Query integrity verification in presence of access restrictions

Sabrina De Capitani di Vimercati[1][0000−0003−0793−3551], Sara Foresti[1][0000−0002−1658−6734], Sushil Jajodia[2][0000−0003−3210−558X], Samuele Manclossi[1][0009−0008−3549−9258], Stefano Paraboschi[3][0000−0003−0399−1738], and Pierangela Samarati[1][0000−0001−7395−4620]

[1] Università degli Studi di Milano, Italy
{sabrina.decapitani,sara.foresti,pierangela.samarati}@unimi.it
samuele.manclossi@studenti.unimi.it
[2] George Mason University, USA
jajodia@gmu.edu
[3] Università degli Studi di Bergamo, Italy
stefano.paraboschi@unibg.it

Abstract. Merkle Hash Trees represent a well-established solution for providing integrity guarantees to query results computed by an external provider. Similarly, several techniques have been proposed to enforce access control restrictions over outsourced data. However, when both query integrity and access control enforcement need to be guaranteed, their combined adoption can raise data confidentiality issues, potentially violating the access control policy. In this paper, we propose and compare different approaches combining Merkle Hash Trees for query integrity with access control, while protecting against unintended data leakage. Indeed, the analyzed solutions are characterized by distinct Merkle Hash Tree sizes, costs for integrity verification, and protection guarantees, making each of them suitable for a specific application scenario.

Keywords: Query integrity · access control · Merkle hash tree

1 Introduction

The increasing availability of cloud services for storing and managing data has made data outsourcing a widely adopted solution for organizations wishing to reduce the costs and complexity of maintaining in-house infrastructures. Besides reducing infrastructure costs, outsourcing makes data available anytime and anywhere, allowing users to easily access and query them. Since the external provider managing the outsourced data is typically outside the data owner's control, it might not be fully trustworthy and its (possibly lazy or malicious) behavior should be controlled. Users querying the outsourced data therefore need guarantees that the query results returned by the external provider are both correct and complete (query integrity). A common approach to enable users querying

data to assess the integrity of the query result is based on the definition of authenticated data structures, such as Merkle Hash Trees (MHTs), which allow the data owner to sign a compact representation of the outsourced data (e.g., the root of the MHT) [5]. Query results are then complemented with a verification object, built on these structures, allowing users to verify the integrity of query results.

Solutions addressing the query integrity problem typically assume that users issuing queries are authorized to access the whole outsourced dataset. In many real-world scenarios, access is selective and users may be authorized to access only a subset of the outsourced dataset. Solutions enforcing access control on data outsourced to an external, possibly untrusted, server have also been extensively investigated and are mainly based on encryption-based techniques supporting fine-grained data access (e.g., selective encryption) [4]. While the combined adoption of techniques for query integrity and for enforcing access control might seem straightforward, it introduces novel confidentiality issues. Since authenticated data structures reveal information on the outsourced data, they may leak information about data that a user is not authorized to access, potentially violating the access control policy. Clearly, restricting visibility on the MHT structure is not a viable solution, as it would compromise the possibility to verify the integrity of query results.

In this paper, we investigate different ways of combining Merkle Hash Trees and access control enforcement while preventing unintended information leakage. In particular, we compare authenticated structures built on users' capability lists with structures built on access control lists. The proposed solutions exhibit different trade-offs in terms of the size of the authenticated data structure, query verification overhead, and confidentiality guarantees, making each of them suitable for different application scenarios.

The remainder of this paper is organized as follows. Section 2 introduces the basic concepts. Section 3 illustrates the problem addressed in the paper. Section 4 presents the solution based on the definition of a capability-based MHT. Section 5 introduces an alternative solution based on the definition of ACL-based MHTs. Section 6 compares the proposed solutions in terms of storage overhead, verification costs, and confidentiality guarantees. Section 7 reports the experimental results. Section 8 presents the related work. Finally, Section 9 concludes the paper.

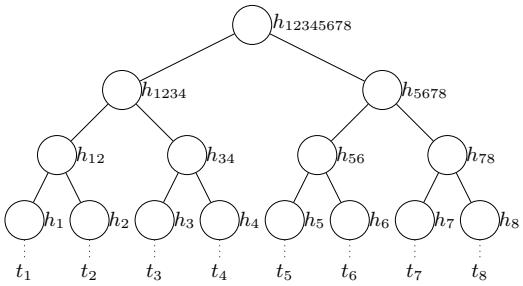
2 Basic concepts

We consider a data outsourcing scenario where a data owner stores, at an external service provider, a relation R defined over a set of attributes and containing a set of tuples. The provider is trusted to store the relation and execute queries on it on behalf of users, but it is not trusted to access its content. Different users may be authorized to access different subsets of the outsourced tuples. At the same time, users require guarantees on the correctness and completeness of query results (query integrity).

EMPLOYEES

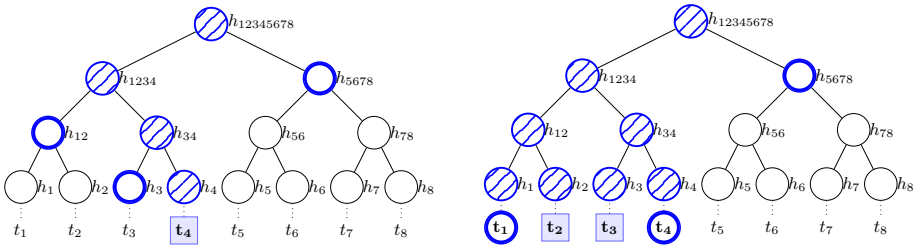
	Name	City	Salary
t_1	Hellen	Athens	1000
t_2	Rob	Rome	2000
t_3	Miky	Paris	3000
t_4	Zoe	NY	4000
t_5	Willy	Madrid	5000
t_6	Dan	Paris	6000
t_7	Nick	London	7000
t_8	Laura	NY	8000

(a)



(b)

Fig. 1. An example of relation (a) and corresponding MHT built over attribute `Salary`



(a) (b)

Fig. 2. An example of VO (blue-bordered nodes and tuples), tuples in the query result (light blue background), and nodes reconstructed by the client (blue diagonal striped nodes) for the verification of a point query (a) and of a range query (b)

Query integrity verification. We use a *Merkle Hash Tree* (MHT) [5] to provide integrity guarantees on query results. An MHT defined over a relation R is a binary tree having one leaf for each tuple in the relation. Each leaf stores the hash of the tuple it represents, computed through a one-way hash function h (e.g., a collision-resistant hash function such as SHA-1). The leaves are ordered according to the values of an attribute a of relation R defined over a totally ordered domain. Each internal node stores the hash of the concatenation of the values stored in its children. In other words, given an internal node n with children n_l and n_r, the value of n is computed as $n{=}h(n_l||n_r)$, where $||$ denotes concatenation. For simplicity, in the following we assume that no two tuples in R have the same value for the attribute a used to build the MHT. If a node has only one child, the second child is replaced by a *dummy node*. For instance, Figure 1 illustrates relation EMPLOYEES with three attributes (`Name`, `City`, and `Salary`) together with the corresponding MHT built over attribute `Salary`.

For the verification of the integrity of query results, the data owner signs and publishes the root of the MHT. Query results are complemented with a *Verification Object* (VO) that enables the querying user to reconstruct the root

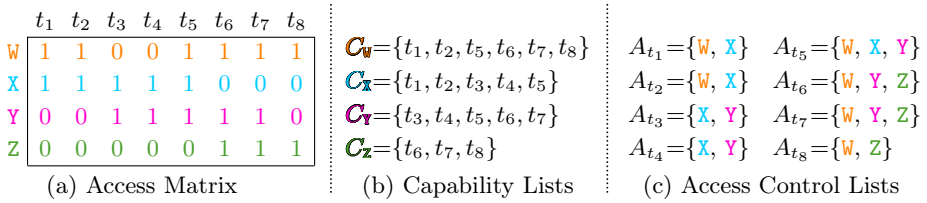


Fig. 3. An example of authorization policy for relation EMPLOYEES (a) and corresponding capability lists (b) and access control lists (c)

and verify the correctness and completeness of the result. For instance, consider a query retrieving the employee with `Salary`=4000 (tuple t_4 with a light blue background in Figure 2(a)). The corresponding VO includes the sibling of each node along the path from h_4 (hash of t_4) to the root (i.e., nodes h_3, h_{12}, and h_{5678} with a blue border in Figure 2(a)). Using the query result and the VO, the user can recompute the root and verify that it matches the signed root published by the data owner. Figure 2(a) illustrates the nodes recomputed by the user with a blue diagonal stripe. For range queries, the VO additionally includes the tuples immediately preceding and immediately following the result range. For instance, a query retrieving employees with `Salary` between 1100 and 3500 returns tuples t_2 and t_3, together with the VO that includes tuples t_1, t_4, and h_{5678}, proving that no tuple has been omitted from the result (Figure 2(b)). Similarly, for empty-result queries, the VO contains the tuples having, for attribute a, the values immediately preceding and immediately following the query value or range, allowing users to verify that the result is indeed empty.

Access control. The authorization policy defined over relation R specifies, for each tuple, the users, in the set $\mathcal{U} = \{u_1, \ldots, u_n\}$, who are authorized to read the tuple. The policy is represented via an access control matrix $\mathcal{A}$, with a row for each user in $\mathcal{U}$ and a column for each tuple in R. In the matrix, $\mathcal{A}[u, t]$=1 if user u is authorized to access tuple t; 0, otherwise. Figure 3(a) illustrates an example of access matrix defined for relation EMPLOYEES in Figure 1(a). In the following, we use $C_u \subseteq R$ to denote the *capability list* of user u, that is, the set of tuples the user is authorized to access, and $A_t \subseteq \mathcal{U}$ to denote the *access control list* of tuple t, that is, the set of users authorized to access tuple t. Figures 3(b)-(c) illustrate the capability lists and ACLs, respectively, corresponding to the access matrix in Figure 3(a).

We assume that the authorization policy $\mathcal{A}$ is enforced through a self-protection layer based on encryption. The specific technique adopted is outside the scope of this paper. For instance, the authorization policy can be enforced through *selective encryption*, where each tuple is encrypted with a key that all and only authorized users know or can derive [1]. We also assume that the outsourced encrypted relation is combined with standard techniques supporting point and range queries over the encrypted attribute a (e.g., [4]). Query processing over encrypted data is orthogonal to our contribution. In the following, for simplicity, examples will refer to the plaintext relation.

3 Problem definition

Providing integrity guarantees in the presence of access restrictions raises new privacy risks. Indeed, the information disclosed for integrity verification may reveal information about tuples that a user is not authorized to access.

A first problem is due to the topology of the MHT. Since the leaves are ordered according to the values of attribute a, the information disclosed through the VO may reveal that an unauthorized tuple lies between two tuples for which the user is authorized, allowing the user to infer an interval for the value of a. To illustrate the problem, suppose that user `W` submits a query retrieving all tuples for which the value of `Salary` is between 1100 and 6500. If the authorization policy is not considered, the result includes tuples $\langle t_2, \ldots, t_6 \rangle$, together with VO=$\{t_1, t_7, h_8\}$. Since tuples are encrypted in such a way to enforce the authorization policy in Figure 1(b), user `W` cannot decrypt tuples t_3 and t_4. From the topology of the MHT, `W` can infer that both t_3 and t_4 have a `Salary` between 2000 and 5000, and that t_3[`Salary`]$<t_4$[`Salary`]. In other words, although the exact values remain protected, user `W` learns information about tuples that the user is not authorized to access.

A second problem can arise from the VO itself. The VO of a range query or of a query with an empty result includes the tuples immediately preceding and following the result range (or, in some schemes, information derived from them). Again, if such boundaries are not authorized for the user, returning them as part of the VO would violate the authorization policy since it would leak information that the user is not authorized to know. For instance, assume user `X` submits a query retrieving all tuples with `Salary` between 3500 and 5500. The tuples to be returned would be the ones satisfying the condition (t_4 and t_5) as well as, in the VO, the boundary tuples t_3 and t_6. However, returning t_6 would violate the authorization policy since `X` is not authorized for it.

The main reason for both problems is that the MHT is built over all the tuples in the relation, regardless of authorizations holding on them. To avoid improper leakage, the authenticated data structure for a user should be built only on the tuples that the user can access. In this paper, we investigate two alternative solutions for enforcing such restriction. The first builds an MHT for each capability list (i.e., for each user), considering all and only the tuples in the capability (i.e., only the tuples accessible by that user) (Section 4). The second organizes tuples according to access control lists, grouping together tuples visible to the same set of users in specific MHTs (Section 5). Each solution is characterized by different sizes of the authenticated structure, size of the VOs complementing query results, and guarantees of confidentiality of the data and of the authorization policy (Section 6).

4 Capability-based MHT

A simple solution to avoid the inference problems previously described consists of building a MHT for each capability list (i.e., for each user), which contains

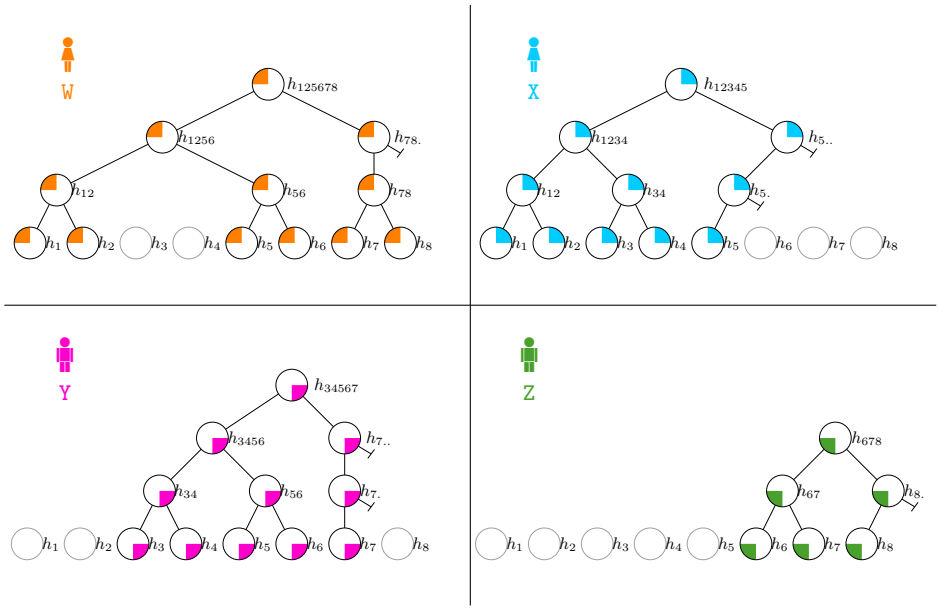


Fig. 4. C-MHTs built over the capability lists of our running example in Figure 3(b)

all and only the tuples in the capability list (i.e., accessible to the corresponding user). Every query submitted by a user u is then verified against the *capability-based MHT* (C-MHT) built over the tuples in C_u. Since the C-MHT refers only to authorized tuples, the corresponding VOs cannot reveal any information about unauthorized tuples, not even their existence. The C-MHT for a user u is formally defined as follows.

Definition 1 (C-MHT). *Let R be a relation, $\mathcal{U}$ be the set of users authorized for tuples in R, and $\mathcal{A}$ be the authorization policy over $\mathcal{U}$ for tuples in R. The* C-MHT *for a user $u \in \mathcal{U}$ over attribute a in R, denoted M_u, is an MHT built over the set $C_u = \{t \in R \mid \mathcal{A}[u,t] = 1\}$ of tuples.*

Figure 4 illustrates the four C-MHTs of our running example. In this figure (and in the following), a short segment attached to a node denotes a dummy node replacing a missing child. Queries submitted by a user u will be verified against the C-MHT built over the user's capability list. Hence, both the query result and the VO contain only tuples that the user is authorized to access and no improper information leakage can occur. However, having one MHT for each capability list results in a larger authenticated data structure. The size of the overall authenticated data structure grows with the number of authorizations (i.e., the number of 1s in $\mathcal{A}$). However, different C-MHTs may contain identical nodes that can be factorized and shared among multiple C-MHTs. As a matter of fact, the hash of tuples will appear in the C-MHT of every user authorized for them. For instance, in Figure 4, node h_2 appears in both $\mathrm{M}_{\mathtt{W}}$ for user W and in $\mathrm{M}_{\mathtt{X}}$

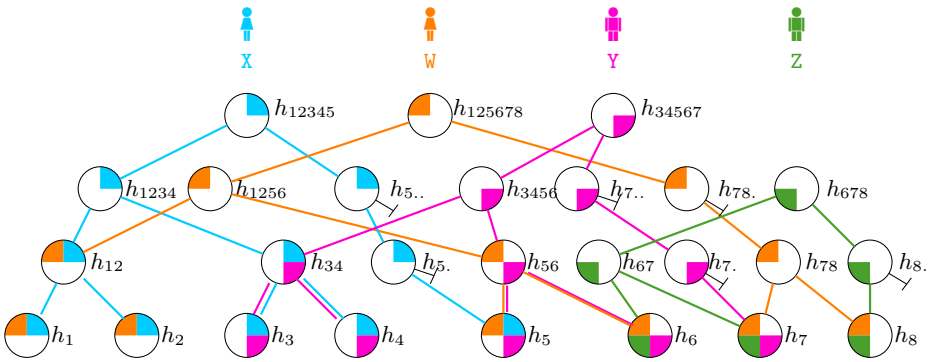


Fig. 5. Factorized version of the C-MHTs in Figure 4

for user X. Furthermore, if a set of tuples having contiguous values for attribute a share the same ACL, the C-MHTs of the users in the ACL may also share internal nodes. For instance, since both t_1 and t_2 are visible to W and X, not only leaf nodes h_1 and h_2, but also their parent node h_{12} appears in both $\mathrm{M}_{\mathtt{W}}$ and $\mathrm{M}_{\mathtt{X}}$ (see Figure 4). Intuitively, the longer the sequence of contiguous tuples visible to the same set of users, the larger the portion of C-MHT they have in common. Figure 5 illustrates the structure obtained by factorizing common nodes in the C-MHTs in Figure 4. This factorization saves 14 nodes. Dummy nodes can also be factorized: a single dummy node can be shared by all nodes at the same level that miss a child. With reference to Figure 5, two dummy nodes, instead of six, would suffice.

The VO of a query formulated by u is computed over M_u. Independently of node factorization, the VO is typically smaller than the VO built over the MHT defined on the whole relation R, since M_u is expected to contain fewer tuples (only those for which user u is authorized). As a consequence, different users may receive different VOs for the same query, even when the query result is identical. For instance, suppose user X submits a query for retrieving the employee with Salary=4000. The result of the query includes tuple t_4. The VO computed over $\mathrm{M}_{\mathtt{X}}$ in Figure 4 (and in Figure 5) includes h_3, h_{12}, and $h_{5..}$. If the same query is submitted by user Y, the query result is unchanged but the VO computed over $\mathrm{M}_{\mathtt{Y}}$ consists of h_3, h_{56}, and $h_{7..}$.

5 ACL-based MHT

We now consider an alternative approach that consists of partitioning tuples according to their ACLs. In other words, for each set of tuples accessible to the same set U of users we build an *ACL-based MHT* (A-MHT) over such tuples for the set U of users, which is formally defined as follows.

Definition 2 (A-MHT). *Let R be a relation, $\mathcal{U}$ be the set of users authorized for tuples in R, and $\mathcal{A}$ be the authorization policy over $\mathcal{U}$ for tuples in R. The*

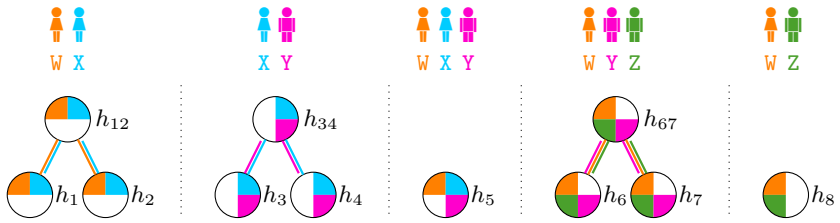


Fig. 6. A-MHTs built over the ACLs of our running example in Figure 3(c)

A-MHT *for a set* $U \subseteq \mathcal{U}$ *of users over attribute* a *in* R*, denoted* M_U*, is an MHT built over the set* $\{t \in R \mid A_t = U\}$ *of tuples.*

Compared to the C-MHTs introduced in the previous section, A-MHTs naturally avoid the duplication of leaf and internal nodes because each tuple belongs to exactly one ACL and it is therefore represented in a single MHT. Furthermore, we note that if the number of ACLs in the authorization policy is limited and many users share the same privileges (i.e., ACLs are large), A-MHTs are smaller than C-MHTs. For instance, Figure 6 illustrates the A-MHTs for our running example. It is easy to see that the overall number of nodes is lower than the number of nodes in Figure 5.

Since tuples are partitioned according to their ACLs, verifying the integrity of a query result for user u may require considering multiple A-MHTs. More precisely, for range and empty queries, the result must be complemented with a VO (computed as discussed in Section 2) for each A-MHT M_U such that $u \in U$, allowing the user to verify the integrity of the query result with respect to the tuples represented by M_U. Note that since A-MHTs are built over an attribute with unique values, the VO of a point query is built over the A-MHT containing the resulting tuple. For instance, consider the A-MHTs in Figure 6 and suppose that user `X` submits a range query. The query result must be associated with a VO for $\mathrm{M}_{\mathtt{WX}}$, $\mathrm{M}_{\mathtt{XY}}$, and $\mathrm{M}_{\mathtt{WXY}}$. To verify that queries have been evaluated over all the relevant A-MHTs, users should know all the ACLs in which they are involved. Otherwise, a malicious server could omit the VO of one or more relevant A-MHTs, and the user would not be able to detect this omission. We therefore complement the A-MHTs with an upper authenticated structure built over their roots, allowing users to verify that all relevant A-MHTs (i.e., all M_U such that $u \in U$) have been considered. Intuitively, the verification of a query result will include the VOs on relevant A-MHTs, proving that no tuple has been omitted, and the VO on the upper-level structure, proving that no A-MHT has been omitted. The use of an upper authenticated structure also avoids requiring users to keep multiple signed roots (one for each ACL in which they are involved). The upper authenticated structure can be built according to three strategies: a *single-tree* (SA-MHT) shared among all users; *multiple-trees* (MA-MHT), one for each user; or *hash-based* (HA-MHT), one node for each user. These strategies differ in how users verify that no relevant A-MHT has been omitted.

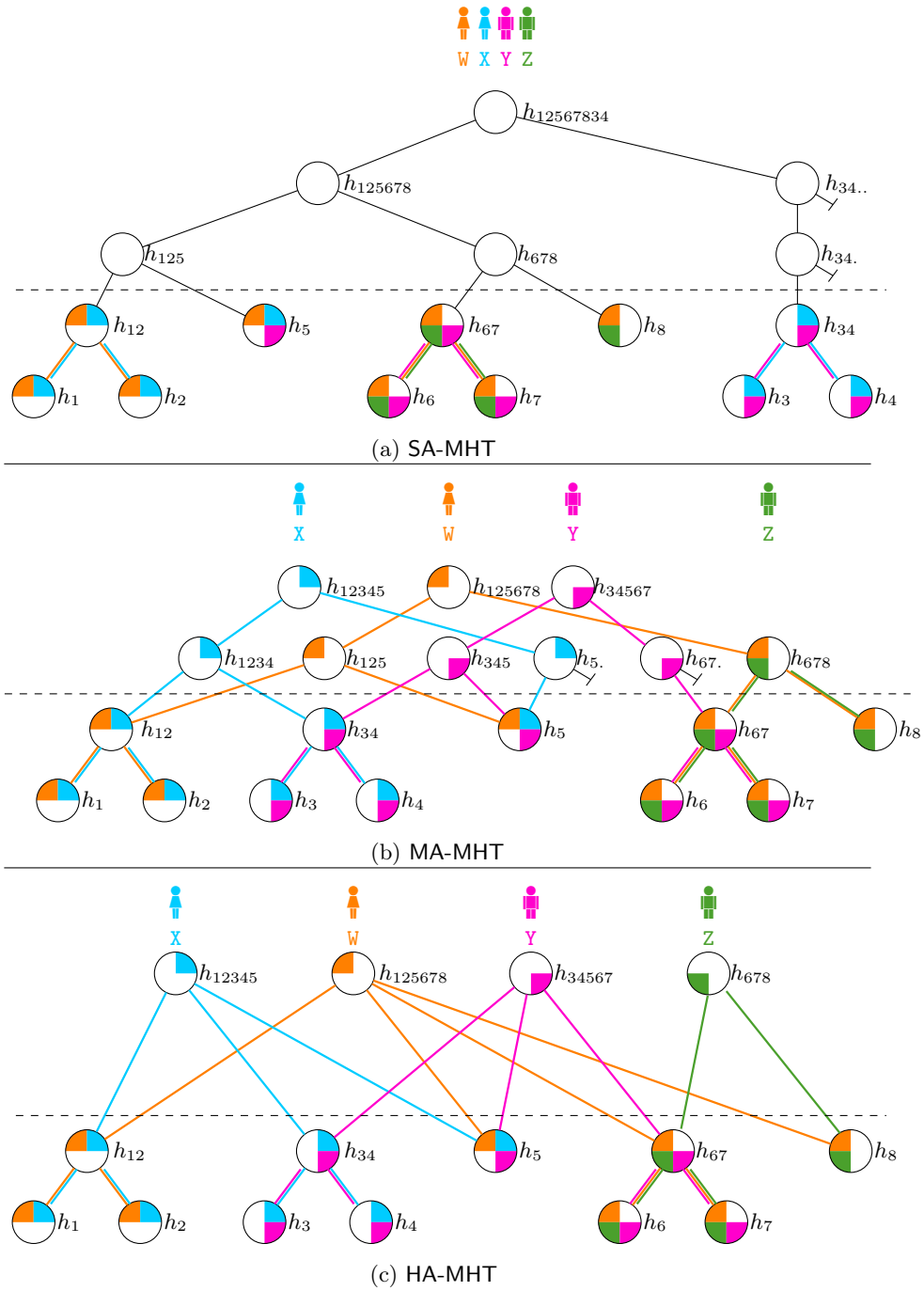


Fig. 7. An example of upper authenticated structure with the single-tree (a), multiple-tree (b), and hash-based (c) strategies

Single-tree (SA-MHT). A simple approach consists of building an MHT whose leaves are the roots of the A-MHTs. Since an MHT is defined over an ordered sequence of leaves, a total order among the ACL values must be defined. Figure 7(a) illustrates the SA-MHT structure built on the A-MHTs in Figure 6 with an upper (single) MHT combining their roots, assuming a lexicographic ordering of ACLs. Note that the choice of the order among ACLs does not affect the size of the upper structure, since the number of leaves does not change. However, for range and empty queries, the order might have an impact on the size of the VO for the upper-level structure, which is necessary to prove that no relevant A-MHT has been omitted. The size of such a VO depends on how the ACLs in which the requesting user is involved are distributed among the leaves of the upper structure. For instance, consider a query submitted by user W who belongs to the ACLs WX, WXY, WYZ, and WZ. Since the roots of M_{WX}, M_{WXY}, M_{WYZ}, and M_{WZ} are adjacent leaves of the upper structure, the VO of the upper structure for W includes only one node (i.e., $h_{34..}$). The VO of the upper structure for the same query for user Y instead includes two nodes (i.e., h_{12} and h_8) because the roots of the A-MHTs relevant for Y (i.e., M_{WXY}, M_{WYZ}, M_{XY}) are not adjacent. This observation suggests that ACLs should be ordered so that those involving users who submit queries more frequently appear as contiguous leaves of the upper MHT. This would reduce the size of the VO of the upper structure for such users.

Multiple-tree (MA-MHT). Since the integrity verification of the result of a query submitted by a user u must only involve the A-MHTs M_U such that $u \in U$, we build one upper MHT for each user, whose leaves are the roots of these relevant A-MHTs. Figure 7(b) illustrates the MA-MHTs built on the A-MHTs in Figure 6 with an upper MHT for each user u. As an example, the upper MHT for user W has, as leaf nodes, the roots of M_{WX}, M_{WXY}, M_{WYZ}, and M_{WZ}. Like for the SA-MHT strategy, the leaves of each upper MHT must follow a predefined order (again we use the lexicographical order in our examples) to enable proper reconstruction of the root of the upper structure. However, unlike the SA-MHT strategy, such ordering does not affect the size of the VO. For range and empty queries, user u already reconstructs the roots of all relevant A-MHTs, which correspond to all the leaves of the upper structure (i.e., the upper VO is empty). For point queries, u reconstructs the root of one A-MHT, which is a leaf of the upper structure. In this case, the VO of the upper-level structure enables the reconstruction of its root.

Hash-based (HA-MHT). Instead of organizing the roots of the A-MHTs relevant for a user into an upper MHT, we combine these roots into a single hash value signed by the data owner. In other words, the upper structure has a node for each user u, having as children the roots of the A-MHTs M_U with $u \in U$. Figure 7(c) illustrates the HA-MHT combining the A-MHTs in Figure 6. This structure is much smaller than the SA-MHT and MA-MHT structures. As an example, the upper MHT for user W has a single node with the roots of M_{WX}, M_{WXY}, M_{WYZ}, and M_{WZ} as children. During query verification, the user applies the hash function h to the roots of the relevant A-MHTs and verifies that it matches the value signed by the data owner.

6 Comparison among authenticated data structures

The C-MHT and A-MHT approaches previously discussed have different advantages and disadvantages. We compare them according to three main criteria: *1)* the overall size of the authenticated data structure, *2)* the size of the VOs, and *3)* the confidentiality guarantees they provide. Since none of the approaches outperforms the others according to all the criteria, the choice depends on the trade-off that best satisfies the data owner's and users' requirements.

Structure size. A-MHTs reduce node duplication compared with C-MHTs, since each tuple belongs to exactly one ACL and is therefore represented only once. As a consequence, their size is only marginally affected by the number of authorizations (see Section 7). Among the three alternatives SA-MHT, MA-MHT, and HA-MHT, the HA-MHT is the smallest authenticated data structure, since the upper-level consists of a single node for each user. SA-MHT introduces only a limited overhead, and the MA-MHT is larger because maintaining a different upper-level MHT for each user significantly reduces factorization.

VO size. The size of the VO depends not only on the authenticated data structure used, but also on the query evaluated (i.e., point or range), and on the size of the query result (i.e., an empty result or a result including one or more tuples). With the C-MHT approach, the integrity of a query result is verified against a single MHT, and therefore the VO is built from that MHT only. With the A-MHT approach, the query result must instead be verified against all the relevant A-MHTs (as discussed in the previous section). Depending on the adopted upper-level strategy (i.e., SA-MHT, MA-MHT, HA-MHT), the VO may also include the information from the upper authenticated structure. This implies that with the C-MHT approach, the VOs are generally smaller than the VOs generated with the A-MHT approach (see Section 7 for an experimental evaluation of these differences). However, different authorization policies, instances of the relation, and types of queries can be characterized by slightly different verification costs for both the C-MHT and the A-MHT approaches.

Confidentiality. Both C-MHT and A-MHT protect the confidentiality of the outsourced relation by ensuring that every authenticated structure visible to a user u contains only tuples that u is authorized to access. For instance, with respect to our running example, user `W` would be able to infer the existence of tuples t_3 and t_4 and that their `Salary` is a value between 2000 and 5000, when the integrity verification is performed over the MHT built over the whole relation (Figure 1(b)). User `W` would instead not even suspect the existence of such tuples when using the C-MHT or A-MHT approaches. Indeed, none of the structures built with these approaches include t_3 and t_4 as leaves of a tree that `W` can visit. The adoption of A-MHTs could, however, leak information about the authorization policy. For instance, consider the SA-MHT structure in Figure 7(a). Since each leaf of the upper-level MHT corresponds to the root of an A-MHT, each user can observe the number of ACLs in the system. A user u can then infer that there are ACLs to which u does not belong, and consequently the existence of

tuples that u is not authorized to access (whose content is however protected). For instance, user W can infer the existence of an ACL that does not include W, thus inferring the existence of tuples that W cannot access. The MA-MHT and HA-MHT provide stronger protection of the authorization policy. Since there is an upper structure for each user, these approaches do not reveal the existence of ACLs that do not include a specific user. They still disclose, however, that the user belongs to multiple ACLs, indirectly revealing the existence of other users with different privileges. For instance, the structures in Figures 7(b)-(c) reveal to user W that W belongs to four different ACLs, implying the existence of at least two additional users with privileges different from those of W. Indeed, two additional users are enough to generate four distinct ACLs.

7 Experimental results

We now report the results of our experimental evaluation, whose goal is to validate the qualitative comparison presented in Section 6. We implemented the C-MHT and A-MHT strategies in C, using a Python orchestrator for generating the input data, running batches of experiments, and aggregating the results. The experiments have been performed on a machine with Linux Ubuntu 24.04 LTS equipped with an Intel Core i5-5300U processor (4 cores). To analyze the efficiency of the proposed solutions, we have evaluated both the initialization phase (i.e., the construction of the authenticated data structures) and query verification. We have tested several configurations by varying the number of tuples in the relation, the number of users, and the authorization policy (e.g., random policies, fixed number of ACLs, and ACLs of fixed size). In the following, we report the results obtained for a representative configuration that consists of a relation with 1024 tuples, 64 users, and the authorization policy that permits to better observe the peculiarities of the C-MHT and A-MHT strategies. In this configuration, each user can access one or more randomly selected subsets of tuples with contiguous values for the attribute used to build the MHT. In our experiments, we considered the MHT built over the whole relation without consideration of the authorization policy as our baseline (black line in the figures).

Size of the authenticated data structure. Figure 8 compares the median size, measured as the number of nodes, of the C-MHT and A-MHT structures while varying the number of authorizations in the policy (i.e., the number of entries equal to 1 in the access matrix). For each tested configuration, the figure also reports the variance computed over 1000 runs. Note that varying the number of users and/or of tuples produces similar results, since the size of the authenticated data structure mainly depends on the ratio between granted authorizations and the maximum possible number of authorizations. As visible from the figure, C-MHTs are considerably larger than A-MHTs, independently from the strategy for creating the upper structure, except for scenarios characterized by a very limited number of authorizations. Furthermore, the size of C-MHTs quickly grows as the number of authorizations increases. The size of A-MHTs is less affected by the number of authorizations and mainly depends on the strategy adopted to build

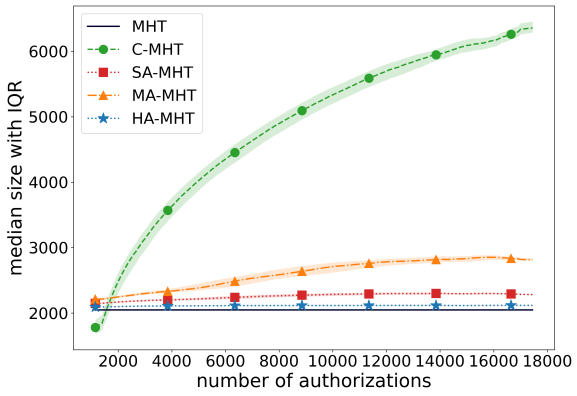


Fig. 8. Median size of the authenticated structure, varying the number of authorizations

the upper authenticated structure. As expected, the hash-based strategy minimizes the size of the authenticated structure, which remains close to the baseline and is not influenced by the number of authorizations. Between the remaining two strategies, the single-tree implies a lower overhead (up to 10% with respect to the baseline) than the multiple-tree strategy (up to 50% with respect to the baseline), although both structures grow with the number of authorizations.

Size of the VO. While the size of the authenticated data structure has an impact on the required storage space, the size of the VO reflects the overhead caused by integrity verification on the query execution. Indeed, a larger VO implies higher communication overhead between the service provider and the client, and a higher computational overhead for the verification of completeness of the query result. Since the size of the VO depends on the kind of authenticated data structure, on its size, as well as on the query result, we consider the worst case scenario analyzing the size of the largest VO (i.e., the longest path in the authenticated structure) in two different scenarios: when the (point) query result is empty and when it includes the target tuple. We do not specifically analyze the size of the VO in case of range queries as it is smaller or, in the worst case, of the same size as the VO returned in case of empty query result. As visible from Figure 9, both when the query is successful and when it returns an empty result, C-MHTs are characterized by VOs having a size comparable (or even smaller, for small configurations) to the baseline, and almost independent from the number of authorizations. The size of the VO remains almost independent from the number of authorizations and slightly (∼70%) higher than the baseline for SA-MHT and MA-MHT when the query result is not empty, while it grows with the number of authorizations for HA-MHT. When the query result is empty, the size of the VO grows considerably with the number of authorizations when using A-MHTs, independently from the strategy used for combining roots, with SA-MHT presenting a slightly larger VO. The size of the VOs necessary to verify

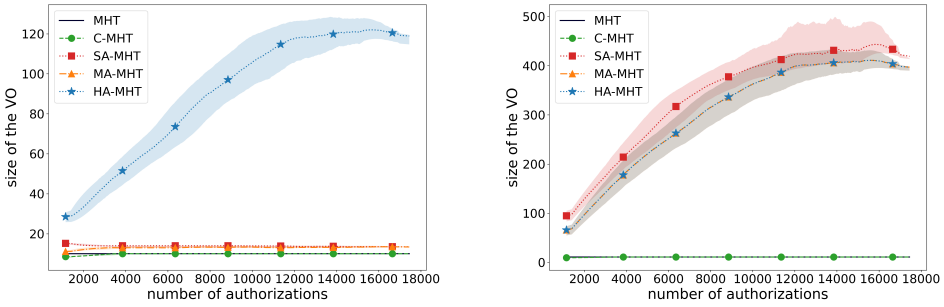


Fig. 9. Size of the VO for point queries when the target is found (a) and when it is not found (b), varying the number of authorizations

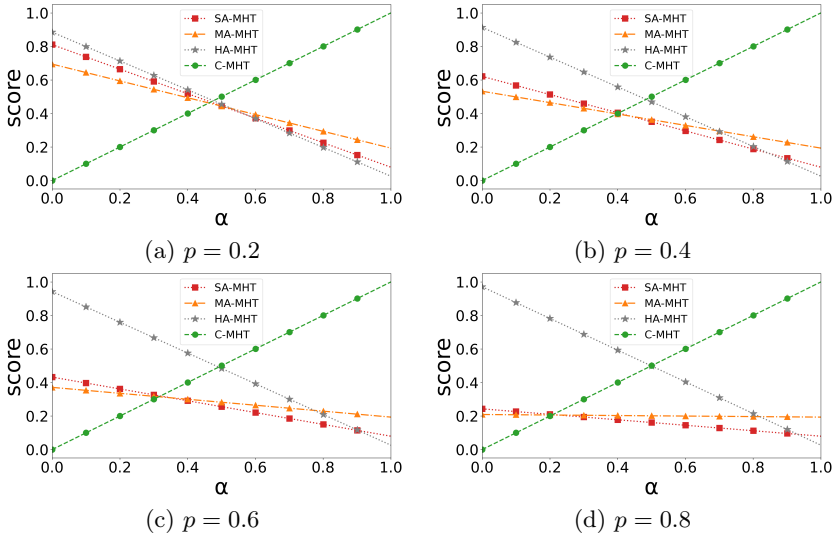
the correctness of empty query results is larger, for all the proposed strategies, with respect to the verification of non-empty results. This is due to the need to prove, for empty results, that all the A-MHTs do not include tuples satisfying the query condition. On the contrary, only the A-MHT containing the resulting tuple contributes a non-empty VO, since at most one tuple can satisfy a point query. C-MHTs do not suffer from this drawback, as the VO for a query formulated by a user operates on one MHT only (the one of the user). Note that the growth in the size of the VOs for A-MHTs slows down when the number of authorizations becomes very large. This might be due to the fact that the number of ACLs in the system decreases when the number of authorizations grows considerably, hence maintaining the size of the VO more stable.

Balancing storage and query verification overhead. The previous experiments show that no authenticated data structure is better than the others in terms of both storage overhead (i.e., the size of the authenticated data structure) and query verification overhead (i.e., the size of the VO).

We therefore analyze this trade-off by defining a scoring function that combines these two metrics, weighted according to the relative importance of storage with respect to query verification and the probability that a query returns one or more tuples. We use the following *weighted node metric* to combine the storage and verification overhead into a single score value:

$$weighted\_nodes = \alpha \cdot s + (1 - \alpha) \cdot (p \cdot VO_f + (1 - p) \cdot VO_e)$$

with $\alpha \in [0, 1]$ the relevance of storage with respect to query verification, p the probability of the result of a query to be not empty (and hence $1 - p$ the probability of the query result to be empty), s the size (number of nodes) of the authenticated data structure, VO_f and VO_e the size (number of nodes) of the largest VO when the query result includes at least one tuple and when it is empty, respectively. Figure 10 illustrates, varying α from 0 to 1, the weighted nodes metric for the proposed authenticated data structures assuming $p = 0.2$,



(a) $p = 0.2$ (b) $p = 0.4$

(c) $p = 0.6$ (d) $p = 0.8$

Fig. 10. Weighted node metric for different values of α and p

$p = 0.4$, $p = 0.6$, and $p = 0.8$. As expected, C-MHTs are preferred when α is small (below 0.32 with $p = 0.6$), with the size of the VO being more important than the size of the structure (i.e., the verification of each query result is valued more than the storage of the structure). When α grows, and hence storage becomes more relevant, A-MHTs are to be preferred over C-MHTs. In particular, for values of α close to 1 (higher than 0.91 with $p = 0.6$), HA-MHTs represent the best choice. For intermediate values of α, representing configurations where both storage and query verification are important, SA-MHTs provide the best trade-off. Note that, as p increases, the value of α below which C-MHTs are preferred decreases (from 0.50 with $p = 0.2$ to 0.20 with $p = 0.8$).

In conclusion, C-MHT is preferable to limit verification costs and protect the authorization policy, while HA-MHT is preferable to limit storage costs. Both SA-MHT and MA-MHT nicely balance query verification and storage costs, with SA-MHT guaranteeing a lower cost and MA-MHT offering better protection to the authorization policy.

8 Related work

Our work is related to two research areas: techniques for guaranteeing integrity of query results and access control enforcement over outsourced data.

Approaches for providing integrity guarantees (i.e., correctness and completeness) for query results computed by an external party have mainly followed two directions: deterministic and probabilistic techniques. Deterministic solutions are based on authenticated data structures (e.g., signature chains, authenticated skip lists, and MHTs), which allow users to verify the correctness and completeness of

query results through verification objects (e.g., [5, 6, 8, 9, 12, 13, 20]). Probabilistic approaches are instead based on the injection of sentinels (i.e., fake tuples not recognizable as such) and/or twins (i.e., replicated tuples), whose absence signals incomplete results (e.g., [3, 10, 14, 16]). The problem of providing integrity guarantees to computation results has been studied considering, besides traditional relational data, also other kinds of data collections (e.g., vector databases [11]). Both deterministic and probabilistic approaches do not consider authorizations.

A second research direction addresses access control enforcement in outsourcing scenarios. Existing approaches are mainly based on selective encryption or Attribute-Based Encryption (ABE). Selective encryption consists of encrypting each tuple with a key known only to users authorized to access it, and distributing keys to users according to their privileges (e.g., [1]). ABE associates users with sets of attributes and encrypts each tuple with a condition, defined over users' attributes, allowing only users satisfying the condition to decrypt it (e.g., [7, 15, 21]). The risks arising when combining selective encryption and indexing techniques to support fine-grained access have been analyzed in [2], proving that indexes, similarly to authenticated structures, might reveal information to unauthorized users. Existing approaches, however, do not provide integrity guarantees for query results.

Only a few proposals have investigated the combination of authenticated query processing and access control [17–19]. The solutions in [17, 18] propose a new authenticated data structure based on access-policy-preserving (APP) signatures. APP signatures are aggregated into a grid-index data structure, used for building VOs. The work in [19] proposes a different authenticated query processing framework for the digital governance scenario. Our approach differs from these solutions as we rely on traditional MHTs for providing integrity guarantees, while preventing information leakage caused by access restrictions.

9 Conclusions

We addressed the problem of providing integrity guarantees while enforcing access control restrictions over data stored and managed by external providers. Since the combined use of authenticated data structures for query integrity and access control enforcement may disclose information to users not authorized to access it, we proposed and compared different authenticated data structures based on capability lists and access control lists. We qualitatively compared the proposed solutions in terms of size of the authenticated structures, size of the VOs, and confidentiality guarantees they provide. The comparison shows that capability-based MHTs provide the strongest confidentiality guarantees, while ACL-based MHTs offer different trade-offs between storage overhead, query verification cost, and protection of the authorization policy. An experimental evaluation confirms the qualitative analysis and shows that ACL-based MHTs are typically smaller, while capability-based MHTs generally require smaller VOs. The paper leaves room for future work, including the consideration of updates to data values in the tuples and to the access control policy.

Acknowledgments. This work was supported in part by project SERICS (PE00000014) under the MUR NRRP funded by the EU - NGEU. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor Italian MUR can be held responsible for them.

References

1. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM Transactions on Database Systems **35**(2), 1–46 (April 2010)
2. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Private data indexes for selective access to outsourced data. In: Proc. of the 10th Workshop on Privacy in the Electronic Society (WPES 2011). Chicago, IL, USA (October 2011)
3. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Sassi, R., Samarati, P.: Sentinels and twins: Effective integrity assessment for distributed computation. IEEE Transactions on Parallel and Distributed Systems **34**(1), 108–122 (January 2023)
4. De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P.: Practical techniques building on encryption for protecting and managing data in the cloud. In: Ryan, P., Naccache, D., Quisquater, J.J. (eds.) The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday. Springer (2016)
5. Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.G.: Authentic third-party data publication. In: Proc. of the 14th Working Conference on Data and Application Security (DBSec 2000). Schoorl, The Netherlands (August 2000)
6. Di Battista, G., Palazzi, B.: Authenticated relational tables and authenticated skip lists. In: Proc. of the 21st Annual Conference on Data and Applications Security and Privacy (DBSec 2007). Redondo Beach, CA, USA (July 2007)
7. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proc. of the 13th ACM Conference on Computer and Communications Security (CCS 2006). Alexandria, VA, USA (October-November 2006)
8. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proc. of the ACM International Conference on Management of Data (SIGMOD 2006). Chicago, IL, USA (June 2006)
9. Li, M., Gao, J., Zhang, Z., Conti, M., Alazab, M.: Secure, available, verifiable, and efficient range query processing on outsourced datasets. In: Proc. of the IEEE International Conference on Communications (ICC 2024). Denver, CO, USA (June 2024)
10. Liu, R., Wang, H.: Integrity verification of outsourced XML databases. In: Proc. of the International Conference on Computational Science and Engineering (CSE 2009). Vancouver, Canada (August 2009)
11. Pan, J., Wang, Z.: VEkNN: Verifiable and encrypted kNN search over high-dimensional vectors. In: Proc. of the 9th International Joint Conference on Web and Big Data (APWeb-WAIM 2025). Shenyang, China (August 2025)
12. Pang, H., Jain, A., Ramamritham, K., Tan, K.L.: Verifying completeness of relational query results in data publishing. In: Proc. of the ACM International Conference on Management of Data (SIGMOD 2005). Baltimore, MD, USA (June 2005)

13. Pang, H., Tan, K.L.: Authenticating query results in edge computing. In: Proc. of the 20th IEEE International Conference on Data Engineering (ICDE 2004). Boston, MA, USA (April 2004)
14. Wang, H., Yin, J., Perng, C.S., Yu, P.S.: Dual encryption for query integrity assurance. In: Proc. of the 17th ACM Conference on Information and Knowledge Management (CIKM 2008). Napa Valley, CA, USA (October 2008)
15. Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In: Proc. of the 14th International Conference on Practice and Theory in Public Key Cryptography (PKC 2011). Taormina, Italy (March 2011)
16. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: Proc. of the 33rd International Conference on Very Large Data Bases (VLDB 2007). Vienna, Austria (September 2007)
17. Xu, C., Chen, Q., Hu, H., Xu, J., Hei, X.: Authenticating aggregate queries over set-valued data with confidentiality. IEEE Transactions on Knowledge and Data Engineering **30**(4), 630–644 (April 2018)
18. Xu, C., Xu, J., Hu, H., Au, M.H.: When query authentication meets fine-grained access control: A zero-knowledge approach. In: Proc. of the ACM International Conference on Management of Data (SIGMOD 2018). Houston, TX, USA (June 2018)
19. Xue, J., Liu, K., Li, F., Zhang, W., Zhang, X.: GovLink: Auditable FPH-CP-ABE for dynamic revocation and query integrity in e-government. IEEE Internet of Things Journal **13**(2), 3331–3343 (January 2026)
20. Yang, Y., Papadias, D., Papadopoulos, S., Kalnis, P.: Authenticated join processing in outsourced databases. In: Proc. of the ACM International Conference on Management of Data (SIGMOD 2009). Providence, RI, USA (June-July 2009)
21. Zhao, F., Nishide, T., Sakurai, K.: Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems. In: Proc. of the International Conference on Information Security Practice and Experience (ISPEC 2011). Guangzhou, China (May-June 2011)