferretti, Michele Colajanni, Mirco Marchetti. Guaranteeing Correctness of Bulk Operations in Outsourced Databases. 30th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec), Jul 2016, Trento, Italy. pp.37-51, 10.1007/978-3-319-41483-6_3 . hal-01633666

HAL Id: hal-01633666

https://inria.hal.science/hal-01633666

Submitted on 13 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Guaranteeing Correctness of Bulk Operations in Outsourced Databases

Luca Ferretti, Michele Colajanni, and Mirco Marchetti

Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia

{luca.ferretti,michele.colajanni,mirco.marchetti}@unimore.it

Abstract. The adoption of public cloud services, as well as other data outsourcing solutions, raises concerns about confidentiality and integrity of information managed by a third party. By focusing on data integrity, we propose a novel protocol that allows cloud customers to verify the correctness of results produced by key-value databases. The protocol is designed for supporting efficient insertion and retrieval of large sets of data through bulk operations in read and append-only workloads. In these contexts, the proposed protocol improves state-of-the-art by reducing network overheads thanks to an original combination of aggregate bilinear map signatures and extractable collision resistant hash functions.

Keywords: database · outsourcing · cloud · integrity · authenticity · completeness · aggregate · signature · accumulator · bilinear map · ECRH · BLS

1 Introduction

The adoption of cloud services and other data outsourcing solutions is often hindered by data confidentiality needs and by limited trust about the correctness of operations performed by the service provider. Data confidentiality issues are addressed by several proposals based on encryption schemes (e.g., [10, 12, 21]). The correctness may be guaranteed through standard *authenticated data structures* [15, 24] based on message authentication codes [1] and digital signatures [19] that are affected by large network overheads and by limited database operations. Recent proposals, such as [13, 16, 17, 20], improve standard protocols but they cannot be adopted to guarantee results correctness in outsourced key-value databases because they incur either in network overheads [13, 16, 20] or in high computational costs [16, 17, 9]. For these reasons, we propose Bulkopt, a novel protocol that allows us to detect unauthorized modifications on outsourced data, as well as the correctness of all results produced by a cloud database service. Bulkopt guarantees authenticity, completeness and freshness of results produced by outsourced databases including cloud related services. It is specifically designed to work efficiently in read and append-only workloads possibly characterized by bulk operations, where large amounts of records may be inserted in the key-value database through one write operation. Moreover, Bulkopt supports

efficient fine-grained data retrievals by reducing network overhead related to the verification of bulk read operations in which multiple, possibly dispersed, keys are retrieved at once.

Closer cryptographic protocols [8, 14] proposed for memory checking data model [5] efficiently support operations on large numbers of records, but they do not support standard database queries and they cannot be immediately extended to database outsourcing scenarios. Bulkopt supports standard insert and read operations on key-value databases and limits communication overhead and verification costs of bulk operations. It recasts the problem of verifying the correctness of results produced by an untrusted database in terms of set operations by leveraging an original combination of *bilinear map aggregate signatures* [7] and *extractable collision resistant* (ECR) hash functions [4, 8].

The remainder of the paper is structured as following. Section 2 outlines the system and threat models assumed by the Bulkopt protocol. Section 3 describes the main ideas behind the Bulkopt protocol and outlines the high-level design of the solution. Section 4 proposes the implementation based on aggregate signatures and ECR hash functions. Section 5 outlines the Bulkopt main contributions and compares it with related work. Finally, Section 6 concludes the paper and outlines future work.

2 System and threat models

We adopt popular terminology for database outsourcing [23]. We identify a data *owner* that stores data on a database *server* managed by an untrusted *service provider*, and many authorized *users* that retrieve data from the server. The server offers a query interface that can be accessed by the data owner and the authorized users to retrieve values by providing a set of keys. We consider a *publicly verifiable* setting [23] and assume that only the data owner knows his private key, that is required to insert data into the database, and that authorized users know the public key of the owner that is required to verify results produced by the server. We note that in this first version of the protocol, we do not consider delete and update operations and focus on efficient insert and read database operations.

Our threat model assumes that the owner and all users are honest, while the server is untrusted. In particular we assume that the server (or any other unauthorized party, that does not have legitimate access to the private key) may try to insert, modify and delete data on behalf of the owner. The Bulkopt protocol allows all users and the owner to verify the correctness of all results produced by the server. We distinguish three types of results violations:

- **authenticity**: results that contain records that have never been previously inserted by the data owner or that have been modified after insertion;
- **completeness**: results that do not include all keys requested by the client but that have been previously inserted by the data owner;
- **freshness**: results that are based on an old version of the database. In the considered operation workload the server can only violate freshness if he

returns results that are both authentic and complete, but refer to an old version of the database.

3 Protocol overview

We describe the formal model used by Bulkopt to represent data and operations (Section 3.1) and to express authenticity and completeness guarantees as set operations (Sections 3.2 and 3.3). We note that since in this version of the protocol we do not consider delete and updates, the server can only violate freshness if he returns results that are both authentic and complete, but that refer to an old version of the database. As a result, clients can detect freshness violations by always using updated cryptographic digest to compute authenticity and completeness proofs. For details about verification operations please refer to the candidate implementation of the protocol described in Section 4.

3.1 Data model

We model the key-value database as a set of tuples $D = \{(k, v)\}$, where k is the key and v is the value associated to k. The owner populates the key-value database by executing one or more insert operations. For each insert operation the owner sends a set of tuples $B_i = \{(k, v)\}$, where i is an incremental counter that uniquely identifies an insert operation. The set B_i contains at least one tuple, and may contain several tuples in case of bulk insertions. Without loss of generality, in the following we refer to each set of tuples B_i as a *bulk*. We define as K_i the set of keys included in B_i, and $D_n = \cup_{i=1}^{n} B_i$ the set of records stored in the database after n bulk insertions.

We assume that the server has access to a lookup function that given a set of keys $\{k\}$ allows him to retrieve the set of insert operation identifiers $\{i\}$ in which these keys were sent by the owner. Such function can be obtained by deploying any standard indexing data structure of preference (e.g., a B-tree).

Any client (including the owner) can issue a *read operation* requesting an arbitrary set of keys $X = \{k\}$. If the server behaves correctly he must return the subset of the database A, defined as:

$$A = \{(k, v) \in D_n \mid k \in X\} \tag{1}$$

We define R as the set of keys included in A, that is:

$$R = \{k \in X \mid (k, v) \in A\} \tag{2}$$

While executing read operations issued by clients, the server distinguishes two different sets of keys: T and $\bar{T}$.

T is the union of all sets K_i that contain at least one key among those requested by a client:

$$T = \bigcup K_i \mid K_i \cap X \neq \emptyset \tag{3}$$

Within each K_i we identify two subsets of keys: $R_i = K_i \cap X$ and $Q_i = K_i \backslash R_i$. We define Q as the union of all sets Q_i, and we note that the union of all sets R_i is equal to set R (see Equation (2)). Thus, set Q is the complement of R in T.

$\bar{T}$ is the union of all sets K_i that do not contain any key among those requested by a client:

$$\bar{T} = \bigcup K_i \mid K_i \cap X = \emptyset \tag{4}$$

To better explain how these sets are built and the relationships among them, we refer to a simple example shown in Figure 1. In this example we have a key-

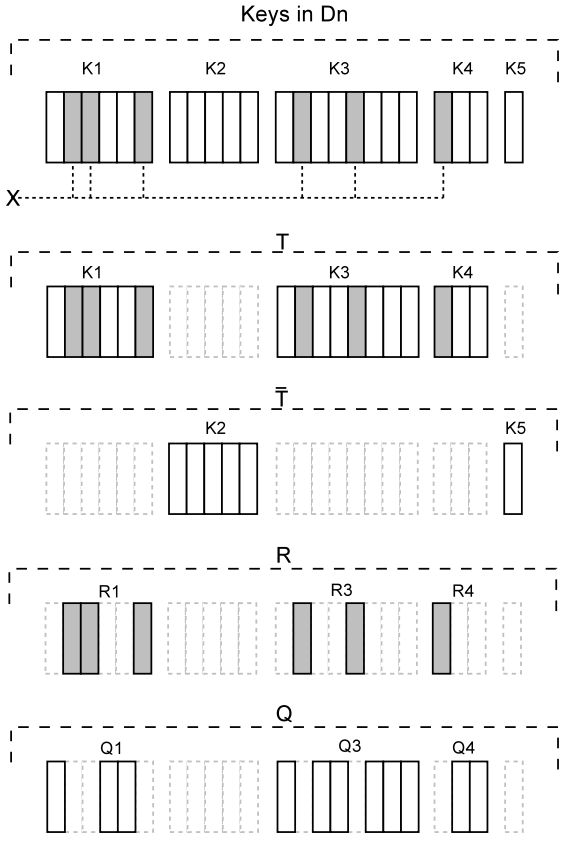

Fig. 1. Example of sets computed over a key-value database.

value database on which the owner already executed five bulk insert operations, each involving a different amount of tuples. The keys included in the database are represented by sets K_1 to K_5. We assume that a legitimate client executes

a read operation, asking to retrieve six keys belonging to three different bulks. The set of keys requested is represented by X. Since X includes keys belonging to bulks K_1, K_3 and K_4, all keys of these bulks belong to T, while $\bar{T}$ includes all keys belonging in the remaining bulks (K_2 and K_5). Sets R_1, R_3 and R_5 include only the keys requested by the client and belonging to K_1, K_3 and K_5, respectively. Set R includes all the keys belonging to the union of R_1, R_3 and R_5. Sets Q_1, Q_3 and Q_5 include only the keys that were not requested by the client and that belong to K_1, K_3 and K_5, respectively. Finally, set Q includes all the keys belonging to the union of Q_1, Q_3 and Q_5.

Sets Q and $\bar{T}$ are the main building blocks that Bulkopt leverages to identify a violation of the security properties or to prove the correctness of results produced by the server.

3.2 Authenticity

Bulkopt builds proofs of authenticity by demonstrating that:

$$R \cup Q \cup \bar{T} = K_D \tag{5}$$

where K_D represents the set of keys included in D_n. We recall from Section 2 that authenticity is violated if the server produces a result containing a key that has not been inserted by the owner. Let us assume that R includes a fake key k_f that has been created by the server but does not belong to K_D. Then it is obvious that Equation (5) does not hold, since R is not a subset of K_D.

An obvious solution to demonstrate that R is a subset of K_D would be for the client to have the complete set K_D. Of course this is not applicable, since it would require all clients to maintain a local copy of the whole key-value database.

To overcome this issue, Bulkopt requires the owner to maintain a cryptographic accumulator $\sigma(K_D)$ that represents the state of the keys stored in the database D_n. This accumulator is updated after each insert operation and has to be available to all users. Moreover, the server builds two witness data structures W_Q and $W_{\bar{T}}$ that represent the sets Q and $\bar{T}$, and sends them to the client together with its response A. We remark that cryptographic accumulators and witnesses are small and fixed-size data structures, that can be transmitted with minimal network overhead [3, 6].

To verify Equation (5) a client can extract the set of keys R from A, and use two accumulators verification functions. In particular, it checks whether the witness data structures received by the database validates the results with respect to the requested data and the current state of the database that is maintained locally. Intuitively, the client verification process can be represented as following:

$$\mathit{verify}\left(\mathit{verify}\left(\sigma(R), W_Q\right), W_{\bar{T}}\right) \stackrel{?}{=} \sigma(K_D) \tag{6}$$

where *verify* denotes accumulators verification functions.

If Equation (6) is verified, then the user knows that the two witnesses produced by the server are correct and that Equation (5) is also verified. Hence R is

a subset of K_D and authenticity holds. On the other hand, if Equation (6) is not verified, either the witnesses produced by the server are not correct or R is not a subset of K_D. In both cases, the client is able to efficiently detect a misbehavior of the server.

3.3 Completeness

Bulkopt builds proofs of completeness by demonstrating that:

$$X \cap (K_D \backslash R) = \emptyset \tag{7}$$

that is, the set of keys requested by the client X and the set of keys not returned by the server $K_D \backslash R$ share no common keys. We recall that $K_D \backslash R$ is equal to $Q \cup \bar{T}$, hence Equation (7) can be expressed as the following equation:

$$X \cap (Q \cup \bar{T}) = \emptyset \tag{8}$$

Bulkopt proves such conditions by leveraging properties of ECR hash functions. In particular, as shown by [8], ECR hash functions can be used to efficiently express set intersections by using polynomial representations of sets. That is, an empty intersection between sets correspond to polynomials having *great common divisor* (*gcd*) equal to 1 (that is, informally we say that since the sets do not share any common elements, the corresponding polynomials do not have common roots).

Let us denote as $\mathcal{C}_M(s)$ a polynomial representation of a generic set M w.r.t. variable s [8, 11], and a set $P = Q \cup \bar{T}$. To prove that the *gcd* of the polynomials is 1, the server must generate two polynomials $\dot{p}, \dot{x}$ such that:

$$\mathcal{C}_P \cdot \dot{p} + \mathcal{C}_X \cdot \dot{x} = 1, \tag{9}$$

The server sends witnesses W_P, $W_{\dot{p}}$ and $W_{\dot{x}}$ in addition to W_Q and $W_{\bar{T}}$ that were already sent to prove authenticity. A user can now exploit verification functions of the considered cryptographic signature to verify Equation (9). If Equation (9) is verified, then the client knows that the witnesses produced by the server are correct and that Equation (7) is also verified. Hence R includes all keys X requested by the client that are available in the server database, and completeness holds. On the other hand, if Equation (9) is not verified, either the witnesses produced by the server are not correct or X shares common elements with sets of keys Q or $\bar{T}$ that were not sent by the server, thus violating completeness. In both cases, the client is able to efficiently detect a misbehavior of the server.

4 Protocol Implementation

In this section we describe the Bulkopt protocol by referring to its main three phases: *setup and key generation* (Section 4.1), *insert operations* (Section 4.2) and *read operations* (Section 4.3).

4.1 Setup and key generation

Setup. Let g be a generator of the cyclic multiplicative group $\mathbb{G}$ of prime order p, $\mathbb{G}_T$ a cyclic multiplicative group of the same order and $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the pairing function that satisfies the following properties: bilinearity: $\hat{e}(m^a, n^b) = \hat{e}(m, n)^{ab}\, \forall m, n \in \mathbb{G}, a, b \in \mathbb{Z}_p^*$; non-degeneracy: $\hat{e}(g, g) \neq 1$; computability: there exists an efficient one-way algorithm to compute $\hat{e}(m, n)$, $\forall m, n \in \mathbb{G}$.

Let h be a cryptographic hash function and $h_z(\cdot), h_g(\cdot)$ be two full domain hash functions (FDH) secure in the random oracle model [2, 7] defined as following:

$$h_z : \{0,1\}^* \to \mathbb{Z}_p^* \tag{10}$$

$$h_g : \{0,1\}^* \to \mathbb{G} \tag{11}$$

Let us denote as $C_M(s)$ the characteristic polynomial that uniquely represents the set M, generated by using as roots of the polynomial the sum opposite of the elements of the set and as variable the secret key s [22]. Polynomial $C_M(s)$ can be computed as following:

$$C_M(s) = \prod_{m \in M} (m + s) \tag{12}$$

Let $F_M = (f(M), f'(M))$ be the output of an extractable collision resistant (ECR) hash function [4] with secret key $(s, \alpha) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ and public key $[g, g^s, \ldots, g^{s^q}, g^\alpha, g^{\alpha s}, \ldots, g^{\alpha s^q}]$, where M denotes a set of values $m \in \mathbb{Z}_p^*$. The output of the function can be computed through two different algorithms depending on the knowledge of the secret key s. For this reason, we denote as $\big(f_{sk}(M), f'_{sk}(M)\big)$ the computation of $(f(M), f'(M))$ with knowledge of the secret key and $\big(f_{pk}(M), f'_{pk}(M)\big)$ the computation of $(f(M), f'(M))$ with only knowledge of the public key. We will use notation $F_M, f(M)$ and $f'(M)$ to identify the black-box outputs of the functions when it is indifferent if they were computed with or without knowledge of the secret key. Functions $f_{sk}(M)$ and $f_{sk}(M)$ can be computed by using straightforwardly the polynomial $C_M(s)$ shown in Equation (12) as following:

$$f_{sk}(M) = g^{C_M(s)} = g^{\prod_{i=1}^{|M|}(m_i + s)}, \tag{13}$$

$$f'_{sk}(M) = g^{\alpha C_M(s)} = g^{\alpha \prod_{i=1}^{|M|}(m_i + s)}, \tag{14}$$

Functions $f_{pk}(M)$ and $f'_{pk}(M)$ can be computed by using the coefficients of the polynomial $C_M(s)$. That is, if we consider the set of the coefficients $\{a_i\}_{i=[1,\ldots,|M|]}$ of the polynomial $C_M(s)$ such that $C_M(s) = \sum_{i=1}^{|M|} a_i \cdot s^i$, $f_{pk}(M)$ and $f'_{pk}(M)$

can be computed as following:

$$f_{pk}(M) = \prod_{i=1}^{|M|} \left(g^{s^i}\right)^{a_i} \tag{15}$$

$$f'_{pk}(M) = \prod_{i=1}^{|M|} \left(g^{\alpha s^i}\right)^{a_i} \tag{16}$$

Although functions $(f_{sk}(\cdot), f'_{sk}(\cdot))$ and $(f_{pk}(\cdot), f'_{pk}(\cdot))$ have the same behavior, computing of $(f_{sk}(\cdot), f'_{sk}(\cdot))$ is more efficient due to the computation of only one exponentiation in the group $\mathbb{G}$. Without knowledge of the secret key, ECR hash functions can be verified as following:

$$\hat{e}(f(M), g^{\alpha}) \stackrel{?}{=} \hat{e}(f'(M), g) \tag{17}$$

Otherwise, the secret key allows a more efficient verification:

$$f(M)^{\alpha} \stackrel{?}{=} f'(M) \tag{18}$$

Although knowledge of the secret key improves the algorithm efficiency, it allows one to cheat in the computation of the hash function. Hence, it cannot be given to parties that have advantages in breaking the security of the ECR hash function.

Key Generation. We denote the owner's secret and public keys as sk and pk and generate them as follows:

$$sk = (u, s, \alpha),\ (u, s, \alpha) \xleftarrow{R} \mathbb{Z}_p^* \times \mathbb{Z}_p^* \times \mathbb{Z}_p^* \tag{19}$$

$$pk = (U, [g^s, \ldots, g^{s^q}, g^{\alpha}, g^{\alpha s}, \ldots, g^{\alpha s^q}]),\ U = g^u \tag{20}$$

where $q \in \mathbb{N}$ must be greater than or equal to the maximum number of records involved for each insert or read operation, and u, s and α be different from each other.

4.2 Insert operations

The owner issues an insert operation by sending the tuple $(B_i, \sigma_i, \Gamma_i)$, where:

- $i \in \mathbb{N}$ is the *operation identifier*, that is the incremental counter maintained locally by the owner and by the server that identifies the insert operation (see Section 3);
- $B_i = \{(k, v)\}$ is the set of keys and records inserted in the database at operation i. We also denote as K_i the set of the keys $\{\mathrm{k}\}$ inserted in this operation;
- σ_i is the *bulk signature* of the set of keys K_i inserted at operation i. It is computed by the tenant as:

$$\begin{aligned} \sigma_i(K_i) &= \left([h_g(i) \cdot f_{sk}(K_i)]^u, \left[h_g(i) \cdot f'_{sk}(K_i)\right]^u\right) = \\ &= \left(\left[h_g(i) \cdot g^{\prod_{k \in K_i}(k+s)}\right]^u, \left[h_g(i) \cdot g^{\alpha \prod_{k \in K_i}(k+s)}\right]^u\right) \end{aligned} \tag{21}$$

– Γ_i is the set of the *record signatures* of the records B_i, computed by using a BLS aggregate signature scheme [7]:

$$\Gamma_i(B_i) = \{\gamma_i(k,v)\}_{(k,v)\in B_i} \tag{22}$$

$$\gamma(k,v) = h_g(k \parallel v)^u \tag{23}$$

where $\parallel$ denotes the concatenation operator. We assume that the concatenation of the values k and v does not compromise the security of $h_g(\cdot)$. If the security of the candidate implementation of $h_g(\cdot)$ in this context, one should apply a collision resistant hash function or a message authentication code algorithm on the value v previous to the concatenation operation [1].

We note that the *bulk signature* σ_i (Equation (21)) is similar to the computation of a bilinear map accumulator [18]. The original scheme would compute the signature of $f_{sk}(K_i)$ as $f_{sk}(K_i)^u$. Our scheme differs for the factor $h_g(i)^u$, that could be seen as a BLS signature of the operation identifier i. This variant allows us to bind the bulk signature $\sigma_i(K_i)$ to the operation identifier i in which the insert operation is executed. As we describe in Section 4.3, this design choice also allows us to verify correctness of the server answers by using security proofs that were originally proposed for the memory checking setting [8].

Both the owner and the server keep track of the operation identifier i locally, without exchanging it in each insert operation. After each insert operation, the server stores all records B_i, the bulk signatures σ_i and the record signatures Γ_i in the database associated to the operation identifier i.

The owner does not store any bulk signature σ_i or record Γ_i, but he maintains a cryptographic structure of constant size to keep track of the state of the database. We call it the *database signature* $\mathcal{D} = (\sigma^\star{}_{last}, F_{D_{last}})$, where *last* is the value of the operation identifier i for the last insert operation executed on the server, and $\sigma^\star_{last}$ and $F_{D_{last}}$ are the bulk signature and ECR hash function of all the keys inserted in the database.

The owner computes the bulk signature $\sigma^\star_{last}$ as following:

– after the first insertion ($i = 1$) he sets the initial value of the database signature as $\sigma^\star_1 = \sigma_1$;
– after any other insert operation ($i > 1$), the owner computes the database signature $\sigma^\star_i$ by computing the product of the current version of the database signature $\sigma^\star_{i-1}$ and the bulk signature σ_i of the last executed insert operation as $\sigma^\star_i = \sigma^\star_{i-1} \cdot \sigma_{i-1}$.

As a result, the value of the database signature $\sigma^\star_{last}$ is equal to the product of all the bulk signatures σ_i ever sent by the owner to the server:

$$\sigma^\star_{last} = \prod_{i=1}^{i=last} \sigma_i \tag{24}$$

The owner computes the database ECR hash function $F_{D_{last}}$ as following:

– after the first operation ($i = 1$), the database accumulator is equal to the ECR hash function of the keys included in the first bulk of data, that is $F_{D_1} = (f_{sk}(K_1), f'_{sk}(K_1))$;
– after any other operation ($i > 1$), the database accumulator is computed as $F_{D_i} = F_{D_{i-1}}^{C_{K_i}(s)}$.

As a result, the value of $F_{D_{last}}$ after the last insert operation is the following:

$$F_{D_{last}} = (g^{\prod_{i=1}^{last} C_{K_i}(S)}, g^{\alpha \prod_{i=1}^{last} C_{K_i}(S)}) \tag{25}$$

4.3 Read operations

To execute a read operation a client must send a set of keys $X = \{k\}$ to the server. The server returns the following tuple:

$$response\,(X) := (I, A, \pi_{auth}, \pi_{comp}, \pi_{rec}) \tag{26}$$

where $I = \{i\}$ is the set of the operation identifiers associated to the bulks that include at least one of the keys X requested by the client; $A = \{A_i\}_{i \in I}$ is the set of the key-value records that compose the actual response to the client, grouped by the corresponding operation identifier i from which the server retrieved it; π_{auth}, π_{comp} and π_{rec} are the *keys authenticity proof*, the *keys completeness proof* and the *records authenticity proof* used to prove keys authenticity, completeness for the returned keys and authenticity of the values associated to the keys, respectively. Although from a security perspective keys authenticity and completeness proofs depend on each other, we distinguish them for the sake of clarity. We also observe that guaranteeing records correctness does not require any completeness proof because we are considering a key-value database where projection queries are not allowed. We recall from Section 3 that the elements of each set of the response A_i is a key-value tuple (k, v), and we denote as R_i the set of the keys included in the set A_i. In the following we describe separately the generation and the verification processes for keys authenticity proofs, keys completeness proofs and records authenticity proofs.

Keys authenticity. The keys authenticity proof is a tuple that includes the following values:

$$\pi_{auth} = (\{F_{Q_i}\}_{i \in I}, F_T, W_{\bar{T}}), \tag{27}$$

where $\{F_{Q_i}\}_{i \in I}$ is the set of the *bulk witnesses*, F_T is the aggregate ECR hash function of bulks that include at least one of the keys requested by the client, $W_{\bar{T}}$ is the aggregate bilinear signature of the bulks that do not include any of the keys requested by the client.

The server generates each bulk witness F_{Q_i} by computing the ECR hash function f_{pk} (see Equation (16)) on the set complement Q_i of R_i with respect to K_i, as following:

$$\begin{aligned} F_{Q_i} &= \big(f_{pk}(Q_i), f'_{pk}(Q_i)\big) = \big(f_{pk}(K_i \backslash R_i), f'_{pk}(K_i \backslash R_i)\big) = \\ &= \left(g^{C_{K_i \backslash R_i}(s)}, g^{\alpha \cdot C_{K_i \backslash R_i}(s)}\right), \forall i \in I \end{aligned} \tag{28}$$

Moreover, the server computes the aggregate bilinear signature $W_{\bar{T}}$ as the witness for bulks that do not include any keys requested by the client by aggregating the owner signatures as following:

$$W_{\bar{T}} = \prod_{i \in I} \sigma_i(K_i) = \left[\prod_{i \in I} h_g(i) g^{C_{K_i}} \right]^u \tag{29}$$

The client verifies authenticity of the keys $\{R_i\}$ returned by the server by using values included in the authentication proof π_{auth} and the database signature $\sigma^{\star}_{last}$ stored locally (see Equation (24)). The client verifies correctness of the ECR hash function F_T by using Equation (17). Then, the client verifies that the ECR hash function F_T is built correctly with respect to the aggregate bilinear signature $W_{\bar{T}}$ by using the locally maintained database signature $\sigma^{\star}_{last}$, as following:

$$\hat{e}\left(F_T, U\right) \stackrel{?}{=} \hat{e}\left(\frac{\sigma^{\star}_{last}}{W_{\bar{T}}}, g\right) \tag{30}$$

Finally, the client uses F_T to verify authenticity of the returned records $\{R_i\}_{i \in I}$ by using the bulk witnesses $\{F_{Q_i}\}_{i \in I}$, as following:

$$\hat{e}\left(\prod_{i \in I} h_g(i), g\right) \prod_{i \in I} \hat{e}\left(f_{pk}(R_i), F_{Q_i}\right) \stackrel{?}{=} \hat{e}\left(F_T, g\right) \tag{31}$$

After this verification process the client is sure about the following guarantees:

- F_T is a valid witness for the bilinear aggregate signature $W_{\bar{T}}$, as the probability of generating or extracting any other owner signature would break the non-extractability guarantees of aggregate bilinear signatures [7];
- all the returned keys $\{R_i\}_{i \in I}$ are authentic, because the server proved existence of the witnesses Q_i with respect to bulks aggregate hash function F_T and generating false witnesses would break extractable collision resistance (ECR) guarantees of the ECR hash function $(f(\cdot), f'(\cdot))$ [8];
- all the operation identifiers $i \in I$ sent by the client are authentic, as generating identifiers that satisfy Equation (31) would break either the FDH function $h_g(\cdot)$ or the collision resistance guarantees of aggregate bilinear signatures [7].

Keys completeness. As described in Section 3.3, to prove completeness of the response the server must produce witnesses that prove disjunction the requested keys X with respect to the complement sets Q and $\bar{T}$. The completeness proof is a tuple that includes such witnesses, and additional values that allow the client to verify that the server generated them correctly:

$$\pi_{comp} = (F_P, F_{\dot{p}}, F_{\dot{x}}), \tag{32}$$

where F_P is the ECR hash function of the set union including the complement sets Q and $\bar{T}$, ($F_{\dot{q}}$ and $F_{\dot{x}}$) the witnesses that prove disjunction of the set of the requested keys X with respect to sets $\bar{T}$ and Q.

First, the server computes the ECR hash function of $Q \cup \bar{T}$ as:

$$F_P = \left(f_{pk}(Q \cup \bar{T}), f'_{pk}(Q \cup \bar{T})\right) = \left(g^{C_{Q \cup \bar{T}}(s)}, g^{\alpha \cdot C_{Q \cup \bar{T}}(s)}\right) \tag{33}$$

The two witnesses $F_{\dot{p}}$ and $F_{\dot{x}}$ of polynomials $\dot{x}$ and $\dot{p}$ are generated by the server to show that the *gcd* between the characteristic polynomials C_X and $C_{Q \cup \bar{T}}$ of sets X and $Q \cup \bar{T}$ is 1, that is equivalent to prove disjunction of sets X, Q and $\bar{T}$, as shown in [8]:

$$\dot{x}, \dot{p} : C_X(s) \cdot \dot{x} + C_P(s) \cdot \dot{p} = 1 \tag{34}$$

$$F_{\dot{p}} = \left(f_{pk}(\dot{p}), f'_{pk}(\dot{p})\right) \tag{35}$$

$$F_{\dot{x}} = \left(f_{pk}(\dot{x}), f'_{pk}(\dot{x})\right) \tag{36}$$

The client verifies correctness of the ECR hash functions $F_P, F_{\dot{q}}$ and $F_{\dot{x}}$ sent by the server by using Equation (17). Then, he verifies whether F_P represents the set complement of R with respect to D by checking the value of F_P against the database accumulator $F_{D_{last}}$ (see Equation (25)) publicly distributed by the owner:

$$\hat{e}\left(f_{pk}(R), F_P\right) \stackrel{?}{=} \hat{e}\left(F_{D_{last}}, g\right) \tag{37}$$

Now that the client verified the correct generation of the witnesses F_P, he can verify disjunction of X, Q and $\bar{T}$ by testing Equations (34) as following:

$$\hat{e}(f_{pk}(X), F_{\dot{x}}) \cdot \hat{e}(F_P, F_{\dot{p}}) \stackrel{?}{=} \hat{e}(g, g) \tag{38}$$

Records authenticity. The server computes the proof of authenticity π_{rec} by aggregating all the record signatures $\gamma_{k,v} = \gamma(k, v)$ previously received by the owner for all the records returned to the client, as following:

$$\pi_{rec} = \prod_{(k,v) \in A_i, \forall A_i \in A} \gamma_{k,v} \tag{39}$$

The client verifies authenticity of the response A given the server integrity proof π_{int} and the owner public key U by verifying the following condition:

$$\hat{e}\left(\prod_{(k,v) \in A_i, \forall A_i \in A} h_g(k \parallel v), U\right) \stackrel{?}{=} \hat{e}(\pi_{rec}, g) \tag{40}$$

This concludes the description of the protocol: any client that is enabled to query the database and that knows the owner's public key pk and the state of the database $\mathcal{D}$ can verify correctness of the results by using the described verification operations. We recall that if a client knows the secret key sk, such as in symmetric settings, he can verify results correctness more efficiently by using the secret exponents u and α.

5 Related Work

Most literature related to security of data outsourcing and cloud services aims to protect data confidentiality of tenant data against malicious insiders of cloud providers. These works typically assume the honest-but-curious threat model where an insider within the cloud provider may access and copy tenant data without corrupting or deleting them. To solve this issue several works already proposed in the literature leverage architectures based on partially homomorphic and property preserving encryptions that allow cloud computations and efficient retrieval on encrypted data (e.g., [10, 12, 21]). Unlike these works, in this paper we do not trust the cloud provider to behave correctly, but we assume a threat model where the cloud provider can violate authenticity and completeness of tenant data, either due to hardware/software failures or deliberate attacks. The main problem in this context is to combine authenticity and completeness guarantees without affecting the database performance and functionalities. As an example, standard message authentication codes or digital signatures can guarantee authenticity of outsourced data. However, they cannot guarantee results completeness without incurring in great network overhead.

A well-known solution to guarantee results correctness is to adopt Merkle hash trees [9], that allow to build efficient proofs for range queries by authenticating the sorted leafs of the tree with respect to an index defined at design time. However, they do not support efficient queries on arbitrary values and efficient proofs on dispersed key values. Other solutions allow the tenant to verify authenticity and completeness of outsourced data by means of RSA accumulators [16, 17, 13]. Although RSA accumulators provide constant asymptotic complexity for read and update operations, their high constant computational overhead often prevent their practical application in most scenarios [9]. A different approach is proposed in [25], that relies on the insertion of a number of fake records in the database. These records are then retrieved to verify their presence, and possibly identify completeness violations. However, since no cryptographic verification is executed on the real database, such a solution provides lower security guarantees based on probabilistic completeness verification. The protocols proposed in [8] guarantees authenticity of operations in a memory-checking model by maintaining an N-ary tree of constant height. Since only the values of the nodes change (but not the number of cells), these protocols can produce proofs of constant size with respect to the cardinality of the sets stored in each memory cell. However, their proposal cannot be easily adopted in the data outsourcing scenario because the amount of sets is not constant and the tree structure would require expensive re-balancing operations.

6 Conclusion

This paper proposes Bulkopt, a novel protocol that provides authenticity and completeness guarantees for key-value databases. Bulkopt is specifically designed for providing data security guarantees in the context of cloud-based services subject to read/write workloads, and efficiently support bulk insert operations, as

well as read requests that involve the retrieval of multiple and not contiguous keys at once. Efficient verification of bulk operations is achieved by modeling data security constraints in terms of set operations, and by leveraging cryptographic proofs based for set operations. In particular, Bulkopt is the first protocol that combines extractable collision resistant hash functions and aggregate bilinear map signatures to achieve novel cryptographic constructions that allow the verification of authenticity and completeness over large sets of data by relying on small cryptographic proofs. More work is needed to tune the protocol performance by using data structures to cache partial proofs at the server side, as well as further developments to also support update operations.

Acknowledgments. The authors acknowledge the support of HORIZON 2020 FCT-1-2015 project ASGARD "Analysis System for Gathered Raw Data".

References

1. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Proc. 1996 IACR Int'l Conf. Advances in Cryptology*. Springer.
2. M. Bellare and P. Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *Proc. 1996 Int'l Conf. Advances in Cryptology*. Springer.
3. J. Benaloh and M. De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Proc. 1993 IACR Int'l Conf. Advances in Cryptology*. Springer.
4. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proc. 2012 ACM Third Int'l Conf. Innovations in Theoretical Computer Science*.
5. M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2-3), 1994.
6. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in cryptology*. Springer, 2003.
7. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Proc. 2001 IACR Int'l Conf. Advances in Cryptology*. Springer.
8. R. Canetti, O. Paneth, D. Papadopoulos, and N. Triandopoulos. Verifiable set operations over outsourced databases. In *Proc. 2014 IACR Int'l Conf. Public-Key Cryptography*. Springer.
9. S. A. Crosby and D. S. Wallach. Authenticated dictionaries: Real-world costs and trade-offs. *ACM Trans. Information and System Security*, 14(2):17, 2011.
10. L. Ferretti, M. Colajanni, and M. Marchetti. Distributed, concurrent, and independent access to encrypted cloud databases. *IEEE Transactions on Parallel and Distributed Systems*, 25(2), 2014.
11. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proc. 2004 IACR Int'l Conf. Advances in Cryptology*. Springer.
12. H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proc. 2002 ACM SIGMOD Int. Conf. Management of Data*, 2002.
13. J. Hong, T. Wen, Q. Gu, and G. Sheng. Query integrity verification based-on mac chain in cloud storage. In *Proc. 13th IEEE/ACIS Int. Conf. Computer and Information Science*, 2014.

14. A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos. Trueset: Faster verifiable set computations. In *Proc. 23rd USENIX Int'l Conf. Security Symposium*, 2014.
15. A. Miller, M. Hicks, J. Katz, and E. Shi. Authenticated data structures, generically. *ACM SIGPLAN Notices*, 49(1), 2014.
16. E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *ACM Trans. Storage*, 2(2), May 2006.
17. M. Narasimha and G. Tsudik. Dsac: An approach to ensure integrity of outsourced databases using signature aggregation and chaining. Cryptology ePrint Archive, Report 2005/297, 2005.
18. L. Nguyen. Accumulators from bilinear pairings and applications. In *Proc. 2005 Int'l Conf. Topics in Cryptology*. Springer, 2005.
19. Nist. Digital signature standard. Technical report, Jul. 2013.
20. R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang. Enabling security in cloud storage slas with cloudproof. In *Proc. USENIX Annual Technical Conf.*, 2011.
21. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *Proc. 23rd ACM Symp. Operating Systems Principles*, Oct. 2011.
22. F. P. Preparata and D. V. Sarwate. Computational complexity of fourier transforms over finite fields. *Mathematics of Computation*, 31(139), 1977.
23. P. Samarati and S. D. C. di Vimercati. Data protection in outsourcing scenarios: Issues and directions. In *Proc. 5th ACM Symp. Information, Computer and Communications Security*, 2010.
24. R. Tamassia. Authenticated data structures. In *Algorithms-ESA*. Springer, 2003.
25. M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *Proc. 33rd Int. Conf. Very Large Data Bases*. VLDB Endowment, 2007.