Certification and Authentication of Data Structures

Roberto Tamassia* Nikos Triandopoulos†

Abstract

We study *query authentication schemes*, algorithmic and cryptographic constructions that provide efficient and secure protocols for verifying the results of queries over structured data in untrusted or adversarial data distribution environments. We formally define the problem in a new data query and authentication setting that involves general query types answered in the RAM model of computation, and put forward a new approach for designing secure query authentication schemes that, through the new concept of *query certification*, aims to authenticate the *validity of the answer*, rather than the entire process that generates the answer. Our main results state that this new authentication framework achieves *generality*, namely any query type admits a secure query authentication scheme, and also supports an important type of *modularity*, namely the authentication of general queries based on the evaluation of relations over the data elements is reduced to the authentication of set-membership queries. Thus, in addition to general possibility results under general assumptions and characterization results using existing cryptographic techniques, we contribute a clear separation between algorithmics and cryptography in data-authentication protocol design, and sufficient conditions for achieving super-efficient answer verification in time asymptotically less than the time needed to answer the query.

1 Introduction

Data authentication is a fundamental problem in data management, where we wish to design secure and efficient protocols that prove the authenticity of computations in untrusted or adversarial data distribution environments. The problem is of both practical and theoretical importance. More and more in distributed, pervasive or Internet computing, information is delivered through untrusted computing entities, raising crucial security threats with respect to data authenticity. From a theoretical point of view, data authentication introduces new dimensions both in the design of algorithms and in cryptography. On one hand, known data management and data structuring techniques often need to be reexamined, in new data dissemination settings, where the data distributor and the data owner are different entities. On the other hand, directly applying traditional and well-studied message authentication techniques for data authentication—where data cannot be treated as a whole—is often inadequate to provide efficient solutions.

Most of the existing work in the design of authentication protocols has focused on authentication schemes for verifying the results of specific query types, using explicit constructions that combine data structuring techniques with related cryptographic primitives (mostly signatures and hashing). Being problem-specific, the merging of algorithmics and cryptography usually leads to more complex and less modular authentication schemes, with usually more elaborate associated security proofs. Moreover, although general authentication methodologies for certain classes of queries do exist, they authenticate the query results by essentially verifying step-by-step the computation that generates the answer (e.g., the entire search process in the data structure), and they provide solutions only for the static case, where no updates are allowed in the data set. Therefore, the corresponding protocols are usually less practical and more costly than needed.

In this paper we study data authentication over structured data, where data is disseminated by issuing queries, from a theoretical, protocol-design point of view. Our goal is to provide a general authentication

*Brown University, Computer Science Department, Providence, RI 02912, USA, rt@cs.brown.edu

†University of Aarhus, Computer Science Department, Aarhus N, DK-8200, Denmark, nikos@daimi.au.dk

framework that advances the design of authentication protocols and offers useful techniques for systematically building secure schemes. We depart from previous approaches by proposing the decoupling of the answer-verification process and the answer-generation process for any query types in a general query model over dynamic data, and also separating the algorithmic and cryptographic components in data authentication.

Aiming at general results, we use a very general computational model, the RAM model, and a very general data type and query model, where data is organized according to the relational data model, slightly modified to fit the RAM model. We provide a formal definition for the problem of authenticating answers to queries through a *query authentication scheme*, in a setting where the (honest) data owner and the (malicious) query responder are distinct entities, and where end-users do not trust (the authenticity of) the answers to their queries. Central idea in our work is the following: in contrast to approaches that authenticate the algorithm that answers a query, we propose an answer-based approach where only the information that is sufficient (or necessary) for the answer verification is being authenticated. To achieve this, we introduce the concept of *query certification*, which models answer verification in the information theoretic model. In particular, a certification data structure for a query type defines the type of information and corresponding algorithms that are sufficient to verify the correctness of the answer to any concrete query. We identify the inherent relationship between query authentication and query certification, [1] and put forward a new approach for data authentication: we show that for any query type we can build an authenticated data structure that provides authenticated queries in the bounded computational model, by first designing a certification data structure for the same query type and then applying simple cryptographic constructions to its functionality.

Moreover, this transformation of any certification data structure to an authenticated data structure, satisfies, by construction, an important property: to authenticate an answer to a general query, the authenticated data structure can only use a query authentication scheme for set-membership queries, that is, protocols that verify membership in sets (actually, only positive answers to these queries). To show this we introduce the concept of (query) problem reducibility in data authentication. Informally, we say that query of type A is authenticated reduced to query of type B, when a query authentication scheme for B leads to a query authentication scheme for A. We thus show an important reduction: any query problem in our query model is authenticated reduced to the fundamental set membership problem[2]. Although for unstructured data (i.e., computations on memory cells) this reduction is implied by the results on memory correctness by Blum *et al.* [2], our reduction is the first known for structured data, and has, as we show, some important consequences, given that, at present, concrete cryptographic constructions exist only for set-membership query authentication schemes (e.g., Merkle tree [23], its distributed extension [31], and one-way accumulators [6]).

Finally, by showing that for any query type there exists an efficient certification data structure, our authentication framework contributes not only general possibility results in the design of efficient authenticated data structures, but also a useful design tool for *super-efficient* data authentication, where verifying the answer is asymptotically faster than answering the query. Indeed, although the above completeness result is proven by verifying the query-answering algorithm, thus extending previous possibility results to *general queries over dynamic data*, we demonstrate that super-efficient certification structures exist for certain query types. This way, we can exploit the computational gap that is often observed between answering a query and verifying its answer (see, e.g., [14, 20]). Accordingly, using our framework super-efficient verification can be achieved by designing new super-efficient certification structures, or by constructing new cryptographic primitives for optimal (constant-time) set-membership verification, or by improving on both directions.

1.1 Related Work

Authenticated Data Structures. Extensive work exists on authenticated data structures [26], which model the security problem of data querying in untrusted or adversarial environments. This model augments a

[1] This is inspired by *certifying algorithms* [20], which study program-correctness checking in erroneous implementations.

[2] This is a non-trivial reduction, meaning that efficiency is preserved in our reduction. A trivial reduction authenticates the answer to a query by authenticating all possible query-answer pairs, which for most query problems is a set of infinite cardinality.

data structure such that along with an answer to a query, a cryptographic proof is provided that can be used to verify the answer authenticity. Research initially focused on authenticating membership queries (mostly in the context of the certificate revocation problem), where various authenticated dictionaries based on extensions of the *hash tree* introduced by Merkle [23] have been studied [1, 5, 16, 26, 30]. In [6, 13] it is showed how the use dynamic accumulators can realize a dynamic authenticated dictionary and in [14, 28] schemes that use an interesting combination of hashing with accumulators are presented that improve the efficiency of one-dimensional authenticated range searching. More general queries, beyond membership queries, have been studied as well, where extension of hash trees are used to authenticate various queries, including: basic operations (e.g., select, join) on databases [9, 25], pattern matching in tries and orthogonal range searching [19], path queries and connectivity queries on graphs and queries on geometric objects (e.g., point location queries and segment intersection queries) [15] and queries on XML documents [8]. Our work provides a useful framework for the design of new efficient authenticated data structures of any query type.

General Authentication Techniques. There has been also substantial progress in the design of generic authentication techniques, that is, development of general authentication frameworks that can be used for the design of authenticated data structures for authenticating concrete queries, or design of general authentication patterns that authenticate classes of queries. Work of this type is as follows. In [19] it is described how by hashing over the search structure of data structures in a specific class a broad class we can get authenticated versions of these data structures. The class of data structures is such that (i) the links of the structure form a directed acyclic graph G of bounded degree and with a single source node; and (ii) queries on the data structure correspond to a traversal of a subdigraph of G starting at the source. The results hold for the pointer machine model of computation, where essentially the entire search algorithm is authenticated. This way, an answer carries a proof that is proportional to the search time spent for generating the answer itself, and the answer verification has analogous time complexity. The method only handles static problems. In [15], it is shown how extensions of hash trees can be used to authenticate abstract properties of data that is organized as paths, where the properties are decomposable, i.e., the properties of two subpaths can be combined to give the property of the resulting path. Also the authentication of the general fractional cascading data-structuring technique [7] is presented. This technique can lead to authentication of data structures that involve iterative searches over catalogs. The underlying model is same as before, i.e., the pointer machine model. Although, the techniques do not explicitly authenticate the corresponding search algorithm, the complexity of the resulted authenticated data structures is of the same order of magnitude as the searching algorithm. Finally, in [29] a general technique is described for designing consistency proofs for static committed databases—a different problem than data authentication. However, the technique can be extended to provide a general framework for designing also authenticated data structures in the static case (that actually enjoy additional properties). The authentication technique is similar to the one in [19]: the searching algorithm that is used to produce the answer is authenticated. Also, the used model is the pointer machine; the RAM model can be captured at a $O(\log M)$ overhead, where M is the total memory used. Our results operate on the RAM model, thus, they include a broader class of both static and dynamic query problems and can lead to more efficient constructions, where the answer validity and not the algorithm is verified. Finally, in [30] it is shown that for the dictionary problem and hash-based data authentication, the querying problem and the authentication problem are equivalent. That is, for authenticated dictionaries of size n, all costs related to authentication are at least logarithmic in n in the worst case.

Consistency Proofs and Privacy. Recently, the study of an additional security property related to authenticated data structures has been initiated. Assuming a more adversarial for the user setting, one can consider the case where the data source can act unreliably. The new requirement is then data consistency, namely, the incapability of the data source to provide different, i.e., contradictory, verifiable answers to the same query. Buldas *et al.* [5] study this issue for hash trees and show how to enforce data consistency by augmenting hash trees. In [24] zero-knowledge sets are introduced, where a prover commits to a value for a set and

membership queries can be verified by a verifier consistently (and in zero-knowledge). In [29] consistency proofs are extended to range queries and where also sufficient conditions are given for schemes to achieve consistency. The works in [24, 29] provide privacy-preserving verification.

Certifying Algorithms and Checking Primitives. Extensive work on certifying algorithms [4, 10, 21, 22] model a computational gap between the computation of a program and the verification of this correctness. This is related with the idea behind our authentication framework. Our methodology to decouple the searching algorithm from the answer verification is modeled through a certification data structure, defined in Section 4, which can be viewed as an extension of methodology of the certifying algorithms for data structures. Related also work appears in [2, 3, 12, 27].

In Sections 2 we define our query model and in Section 3 we present our definitional framework for query authentication schemes, defining their security requirements. In Section 4 we introduce the certification data structures which model the core concept in our authentication framework of answer testability, we provide a constructive proof of their existence and discuss the importance and of these structures as expressed by the decoupling of the answer-verification and answer-generation processes. In Section 5 we introduce the reducibility among authentication schemes and prove our main results. Section 6 presents additional applications of our framework. This extended abstract omits several details of our work, which will appear in the full version of the paper. Proofs or sketches of proofs are included in the Appendix.

2 Preliminaries

We first define our relation-based data querying model, which is based on the RAM model of computation.

Definition 1 (Structured Data Set) *A* structured data set *(or, simply, a* data set*)* $S = (\mathcal{E}, \mathcal{R})$ *consists of: (i) a collection* $\mathcal{E} = \{E_1, \ldots, E_t\}$ *of sets of data elements such that, for* $1 \leq i \leq t$*, set* E_i *is a subset of a* universe $\mathcal{U}_i$*, and (ii) a collection* $\mathcal{R} = \{R_1, \ldots, R_k\}$ *of indexed sequences of tuples of data elements such that, for* $1 \leq i \leq k$*, sequence* $R_i = (R_i[1], \ldots, R_i[m_i])$ *consists of* m_i *distinct* p_i*-tuples from* $E_{j_1} \times \ldots \times E_{j_{p_i}}$*, where* $1 \leq j_1 \leq \ldots \leq j_{p_i} \leq t$ *and* $p_i < p$ *for some integers* p *and* m_i*. The* size n *of data set* $S = (\mathcal{E}, \mathcal{R})$ *is defined as* $n = \sum_{i=1}^{t} |E_i|$*. Also, we assume that* t*,* k *and* p *are constants (with respect to* n*).*

Our definition shares concepts from the relational data model for databases (see, e.g., [17]). A relation, mathematically defined as a subset of the Cartesian product of sets, is typically viewed as a set of tuples of elements of these sets. Our model actually uses indexed sequences of tuples, i.e., each member R_i of $\mathcal{R}$ is an array of tuples, where each tuple can be indexed by an integer. Thus, very general data organization and algorithmic paradigms are captured. For instance, a graph $G = (V, E)$ may correspond to data set $S_G = (\mathcal{E}, \mathcal{R})$, where $\mathcal{E} = V$ and $\mathcal{R}$ consists of a single sequence of indexed pairs representing relation E (edges in G). More complex graphs, e.g., with edge directions, weights, costs or associated data elements, can be represented by appropriately including new primitive data-element sets in $\mathcal{E}$ and corresponding sequences in $\mathcal{R}$ describing data elements' structure and various relations among them. (See examples in Appendix B.)

Definition 2 (Querying Model) *Let* $S = (\mathcal{E}, \mathcal{R})$ *be a structured data set. A* query operation Q_S *on* S *is a computable function* $Q_S : \mathcal{Q} \rightarrow \mathcal{A}_S$*, where* $\mathcal{Q}$ *is the query space (the set of all possible queries* q *of specific type that can be issued about* S*) and* $\mathcal{A}_S$ *is the answer space (the set of all possible answers to queries on* S *drawn from* $\mathcal{Q}$*). The* answer *of a query* $q \in \mathcal{Q}$ *under* Q_S *is* $Q_S(q) \in \mathcal{A}_S$*. An element* $a \in \mathcal{A}_S$ *of the answer space is the* correct answer *for query* q *if and only if* $Q_S(q) = a$*.*

Observe that the above definitions capture general query operations[3] on data sets that are based on relations. The only requirement is that any query in the query space is mapped to a unique answer in the

[3] Alternatively but less conveniently, query operation Q_S can be defined independently of the data set S, such that the answer to query q is $Q(S, q)$. In this case, the query and answer spaces are also independent of S.

answer space and that any answer corresponds to some query[4]. For instance, if $S_G = (\mathcal{E}, \mathcal{R})$ represents a monotone subdivision of the plane into the polygons induced by the vertices and edges of a planar graph G, the point location query operation maps a point in the plane (query) to the unique region of the subdivision (answer) containing it. Regarding the complexity of query answering, we only require that query operation Q_S is efficiently computable. Typically, function Q_S is evaluated on query $q \in \mathcal{Q}$ by a query answering algorithm that operates over S through an appropriate for the type of queries in $\mathcal{Q}$ query data structure.

Definition 3 (Query Data Structure) *A* query data structure $D(Q_S)$ *for query operation* $Q_S : \mathcal{Q} \rightarrow \mathcal{A}_S$ *on data set* $S = (\mathcal{E}, \mathcal{R})$ *consists of a structured data set* $(\mathcal{E}_Q, \mathcal{R}_Q)$*, such that* $\mathcal{E} \subset \mathcal{E}_Q$ *and* $\mathcal{R} \subset \mathcal{R}_Q$ *and an algorithm* Answer*, which on input a query* $q \in \mathcal{Q}$ *and data set* $(\mathcal{E}_Q, \mathcal{R}_Q)$ *returns* $Q_S(q) \in \mathcal{A}_S$ *in time polynomial in* n *and* $|q|$ *by accessing and processing tuples in* $\mathcal{R}$*.*[5] *We write* $D(Q_S) = (\mathcal{E}_Q, \mathcal{R}_Q, \mathsf{Answer})$.

On input query q, algorithm Answer operates over S through the use of $D(Q_S)$: by processing relations in $\mathcal{R}_Q$, Answer accesses relations in $\mathcal{R}$, evaluates conditions over elements in S and produces the answer. For instance, for a point location algorithm that is based on segment trees and operates on planar subdivision $S_G = (\mathcal{E}, \mathcal{R})$, data set $(\mathcal{E}_Q, \mathcal{R}_Q)$ represents a two-level search structure locating points in logarithmic time; here, data set S_G includes information about the regions defined by the edges of graph G.

A data set S is *static* if it stays the same over time and *dynamic* if it evolves over time through *update operations* performed on S. An update operation U_S for S is a function that given an update $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of all possible updates, results in changing one or more data elements in $\mathcal{E}$ and accordingly one or more tuples in $\mathcal{R}$. If S is static (resp. dynamic), data set $(\mathcal{E}_Q, \mathcal{R}_Q)$ can be constructed (resp. updated) by some algorithm Constr_Q (resp. Update_Q) that runs on input S (resp. S and $y \in \mathcal{Y}$) in polynomial time in n.

Our data querying model achieves generality by combining the expressiveness of relational databases with the power of the RAM computation model. By using index-annotated relations, complex data organizations are easily represented and accessed. For instance, indirect addressing is supported by treating indexes as a distinct data type which is included in $\mathcal{E}$, thus our model strictly contains the pointer machine model.

The cryptographic primitives that we use are presented in the Appendix A.

3 Authenticated Data Structures

In this section, we formally describe a general model for data authentication in untrusted and adversarial environments by introducing *query authentication schemes*, cryptographic protocols (algorithms that use cryptography to satisfy certain properties) for the authentication of general queries over collections of structured data. Conceptually, query authentication schemes extend certification structures in that answer validation is not performed in a collaborative setting; instead, the prover may be adversarial and answer verification is now achieved in the bounded computational model. In particular, we examine data authentication in a non-conventional setting, where the creator (or owner) of a data set is not the same entity with the one answering queries about the set and, in particular, the data owner does not control the corresponding data structure that is used to answer a query. In this setting, an intermediate, untrusted party answers the queries about the data set that are issued by an end-user. We formally define this model of data querying.

Definition 4 (Three-Party Data Querying Model) *A* three-party data querying model *consists of a* source $\mathcal{S}$*, a* responder $\mathcal{R}$ *and a* user $\mathcal{U}$*, where: (i) source* $\mathcal{S}$ *creates (and owns) a dynamic data set* S*, which is maintained by query data structure* $D(Q_S)$ *for query operation* $Q_S : \mathcal{Q} \rightarrow \mathcal{A}_S$ *on* S*; (ii) responder* $\mathcal{R}$ *stores* S*, by maintaining a copy of* $D(Q_S)$ *and some auxiliary information for* S*; (iii) user* $\mathcal{U}$ *issues queries*

[4]Unique answers are used without loss of generality. Of course, there are query problems for which Q_S is a mapping not a function. That is, more that one answers can exist for a given query. For instance, a path query on a graph, given two vertices asks for any connecting path, if it exists. We can appropriately augment the query space for this type of queries to include the index of the answer (according to some fixed ordering) that we wish to obtain.

[5]By Definition 1, for any data set $(\mathcal{E}, \mathcal{R})$ of size n, the total number of relations that exist in $\mathcal{R}$ (and thus can be possibly accessed by Answer) is $O(n^p) = poly(n)$. This implies that the storage size of data set $(\mathcal{E}, \mathcal{R})$ is polynomially related to its size.

about S to responder $\mathcal{R}$ by sending to $\mathcal{R}$ a query $q \in \mathcal{Q}$; (iv) on a query $q \in \mathcal{Q}$ issued by $\mathcal{U}$, $\mathcal{R}$ computes answer $a = Q_S(q)$ and sends a to $\mathcal{U}$; (v) on an update $y \in \mathcal{Y}$ for S issued by the source, S and $D(Q_S)$ are appropriately updated by $\mathcal{S}$ and $\mathcal{R}$.

The model achieves generality and has many practical applications. Regarding data authentication, we wish that the user can verify the validity of the answer given to him by the responder. For this verification process, we wish that the responder, along with the answer, gives to the user a proof that can be used in the verification. To capture this verification feature, we define the notion of a *query authentication scheme*.

Definition 5 (Query Authentication Scheme) *A* query authentication scheme *for query operation $Q_S : \mathcal{Q} \to \mathcal{A}_S$ on structured data set S is a quadruple of PPT algorithms* $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ *such that:*

Key generation *Algorithm* KeyG *takes as input a security parameter 1^κ, and outputs a key pair (PK, SK). We write $(PK, SK) \leftarrow \mathsf{KeyG}(1^\kappa)$.*

Authenticator *Algorithm* Auth *takes as input the secret and public key (SK, PK), the query space $\mathcal{Q}$ (or an encoding of the query type) and data set S of size n and outputs an authentication string α and a verification structure V, that is $(\alpha, \mathsf{V}) \leftarrow \mathsf{Auth}(SK, PK, \mathcal{Q}, S)$, where $\alpha, \mathsf{V} \in \{0,1\}^*$.*

Responder *Algorithm* Res *takes as input a query $q \in \mathcal{Q}$, a data set S of size n and a verification structure $\mathsf{V} \in \{0,1\}^*$ and outputs an answer-proof pair $(a, p) \leftarrow \mathsf{Res}(q, S, \mathsf{V})$, where $a \in \mathcal{A}_S$ and $p \in \{0,1\}^*$.*

Verifier *Algorithm* Ver *takes as input the public key PK, a query $q \in \mathcal{Q}$, an answer-proof pair $(a, p) \in \mathcal{A}_S \times \{0,1\}^*$ and an authentication string $\alpha \in \{0,1\}^*$ and either accepts the input, returns 1, or rejects, returns 0, that is, we have that $\{0,1\} \leftarrow \mathsf{Ver}(PK, q, (a, p), \alpha)$.*

Updates *For the dynamic case, we additionally require the existence of an update algorithm Auth_U that complements algorithm* Auth *and handles updates; namely, Auth_U given update $y \in \mathcal{Y}$, it updates the authentication string and the verification structure: $(\alpha', \mathsf{V}') \leftarrow \mathsf{Auth}_U(SK, PK, \mathcal{Q}, S, y, \alpha, \mathsf{V})$.*

We now define the first requirement for a query authentication scheme, which is *correctness*. Intuitively, we wish the verification algorithm to accept answer-proof pairs generated by the responder algorithm and these answers always to be correct. We also discuss the *security* requirement of any query authentication scheme. Starting from the basis that in our three-party data querying model, the user $\mathcal{U}$ trusts the data source $\mathcal{S}$ but not the responder $\mathcal{R}$, it is the responder that can act adversarially. We first assume that $\mathcal{R}$ always participates in the three-party protocol, i.e., it communicates with $\mathcal{S}$ and $\mathcal{U}$, as the protocol dictates. Thus, we do not consider denial-of-service attacks; they do not form an authentication attack but rather a data communication threat. However, $\mathcal{R}$ can adversarially try to cheat, by not providing the correct answer to a query and forging a false proof for this answer. Accordingly, the security requirement is that given any query issued by $\mathcal{U}$, no computationally bounded $\mathcal{R}$ can reply with a pair of answer and an associated proof, such that both the answer is not correct and $\mathcal{U}$ verifies the authenticity of the answer and, thus, accepts it. The above requirements are expressed as the following two conditions for query authentication structures.

Definition 6 (Correctness) *A query authentication scheme* $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ *is said to be* correct *if for all queries $q \in \mathcal{Q}$, if $(\alpha, \mathsf{V}) \leftarrow \mathsf{Auth}(SK, PK, \mathcal{Q}, S)$ and additionally $(a, p) \leftarrow \mathsf{Res}(q, S, \mathsf{V})$, then with overwhelming probability it holds that $1 \leftarrow \mathsf{Ver}(PK, q, (a, p), \alpha)$ and $Q_S(q) = a$.*

Definition 7 (Security) *A query authentication scheme* $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ *for query operation $Q_S : \mathcal{Q} \to \mathcal{A}_S$ on structured data set S is said to be* secure*, if no probabilistic polynomial-time adversary $\mathcal{A}$, given any query $q \in \mathcal{Q}$, public key PK and oracle access to the authenticator algorithm* Auth*, can output an authentication string α, an answer a' and a proof p', such that a' is an incorrect answer that passes the verification test, that is, $a' \neq Q_S(q)$ and $1 \leftarrow \mathsf{Ver}(PK, q, (a', p'), \alpha)$. (See formal definition in Appendix C.)*

Definition 8 (Authenticated Data Structure) *An* authenticated data structure *for queries in query space $\mathcal{Q}$ on a data set S is a* correct *and* secure *query authentication scheme* $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$*, or, as it is implied, a scheme where, given an authentication string α, for algorithm* Ver *it holds that, for all queries*

$q \in \mathcal{Q}$, with all but negligible probability (measured over the probability space of the responder algorithm): $Q_S(q) = a$ if and only if there exists p s.t. $1 \leftarrow \mathsf{Ver}(PK, q, (a, p), \alpha)$.

4 Certification Data Structures

In this section, we explore the decoupling of query answering and answer verification. We start by defining the notion of *answer testability*, formally expressed through a *certification data structure*. Intuitively, this notion captures the following important property in data querying: query operations on any data set return validated answers that can be tested to be correct given a (minimal) subset of specially selected relations over elements of the data set. In essence and in an information-theoretic sense, queries are certified to return valid answers; actually this holds in a safe way (i.e., cheating is effectively disallowed).

Definition 9 (Certification Data Structure) *Let $D(Q_S) = (\mathcal{E}_Q, \mathcal{R}_Q, \mathsf{Answer})$ be a query data structure for query operation $Q_S : \mathcal{Q} \to \mathcal{A}_S$ on data set $S = (\mathcal{E}, \mathcal{R})$ of size n. A* certification data structure *for S with respect to $D(Q_S)$ is a triplet $C(Q_S) = ((\mathcal{E}_C, \mathcal{R}_C), \mathsf{Certify}, \mathsf{Verify})$, where $(\mathcal{E}_C, \mathcal{R}_C)$, called the* certification image *of S, is a structured data set and* Certify *and* Verify *are algorithms such that:*

Answer tests: *On input query $q \in \mathcal{Q}$ and data sets $(\mathcal{E}_Q, \mathcal{R}_Q)$ and $(\mathcal{E}_C, \mathcal{R}_C)$,* Certify *returns answer $a = Q_S(q)$ and an* answer test *τ, which is a sequence of pairs (i, j), each indexing a tuple $R_i[j]$ of $\mathcal{R}_C$. Answer test τ defines a subset $\mathcal{R}_C(\tau) \subseteq \mathcal{R}_C$, called the* certification support *of answer a.*

Answer testability: *On input query $q \in \mathcal{Q}$, data set $(\mathcal{E}_C, \mathcal{R}_C)$, answer $a \in \mathcal{A}_S$ and answer test τ,* Verify *accesses and processes* only *relations in $\mathcal{R}_C(\tau)$ and returns either 0 (rejects) or 1 (accepts).*

Completeness: *For all queries $q \in \mathcal{Q}$, it holds that* $\mathsf{Verify}(q, \mathcal{R}_C, \mathsf{Certify}(q, (\mathcal{E}_Q, \mathcal{R}_Q), (\mathcal{E}_C, \mathcal{R}_C))) = 1$.

Soundness: *For all queries $q \in \mathcal{Q}$, answers a, answer tests τ, when* $\mathsf{Verify}(q, \mathcal{R}_C, a, \tau) = 1$, $a = Q_S(q)$.

Regarding complexity measures for certification data structure $C(Q_S)$, we say: (1) $C(Q_S)$ is answer-efficient *if the time complexity $T_C(n)$ of* Certify *is asymptotically at most the time complexity $T_A(n)$ of* Answer*, i.e., $T_C(n)$ is $O(T_A(n))$; (2) $C(Q_S)$ is* time-efficient *(resp.* time super-efficient*) if the time complexity $T_V(n)$ of* Verify *is asymptotically at most (resp. less than) the time complexity $T_A(n)$ of* Answer*, i.e., $T_V(n)$ is $O(T_A(n))$ (resp. $o(T_A(n))$); and analogously, (3) $C(Q_S)$ is* space-efficient *(resp. space super-efficient) if the space requirement $S_C(n)$ of $(\mathcal{E}_C, \mathcal{R}_C)$ is asymptotically at most (resp. less than) the space requirement $S_Q(n)$ of $(\mathcal{E}_Q, \mathcal{R}_Q)$, i.e., $S_C(n)$ is $O(S_Q(n))$ (resp. $o(S_Q(n))$). If S is static, data set $(\mathcal{E}_C, \mathcal{R}_C)$ can be constructed by some algorithm Constr_C that runs on input S in polynomial time in n.*

For simplicity, the above definition corresponds to the static case. The dynamic case can be treated analogously. Informally, an update algorithm Update_C is responsible to handle updates in data set S by accordingly updating $C(Q_S)$; that is, it produces the updated set $(\mathcal{E}'_C, \mathcal{R}'_C)$ and, in particular, the set of tuples where $\mathcal{R}'_C$ and $\mathcal{R}_C$ differ at. Algorithm Update_C additionally produces an *update test* (as the answer test above, a set of indices for tuples in $\mathcal{R}_C$) that validates the performed changes. Similarly, an update testing algorithm Updtest, on input an update $y \in \mathcal{Y}$, set $\mathcal{R}_C$, a set of tuples (changes in $\mathcal{R}_C$) and an update test, accepts if and only if the tuples correspond to the correct, according to y, new or deleted tuples in $\mathcal{R}_C$. Similarly, we can define *update efficiency* and *update-testing (super-)efficiency* for $C(Q_S)$, with respect to the time complexity of Update_C and Updtest respectively, as they asymptotically compare to Update_Q.

Certification data structures introduce a general framework for studying data querying with respect to the answer validation and correctness verification. They support certification of queries in a computational setting where the notions of query answering and answer validation are conceptually and algorithmically separated in a clean way. In particular, answer validation is based *merely* on the certification image $(\mathcal{E}_C, \mathcal{R}_C)$ of data set $S = (\mathcal{E}, \mathcal{R})$; the two data sets are related by sharing tuples, possibly, through a subset relation. Also, query certification depends *only* on the certification support of the answer, i.e., subset $\mathcal{R}(\tau)$.

Our first result shows that for every query structure there is an efficient certification structure, that is, a completeness result showing that all queries can be certified without loss of efficiency.

Theorem 1 *Any query data structure for any query operation on any structured data set admits an answer-, time-, update-, update-testing- and space-efficient certification data structure. (Proof in Appendix D.)*

Discussion. Certification data structures are designed to accompany two-party data query protocols in the straightforward way: party A possesses data sets $(\mathcal{E}_Q, \mathcal{R}_Q)$ and $(\mathcal{E}_C, \mathcal{R}_C)$ and runs algorithm Certify and party B possesses data set $(\mathcal{E}_C, \mathcal{R}_C)$ and runs algorithm Verify. The underlying dynamic set S is controlled by B by creating update and query operations for S. Although both operations are performed at A, B is able to verify their correctness. Thus, this setting models *certified outsourced computation*: at any point in time, B maintains a correct certification image of outsourced set S allowing verification *without* loss of efficiency, by Theorem 1. And although its existential proof is trivial (both parties execute the same algorithms on the same data) yet, its significance is justified by that: (1) in addition to showing that Definition 9 is meaningful, Theorem 1 proves the feasibility of answer testability for any computable query in a general querying and computational model; (2) time super-efficient certification is in general feasible (see Appendix B for some specific examples) so outsourced computations are important, and (3) in the bounded-computational model and using cryptography, certification data structures have important applications to popular and practical models of data querying, namely, *authentication* and *consistency* in third party models and space super efficiency of certified outsourced computations in the client-server model (Sections 5 and 6).

Relation to Certifying Algorithms. Our certification data structures are related to and inspired by certified algorithms (see, e.g., [18, 20]). Both model the property of answer testability (of a program or an algorithm for a data structure) as distinct from algorithm execution. The main difference, though, is that here we model the intrinsic property of a data structure to provide proof of correctness for verification purposes. Certifying algorithms are designed to guard against an erroneous implementation of an algorithm. Definition 9 can be viewed as an extension of the theory of certifying algorithms to data structures: certifying algorithms for data structures use the implementation of a data structure as a black box and add a wrapper program to catch errors; instead, here, the data structure is augmented to facilitate query certification.

5 Authentication Reductions and General Authentication Results

We are now ready to use the definitional framework of the previous sections and describe and prove the main results of our work. The road map is as follows. First we introduce the notion of reducibility in data authentication, namely by defining reductions between query authentication schemes. We then prove, using our framework of certification data structures, that the authentication of any query in our model is reduced to the authentication of *set membership* queries. In fact, we need to authenticate only positive answers—that is, relation $\in$ and not $\notin$ needs to be authenticated. We then present implications of this result, in terms of concrete constructions. Using certification structures, we provide a general methodology for constructing correct and secure query authentication schemes and we show that any search structure for any query type in our querying model can be transformed into an authenticated data structure. Also, based on super-efficient query certification, we develop a new approach for data authentication, where only the information necessary for the answer verification is authenticated, and not the entire information used by search algorithm, which leads to a powerful framework for the design of authentication structures with super-efficient verification.

Let $QAS(Q_S, S)$ denote a query authentication scheme (or QAS) for query operation Q_S and data set S. Intuitively, authenticated reductions among QASs allow the design of a QAS using no other cryptographic tools but what another QAS provides and in a way that preserves correctness and security.

Definition 10 (Reductions of Query Authentication Schemes) *Let S and S' be data sets, $Q_S : \mathcal{Q} \to \mathcal{A}_S$, $Q'_S : \mathcal{Q}' \to \mathcal{A}'_S$ be query operations on S and S' respectively, and $QAS(Q_S, S) = (\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$, $QAS(Q'_S, S') = (\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$ be query authentication schemes for Q_S on S and Q'_S on S' respectively. We say that $QAS(Q_S, S)$ is* authenticated reduced *to $QAS(Q'_S, S')$, if key generation algorithms* KeyG *and* KeyG′ *are identical, $QAS(Q_S, S)$ uses the public and secret keys generated by* KeyG

never explicitly, *but* only implicitly *through black-box invocations of algorithms* Auth', Res' *and* Ver', *and* $QAS(Q_S, S)$ *is correct and secure whenever* $QAS(Q'_S, S')$ *is correct and secure.*

A general query authentication scheme. Let $S = (\mathcal{E}, \mathcal{R})$ be a structured data set and $Q_S : \mathcal{Q} \rightarrow \mathcal{A}_S$ be any query operation. Let $D(Q_S) = (\mathcal{E}_Q, \mathcal{R}_Q, \mathsf{Answer})$ be a query data structure for Q_S. By Theorem 1, we know that there exists a certification data structure $C(Q_S) = ((\mathcal{E}_C, \mathcal{R}_C), \mathsf{Certify}, \mathsf{Verify})$ for S with respect to Q_S. Let $Q_\in : \mathcal{Q}(\mathcal{R}_C) \rightarrow \{\mathsf{yes}, \mathsf{no}\}$ be the set membership query operation, where the query space $\mathcal{Q}(\mathcal{R}_C)$ is the indexed tuples that exist in $\mathcal{R}_C$. Assuming the existence of a secure and correct $QAS(Q_\in, \mathcal{R}_C) = (\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$, we next construct $QAS(Q_S, S) = (\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$, a query authentication scheme for Q_S and S, parameterized by $QAS(Q_\in, \mathcal{R}_C)$ for set membership queries.

(A) Key-generation algorithm. By definition it is the same as KeyG', thus, $SK = SK'$ and $PK = PK'$.

(B) Authenticator. The authenticator algorithm Auth using S and Constr_C computes the structured data set $S_C = (\mathcal{E}_C, \mathcal{R}_C)$ of the corresponding certification structure $C(Q_S) = ((\mathcal{E}_C, \mathcal{R}_C), \mathsf{Certify}, \mathsf{Verify})$. Then Auth runs algorithm Auth' on input SK', PK', $Q_\in$ and $\mathcal{R}_C$. That is, algorithm Auth computes the pair $(\alpha', \mathsf{V}') \leftarrow \mathsf{Auth}'(SK', PK', Q_\in, \mathcal{R}_C)$, and then Auth outputs (α', V').

(C) Responder. The responder algorithm Res first computes the structured data sets $S_Q = (\mathcal{E}_Q, \mathcal{R}_Q)$ and $S_C = (\mathcal{E}_C, \mathcal{R}_C)$ using S and algorithms Constr_Q and Constr_C. Then, on input q, S_Q and S_C it simply runs algorithm $\mathsf{Certify}$ to produce its pair (a, τ). Then Res constructs the certification support $\mathcal{R}_C(\tau)$ of answer a by accessing set $\mathcal{R}_C$ with the use of indices in τ. For every tuple $< t >$ in $\mathcal{R}_C$, algorithm Res runs the responder algorithm Res' on inputs $< t >$, $\mathcal{R}_C$ and V' to get $(a'(t), p'(t)) \leftarrow \mathsf{Res}'(< t >, \mathcal{R}_C, \mathsf{V}')$ and, if $(t_1, \ldots, t_{|\tau|})$ is the sequence of tuples accessed in total, Res creates sequence $p' = (p'(t_1), \ldots, p'(t_{|\tau|}))$, sets $p = (\tau, \mathcal{R}_C(\tau), p')$ and finally outputs (a, p).

(D) Verifier. The verifier algorithm Ver first checks if the proof p and answer a are both well-formed and, if not, it rejects. Otherwise, by appropriately processing the proof p, algorithm Ver runs algorithm Verify on inputs q, $\mathcal{R}_C(\tau)$, a and τ. Whenever algorithm Verify needs to access and process a tuple $< t_i >$, where $< t_i >$ is the i-th tuple accessed by Verify, algorithm Ver runs algorithm Ver' on inputs PK', $< t_i >$, $(\mathsf{yes}, p'(t_i))$ and α' and if $0 \leftarrow \mathsf{Ver}'(PK', < t_i >, (\mathsf{yes}, p'(t_i)), \alpha')$, algorithm Ver rejects. Otherwise, Ver continues with the computation. Finally, Ver accepts if and only if Verify accepts.

We have thus constructed $QAS(Q_S, S)$, where Q_S is a general query operation of set S, parameterized by $QAS(Q_\in, \mathcal{R}_C)$, where $Q_\in$ is the set membership query operation and $\mathcal{R}_C$ is the certification image of S with respect to the certification data structure in use. We can show the following results.

Theorem 2 *Let* $QAS(Q_\in, \mathcal{R}_C)$ *be any query authentication structure for set membership queries and* $QAS(Q_S, S)$ *our QAS as constructed above. For any query operation* Q_S *and any data set* S, $QAS(Q_S, S)$ *is correct and secure if* $QAS(Q_\in, \mathcal{R}_C)$ *is correct and secure. (Proof in Appendix D.)*

Theorem 3 *For any query operation* Q_S *on any data set* S, *there exists a secure and correct query authentication structure* $QAS(Q_S, S)$ *based on a certification data structure* $C(Q_S)$. *Moreover,* $QAS(Q_S, S)$ *is authenticated reduced to any secure and correct query authentication structure* $QAS(Q_\in, \mathcal{R}_C)$ *for the set membership query operation* $Q_\in$ *on some certification image* $\mathcal{R}_C$ *of* $C(Q_S)$. *(Proof in Appendix D.)*

We now show what are the implications of Theorems 2 and Theorem 3 in terms of time and space complexity. First, let us define the cost measures that are of interest in a query authentication scheme $QAS(Q_S, S) = (\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ for a set of size n. Let $T_a(n)$, $T_r(n)$, $T_v(n)$ denote the time complexity of algorithms Auth, Res and Ver respectively, $S_a(n)$, $S_r(n)$ denote the space complexity of Auth, Res. Also for $QAS(Q_\in, \mathcal{R}_C) = (\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$, let $T'_a(n)$, $T'_r(n)$, $T'_v(n)$ denote the time complexity of algorithms Auth', Res' and Ver' respectively, $S'_a(n)$, $S'_r(n)$ denote the space complexity of Auth', Res'. Recall from Section 4 that for certification data structure $C(Q_S) = ((\mathcal{E}_C, \mathcal{R}_C), \mathsf{Certify}, \mathsf{Verify})$, $T_A(n)$, $T_C(n)$, $T_V(n)$, $S_Q(n)$ and $S_C(n)$ denote various time and space complexity measures. Also let $p(n)$ denote the proof size in $QAS(Q_S, S)$ and $p'(n)$ the proof size in $QAS(Q_\in, \mathcal{R}_C)$. We have the following.

Lemma 1 *Let S be a structured data set and $C(Q_S) = ((\mathcal{E}_C, \mathcal{R}_C), \mathsf{Certify}, \mathsf{Verify})$ be a certification data structure for S. Let n be the size of S and let $m(n) = |\mathcal{R}_C|$ denote the size of the certification image and $s(n) = |\mathcal{R}_C(\tau)|$ the size of the certification support of an answer.*

For any query operation Q_S, the query authentication scheme $QAS(Q_S, S) = (\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ that is based on $QAS(Q_\in, \mathcal{R}_C) = (\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$ and uses certification data structure $C(Q_S)$ has the following performance:

1. *w.r.t. time, $T_a(n) = O(T'_a(n))$, $T_r(n) = O(s(n)T'_r(n) + T_C(n))$, $T_v(n) = O(s(n)T'_v(n) + T_V(n))$;*
2. *w.r.t. space complexity, $S_a(n) = O(S'_a(n) + n + m(n))$, $S_r(n) = O(S'_r(n) + S_Q(n) + m(n))$;*
3. *w.r.t. proof size, $p(n) = O(s(n)p'(n))$.*

We can now use the above Lemma to have general complexity results in terms of our parameterized query authentication scheme $QAS(Q_S, S)$. By appropriately choosing known (secure and correct) constructions for authenticating set membership queries we can achieve trade-offs on the efficiency of general query authentication schemes. Here, we are interested only in asymptotic analysis, omitting improvements of constant factors. So, we only study the related costs with respect to the set size n and not the exact implementation of the cryptographic primitives.

Theorem 4 *Assume the settings of Lemma 1. For any query operation Q_S, the query authentication scheme $QAS(Q_S, S) = (\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ that is based on query authentication scheme $QAS(Q_\in, \mathcal{R}_C) = (\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$ and uses certification data structure $C(Q_S)$ has the following performance.*

Static Case: *Using only signatures, we have the following performance: w.r.t. time complexity, $T_a(n)$ is $O(m(n))$, $T_r(n)$ is $O(s(n) + T_C(n))$, $T_v(n)$ is $O(s(n) + T_V(n))$; w.r.t. space complexity, $S_a(n)$ is $O(n + m(n))$, $S_r(n)$ is $O(S_Q(n) + m(n))$; w.r.t. the proof size, $p(n)$ is $O(s(n))$.*

Dynamic Case: *Using signature amortization, we have the following performance:*

Hash Tree: *w.r.t. time complexity, $T_a(n)$ is $O(m(n))$, $T_r(n)$ is $O(s(n) \log n + T_C(n))$, $T_v(n)$ is $O(T_V(n) + s(n) \log n)$; w.r.t. space complexity, $S_a(n)$ is $O(n + m(n))$, $S_r(n)$ is $O(n + S_Q(n) + m(n))$; w.r.t. the proof size, $p(n)$ is $O(s(n) \log n)$; if k tuples are updated, these can be handled in $O(k \log n)$ time.*

Dynamic Accumulator: *w.r.t. time complexity, $T_a(n)$ is $O(m(n))$, $T_r(n)$ is $O(s(n)\sqrt{n} + T_C(n))$, $T_v(n)$ is $O(s(n) + T_V(n))$; w.r.t. space complexity, $S_a(n)$ is $O(n + m(n))$, $S_r(n)$ is $O(n + S_Q(n) + m(n))$; w.r.t. the proof size, $p(n)$ is $s(n)$; if k tuples are updated, these can be handled in $O(k\sqrt{n})$ time.*

By Theorem 3, all query operations can be authenticated in the three-party authentication model given a corresponding certification structure. Theorem 4 gives a detailed complexity analysis of the authenticated data structures derived by the corresponding query authentication schemes. The complexity for query authentication depends on the complexity of the query certification used. Our results hold for the RAM model of computation, which strictly includes the pointer machine model and, by Theorem 1, all query problems that have a query data structure have a certification data structure, thus our results generalize and improve previous known possibility results. Additionally, our framework provides insights for super-efficient verification, as described in the following meta-theorem (in [14], we exhibit such results for 1D range searching).

Theorem 5 *Let S be a structured data set and $\mathcal{Q}_s$ be a query operation on S. If there exists a time (space) super-efficient certification data structure for Q_S, then there exists a time (space) super-efficient authenticated data structure for Q_S.*

6 Applications of Our Framework

We discuss, rather informally, due to space limitations, three additional applications of our framework.

We have seen how certification data structures support outsourced computations in data querying. In the bounded computational model, we can actually also achieve *storage outsourcing*, where the certification image $(\mathcal{E}_C, \mathcal{R}_C)$ of data set S is entirely outsourced to an untrusted entity. Consider a certification data structure, where party A (outsourcer) runs Certify and party B (source of data) runs Verify. It is possible for

B to store only a cryptographic commitment of $(\mathcal{E}_C, \mathcal{R}_C)$ and still be able to verify its integrity throughout a series of updates on an initially empty set S. We refer to this property as *consistency*: data source B checks that any update on S, and thus on $(\mathcal{E}_C, \mathcal{R}_C)$, is in accordance with the history of previous updates. The idea is to use cryptographic primitives (e.g., hashing) that provide commitments of sets, subject to which membership can be securely checked. The following result finds applications in the popular and practical *client-server data outsourcing* model, where a client (small computational device, B) uses space super-efficient protocols to check its data that resides at a remote and untrusted server (A) (see Appendix E).

Theorem 6 *In the bounded computational model, any certification data structure can be transformed to a secure, consistent, space-optimal data outsourced scheme in the client-server communication model.*

The above result is of independent interest but can be actually applied to third-party data authentication. In Section 3, the data source stores the certification image, whereas the responder stores the data set and the certification image (parties B and A in our previous discussion). We see that using a secure data outsourced scheme, the data source can oursource the certification image to the responder and still be able to check that the data set is correctly maintained and space-optimality is achieved at the data source (similar to [2]).

Additionally, using the distributed Merkle tree construction over peer-to-peer networks in [31] (it realizes a distributed authenticated dictionary that is a secure *distributed* query authentication scheme for membership queries) and the results of the previous section, we get that all queries can be authenticated even when the responder is a distributed peer-to-peer network.

Theorem 7 *For any query operation on structured data sets there exists a space-optimal (at the data source side) and distributed (at the responder side) authenticated data structure.*

References

[1] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In *Advances in Cryptology – CRYPTO ' 98*, LNCS. Springer-Verlag, 1998.

[2] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[3] M. Blum and H. Wasserman. Program result-checking: A theory of testing meets a test of theory. In *Proceedings of the 35th Annual Symp. on Foundations of Computer Science*, pages 382–393, 1994.

[4] J. D. Bright and G. Sullivan. Checking mergeable priority queues. In *Digest of the 24th Symposium on Fault-Tolerant Computing*, pages 144–153. IEEE Computer Society Press, 1994.

[5] A. Buldas, P. Laud, and H. Lipmaa. Accountable certificate management using undeniable attestations. In *ACM Conference on Computer and Communications Security*, pages 9–18. ACM Press, 2000.

[6] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proc. CRYPTO*, 2002.

[7] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(3):133–162, 1986.

[8] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. In *Proc. ACM Conference on Computer and Communications Security*, pages 136–145, 2001.

[9] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic data publication over the Internet. *Journal of Computer Security*, 11(3):291 – 314, 2003.

[10] U. Finkler and K. Mehlhorn. Checking priority queues. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, pages S901–S902, 1999.

[11] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr. 1988.

[12] M. T. Goodrich, M. J. Atallah, and R. Tamassia. Indexing information for data forensics. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *Proc. Int. Conf. on Applied Cryptography and Network Security (ACNS)*, volume 3531 of *LNCS*, pages 206–221. Springer-Verlag, 2005.

[13] M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In *Proc. of Information Security Conference (ISC)*, volume 2433 of *LNCS*, pages 372–388. Springer-Verlag, 2002.

[14] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *Proceedings of CT–RSA Conference*, April 2008.

[15] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Proc. RSA Conference—Cryptographers' Track*, volume 2612 of *LNCS*, pages 295–313. Springer, 2003.

[16] P. C. Kocher. On certificate revocation and validation. In *Proc. Int. Conf. on Financial Cryptography*, volume 1465 of *LNCS*. Springer-Verlag, 1998.

[17] H. F. Korth and A. Silberschatz. *Database system concepts*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[18] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs.

[19] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[20] K. Mehlhorn, A. Eigenwillig, K. Kanegossi, D. Kratsch, R. McConnel, U. Meyer, and J. Spinrad. Certifying algorithms (A paper under construction). Manuscript, 2005.

[21] K. Mehlhorn and S. Näher. *Checking Geometric Structures*, Dec. 1996. Program Documentation.

[22] K. Mehlhorn, S. Näher, M. Seel, R. Seidel, T. Schilz, S. Schirra, and C. Uhrig. Checking geometric programs or verification of geometric structures. *Comput. Geom. Theory Appl.*, 12(1–2):85–103, 1999.

[23] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO '89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1989.

[24] S. Micali, M. Rabin, and J. Kilian. Zero-Knowledge sets. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 2003.

[25] G. Miklau and D. Suciu. Implementing a tamper-evident database system. In *Asian Computing Science Conference, Data Management on the Web*, 2005.

[26] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.

[27] M. Naor and G. N. Rothblum. The complexity of online memory checking. In *Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005.

[28] G. Nuckolls. Verified query results from hybrid authentication trees. In *DBSec05*, 2005.

[29] R. Ostrovsky, C. Rackoff, and A. Smith. Efficient consistency proofs for generalized queries on a committed database.

[30] R. Tamassia and N. Triandopoulos. Computational bounds on hierarchical data processing with applications to information security.

[31] R. Tamassia and N. Triandopoulos. Efficient content authentication in peer-to-peer networks. In *Proceedings of Applied Cryptography and Network Security*, pages 354–372, 2007.

A Definitions of Cryptographic Primitives

We now overview some cryptographic primitives that are useful for our exposition. We give a definition of a signature scheme as in [11], which has become the standard definition of security for signature schemes. Schemes that satisfy it are also known as signature schemes secure against *adaptive chosen-message attack*. A function $\nu : \mathbb{N} \rightarrow \mathbb{R}$ is *negligible* if for every positive polynomial $p(\cdot)$ and for sufficiently large k, $\nu(k) < \frac{1}{p(k)}$.

Definition 11 (Signature scheme) *The triple of PPT algorithms* $(G(\cdot), \mathsf{Sign}_{(\cdot)}(\cdot), \mathsf{Verify}_{(\cdot)}(\cdot,\cdot))$*, where* G *is the key generation algorithm,* Sign *is the signature algorithm, and* Verify *the verification algorithm, constitute a digital signature scheme for a family (indexed by the public key* PK*) of message spaces* $\mathcal{M}_{(\cdot)}$ *if the following two properties hold:*

Correctness *If a message* m *is in the message space for a given public key* PK*, and* SK *is the corresponding secret key, then the output of* $\mathsf{Sign}_{SK}(m)$ *will always be accepted by the verification algorithm* Verify_{PK}*. More formally, for all values* m *and* k*:*

$$\Pr[(PK, SK) \leftarrow G(1^k); \sigma \leftarrow \mathsf{Sign}_{SK}(m) \ : \ m \leftarrow \mathcal{M}_{PK} \wedge \neg \mathsf{Verify}_{PK}(m, \sigma)] = 0.$$

Security *Even if an adversary has oracle access to the signing algorithm that provides signatures on messages of the adversary's choice, the adversary cannot create a valid signature on a message not explicitly queried. More formally, for all families of probabilistic polynomial-time oracle Turing machines* $\{A_k^{(\cdot)}\}$*, there exists a negligible function* $\nu(k)$ *such that*

$$\begin{aligned} \Pr[(PK, SK) \leftarrow G(1^k); (Q, m, \sigma) \leftarrow A_k^{\mathsf{Sign}_{SK}(\cdot)}(1^k) : \\ \mathsf{Verify}_{PK}(m, \sigma) = 1 \wedge \neg(\exists \sigma' \mid (m, \sigma') \in Q)] \quad = \quad \nu(k). \end{aligned}$$

A cryptographic hash function h operates on a variable-length message M producing a fixed-length hash value $h(M)$. Moreover, all the desired security results are achieved by means of *collision-resistance*, an additional security property required for hash function h. A cryptographic hash function h is called *collision-resistant* if (i) it takes as input a string of arbitrary length and outputs a short string; and (ii) it is infeasible to find two different strings $x \neq y$ that hash to the same value, i.e., form a *collision* $h(x) = h(y)$. For completeness, we give a standard definition of a family of collision-resistant hash functions.

Definition 12 (Collision-resistant Hash Function) *Let* $\mathcal{H}$ *be a probabilistic polynomial-time algorithm that, on input* 1^k*, outputs an algorithm* $h \ : \ \{0,1\}^* \mapsto \{0,1\}^k$*. Then* $\mathcal{H}$ *defines a family of collision-resistant hash functions if:*

Efficiency *For all* $h \in \mathcal{H}(1^k)$*, for all* $x \in \{0,1\}^*$*, it takes polynomial time in* $k + |x|$ *to compute* $h(x)$*.*

Collision-resistance *For all families of probabilistic polynomial-time Turing machines* $\{A_k\}$*, there exists a negligible function* $\nu(k)$ *such that*

$$\Pr[h \leftarrow \mathcal{H}(1^k); (x_1, x_2) \leftarrow A_k(h) \ : \ x_1 \neq x_2 \wedge h(x_1) = h(x_2)] = \nu(k).$$

We shall use a cryptographic collision-resistant hash function h to compute the digest of a structured data set $S = (\mathcal{E}, \mathcal{R})$. For this, we assume some fixed, well-defined binary representation for any data element e in $\mathcal{E}$, so that h can operate on e. That is, we over-notate h to operate on data elements. Also, we assume that rules have been defined so that h can operate over any finite sequence of elements. That is, we, essentially, further over-notate h to also denote a *multi-variate* hash function. In particular, $h(e_{i_1}, e_{i_2}, ..., e_{i_k})$ is used to represent a hash value computed from elements $e_{i_1}, e_{i_2}, ..., e_{i_k}$, what we call a digest of these elements using hash function $h(\cdot)$. For now, we leave the exact definition of the multivariate extension of h unspecified. For instance, $h(e_{i_1}, e_{i_2}, ..., e_{i_k})$ may denote that $h(\cdot)$ operates on the concatenation of some well-defined and fixed-size binary representation of elements $e_{i_1}, e_{i_2}, ..., e_{i_k}$ or on the concatenation of the individual

hashes using $h(\cdot)$ of some well-defined binary representation of elements $e_{i_1}, e_{i_2}, ..., e_{i_k}$. In both of these examples, h essentially operates on one binary string σ, where the cost of the operation of h on σ is at least proportional to the its length $|\sigma|$.

The following cryptographic primitive is based on the Merkle hash tree [23].

Definition 13 (Hash Tree) *For a set of n elements a hash tree is a binary tree, where each node stores a hash value computed using a collision-resistant hash function. At leaf nodes the hash of the corresponding element is stored; at internal nodes the hash of the concatenation of the hash values of the children nodes.*

Finally, we review dynamic accumulators. We here use a standard definition similar to the one in [6].

Definition 14 ((One-way) Dynamic Accumulator) *An accumulator for a family of inputs $\{\mathcal{X}_k\}$ is a family of families of functions $\mathcal{G} = \{\mathcal{F}_k\}$ with the following properties.*

Efficient Generation *There is an efficient algorithm Gen that on input 1^κ generates a random element f of $\mathcal{F}_k$, an auxiliary information aux_f and a trapdoor information trd_f. Both aux_f and trd_f have sizes that are linear in k.*

Efficient Evaluation *Function f is a computable function $f : \mathcal{A}_f \times \mathcal{X}_k$, where $\mathcal{A}_f$, an efficiently samplable set of accumulation values and $\mathcal{X}_k$, the proposed set of elements to be accumulated, constitute the input domain of f. Function f is polynomial-time computable given the auxiliary information aux_f.*

Quasi-Commutativity *For all $f \in \mathcal{F}_k$, $a \in \mathcal{A}_f$ and $x_1, x_2 \in \mathcal{X}_k$, it holds that*

$$f(f(a, x_1), x_2) = f(f(a, x_2), x_1).$$

Witnesses *Let $a \in \mathcal{A}_f$ and $x \in \mathcal{X}_k$. A value $w \in \mathcal{A}_f$ is called a* witness *for x in a, under f, if $a = f(w, x)$.*

Updates *Let $X \subset \mathcal{X}_k$, $x \in X$, $a_0, a_X, w \in \mathcal{A}_f$, such that $f(a_0, X) = f(w, x) = a_X$. Let $OP = \{insert, delete\}$ be the set of update operations on set X, such that $insert(\bar{x}) = X \cup \{\bar{x}\}$, $\bar{x} \in \mathcal{X}_k - X$ and $delete(\bar{x}) = X - \{\bar{x}\}$, $\bar{x} \in X$. An one-way accumulator is dynamic if there exist efficient algorithms U_{op}, W_{op}, $op \in OP$, such that:*

- *$U_{op}(trp_f, a_X, \bar{x}) = a_{\bar{X}} \in \mathcal{A}_f$ such that $a_{\bar{X}} = f(a_0, op(\bar{x}))$, that is $a_{\bar{X}} = a_{X \cup \bar{x}}$ or $a_{\bar{X}} = a_{X - \bar{x}}$,*
- *$W_{op}(f, aux_f, a_X, a_{\bar{X}}, x, \bar{x}) = w' \in \mathcal{A}_k$ such that $a_{\bar{X}}$ is as above and $a_{\bar{X}} = f(w', x)$.*

Security *An accumulator is one-way (secure) if the following holds true. Let $\mathcal{A}'_f \times \mathcal{X}'_k$ denote the domains for which the computational procedure for function $f \in \mathcal{F}_k$ is defined. That is, in principle, $\mathcal{A}'_f \supseteq \mathcal{A}_f$ and $\mathcal{X}'_k \supseteq \mathcal{X}_k$. For all probabilistic polynomial-time adversaries Adv_k*

$$\begin{aligned}\Pr[f \leftarrow Gen(1^k); a_0 \leftarrow \mathcal{A}_f; (x, w, X) \leftarrow Adv_k(f, aux_f, \mathcal{A}_f, a_0) : \\ X \subset \mathcal{X}_k; w \in \mathcal{A}'_f; x \in \mathcal{X}'_k; x \notin X; f(w, x) = f(a_0, X)] = \nu(k).\end{aligned}$$

B Time Super-Efficient Certification of Data Structures

We describe examples of time super-efficient certification data structures, further justifying the importance of the notion of answer testability. For time super-efficient certification structures, although the certification image may be as large as the query structure, the certification support of the answer to any query has size asymptotically less than the "searching trail" of the query answering algorithm Answer. In this case, a super-efficient certification data structure exploits this gap in certifying queries.

A very simple case is the dictionary problem, where $S = (\mathcal{E}, \mathcal{R})$ is an ordered key-value set of size n: $\mathcal{E}$ is a set K of n key elements with a totally ordering and a set V of n values, and $\mathcal{R}$ consists of two indexed relations, the key-value relation R_{KV} and the successor relation R_S over keys. The query operation Q_S has query space the universe that key elements are drawn from and answer space the set of all possible key-value pairs; to any query (key) q, Q_S maps the answer (key-value pair) (k, v) if $q = k$, $q \in K$ and $(k, v) \in R_{KV}$ (v is the value of k), or the answer $\perp$ (denoting negative membership answer) if no such condition is satisfied. Consider any search tree that implements the dictionary query data structure. Then

the set $(\mathcal{E}_Q, \mathcal{R}_Q)$ of a query data structure is an augmentation of $(\mathcal{E}, \mathcal{R})$, for instance $\mathcal{E}_Q$ now includes tree nodes and pointers, or $\mathcal{R}_Q$ now includes the node-data and parent-child relations. There exists a time super-efficient (and space-efficient) certification data structure for the dictionary problem. Set $(\mathcal{E}_C, \mathcal{R}_C)$ is simply $(\mathcal{E}, \mathcal{R})$. On input a query q, algorithm Certify returns as an answer test the indices in $\mathcal{R}_C$ of two tuples: if $q \in K$, the indices of tuple $< q, suc(q) >$ of the successor relation R_S and of tuple $< q, v >$ of the key-value relation R_{KV} are returned, otherwise, the indices of tuples $< x, suc(x) >, < y, suc(y) > \in R_S$, such that x is the maximum element and y is the minimum element satisfying $x < q < y$, according to the total order of K. Algorithm Verify, accesses these tuples and accepts or rejects accordingly. For instance, if $a = \perp$ and the indices of two tuples $< x, suc(x) >, < y, suc(y) >$ of the successor relation are in answer test τ, then it accepts if $x < q < y$ and $suc(x) = y$; Verify rejects in all other cases.

It is easy to see that the completeness and soundness conditions hold. We note that the soundness property depends on both the answer testing algorithm Verify and on the certification image $(\mathcal{E}_C, \mathcal{R}_C)$. For instance, although a different (than the successor) relation could satisfy the completeness property, this choice may not satisfy soundness. For instance, the "odd-rank-difference" relation (two keys have ranks in the sorted set $\mathcal{E}$ with odd difference), which includes the successor relation, satisfies only the completeness condition. Note that $T_V(n) = O(1)$ although $T_A(n) = O(\log n)$; also $S_C(n) = O(S_Q(n)) = O(n)$. The dynamic extension of this certification data structure is straightforward. We note that the successor relation can be used to support in a very similar way a time super-efficient certification data structure for one-dimensional range searching.

Also, consider the point location problem, where we ask to find the region of a planar subdivision of size n containing a given query point. Using existing efficient point-location algorithms point location queries can be answered in time $O(\log n)$. A time super-efficient certification data structure stores the trapezoidal decomposition of the subdivision. Each trapezoid is expressed as a tuple of five data elements: two vertices (defining the top and bottom sides), two edges (defining the left and right sides), and a region (containing the trapezoid). The answer test is the index of the trapezoid containing the query point, which can be computed by a simple modification of the query algorithm. The inclusion of the answer point in the answer test trapezoid is tested in $O(1)$ time. That is, again, $T_V(n) = O(1)$ although $T_A(n) = O(\log n)$. This certification data structure has also a dynamic extension. Additional examples include data structures for other geometric problems (e.g., convex hull) and also index structures for queries on relational databases, where, for instance, the correctness of the results of complex SQL type of queries ("SELECT $(\cdot)$, FROM $(\cdot)$, WHERE $(\cdot)$") seems to be verifiable independently of the searching through multi-dimensional tree-like index-structures (thus, with time complexity that is better by at least a logarithmic factor).

C Formal Security Definition for Query Authentication Schemes

Definition 15 (Security – Formal version of Definition 7) *Let* $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ *be a query authentication scheme for query operation* $Q_S : \mathcal{Q} \rightarrow \mathcal{A}_S$ *on structured data set* S*. We say that query authentication scheme* $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ *is* secure *if no probabilistic polynomial-time adversary* $\mathcal{A}$ *can win non-negligibly often in the following game:*

1. *A key pair is generated:*
$$(PK, SK) \leftarrow \mathsf{KeyG}(1^\kappa).$$
2. *The adversary* $\mathcal{A}$ *is given:*
   - *The public key* PK *as input.*
   - *Oracle access to the authenticator, i.e., for* $1 \leq i \leq poly(k)$*, where poly*$(\cdot)$ *is a polynomial, the adversary can specify a data set* S_i *of size* n *and obtain* $(\alpha_i, \mathsf{V}_i) \leftarrow \mathsf{Auth}(SK, PK, \mathcal{Q}, S_i)$*. However, the adversary cannot issue more than one query with the data set* S_i*. That is, for all* $i \neq j$*,* $S_i \neq S_j$*.*
   - *A query* $q \in \mathcal{Q}$*.*

3. *At the end, $\mathcal{A}$ outputs an authentication string α, an answer a' and a proof p.*

The adversary wins *the game if the following violation occurs:*

Violation of the security property: *The adversary did manage to construct an authentication string α in such a way, that given a query $q \in \mathcal{Q}$, the adversary outputs an incorrect answer-proof pair (a', p') that passes the verification test. Namely, the adversary wins if* one *of the following hold:*

- *The authenticator was never queried with S and yet the verification algorithm does not reject, i.e., $1 \leftarrow \mathsf{Ver}(PK, q, (a', p'), \alpha)$.*
- *The authenticator was queried with S and yet $a' \neq Q_S(q)$ and the verification algorithm accepts, i.e., $1 \leftarrow \mathsf{Ver}(PK, q, (a', p'), \alpha)$.*

D Proofs

Proof of Theorem 1. We first discuss the static case. Let $S = (\mathcal{E}, \mathcal{R})$ be a structured data set of size n and $Q_S : \mathcal{Q} \to \mathcal{A}_S$ be a query operation on S. Let $D(Q_S) = (\mathcal{E}_Q, \mathcal{R}_Q, \mathsf{Answer})$ be a query data structure for Q_S. We now describe a certification data structure $C(Q_S) = ((\mathcal{E}_C, \mathcal{R}_C), \mathsf{Certify}, \mathsf{Verify})$ for S with respect to query data structure $D(Q_S)$. First we set $(\mathcal{E}_C, \mathcal{R}_C) = (\mathcal{E}_Q, \mathcal{R}_Q)$. Algorithm Certify is an augmented version of Answer. Given a query $q \in \mathcal{Q}$ and sets $(\mathcal{E}_C, \mathcal{R}_C)$, $(\mathcal{E}_Q, \mathcal{R}_Q)$, Certify creates an empty sequence τ of indices of tuples in $\mathcal{R}_C$ and then it runs Answer on input $(q, (\mathcal{E}_Q, \mathcal{R}_Q))$ to produce the answer $Q_S(q)$. Also, any time algorithm Answer accesses a tuple $R_i[j]$ in $\mathcal{R}_Q$, algorithm Certify adds (i, j) to the end of sequence τ. When Answer terminates, so does Certify, and returns the output $a = Q_S(q)$ produced by Answer and sequence τ as the corresponding answer test.

We define algorithm Verify as an augmentation of Answer operating as follows. On input a query $q \in \mathcal{Q}$, set $(\mathcal{E}_C, \mathcal{R}_C)$, an answer a and a sequence τ, algorithm Verify starts executing algorithm Answer on input $(q, (\mathcal{E}_Q, \mathcal{R}_Q))$ and checks the execution of Answer subject to sequence τ. That is, each time Answer retrieves a tuple $R_i[j]$ in $(\mathcal{E}_Q, \mathcal{R}_Q)$, Verify removes the first element of τ and compares it to (i, j), rejecting the input if the comparison fails. When Answer terminates, the answer computed by Answer is compared with the answer provided as input: if the two answers agree (are equal) then Verify accepts its input, otherwise it rejects.

We now show that the completeness and soundness conditions are satisfied. Completeness is easily seen to hold, since the tuple-access trail of the same—correctly implementing query operation Q_S—algorithm Answer on executions of the same input is tested by algorithm Verify. Thus, we are guaranteed that Certify reports the correct for its input query answer and an answer test that when feeds the computation of Verify does not lead to rejection. With respect to soundness, we easily see that this requirement also holds: when algorithm Verify accepts on input $(q, \mathcal{R}_C, a, \cdot)$, then it is always the case that $a = Q_S(q)$. Indeed, when operating on the valid data set and on input q, algorithm Answer returns the unique, correct answer for q. Finally, it is easy to see that our certification data structure is answer-, time- and space-efficient. This follows from the fact that for any inputs Certify and Verify do a total amount of work that is only by a constant factor more than the work of Answer, thus $T_C(n) = O(T_A(n))$ and $T_V(n) = O(T_A(n))$, and the fact that $(\mathcal{E}_C, \mathcal{R}_C) = (\mathcal{E}_Q, \mathcal{R}_Q)$, thus $S_C(n) = O(S_Q(n))$. Observe that each pair (i, j) in the answer test τ is accessed in constant time.

The dynamic case is treated analogously. This time instead of the query answering algorithm Answer, we augment the update algorithm Update_Q of the query data structure to define the update and the update testing algorithms, Update_C and Updtest respectively, of certification data structure $C(Q_S)$. The completeness, soundness and complexity properties hold in a similar way as in the static case. □

Proof Sketch of Theorem 2. We start by first discussing the correctness property. Suppose that query authentication scheme $(\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$ is correct. We want to show that $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$

is correct. This easily follows from checking that the verifier Ver does not reject when given an answer-proof pair from the responder Res, for any query issued in $\mathcal{Q}$. Indeed, from the completeness property of the certification data structure the answer testing algorithm Verify does not reject, and additionally the correctness of $(\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$ guarantee that Ver does not reject because of a rejection by Ver'.

For the security we argue as follows. Suppose that $(\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$ is secure. Assume that $(\mathsf{KeyG}, \mathsf{Auth}, \mathsf{Res}, \mathsf{Ver})$ is not secure, then with overwhelming probability responder Res responds to a query $q \in \mathcal{Q}$ incorrectly but still the verifier Ver fails to reject its input. Based on the soundness property of the certification data structure in use, we must admit that it is not algorithm $\mathsf{Certify}$ that cheats the verifier, that is, it is not the indices in sequence τ that cause the problem, but rather the fact that algorithm Verify runs on incorrect data. Then there must be at least one tuple in $\mathcal{R}_C$ that although it was verified to be a member of $\mathcal{R}_C$ it is not authentic, meaning that its index is correct but one or more of the data elements in the tuple have been (maliciously) altered. We thus conclude that for at least one query the verification algorithm Ver' of query authentication scheme $(\mathsf{KeyG}', \mathsf{Auth}', \mathsf{Res}', \mathsf{Ver}')$ failed to reject on an invalid query-answer pair. This is a contradiction, since this scheme is assumed to be secure. □

Proof Sketch of Theorem 3. The result follows by our construction $QAS(Q_S, S)$ and the fact that there exist secure query authentication schemes $QAS(Q_\in, \cdot)$ for membership queries on any data set: in particular, digital signatures, Merkle's hash tree and one-way accumulators provide a correct and secure implementation of $QAS(Q_\in, \cdot)$. □

Proof Sketch of Lemma 1. It follows directly by the construction of $QAS(Q_S, S)$ and the use of $QAS(Q_\in, \mathcal{R}_C)$ and $C(Q_S) = (\mathcal{E}_C, \mathcal{R}_C, \mathsf{Certify}, \mathsf{Verify})$. □

Proof Sketch of Theorem 4. For the static case, simply the use of signatures provides a satisfactory time-space trade-off. That is, every indexed tuple in the certification image $\mathcal{R}_C$ is signed. The query authentication scheme $QAS(Q_\in, \mathcal{R}_C)$ in this case is very simple: Auth signs all tuples in $\mathcal{R}_C$ and sets α to be all these signatures with $\mathsf{V} = \perp$; Res, along with the (positive) answer to an $\in$ query, returns the corresponding tuples in $\mathcal{R}_C(\tau)$ and the corresponding signature; and Ver simply verifies a number of signatures.

For the dynamic case, the extensive use of signatures is not an efficient solution, since because of the updates on the set S, after every update all signatures have to be updated. Alternatively, signature amortization can be used, where only one digest of set $\mathcal{R}_C$ is signed (incurring $O(1)$ update (signing) cost). Two alternative options for computing the digest of set $\mathcal{R}_C$ are: (i) the use of a hash tree and (ii) the use of an accumulator. The construction of $QAS(Q_\in, \mathcal{R}_C)$ is straightforward and we omit here the details. Hash trees have linear storage needs, logarithmic access, update and verification times and logarithmic proof size. Dynamic accumulators, on the other hand, have linear storage needs, constant time verification and constant proof, at an increased cost to support updates and processes (of witnesses). Note that the use of the trapdoor information can only be used by algorithm Auth and not by algorithm Res for it would destroy the security of the scheme. In [13] some interesting trade-offs between the update and process times costs are discussed (e.g., one can achieve a $\sqrt{n}$ trade-off). □

E Consistency for Data Outsourcing in the Client-Server Model

We consider the problem of secure data outsourcing in a client-server communication model. In this setting, a *client* Cl completely outsources a data set S that he owns to a remote, untrusted *server* Ser. S is generated and queried through a series of *update* and *query* operations issued by Cl. At any time, the client Cl keeps

some *state information* s that encodes information about the current state of the outsourced set S. Then the communication protocol is as follows:

1. The client Cl keeps state information s and issues an update operation to the server Ser.
2. Server Ser performs the operation, i.e., Ser accordingly updates S to a new version S', and generates a *proof* π, which is then returned to the client Cl. We write $\pi \leftarrow \mathsf{certify}(o, S, S')$. We call the proof π returned to the client a *consistency* proof.
3. Client Cl runs a verification algorithm, which takes as input the current state s, the operation o and the corresponding consistency proof π and either accepts or rejects the input. If the input is accepted, the state s is appropriately updated to a new state s'. We write $\{(yes, s'), (no, \perp)\} \leftarrow \mathsf{verify}(s, \pi)$.

We call the above protocol and pair of algorithms (certify, verify) an *data outsourced scheme*. We next describe the security requirement we wish an data outsourced scheme to satisfy. Intuitively, we want the scheme to satisfy *correctness* and *consistency*, meaning that a correct behavior by the server Ser to any operation will be accepted by the verification algorithm, but any inconsistency or misbehavior by Ser with respect to any single update operation will be immediately detected and rejected.

More formally, consider a data outsourced scheme $(\mathsf{certify}, \mathsf{verify})$ let $\mathsf{operate}(\cdot, \cdot)$ be the algorithm that, given the current set S and an operation o, performs the operation o and brings the file to the updated version S'. We $S' \leftarrow \mathsf{operate}(o, S)$. Let $\tau = (o_1, \ldots, o_t)$ be a sequence of t operations issued by the client Cl on an initially empty set S_0 and initial empty state $s = \perp$ and let S be the set after the last operation is performed. We say that s is a *consistent* state for series τ with respect to the scheme in consideration, if s has been computed by running algorithms operate, certify and verify sequentially for all operations $o_1, \ldots, o_t$ in τ. In this case, we simply say that s is consistent with S.

Definition 16 (Security for data outsourced schemes.) *Let* $(\mathsf{certify}, \mathsf{verify})$ *be a data outsourced scheme with security parameter* κ*. Let* s *be any state that is consistent with the set* S *that corresponds to any series of operations in an initially empty set, and let* o *be any operation. Then,* $(\mathsf{certify}, \mathsf{verify})$ *is said to be* secure *if the following requirements are satisfied.*

Correctness. *Whenever* $\pi \leftarrow \mathsf{certify}(o, S, \mathsf{operate}(o, S))$*, then it holds that* $(yes, s') \leftarrow \mathsf{verify}(s, \pi)$*. That is, if the new operation* o *is performed correctly and the consistency proof is generated using algorithm* certify*, then the verification algorithm accepts and computes the new state* s' *(which is consistent with the new set).*

Consistency. *For any polynomial-time adversary* $\mathcal{A}$*, having oracle-access to algorithms* certify *and* verify*, that on input a set* S *and an operation* o *produces a consistency proof* π*, whenever* $(yes, s') \leftarrow \mathsf{verify}(s, \pi)$*, then the probability that either* $(S') \leftarrow \mathsf{operate}(o, S)$ *does not hold or* s' *is not consistent with* S' *is negligible in the security parameter* κ*. That is, assuming a polynomially bounded adversary that observes a polynomial number of protocol invocations and then produces a pair of consistency proof* π*, if* π *for the new operation* o *is accepted by the verification algorithm, then with overwhelming probability the operation has been performed correctly and the new state is consistent with the new set.*

According to this definition, if the client Cl starts from an empty set and outsources it to the server Ser (through appropriate update operations) using a secure data outsourced scheme, Cl will end up with a consistent state with the final data set. Thus, the data set is consistent with the history of updates and all future operations will be verified. With respect to efficiency, we say that a data outsourced scheme is time-*efficient* if the verification time is sub-linear in the data set size. We say that an authenticated storage scheme is space-*efficient* if the state information stored by the client Cl is sub-linear on the data set size. We say that it is space-*optimal* if the state information is of constant size.