Optimal Authentication of Operations on Dynamic Sets*

Charalampos Papamanthou
Brown University
Providence RI

Roberto Tamassia
Brown University
Providence RI

Nikos Triandopoulos†
RSA Laboratories
Cambridge MA

August 23, 2010

Abstract

We study the problem of authenticating outsourced set operations performed by an untrusted server over a dynamic collection of sets that are owned by a trusted source. We present efficient methods for authenticating fundamental set operations, such as *union* and *intersection* so that the client can verify the correctness of the received answer. Based on a novel extension of the security properties of *bilinear-map accumulators*, our authentication scheme is the first to achieve *optimality* in several critical performance measures: (1) *the verification overhead at the client is optimal*, that is, the client can verify an answer in time proportional to the size of the query parameters and answer; (2) *the update overhead at the source is constant*; (3) *the bandwidth consumption is optimal*, namely constant between the source and the server and operation-sensitive between the client and the server (i.e., proportional only to the size of the query parameters and the answer); and (4) *the storage usage is optimal*, namely constant at the client and linear at the source and the server. Updates and queries are also efficient at the server. In contrast, existing schemes entail high bandwidth and verification costs or high storage usage since they recompute the query over authentic data or precompute answers to all possible queries. We also show applications of our techniques to the authentication of *keyword searches* on outsourced document collections (e.g., inverted-index queries) and of queries in outsourced *databases* (e.g., equi-join queries). Since set intersection is heavily used in these applications, we obtain new authentication schemes that compare favorably to existing approaches.

*Work supported in part by the U.S. National Science Foundation, the Center for Geometric Computing and the Kanellakis Fellowship at Brown University and by the Center for Reliable Information Systems and Cyber Security at Boston University.

†Research performed while the author was at Boston University.

1 Introduction

Cloud services are fast becoming a vital part of today's computing reality. To mitigate equipment, maintenance and storage costs, numerous Internet-based applications are built by utilizing storage and computation resources that are offered and managed by third-party service providers (e.g., Amazon S3, EC2 and SimpleDB Web services). Also, millions of on-line users manage their personal data through remote cloud storage (e.g., Gmail, Google docs). As end-user applications consume information that comes from unknown machines, verifying the integrity of outsourced computations performed over remotely stored data is a crucial security property for (the viability and reliable development of) cloud computing [19].

Indeed, in such third-party data management settings, where remotely stored data resides outside the administrative control of its owner and where computations are performed by remote untrusted machines, the ability to check the integrity of the data and the correctness of the computations performed on this data is highly desirable, or otherwise a faulty or malicious server can tamper with the data or falsify the computation results. In addition to strong integrity guarantees, such a correctness-verification process should operate in a way that is very practical for the clients, or otherwise the benefits of computation outsourcing are dismissed. Ideally, computations should be verified without having to locally rerun them or having to utilize too much extra cloud storage.

The problem of efficiently checking the integrity of outsourced data is best studied in the model of *authenticated data structures* [28, 37], which has been developed over the last decade and is also closely related to memory checking [6]. This model considers three participating entities: a data owner, called *source*, outsources a data structure to multiple untrusted sites, called *servers*, to which, due to scalability reasons, *clients* issue queries. A server augments the answer to a client's query on the data structure with a cryptographic proof which is to be used by the client to verify the validity of the answer, based only on the trust the client has to the source. This trust is usually conveyed through a signature on time-stamped *digest* of the data structure, that is, a collision resistant succinct representation of the data structure (e.g., the root hash of a Merkle tree).

Starting from the fundamental problem of *set-membership authentication* (e.g., [15, 16, 28]), the efficient authentication of many classes of search queries has been studied, including *range search* [18, 25], *orthogonal range search* [3, 18], *graph and geometric search* [18], and *search in XML documents* [8]. These solutions can be proved secure based on the existence of collision-resistant hash functions.

In this paper and motivated from applications related to *keyword search* and *database queries*, we study the authentication of fundamental queries on general *sets* and consider the following problem. Assume that a collection of m sets $S_1, S_2, \ldots, S_m$ is outsourced at an untrusted server. We wish to authenticate queries on the sets, such as *intersection*, *union* and *difference*. For example, given t indices $i_1, i_2, \ldots, i_t$ between 1 and m, we wish to verify the correctness of the returned intersection $S_{i_1} \cap S_{i_2} \cap \ldots \cap S_{i_t}$. However, we wish to avoid the following two straightforward but highly inefficient solutions.

First, one could just provide the client with integrity proofs for the contents of the sets $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$ using well-known set-membership authentication schemes (e.g., Merkle trees [26]) and then let the client run the intersection algorithm locally. In this case, running the intersection algorithm on *correct* (verified) data would also trivially verify the correctness of the answer. Clearly, by first verifying all the elements of the sets participating in the intersection, this solution results in high communication and verification costs. For instance, in the extreme case where all stored sets have an empty intersection, the entire set collection must be verified in order to verify a constant-size answer! This is not desirable as it is against computation outsourcing.

Second, one could store one short proof (e.g., a signature by the source) for the answer of each possible intersection query. This would allow for optimal verification overhead at the client at the cost of increased storage and update overheads at the source and the server. For instance, the insertion of a singleton set in the collection would practically incur the re-computation of all proofs related to the union queries! This is

also not desirable, as it is against storage outsourcing.

In this work, we instead aim at designing set-operation authentication schemes that *minimize the authentication overhead for the three involved parties*. Inspired by the property of *super-efficient verification* that has been studied in certifying algorithms [21, 36] and certification data structures [17, 38], we wish to verify an answer in time asymptotically less than the time spent to produce the answer. In addition, we would like to obtain optimal verification times for set-operation queries; namely, to construct an authentication scheme that verifies answers of size k in $O(k)$ time—clearly an optimal process, since one needs to read the answer in order to verify it. Moreover, we would like to achieve optimal bandwidth usage, meaning that the authentication protocol *asymptotically introduces no extra communication cost*. That is, after a new update (of constant size) on the set collection is performed by the source, we want the source-server communication cost to remain constant. More importantly, we wish that the authentication of a set operation issued by the client is *operation-sensitive*, meaning that it is achieved *by using a proof of size that depends only on the (description and outcome of the) operation* and not on the size of the sets involved. More formally, if a set-operation query involves t sets (e.g., intersection of t sets) and produces an answer of size δ, we would like to proof to be proportional to $t + \delta$. This corresponds to optimal bandwidth usage as the query and answer alone require $O(t + \delta)$ communication. We note that whether the above two optimality properties are achievable for set operations was posed as an open problem in [12]. We close this problem in the affirmative.

Bandwidth-optimal and operation-sensitive authentication of data structure queries is very important from both theoretical and practical perspectives. Not increasing the asymptotic complexity of the answer, and at the same time providing authentication, implies optimal verification, the best one could hope for. Implementing such optimal solutions is important also in practice. For instance, email searches is a common task of a user's daily email activity. The search is performed by means of an *inverted index data structure* [35], that is a collection of sets each consisting of emails that include a specific term. The authentication of a search without an operation-sensitive verification could involve the communication of all the emails that contain one of the keywords of the search (even if the result is much smaller), which would make search authentication fairly impractical.

We note here that the results presented in this work are asymptotic, where the security paramater is considered to be a constant, i.e., the main problem dimensions are the sizes of the data structures, n and m (see Table 1).

1.1 Related work

The great majority of authenticated data structures involve the use of cryptographic hashing to hierarchically compute over the outsourced data one or more secure digests. Hash-based authentication schemes have been designed for various query types, including set-membership (e.g., [6, 16, 25, 28]) and search queries (e.g., [3, 18]). Moreover, authenticated data structures that are based on other cryptographic primitives, in particular cryptographic accumulators, to achieve better complexity trade-offs have been proposed (e.g., [15, 32]). Most of these schemes incur verification costs that are proportional to the time spent to produce the query answer.

Bandwidth-optimal and operation-sensitive authentication of range-search queries appears in [17], where, through a combination of hashing and accumulators, an answer of size t can be verified in $O(t)$ time using a proof of size $O(\log t)$, as opposed to $O(\log n + t)$ corresponding costs that the use of a hash tree would provide. Super-efficient authentication of two-dimensional grid searching appears in [3], where verification proofs have constant size. Finally, *shortest path queries* are authenticated in [24] (not in an *operation-sensitive* way).

Concerning set operations and database queries, Devanbu *et al.* [12] propose the problem of authenticating set operations and join queries on databases, providing solutions that are not operation-sensitive. They identify the importance of coming up with an operation-sensitive authentication scheme and they pose it as

an open problem. Possibly the work closest in context with ours is the work by Morselli *et al.* [27], where the problem of authenticating the intersection, union and difference of sets is studied. Although the size of their proofs is reduced in practice by representing sets with compressed Bloom filters and then performing operations on Bloom filters, still the asymptotic complexity of their solutions is $O(n)$, where n is the size of the sets participating in the set operation. Same linear asymptotic bounds—with an emphasis on efficient practical implementation, though—are achieved by Yang *et al.* [39]. A different approach is taken by Pang and Tan [31] where, in order to achieve operation-sensitivity, expensive pre-processing and exponential space are required (i.e., answers to all possible queries are signed). Palazzi, Pizzonia and Pucacco [30] present a practical method for authenticating general database selection queries on a single table by executing in parallel multiple selection queries referring to a single attribute and terminating all but the first received response.

Finally, related to our work is the systematic study of non-membership proofs for accumulators, firstly introduced for the RSA accumulator in [22] and later for the bilinear-map accumulator in [4, 11].

1.2 Relation to outsourced verifiable computation

Our work is related to the very recent work on *outsourced verifiable computation* [2, 9, 14]. This line of work studies the verification of general functionalities. Therefore, the problem that we study can be solved using such solutions (which, by definition, achieve *answer-sensitive verification*, the main property we shoot for in this paper but for a specific functionality). However, there are important differences between the general approach of verifiable computation and the approach we take in this paper, which we list below.

1. Our contribution refers to a specific functionality (i.e., set intersection of dynamic sets) and does not generalize to other functionalities.
2. We do not consider *privacy* of computation at all here, but we only address the problem of correctness (verifiability) of computation;
3. Our work can be applied in a (two-party) scenario where *public verifiability* is desired since there is no secret keys involved, which cannot be achieved in the general solutions since some secret information at the verifier's side is used as a means of verifying computation;
4. Finally, our work supports *efficient updates* of the evaluated set intersection circuit, which cannot be achieved in generic solutions, as the definition of the evaluated functionality goes into the definition of the system (at the initialization step) and is part of the public key.

Thus, our results are specialized in the verifiability of set operations. As such, they achieve much better levels of efficiency and practicality compared with the ones achieved by the generic constructions.

1.3 Contributions

Our results are summarized as follows:

- We give the first authentication scheme that authenticates sets operations *intersection*, *union* and *difference* optimally, where the size of the proof and the verification time are proportional to the size of the query and the answer. This closes an open problem posed in [12].
- We achieve bandwidth optimality and operation sensitivity by extending in a novel way the properties of *accumulators*, which also allows for efficient updates. Our scheme is proved secure under the (by now well-established) q-strong Diffie-Hellman assumption [7].
- We give applications of our scheme to the authentication of keyword-search queries (e.g., email search) and database queries (e.g., equi-join).

A cost analysis of our scheme shows the expected practical efficiency as well, comparing favorably with existing works, such as [27] and [39]. A detailed comparison of our work with existing schemes appears in

Table 1.

Table 1: Comparison of asymptotic complexity measures in existing work and in this paper for the authentication of an intersection query on t sets, where $1 < t \leq m$. Here, m is the total number of sets, N is the sum of the sizes of the t sets participating in the intersection, M is the sum of the sizes of all the sets in the data structure, $0 < \epsilon < 1$, δ is the *size* of the returned intersection, L is the size of the smallest set participating in the intersection and H is the size of the largest set participating in the intersection. Without using exponential space due to answer precomputation (as done in [31]), though at the expense of an extra $\log^2 N \log\log N$ factor for querying (indicated for simplicity as $\log^3 N$ in the table) our solution provides optimal proofs and optimal verification time, considerably improving in this way the linear upper bounds appearing in [12, 27, 39]. In the second part of the table we present asymptotic results for the case that $t = 2$ and all the set sizes are $\Theta(n)$. In all cases, the client space is $O(1)$ and the source space is $O(m + M)$.

work	**query time**	**proof size**	**verification time**	**server update time**	**source upd time**	**server space**	**update info**
[12, 39]	$(\delta t + L)\log H + \log m$	$(\delta t + L)\log H + \log m$	$(\delta t + L)\log H + \log m$	$\log H + \log m$	$\log H + \log m$	$m + M$	1
[27]	$N + t$	$N + \delta$	$N + t$	$m + M$	$m + M$	$m + M$	N
[31]	t	δ	$t + \delta$	2^m	2^m	$2^m + M$	2^m
this	$t + N \cdot \log^3 N$	$t + \delta$	$t + \delta$	$m^\epsilon + \log H$	$\log H$	$m + M$	1
[12, 39]	$n\log n + \log m$	$n\log n + \log m$	$n\log n + \log m$	$\log n + \log m$	$\log n + \log m$	$m + M$	1
[27]	n	n	n	$m + M$	$m + M$	$m + M$	n
[31]	1	δ	δ	m^2	m^2	$m^2 + M$	m^2
this	$n \cdot \log^3 n$	δ	δ	$m^\epsilon + \log n$	$\log n$	$m + M$	1

2 Preliminaries

In this section, we present some necessary notions that will be useful for describing our constructions.

Definition 2.1 (Negligible function) *Let $f : \mathbb{N} \to \mathbb{R}$. We say that $f(k)$ is $\mathsf{neg}(k)$ iff for any nonzero polynomial $p(k)$ there exits N such that for all $k > N$ it is $f(k) < 1/p(k)$.*

2.1 Bilinear pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be two cyclic multiplicative groups of prime order p, generated by g_1 and g_2 and for which there exists an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $\psi(g_2) = g_1$. Let also $\mathbb{G}_M$ be a cyclic multiplicative group with the same order p and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_M$ be a bilinear pairing with the following properties: (1) Bilinearity: $e(P^a, Q^b) = e(P, Q)^{ab}$ for all $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$; (2) Non-degeneracy: $e(g_1, g_2) \neq 1$; (3) Computability: There is an efficient algorithm to compute $e(P, Q)$ for all $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$. In our setting we have $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$ and $g_1 = g_2 = g$. A *bilinear pairing instance generator* is a probabilistic polynomial time algorithm that takes as input the security parameter 1^k and outputs a uniformly random tuple $(p, \mathbb{G}, \mathbb{G}_M, e, g)$ of bilinear pairings parameters.

2.2 The bilinear-map accumulator

The bilinear-map accumulator [29] is an efficient way to provide short proofs of membership for elements that belong to a set and is extensively used in our construction. It accumulates elements in $\mathbb{Z}_p^*$ (where p is a prime) and the accumulated value is an element in $\mathbb{G}$, such that there exists a bilinear map from

$\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_M$. Given a set of n elements $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ the accumulation value of $\mathcal{X}$ is defined as $\mathsf{acc}(\mathcal{X}) = g^{(x_1+s)(x_2+s)\ldots(x_n+s)}$, where g is a generator of group $\mathbb{G}$ of prime order p and $s \in \mathbb{Z}_p^*$ is a randomly chosen value that constitutes the trapdoor in the scheme. The proof of membership for an element y that belongs to set $\mathcal{X}$ will be the witness $\mathsf{W}_y = g^{\prod_{x \in \mathcal{X}: x \neq y}(x+s)}$. Accordingly, a verifier can test set membership for y by checking that the relation $e(\mathsf{W}_y, g^{y+s}) = e(\mathsf{acc}(\mathcal{X}), g)$ holds. We can generalize the above result for providing a proof that set $\mathcal{Y}$ is a *subset* of set $\mathcal{X}$. Naturally the witness for this case will be

$$\mathsf{W}_{\mathcal{Y}} = g^{\prod_{x \in \mathcal{X}: x \notin \mathcal{Y}}(x+s)}, \tag{1}$$

and the verification is performed by checking the validity of the relation $e(\mathsf{W}_{\mathcal{Y}}, g^{\prod_{y \in \mathcal{Y}}(y+s)}) = e(\mathsf{acc}(\mathcal{X}), g)$. We note that all the operations in the exponent of elements of $\mathbb{G}$ are performed modulo p, since this is the order of the group $\mathbb{G}$. The security of the bilinear-map accumulator is based on the q-strong Diffie-Hellman assumption which can be stated as follows and was introduced in [7]:

Definition 2.2 (q-strong Diffie-Hellman assumption) *Let $\mathbb{G}$ be a cyclic (multiplicative) group of order p. Given the elements $g, g^s, g^{s^2}, \ldots, g^{s^q}$ of $\mathbb{G}$ for some s chosen at random from $\mathbb{Z}_p^*$, then a computationally bounded adversary* Adv *can find $x \in \mathbb{Z}_p$ and output $(x, g^{1/(s+x)})$ with probability* $\mathsf{neg}(k)$.

We note here that we do not require the existence of a bilinear-map for the group $\mathbb{G}$ in Definition 2.2, i.e., the only requirement is that the group is *cyclic* of order p.

2.3 Proving subsets

We now extend the main security claim for the bilinear-map accumulator for computing fake witnesses for *subsets* of elements. We note that this is a generalization of the proof in [29], that was done for a single element (see proof in the Appendix):

Lemma 2.1 *Let k be the security parameter, $(p, \mathbb{G}, \mathbb{G}_M, e, g)$ be a uniformly randomly generated tuple of bilinear pairings parameters and s be chosen at random from $\mathbb{Z}_p^*$. A computationally bounded adversary* Adv *is given a set of elements $\mathcal{X}$ and the elements $g, g^s, g^{s^2}, \ldots, g^{s^q}$ of $\mathbb{G}$, where $q \geq |\mathcal{X}|$. Then* Adv *can find a set $\mathcal{Y}$ and $\mathsf{W}_{\mathcal{Y}}$ such that $\mathcal{Y} \nsubseteq \mathcal{X}$ and $e(\mathsf{W}_{\mathcal{Y}}, g^{\prod_{y \in \mathcal{Y}}(y+s)}) = e(\mathsf{acc}(\mathcal{X}), g)$ with probability* $\mathsf{neg}(k)$.

2.4 Authenticated data structures

In this paper, we develop solutions for checking the validity of set operations in the authenticated data structures computational model [25, 37], which involves three parties: A trusted *source* that owns, updates and outsources his data structure D_i, along with a signed, timestamped, collision resistant digest of it, d_i, to the untrusted *servers* that respond to queries sent by the *clients*. The *servers* should be able to provide with proofs to the queries and the *clients* should be able to verify these proofs based on their trust to the source, by basically using the correct and signed digest d_i. Complexities relevant to the source are the *source update time* (time taken for the source to compute the updated digest), *source space* and *update information* (size of information sent to the servers per update, i.e., the signed digest). Relevant to the servers are *server update time* (time taken by the server per update), *server space*, *query time* (time taken by the server to compute a proof for a query) and *proof size*. Finally, relevant to the client are *verification time* and *client space* with obvious meaning. The client verification is performed using an algorithm $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verify}(q, \Pi(q), \alpha(q), d_i)$, where q is a query on data structure D_i and $\Pi(q)$ is a proof provided by the server for answer $\alpha(q)$. Note that d_i, the digest of D_i, is an input as well.

Let now $\{\mathsf{reject}, \mathsf{accept}\} = \mathsf{check}(q, \alpha(q), D_i)$ be a deterministic algorithm that, given a query q on data structure D_i and an answer $\alpha(q)$ checks to see if this is the correct answer to query q. We can now present the formal security definition, which states that it should be difficult (except with negligible probability) for a computationally bounded adversary to produce verifying proofs for incorrect answers, even after he brings

the data structure to a state of his liking. We present the security definition that applies to any authenticated data structure and captures the points raised before:

Definition 2.3 (Security) *Suppose k is the security parameter and* Adv *is a computationally bounded adversary that is given the public key of the source* pk*. Our data structure D_0 is in the initial state with digest d_0 and is stored by the source. The adversary* Adv *is given access to D_0 and d_0. For $i = 0, \ldots, h = \mathsf{poly}(k)$ either the source or the adversary* Adv *issue an update upd_i in the data structure D_i and therefore the source computes D_{i+1} and d_{i+1}. The outputs D_{i+1} and d_{i+1} are sent to the adversary* Adv*. At the end of this game of polynomially-many rounds, the adversary* Adv *enters the attack stage where he chooses a query q and computes an answer $\alpha(q)$ and a verification proof $\Pi(q)$. We say that the authenticated data structure is secure if*

$$\Pr\left[\begin{array}{l} \{q, \Pi(q), \alpha(q)\} \leftarrow \mathsf{Adv}(1^k, \mathsf{pk}); \\ \mathsf{accept} \leftarrow \mathsf{verify}(q, \Pi(q), \alpha(q), d_h); \\ \mathsf{reject} = \mathsf{check}(q, \alpha(q), D_h); \\ \mathsf{digest}(D_h) = d_h. \end{array}\right] \leq \mathsf{neg}(k).$$

3 Main construction

In this section, we present our main construction for efficient set-operation authentication. The underlying data structure, which we call *sets collection* and describe in detail in the following paragraph, is a generalization of the *inverted index* [5].

3.1 Sets collection

The *sets collection* data structure stores elements from a universe $\mathcal{S}$. It consists of m sets, denoted with $S_1, S_2, \ldots, S_m$, each containing elements from $\mathcal{S}$. Without loss of generality we assume that our universe $\mathcal{S}$ is the set of nonnegative integers less than a k-bit prime p, i.e., $\mathcal{S} = \mathbb{Z}_p$, where k is the security parameter. Every set S_i does not contain duplicate elements, however an element x can appear in more than one set.

Each set is represented by a balanced tree: the elements are stored at the internal nodes sorted according to the inorder traversal and consecutive nodes in the traversal are linked. Thus, the space usage of the sets collection is $O(m + M)$, where M is the sum of the sizes of the sets.

In order to get some intuition, we can view the sets collection as an *inverted index*. The items are documents and each term w_i in the dictionary corresponds to a set S_i, which contains the documents where term w_i appears. In this example, m is the number of terms being indexed, which is typically in the hundreds of thousands, while M is the number of documents being indexed, which is in the billions. However, for the sake of generality, we will keep referring to *items*, instead of documents, and to *sets*, instead of terms.

3.2 Operations and complexity

The operations supported by the sets collection data structure consist of *updates* and *queries*. An update is either an *insertion* of an item into a set or a *deletion* of an item from a set. An update on a set of size n takes $O(\log n)$ time. For simplicity, we assume that the number m of sets does not change.

A query is one of the following standard set operations:

Intersection: Given indices $i_1, i_2, \ldots, i_t$, return set
$\mathsf{I} = S_{i_1} \cap S_{i_2} \cap \ldots \cap S_{i_t}$.

Union: Given indices $i_1, i_2, \ldots, i_t$, return set
$\mathsf{U} = S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_t}$.

Difference: Given indices i and j, return the set
$\mathsf{D} = S_i - S_j$.

Subset query: Given indices i and j, return true if $S_i \subseteq S_j$ and false otherwise.

We denote with δ the size of the answer to a query operation. For an intersection, union, and difference, δ is equal to the size of I, U, and D, respectively. For a subset query, δ is $O(1)$. Using a generalized merge, each query operation can be executed in time $O(N)$, where N is the sum of the sizes of the sets involved in the operation.

In order to verify the integrity of the answers to queries given by the server, we add an authentication data structure on top of the sets collection structure. Our goal is to have no additional asymptotic overhead in the communication between client and server. I.e., for a query with t parameters and answer size δ, we want the proof size to have optimal size $O(t+\delta)$. Also, we want to have efficient storage at the source, server and client and efficient running time for the query, update, and verification algorithms executed by these parties. Before we present our main construction, we overview two inefficient solutions for intersection queries to gain some intuition about the difficulty of the problem.

3.3 Two inefficient solutions

In the first solution, the source precomputes, signs and sends to the server the answers to all the possible intersection queries. The number of all possible intersection queries is $\binom{m}{1} + \binom{m}{2} + \ldots + \binom{m}{m} = 2^m$ and therefore $O(2^m)$ space is needed at the server to store the precomputed answers. When the client issues an intersection query, the server returns the precomputed answer and the source's signature on it. The client, knowing the public key of the source can verify the validity of the answer. Thus, optimal communication is achieved at the expense of exponential space at the server. An update takes also time $\Omega(2^m)$, since the source has to re-sign *all* the precomputed answers, so that replay attacks are avoided. By using a Merke tree, we can transmit only one signature but we still need to recompute all the answers. Note that if intersection queries are limited to two sets, the space at the server becomes $O(m^2)$—still inefficient.

The second solution involves maintaining Merkle trees on top of all the sets. In this way, the client can perform a query operation locally after verifying the validity of all the elements in the relevant sets of the query. This approach incurs high communication and verification costs, since a large number of elements have to be communicated in the worst case, even if the query answer is small. Variations of this technique have been used in relevant works such as [12, 27, 39].

Understanding the limitations of the above solutions, we have developed an authenticated data structure for efficient and *operation-sensitive* authentication of set operations. Since we are in the authenticated data structures model (see Section 2), we need to refer to three separate parties, i.e., the *source* (trusted), the *server* (untrusted) and the *client*.

3.4 The source

Let k be the security parameter. In our construction, the source picks a randomly generated tuple of bilinear pairings parameters $(p, \mathbb{G}, \mathbb{G}_M, e, g)$ and also a random element s from $\mathbb{Z}_p^*$. The value s is kept secret and constitutes the trapdoor of our scheme. The source next computes $g^s, g^{s^2}, \ldots, g^{s^q}$, where $q = \max_{i=1,\ldots,m} |S_i|$ (i.e., q is the maximum number of elements in a set). The authenticated data structure at the source is built as follows:

- For each set S_i, the source computes the accumulation value of S_i i.e., $\mathsf{acc}(S_i) = g^{\prod_{x \in S_i}(s+x)}$ (see Section 2);
- The source builds an *accumulation tree* [32, 33] on top of the pairs $(1, \mathsf{acc}(S_1)), \ldots, (m, \mathsf{acc}(S_m))$. We denote with d be the digest of the accumulation tree;
- Let $(\mathsf{sk}, \mathsf{pk})$ be the source's private-public key pair. The source signs digest d and sends to the server signature $\mathsf{sig}(d, t)$, where t is the current timestamp. The source also computes and sends to the server the pairs (i, g^{s^i}) and their signatures $\mathsf{sig}(i, g^{s^i})$ $(i = 1, \ldots, q)$.

Note that the digest d is a "secure" succinct description of the sets collection structure. Namely, the accumulation tree protects the integrity of the values $\mathsf{acc}(S_i)$ and the accumulation value $\mathsf{acc}(S_i)$ protects the integrity of the items contained in set S_i, for $i = 1, \ldots, m$.

We recall that for any $0 < \epsilon < 1$, an accumulation tree over m items is an authenticated dictionary with $O(1)$ query and communication complexity and $O(m^\epsilon)$ update complexity. In comparison, a Merkle tree or authenticated skip list on m items has $O(\log m)$ query, update and communication complexity.

We now describe how the source performs an update. Suppose the update issued by the source is the insertion of an item $z \in \mathbb{Z}_p$ in set S_k. Then the source can set $\mathsf{acc}(S_k) = \mathsf{acc}(S_k)^{(s+z)}$ (if z was deleted, the source would set $\mathsf{acc}(S_k) = \mathsf{acc}(S_k)^{(s+z)^{-1}}$). This operation updates the accumulation value of the set where z is inserted and takes $O(1)$ time. We note here that we take advantage of the fact that the source knows s, which allows to compute the updated accumulated value very efficiently. Next, the updated accumulation value is sent to the server, which results in a savings of an $O(n \log^2 n)$ computation at the server.

Moreover, the source needs to update the digest of the accumulation tree d, since the value of one of its leaves, i.e., the value $\mathsf{acc}(S_k)$, has changed. This can be done with methods described in [32, 33] in $O(1)$ time (note that the $O(m^\epsilon)$ update time in [32, 33] applies only for the server). After the update has been completed and the new digest d' has been computed, the source signs d', by using a fresh timestamp t' and sends $\mathsf{sig}(d', t')$ to the server.

Finally, if the update causes the size q of the largest set to increase, the source sets $q = q + 1$ and computes and sends to the server the pair (q, g^{s^q}) and the signatures $\mathsf{sig}(q, g^{s^q})$.

Lemma 3.1 *The source keeps $O(m + M)$ space, where m is the number of sets and M is the sum of the sizes of the sets. Also, the source update time is $O(\log n)$, where n is the size of the set where the update is performed. Moreover, the update authentication information has size $O(1)$ and can be computed by the source in $O(1)$ time.*

3.5 The server

The server is the untrusted party that answers queries on behalf of the source and provides proofs of validity of the answers. In addition to a copy of the sets collection structure, the server stores the following authentication structure:

- Pairs (i, g^{s^i}) and their signatures $\mathsf{sig}(i, g^{s^i})$ for all $i = 1, \ldots, q$;
- The accumulation values of the sets $\mathsf{acc}(S_i)$ for $i = 1, \ldots, m$;
- The accumulation tree on top of the pairs $(i, \mathsf{acc}(S_i))$ for $i = 1, \ldots, m$;
- The signature of the *latest* digest of the accumulation tree $\mathsf{sig}(d, t)$.

Suppose the source issues an update, e.g., the insertion of $z \in \mathbb{Z}_p$ in set S_k of size n. First, the server updates in $O(\log n)$ time the sets collection structure. Next, after receiving the updated value $\mathsf{acc}(S_k)$ from the source, the server updates the accumulation tree using techniques from [32, 33] in time $O(m^\epsilon)$, for some $0 < \epsilon < 1$. Therefore, we have:

Lemma 3.2 *The server keeps $O(m + M)$ space, where m is the number of sets and M is the sum of the sizes of the sets. Also, the server update time is $O(m^\epsilon + \log n)$, where n is the size of the set where the update is performed.*

4 Queries and verification

In this section we present the main contribution of this work and show how the server can provide compact proofs for the answers to queries. The proofs have optimal size $O(t + \delta)$, where t is the size of the query

parameters (e.g., $t = 2$ for an intersection of two sets) and δ is the answer size (e.g, $\delta = 1$ if the intersection consists of one element).

4.1 Intersection Query

The parameters of an intersection query are t indices, namely the indices $i_1, i_2, \ldots, i_t$, with $1 \leq t \leq m$. To simplify the notation, we assume without loss of generality that these indices are $1, 2, \ldots, t$. The answer to this query is the set

$$\mathsf{I} = S_1 \cap S_2 \cap \ldots \cap S_t = \{y_1, y_2, \ldots, y_\delta\}.$$

We denote with n_i the size of set S_i ($i = 1, 2, \ldots, t$) and we define $N = \sum_{i=1}^{t} n_i$. I.e., N is the total size of the sets involved in the intersection and δ is the size of the intersection.

We express the correctness of the answer I to the intersection query by means of the following two conditions:

- *Subset condition*:

$$\mathsf{I} \subseteq S_1 \wedge \mathsf{I} \subseteq S_2 \wedge \ldots \wedge \mathsf{I} \subseteq S_t\,, \tag{2}$$

- *Completeness condition*:

$$(S_1 - \mathsf{I}) \cap (S_2 - \mathsf{I}) \cap \ldots \cap (S_t - \mathsf{I}) = \emptyset\,. \tag{3}$$

Note the verification of the completeness condition is necessary since we want to make sure that I contains *all* the common elements. We now show how the server can provide compact proofs for these two conditions (Equations 2 and 3).

Before we proceed, we give the following result, derived from the standard polynomial interpolation based on FFT [10]:

Lemma 4.1 *Let $(s + x_1)(s + x_2)\ldots(s + x_n) = a_n s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0$ be an n-degree polynomial. The coefficients $a_n, a_{n-1}, \ldots, a_0$ can be computed in $O(n \log^2 n)$ time, given $x_1, x_2, \ldots, x_n$.*

For set S_j, we define the following polynomial:

$$P_j(s) = \prod_{x \in S_j,\ x \notin \mathsf{I}} (x + s)\,. \tag{4}$$

Note that polynomial $P_j(s)$ has degree at most $n_j + 1$. An important observation is as follows:

Fact 4.1 *Set I is the intersection of sets $S_1, S_2, \ldots, S_t$ if and only if the polynomials $P_1(s), P_2(s), \ldots, P_t(s)$ have no common factors.*

We now have the following lemma (proof in the Appendix):

Lemma 4.2 *Set I is the intersection of sets $S_1, S_2, \ldots, S_t$ if and only if there exist polynomials $q_1(s), q_2(s), \ldots, q_t(s)$ such that $q_1(s)P_1(s) + q_2(s)P_2(s) + \ldots + q_t(s)P_t(s) = 1$. Moreover, polynomials $q_1(s), q_2(s), \ldots, q_t(s)$ can be computed in $O(t + N \log^2 N \log\log N)$ time.*

Subset condition. For each set S_j, $1 \leq j \leq t$, the server computes the *subset witnesses* $\mathsf{W}_{\mathsf{I},j}$ (the second subscript refers to the set for which the witness is computed), as defined in Equation 1 (Section 2):

$$\mathsf{W}_{\mathsf{I},j} = g^{P_j(s)} = g^{\prod_{x \in S_j,\ x \notin \mathsf{I}} (x+s)}\,. \tag{5}$$

Witness $\mathsf{W}_{\mathsf{I},j}$ will serve as a proof that I is a subset of set S_j. By using Lemma 4.1, each such witness can be computed in time $O(n_j \log^2 n_j)$. Thus, the time for computing all the t subset witnesses is $O(t + N \log^2 N)$.

Completeness condition. Suppose $q_1(s), q_2(s), \ldots, q_t(s)$ are the polynomials computed in Lemma 4.2 that satisfy

$$q_1(s)P_1(s) + q_2(s)P_2(s) + \ldots + q_t(s)P_t(s) = 1\,. \tag{6}$$

The server computes the following values:

- Elements $\mathsf{F}_{\mathsf{I},j} = g^{q_j(s)}$ $(j = 1, \ldots, t)$, called *body witnesses*, which, by Lemma 4.2, can be computed in time $O(t + N \log^2 N \log\log N)$;
- Coefficients $b_\delta, b_{\delta-1}, \ldots, b_0$ of the polynomial $(s+y_1)(s+y_2)\ldots(s+y_\delta)$, i.e., the polynomial whose roots are the elements of the intersection. By Lemma 4.1, these $\delta + 1$ coefficients can be computed in $O(\delta \log^2 \delta)$ time.

Proof components. The proof for the answer I to the intersection query consists the following components:

- Signed timestamped digest of the accumulation tree;
- Pairs $(1, g^s), \ldots, (\delta, g^{s^\delta})$ along with their signatures $\mathsf{sig}(1, g^s), \ldots, \mathsf{sig}(\delta, g^{s^\delta})$, which are stored by the server;
- Coefficients $b_\delta, b_{\delta-1}, \ldots, b_0$ of the polynomial $(s + y_1)(s + y_2)\ldots(s + y_\delta)$ associated with the intersection $\mathsf{I} = \{y_1, y_2, \ldots, y_\delta\}$, which can be computed in $O(\delta \log^2 \delta)$ time;
- Accumulation values $\mathsf{acc}(S_1), \mathsf{acc}(S_2), \ldots, \mathsf{acc}(S_t)$ associated with the sets $S_1, S_2, \ldots, S_t$, along with their respective accumulation tree proofs $\mathsf{proof}(\mathsf{acc}(S_j))$ for $j = 1, \ldots, t$. Each accumulation value is stored by the server. Its proof has constant size and can be computed in constant time by the server;
- *Subset witnesses* $\mathsf{W}_{\mathsf{I},j}$, defined in Equation 5, for $j = 1, \ldots, t$, which can be computed in $O(t + N \log^2 N)$ time;
- *Body witnesses* $\mathsf{F}_{\mathsf{I},j} = g^{q_j(s)}$, where $q_j(s)$ satisfies Equation 6, for $j = 1, \ldots, t$, which can be computed in time $O(t + N \log^2 N \log\log N)$.

Verification. The verification algorithm executed by the client consists of several steps. First, the client uses the accumulation tree proof $\mathsf{proof}(\mathsf{acc}(S_j))$, along with the fresh digest d, to verify the integrity of $\mathsf{acc}(S_j)$, for $j = 1, \ldots, t$. This step takes $O(t)$ time. For details on the verification method for the accumulation tree, refer to [33]. Now the client is assured about the integrity of the accumulation value of each set in the query. The client next proceeds to checking the *subset condition* and the *completeness condition*. We use the following lemma (proof in the Appendix):

Lemma 4.3 *Let $\prod_{i=1}^{\delta}(s + y_i) = \sum_{i=0}^{\delta} b_i s^i$. Define an experiment where the adversary picks $\mathcal{X} = \{x_1, x_2, \ldots, x_{\delta'}\} \neq \{y_1, y_2, \ldots, y_\delta\}$ and outputs $\mathcal{X}$ and the values $b_\delta, b_{\delta-1}, \ldots, b_0$. Subsequently a random $\kappa \in \mathbb{Z}_p$ is picked. Then the probability $\Pr[\sum_{i=0}^{\delta} b_i \kappa^i = \prod_{i=1}^{\delta'}(\kappa + x_i)]$ is $O(\mathsf{poly}(k)/2^k)$.*

By running the experiment of Lemma 4.3, the client checks that the coefficients $b_\delta, b_{\delta-1}, \ldots, b_0$ of the reported intersection are correct. This step takes $O(\delta)$ time since $g^\kappa, g^{\kappa^2}, \ldots, g^{\kappa^\delta}$ can be computed in $O(\delta)$ time. Note that the experiment of Lemma 4.3 avoids the explicit computation by the client of $g^{(s+y_1)(s+y_2),\ldots,(s+y_\delta)}$, which would require computing the coefficients from scratch and therefore would take $O(\delta \log^2 \delta)$ time (see Lemma 4.1).

After verifying the coefficients and the signatures on the values (j, g^{s^j}), $(j = 1, \ldots, \delta)$, the client computes in $O(\delta)$ time $g^{(s+y_1)(s+y_2)\ldots(s+y_\delta)}$ and performs the following tests:

- The client verifies relation

$$e(g^{(s+y_1)(s+y_2)\ldots(s+y_\delta)}, \mathsf{W}_{\mathsf{I},j}) = e(\mathsf{acc}(S_j), g),$$

for all $j = 1, \ldots, t$. This test checks the *subset condition*.

- The client verifies the relation

$$\prod_{j=1}^{t} e(\mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},j}) = e(g, g)\,. \tag{7}$$

Each of the above tests takes $O(t)$ time.

If all the subset and body witnesses have been computed correctly, then Equation 7 is verified, due to Equation 6 and by the bilinear map $e(.,.)$ properties:

$$\prod_{j=1}^{t} e(\mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},j}) = e(g, g)^{\sum_{j=1}^{t} q_j(s) P_j(s)} = e(g, g)\,.$$

If any of the above steps fails, the client rejects the answer.

Security and performance. The core security argument of our construction is given below (proof in the Appendix).

Theorem 4.4 (Security for intersection) *Let* I *be the* claimed *intersection computed by the server for sets that have indices* $1, 2 \ldots, t$*. The server computes accumulation values* $\mathsf{acc}(S_j)$*, accumulation tree proofs* $\mathsf{proof}(\mathsf{acc}(S_j))$*, subset witnesses* $\mathsf{W}_{\mathsf{I},j}$ *and body witnesses* $\mathsf{F}_{\mathsf{I},j}$ *for* $j = 1, \ldots, t$*. If the verification algorithm succeeds and* I *is not the intersection of sets* $S_1, S_2, \ldots, S_t$*, then the server can break the* q*-strong Diffie-Hellman assumption (see Definition 2.2).*

The following lemma summarizes the performance of intersection queries.

Lemma 4.5 *The server executes an intersection query for* t *sets in time* $O(t + N \log^2 N \log\log N)$*, where* N *is the sum of the sizes of the input sets to the query and* δ *is the size of the set reported as answer. Also, the proof has size* $O(t + \delta)$ *and the verification time at the client is* $O(t + \delta)$*.*

4.2 Other Queries

Union query. Suppose that the parameters of a union query are (without loss of generality) indices $1, 2, \ldots, t$. The answer is the set $\mathsf{U} = S_1 \cup S_2 \cup \ldots \cup S_t$. Let $\mathsf{U} = \{y_1, y_2, \ldots, y_\delta\}$. The proof includes values $\mathsf{proof}(\mathsf{acc}(S_j))$, for $j = 1, \ldots, t$ (similarly with the intersection case) and the following items: (1) For each y_i, $i = 1, \ldots, \delta$, the server provides a witness W_{y_i} that proves that y_i belongs to set S_k, represented by $\mathsf{acc}(S_k)$, for some $k = 1, \ldots, t$. These δ witnesses can be constructed in $O(\delta + N \log^2 N)$ time (see Lemma 4.1); (2) For each $j = 1, \ldots, t$ the server provides a *subset* witness W_{S_j} for each set S_j that proves that S_j is a subset of the reported union U.[1] These witnesses have size $O(t)$ and can be constructed in $O(t\delta \log^2 \delta)$ time.

The verification proceeds as follows. The client checks that the reported union does not contain any duplicates. Next, it verifies the integrity of $\mathsf{acc}(S_j)$, for $j = 1, \ldots, t$, which can be done in $O(t)$ time, and that all the elements y_i ($i = 1, \ldots, \delta$) of the reported union belong to some set S_k, for $k = 1, \ldots, t$. This step can be done in $O(\delta)$ time. The final step of the verification checks that all the sets of the query are *subsets* of the union. The client constructs the expression $\mathsf{acc}(\mathsf{U}) = g^{(s+y_1)(s+y_2)\ldots(s+y_\delta)}$ in $O(\delta)$ time (see Lemma 4.3) and checks that the following relation holds for all $j = 1, \ldots, t$:

$$e(\mathsf{acc}(S_j), \mathsf{W}_{S_j}) = e(\mathsf{acc}(\mathsf{U}), g)\,. \tag{8}$$

If the verification algorithm succeeds, then the reported union U is correct, otherwise the q-strong Diffie-Hellman is violated (proof in the Appendix).

[1] $\mathsf{W}_{S_j} = g^{\prod_{x \in \mathsf{U} - S_j} (s+x)}$.

Theorem 4.6 (Security for union) *Let* U *be the* claimed *union of sets that have indices* $1, 2 \ldots, t$*, computed by the server. The server computes accumulation values* $\mathsf{acc}(S_j)$*, accumulation tree proofs* $\mathsf{proof}(\mathsf{acc}(S_j))$*, witnesses* W_j *for all* $j \in \mathsf{U}$ *and subset witnesses* W_{S_j} *for* $j = 1, \ldots, t$*. If the verification algorithm succeeds and* U *is not the union of sets* $S_1, S_2, \ldots, S_t$*, then the server can break the* q*-strong Diffie-Hellman assumption (see Definition 2.2).*

Lemma 4.7 *The server executes a union query for* t *sets in time* $O(t + \delta + N \log^2 N)$ *where* δ *is the size of the union and* N *is sum of the sizes of the sets involved in the union. Also, the proof has size* $O(t + \delta)$ *and the verification time at the client is* $O(t + \delta)$*.*

We note that with our technique, no duplicate elements are processed at the client's side (which do not contribute to the computation of the union anyways) and the client avoids running the union algorithm locally.

Subset query. Suppose the client asks query "Is $S_i \subseteq S_j$?" Let $N = |S_i| + |S_j|$. The proof for a positive answer is easy to construct in $O(N)$ time as described in Section 2. The size of the proof is $O(1)$ (two group elements) and the verification can be achieved in $O(1)$ time (one bilinear-map application).

For the proof of a negative answer, suppose there is an element y such as $S_i = y \cup S_{i2}$, where $y \notin S_j$ and $y \in S_i$. It suffices for the server to provide a *membership* proof for $y \in S_i$ (constructed in $O(n)$ time) and a *non-membership* proof for $y \notin S_j$ (constructed in $O(N \log^2 N)$ time by running the extended Euclidean algorithm for two polynomials, where the divisor is of degree one). Thus, we have the following result.

Lemma 4.8 *The server executes a subset query in* $O(N \log^2 N)$ *time. Also, the proof has size* $O(1)$ *and the verification time at the client is* $O(1)$*.*

Difference query. The difference query is defined by two indices i and j and the answer is set $\mathsf{D} = S_i - S_j$. The server constructs set A so that $S_i = \mathsf{D} \cup A$ and $\mathsf{D} \cap S_j = \emptyset$, and then authenticates these union and intersection operations.

4.3 Summary

We can now present the main result of our work.

Theorem 4.9 *Consider a collection of* m *sets* $S_1, \ldots, S_m$ *and let* $M = \sum_{i=1}^{m} |S_i|$*. For a query operation, let* t *be the number of involved sets,* N *be the sum of the sizes of the involved sets, and* δ *be the answer size. There exists an authenticated sets collection data structure with the following properties: (1) it is secure according to Definition 2.3 and based on the* q*-strong Diffie-Hellman assumption; (2) the client uses* $O(1)$ *space and the source and server use* $O(m + M)$ *space; (3) the source update time is* $O(\log n)$*, the update authentication information has size* $O(1)$ *and the server update time is* $O(m^\epsilon + \log n)$*, for a given constant* $0 < \epsilon < 1$*, where* n *is the size of the set of the update; (4) for an intersection query, the proof size is* $O(t + \delta)$*, the verification time is* $O(t + \delta)$ *and the query time is* $O(t + N \log^2 N \log \log N)$*; (5) for a union query, the proof size is* $O(t + \delta)$*, the verification time is* $O(t + \delta)$ *and the query time is* $O(t + \delta + N \log^2 N)$*; (6) for a subset query, the proof size is* $O(1)$*, the verification time is* $O(1)$*, and the query time is* $O(N \log^2 N)$*. (7) for a difference query, the proof size is* $O(\delta)$*, the verification time is* $O(\delta)$*, and the query time* $O(N \log^2 N \log \log N)$*.*

5 Applications

In this section we describe some applications of our efficient authenticated sets collection data structure.

5.1 Keyword-search

First of all, we notice that our scheme could be easily used to authenticate *keyword-search queries* implemented by the *inverted index* data structure [5]: Each term in the dictionary corresponds to a set in our sets collection data structure which contains as elements all the documents that include this term. A usual text query for terms m_1 and m_2 returns those documents that are included in both the sets that are represented by m_1 and m_2, i.e., their intersection. By using our scheme, we can easily authenticate any such keyword-search query with costs that are proportional to the size of the answer of the query and not proportional to the amount of data that the algorithm reads in order to process the query. Moreover, the derived *authenticated inverted index* can be efficiently updated as well. We continue now with an extension of the authenticated inverted index, the *timestamped keyword-search*.

5.2 Timestamped keyword-search

Apart from applications in web search engines, the inverted index is used in other applications that employ keyword-search as well, such as *email-search*. In email-search, a dictionary is again maintained, the terms of which are mapped into sets of email messages that contain the specific term. Therefore when we are searching our inbox for emails containing terms m_1 and m_2, an inverted index query is executed. However, it is always desirable in email search to be able to introduce a "second" dimension in searching. For example, a query could be "give me the emails that contain terms m_1 and m_2 and which were received between time t_1 and t_2", where $t_1 < t_2$. We call this procedure timestamped keyword-search.

One solution for the authentication of timestamped keyword-search would be to embed a timestamp in the documents (e.g., each email message) and have the client do the filtering locally, after he has verified—using our scheme—the intersection of the sets that correspond to terms m_1 and m_2. However, this is not *operation-sensitive* at all: The intersection can be a lot bigger than the set resulted after the application of the local filtering, making this straightforward solution inefficient.

We now describe an algorithmic construction to solve this problem. Let $t_1, t_2, \ldots, t_r$ be the discrete timestamps that we are interested in (t_i can be viewed as a certain day of the month). We define a new sets collection data structure as follows: Imagine $t_1, t_2, \ldots, t_r$ are the leaves of a binary tree. We build a *segment tree* [34] on top of these timestamps as follows: Each leaf storing timestamp t_i contains the documents (e.g., email messages) that were received at time t_i. Moreover, the internal nodes of the binary tree contain the documents that correspond to the union (note that this union does not have any common elements) of the documents contained in the children's nodes, recursively defining in this way sets of documents for all the nodes of the tree. Therefore we end up with a new sets collection data structure that is built on top of these $2r - 1$ sets (one set per internal tree node of the tree), namely the sets $T_1, T_2, \ldots, T_{2r-1}$. The timestamped keyword-search is therefore authenticated by two sets collection data structures, one built on the text terms, namely the sets $S_1, S_2, \ldots, S_m$, and one built on top of the sets of the timestamps, namely the sets $T_1, T_2, \ldots, T_{2r-1}$. Define now the extension of two timestamps $\mathsf{ext}(t_1, t_2)$ to be the *set of sets* T_i that "cover" the intreval $[t_1, t_2]$, i.e., namely the set that contains sets the union of which equals the set of all timestamps in $[t_1, t_2]$. One can easily see that for every $1 \leq t_1 \leq t_2 \leq r$, $\mathsf{ext}(t_1, t_2)$ is well-defined, and, moreover, that $|\mathsf{ext}(t_1, t_2)| = O(\log r)$.

Suppose now we want to verify the documents that contain terms m_1 and m_2 and which were received between t_1 and t_2. All we have to do is to verify the intersection of the following sets: (a) the union of sets in $\mathsf{ext}(t_1, t_2)$, (b) S_1 (set that refers to term m_1) and, (c) S_2 (set that refers to term m_2). Let $T_1, T_2, \ldots, T_\ell$ be the disjoint sets that are contained in $\mathsf{ext}(t_1, t_2)$, where $\ell = O(\log r)$. The answer to the query is the set $(S_1 \cap S_2) \cap (T_1 \cup T_2 \cup \ldots \cup T_\ell)$ which can be written as $(S_1 \cap S_2 \cap T_1) \cup (S_1 \cap S_2 \cap T_2) \cup \ldots \cup (S_1 \cap S_2 \cap T_\ell)$. Since T_i are disjoint, each term of the union contributes at least one new term to the answer, and therefore we can authenticate this query in a nearly *operation-sensitive* way by authenticating $\log r$ intersections separately (note there is an extra $O(\log r)$ multiplicative factor in the complexities of Theorem 5.1). Therefore we have

the following result:

Theorem 5.1 *Consider a collection of m term sets S_i for $i = 1, \ldots, m$, $2r-1$ timestamp sets $T_1, \ldots, T_{2r-1}$ and let $M = \sum_{i=1}^{m} |S_i|$. For a query operation in a time interval $[t_1, t_2]$, let t be the number of involved sets, N be the sum of the sizes of the involved sets, and δ be the answer size. There exists an authenticated timestamped keyword-search data structure with the following properties: (1) it is secure according to Definition 2.3 and based on the q-strong Diffie-Hellman assumption; (2) the client uses $O(1)$ space and the source and server use $O(m+M+r)$ space; (3) the source update time is $O(\log n)$, the update authentication information has size $O(1)$ and the server update time is $O((m+r)^{\epsilon} + \log n)$, for a given constant $0 < \epsilon < 1$, where n is the size of the set of the update; (4) for an intersection query in $[t_1, t_2]$, the proof size is $O(t \log r + \delta)$, the verification time is $O(t \log r + \delta)$ and the query time is $O(t + N \log r \log^2 N \log \log N)$.*

Note that in the above theorem we do not have a result concerning the authentication of union with timestamps. This is due to the following: Using the same notation as we did for the intersection, the answer to the union query, would be the set $(S_1 \cup S_2) \cap (T_1 \cup T_2 \cup \ldots \cup T_\ell)$. The nature of the answer does not allow for any further algebraic processing and therefore in order to authenticate the whole expression, one needs to authenticate the two unions separately. This leads to a solution that is not operation-sensitive, therefore the operation-sensitive authentication of this type of queries cannot be achieved with our method—at least in a way similar to the techniques we have used so far—. The same applies for the difference queries.

5.3 Equi-join queries

Finally, we show how our technique can be used in efficient authentication of *database queries*, such as *equi-join*. Let $R_1(\alpha, r_{11}, \ldots, r_{1w}), \ldots, R_m(\alpha, r_{m1}, \ldots, r_{mw})$ be m relational tables, that have up to n tuples each, and which share a common attribute α. We now want to compute the equi-join query on the common attribute α on any subset of t of them. This is basically an intersection that can be authenticated by building our scheme on top of the attributes α, for all relations $R_1, R_2, \ldots, R_m$.

To handle duplicate values for the attribute α, we build our authentication scheme on top of distinct α values for all the relations R_i, $i = 1, \ldots, m$ and S we keep a separate structure that maps α to all the related records, for all relations R_i, $i = 1, \ldots, m$. This authenticated structure can be a bilinear-map accumulated value that adds (i.e., the respective witness) only contant overhead per relation (1024 bits in practice) to the proof size, for each element that appears in the equi-join answer. We note that the authentication of these type of queries have also been studied in [39] (see Section 6). Therefore we have the following theorem:

Theorem 5.2 *Consider a collection of m relational tables R_i for $i = 1, \ldots, m$ and let $M = \sum_{i=1}^{m} |R_i|$. For an equi-join query, let t be the number of involved relational tables, N be the sum of the sizes of the involved relational tables, and δ be the answer size. There exists an authenticated data structure for equi-join queries with the following properties: (1) it is secure according to Definition 2.3 and based on the q-strong Diffie-Hellman assumption; (2) the client uses $O(1)$ space and the source and server use $O(m+M)$ space; (3) the source update time is $O(\log n)$, the update authentication information has size $O(1)$ and the server update time is $O(m^{\epsilon} + \log n)$, for a given constant $0 < \epsilon < 1$, where n is the size of the set of the update; (4) for an equi-join query, the proof size is $O(t + \delta)$, the verification time is $O(t + \delta)$ and the query time is $O(t + N \log^2 N \log \log N)$.*

5.4 Systems deployment

Here we describe how our method for timestamped keyword-search can be deployed in a real system to provide efficient integrity checking mechanisms. Our method can be used to provide secure searches in our email inbox, e.g., Gmail searches. The presented algorithms and authenticated data structures can be implemented as an extra "plug-in" service in the browser that loads every time our Gmail inbox is open.

This service runs at an untrusted server (which we call *authentication server*) following the same model that was used in the authentication of Amazon S3 data objects [20].

Suppose now we do a search in our inbox for the emails containing the word "New York" and that were received between March 7th and April 5th. Normally, and without using any authentication service, this query would be sent only to the Gmail server and return some email messages, by running an inverted index query. The results however would not provide any guarantee of correctness. However, with the use of our systems, the query is also sent to the *authentication server* that stores our authenticated data structures (accumulation tree and segment tree) and runs the presented algorithms. The authentication server will assemble a proof of size proportional to the *answer* of our query and send this proof to the client, along with the actual answer. By verifying this proof, the client will be able to verify that the emails that he is receiving as a result of his search are the corect ones, i.e., (a) no email has been ommitted from the answer, (b) no email has been wrongly included in the answer and (c) their content has not been tampered with. Notice that main novelty of our system lies in the fact that all this process can be achieved efficiently, in a way that communication and verification costs are not more than the size of the answer to our query.

6 Analysis

In this section we analyze the costs needed by our solution and compare with experimental results from other works. For bilinear maps and generic-group operations in the bilinear-map accumulator, we used the PBC library [1], a library for pairing-based cryptography, interfaced with C.

6.1 System setup

We choose our system parameters as follows. First of all, type A pairings are used, as described in [23]. These pairings are constructed on the curve $y^2 = x^3 + x$ over the base field $\mathbb{F}_q$, where q is a prime number. The multiplicative cyclic group $\mathbb{G}$ we are using is a subgroup of points in $E(\mathbb{F}_q)$, namely a subset of those points of $\mathbb{F}_q$ that belong to the elliptic curve E. Therefore this pairing is symmetric. The order of $E(\mathbb{F}_q)$ is $q + 1$ and the order of the group $\mathbb{G}$ is some prime factor p of $q + 1$. The group of the output of the bilinear map $\mathbb{G}_M$ is a subgroup of $\mathbb{F}_{q^2}$.

In order to instantiate type A pairings in the PBC library, we have to choose the size of the primes q and p. The main constraint in choosing the bit-sizes of q and p is that we want to make sure that discrete logarithm is difficult in $\mathbb{G}$ (that has order p) and in $\mathbb{F}_{q^2}$. Typical values are 160 bits for p and 512 bits for q. We use the typical value for the size of q, i.e., 512 bits. Note that with this choice of parameters the size of the elements in $\mathbb{G}$ (which have the form (x, y), i.e., points on the elliptic curve) is 1024 bits. Finally, let's assume that the accumulation tree that is built on top of the set digests, has two levels, i.e., $\epsilon = 0.5$ which makes the update time in Lemma 3.2 $O(\sqrt{m})$ and that we are using RSA signatures to sign the digest that are 1024 bits long.

6.2 Communication cost

Here we analyze the communication cost that our scheme has for an intersection of *two* sets. Let's assume that the size of the reported intersection is δ. Then as we saw in Section 4, the proof (apart from the answer itself), consists of the following values: (a) Two subset witnesses, two body witnesses, two accumulation values (each one of the accumulation values comes with two group elements that serve as a proof for it). Therefore the size of all these elements, which are all elements of group $\mathbb{G}$, is not dependent on the size of the intersection and is equal to $2 \times (1024 + 1024 + 1024 + 4 \times 1024)/8 = 14336/8 = 1792$ bytes; (b) The values $g^{s^i} \in \mathbb{G}$ (along with signatures on them) and $b_i \in \mathbb{Z}_p$ (the coefficients of the intersection), for $i = 1, \ldots, \delta$. These have size $\delta(2 \times 1024 + 160)/8 = 276\delta$ bytes; (c) Finally, the digest of the structure and

a signature on it are sent, which add another $2048/8 = 256$ bytes. Therefore the total communication cost is a linear function of δ, i.e., the function $2048 + 276\delta$ (in bytes).

We now compare the communication cost of our scheme with the analysis made in [27]. In Table 2 we compare with the results presented in Table IV of [27] where various set sizes n_1 and n_2 are used and the size of the intersection δ is always $0.01n_2$. Note that in most cases (especially for big sets sizes), our communication cost is a lot less than the one reported in [27]. More importantly, it is *not* dependent on the size of the sets participating in the intersection. In cases that our cost is worse, it is due to the big constants enforced by the use of bilinear pairings and accumulators.

Table 2: Comparison of a 2-intersection authentication overhead (proof size) of the scheme presented in [27] with our scheme. Here n_1 and n_2 are the sets sizes that are intersected and δ is the size of the intersection.

n_1	n_2	δ	KB [27]	KB (this work)
1000	1000	10	3.34	4.69
1000	100	1	1.68	2.26
1000	10	0	1.01	2.02
1000	1	0	0.46	2.02
10000	10000	100	26.88	28.95
10000	1000	10	12.15	4.69
10000	100	1	6.86	2.26
10000	10	0	3.08	2.26
100000	100000	1000	263.25	271.53
100000	10000	100	116.13	28.95
100000	1000	10	63.18	4.69
100000	100	1	26.69	2.26

Table 3: Comparison of an equi-join authentication overhead (proof size) of the scheme presented in [27] with our scheme. Tuple size is in bytes.

tuple size	32	64	128	256	512
MB [39]	15	18.33	30	43.33	66.66
MB (this work)	16.67	17.62	19.51	23.29	30.86

Finally, we compare our solution, in terms of communication cost, with the cost required for authenticating equi-joins with the most efficient algorithm presented in [39], i.e., algorithm AIM (see Table 3). In Figure 17 of [39] two relations R and S are equi-joined and the size of the verification object (VO) is displayed, for multiple tuple sizes (a tuple is a row in the relations) $\mathsf{tup} = 32, 64, 128, 256, 512$ bytes. For this experiment, the size of the answer is 31×10^3 tuples and therefore if we use our scheme the cost is $2048 + 276\delta + \delta\mathsf{tup} + 2\delta \times 128$ bytes, for $\delta = 31 \times 10^3$ (see Section 5). Note that, especially for large tuple sizes, there are considerable savings with our scheme.

6.3 Verification cost

Let exp, mult, add be the times needed to perform an exponentiation, a multiplication and an addition respectively, all modulo p. Let also EXP, MULT, ADD be the respective times in group $\mathbb{G}$ and $\mathcal{EXP}$, $\mathcal{MULT}$, $\mathcal{ADD}$ be the respective times in the target group of the bilinear map $\mathbb{G}_M$. Finally let MAP be the time needed to perform the operation $e(.,.)$ and sig be the time to verify a 1024-bit RSA signature. We

benchmarked all these operations using the PBC library [1] (version $\mathsf{pbc} - 0.5.7$), on a 64-bit, 2.8GHz Intel based, dual-core, dual-processor machine with 4GB main memory, running Debian Linux, and derived the following times, i.e., $\mathsf{MAP} = 5$ms, $\mathcal{MULT} = 0.005$ms, $\mathsf{exp} = 0.02$, $\mathsf{add} = 0.002$ms, $\mathsf{mult} = 0.002$ms and $\mathsf{sig} = 2.7$ms.

We analyze now the verification cost of a 2-intersection, required by our scheme. Let S_i and S_j be the sets of the intersection. The client, as soon as he receives the proof has to perform the following tasks: (a) First the client verifies $\mathsf{acc}(S_i)$ and $\mathsf{acc}(S_j)$, which requires two bilinear-map computations for each value, therefore takes time $4\mathsf{MAP}$, and one signature verification of the final digest, taking time sig; (b) Then he has to run the experiment of Lemma 4.3. The time needed for this part is $\delta(2\mathsf{mult} + 2\mathsf{add} + \mathsf{exp} + \mathsf{sig})$ (the signature cost comes from the fact that the client has to verify signatures on g^{s^i} for $i = 1, \ldots, \delta$); (c) Then he checks the *subset condition* which takes time $4\mathsf{MAP}$; (d) Finally he checks the *completeness condition* that takes times $2\mathsf{MAP} + \mathcal{MULT}$. Therefore we see that the total cost for verification of a 2-intersection of size δ is

$$10\mathsf{MAP} + (\delta + 1)\mathsf{sig} + \delta(2\mathsf{mult} + 2\mathsf{add} + \mathsf{exp}) + \mathcal{MULT}\,,$$

which is a linear function in δ, namely the function $52.7 + 2.728\delta$ (in ms).

References

[1] PBC: The pairing-based cryptography library. `http://crypto.stanford.edu/pbc/`.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[3] M. J. Atallah, Y. Cho, and A. Kundu. Efficient data authentication in an environment of untrusted third-party distributors. In *Proceedings of International Conference on Data Engineering (ICDE)*, 2008.

[4] M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *Proc. Cryptographers' Track at the RSA Conference (CT-RSA)*, pages 295–308. Springer, 2009.

[5] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Publishing Company, Reading, Mass., 1999.

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

[7] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[8] B. Carminati, E. Ferrari, and E. Bertino. Securing XML data in third-party distribution systems. In *Proc. ACM Int. Conf. on Information and Knowledge Management*, CIKM, pages 99–106, 2005.

[9] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption, 2010.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[11] I. Damgård and N. Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. `http://eprint.iacr.org/`.

[12] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.

[13] J. V. Z. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[14] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[15] 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.

[16] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. DARPA Information Survivability Conference and Exposition II (DISCEX II)*, pages 68–82, 2001.

[17] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *Proc. RSA Conference, Cryptographers' Track (CT-RSA)*, volume 4964 of *LNCS*, pages 407–424. Springer, 2008.

[18] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Efficient authenticated data structures for graph connectivity and geometric search problems. *Algorithmica*, 2009. To appear.

[19] B. Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, 2008.

[20] A. Heitzmann, B. Palazzi, C. Papamanthou, and R. Tamassia. Efficient integrity checking of untrusted network storage. In *Proc. ACM CCS Int. Workshop on Storage Security and Survivability (STORAGESS)*, pages 43–54, 2008.

[21] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. In *Proc. Symp. on Discrete Algorithms*, pages 158–167. SIAM, 2003.

[22] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. In *ACNS*, pages 253–269, 2007.

[23] B. Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University, November 2008.

[24] K. M. Man Lung Yiu, Yimin Lin. Efficient verification of shortest path search via authenticated hints. In *ICDE*, page to appear, 2010.

[25] 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.

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

[27] R. Morselli, S. Bhattacharjee, J. Katz, and P. J. Keleher. Trust-preserving set operations. In *INFOCOM*, 2004.

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

[29] L. Nguyen. Accumulators from bilinear pairings and applications. In *Proc. CT-RSA, LNCS 3376, pp. 275-292, Springer.*, 2005.

[30] B. Palazzi, M. Pizzonia, and S. Pucacco. Query racing: Fast completeness certification of query results. In *Proc. Working Conf. on Data and Applications Security and Privacy (DBSEC)*, volume 6166 of *LNCS*, pages 177–192. Springer, 2010.

[31] H. Pang and K.-L. Tan. Authenticating query results in edge computing. In *Proc. of the 20th Int. Conference on Data Engineering*, pages 560–571, 2004.

[32] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 437–448. ACM, October 2008.

[33] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Cryptographic accumulators for authenticated hash tables. Cryptology ePrint Archive, Report 2009/625, 2009. `http://eprint.iacr.org/`.

[34] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[35] P. Raghavan. Information retrieval algorithms: A survey. In *Proc. 18th ACM-SIAM Sympos. Discrete Algorithms*, pages 11–18, Jan. 1997.

[36] G. F. Sullivan, G. M. Masson, and D. Wilson. Certification trails for data structures. In *21st Int. Symp. on Fault-Tolerant Computing (FTCS-21)*, pages 240–247, Montreal, Canada, 1991.

[37] R. Tamassia. Authenticated data structures. In *Proc. European Symp. on Algorithms*, volume 2832 of *LNCS*, pages 2–5. Springer-Verlag, 2003.

[38] R. Tamassia and N. Triandopoulos. Certification and authentication of data structures. In *Proc. Alberto Mendelzon Workshop on Foundations of Data Management*, 2010.

[39] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. Authenticated join processing in outsourced databases. In *SIGMOD Conference*, pages 5–18, 2009.

Appendix

Proof of Lemma 2.1. Suppose Adv finds such a set $\mathcal{Y}$ and such a $\mathsf{W}_{\mathcal{Y}} = \{y_1, y_2, \ldots, y_\ell\}$. Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and $y_j \notin \mathcal{X}$ for some $1 \leq j \leq \ell$. That means that

$$e(\mathsf{W}_{\mathcal{Y}}, g)^{\prod_{y \in \mathcal{Y}}(y+s)} = e(g,g)^{(x_1+s)(x_2+s)\ldots(x_n+s)}\,.$$

Note that $(y_j + s)$ does not divide $(x_1 + s)(x_2 + s)\ldots(x_n + s)$. Therefore there exist polynomial $Q(s)$ of degree $n-1$ and constant λ, such that $(x_1 + s)(x_2 + s)\ldots(x_n + s) = Q(s)(y_j + s) + \lambda$. Thus we have

$$\begin{aligned} e(\mathsf{W}_{\mathcal{Y}}, g)^{(y_j+s)\prod_{1\leq i\neq j\leq \ell}(y_i+s)} &= e(g,g)^{Q(s)(y_j+s)+\lambda} \Rightarrow \\ e(g,g)^{\frac{1}{y_j+s}} &= \left[e(\mathsf{W}_{\mathcal{Y}}, g)^{\prod_{1\leq i\neq j\leq \ell}(y_i+s)} e(g,g)^{-Q(s)}\right]^{\lambda^{-1}}\,. \end{aligned}$$

This means that the adversary can break the q-strong Diffie-Hellmann assumption for the target group $\mathbb{G}_M$.

Proof of Lemma 4.2. ($\Rightarrow$) This direction follows by the fact that we can use the extended Euclidean algorithm and find polynomials $q_1(s), \ldots, q_t(s)$ such that

$$q_1(s)P_1(s) + \ldots + q_t(s)P_t(s) = \mathsf{GCD}(P_1(s), P_2(s), \ldots, P_t(s)).$$

By Fact 4.1, $\mathsf{GCD}(P_1(s), P_2(s), \ldots, P_t(s)) = 1$. That completes the proof of the direct. ($\Leftarrow$) For the inverse, suppose there exist polynomials $q_1(s), q_2(s), \ldots, q_t(s)$ that satisfy the following relation $q_1(s)P_1(s) + q_2(s)P_2(s) + \ldots + q_t(s)P_t(s) = 1$ and I is not the intersection. That means that that the polynomials $P_1(s), P_2(s), \ldots, P_t(s)$ share at least one common factor, e.g., $(s + r)$. Therefore there exists some polynomial $A(s)$ such that $(s + r)A(s) = 1$, i.e., the polynomials $(s + r)A(s)$ and 1 are *equal*, which is a contradiction (note that we want the polynomials to be equal for every $s \in \mathbb{Z}_p$).

In order to compute these coefficients, we use the extended Euclidean algorithm recursively, based on the fact that the greatest common divisor $\mathsf{GCD}(P_1(s), \ldots, P_t(s))$ equals $\mathsf{GCD}(P_1(s), \mathsf{GCD}(P_2(s), \ldots, P_t(s)))$. To compute the greatest common divisor of two $O(n)$-degree polynomials, we can use the algorithm described in [13] that runs in time equal to $O(n \log^2 n \log\log n)$. Since we are using this algorithm t times, the time complexity is $O(tn \log^2 n \log\log n)$. Morever, by the property that $x \log x + y \log y \leq (x+y)\log(x+y)$ and since the size of the sets participating in the intersection is N this equals $O(t + N \log^2 N \log\log N)$, since we also have to read the t polynomials. This algorithm also outputs the required coefficients. If we arrange our data (i.e., t polynomials) on a binary tree, after all the coefficients of the internal nodes have been computed, the final coefficients for all elements at the leaves can be computed in $O(t)$ multiplications (we can avoid the $O(t \log t)$ cost) of $O(n_i)$ degree polynomials, where n_i are the degrees of the polynomials of the leaves. Therefore the result holds. .

Proof of Lemma 4.3. Since it is $\mathcal{X} = \{x_1, x_2, \ldots, x_{\delta'}\} \neq \{y_1, y_2, \ldots, y_\delta\}$ the polynomials $b_\delta \kappa^\delta + b_{\delta-1}\kappa^{\delta-1} + \ldots + b_0$ and $(\kappa + x_1)(\kappa + x_2)\ldots(\kappa + x_{\delta'})$ are not equal for every $\kappa \in \mathbb{Z}_p$. Therefore the probability in question equals the probability of κ being a root of the polynomial $b_\delta \kappa^\delta + b_{\delta-1}\kappa^{\delta-1} + \ldots + b_0 - (\kappa+x_1)(\kappa+x_2)\ldots(\kappa+x_n)$. This polynomial has degree $\max\{\delta, \delta'\}$. Since $\delta, \delta' = O(n) = \mathsf{poly}(k)$, this polynomial has $\mathsf{poly}(k)$ roots. Thus the probability is $\mathsf{poly}(k)/2^k = \mathsf{negl}(k)$.

Proof of Theorem 4.4. Suppose all the verification tests have succeeded. That means that before the verification of the last test (Equation 7) the values $\mathsf{W}_{\mathsf{I},j}$ are indeed the *subset witnesses* for the set I (unless the q-strong Diffie-Hellman assumption has been broken—see Lemma 2.1 for the subset condition and [33] for the verification of the accumulation values—), i.e.,

$$\mathsf{W}_{\mathsf{I},j} = g^{\prod_{x \in S_j : x \notin \mathsf{I}}(x+s)} = g^{P_j(s)}\,, \tag{9}$$

for all $j = 1, \ldots, t$. Suppose now set I is not the *complete* intersection and Equation 7 has been satisfied. This means that the polynomials $P_1(s), P_2(s), \ldots, P_t(s)$ have at least one common factor, say $(s + r)$. Therefore it holds $P_j(s) = (s + r)Q_j(s)$ for some polynomials $Q_j(s)$—computable in polynomial time—, for all $j = 1, \ldots, t$. Therefore, since Equation 7 is satisfied

$$
\begin{aligned}
& e(g,g) = \prod_{j=1}^{t} e\left(\mathsf{W}_{\mathsf{I},j}, \mathsf{F}_{\mathsf{I},j}\right) = \prod_{j=1}^{t} e\left(g^{P_j(s)}, \mathsf{F}_{\mathsf{I},j}\right) \\
= \quad & \prod_{j=1}^{t} e\left(g^{(s+r)Q_j(s)}, \mathsf{F}_{\mathsf{I},j}\right) = \prod_{j=1}^{t} e\left(g^{Q_j(s)}, \mathsf{F}_{\mathsf{I},j}\right)^{(s+r)} \\
= \quad & \left(\prod_{j=1}^{t} e\left(g^{Q_j(s)}, \mathsf{F}_{\mathsf{I},j}\right)\right)^{(s+r)} \\
\Leftrightarrow \quad & e(g,g)^{\frac{1}{s+r}} = \prod_{j=1}^{t} e\left(g^{Q_j(s)}, \mathsf{F}_{\mathsf{I},j}\right) .
\end{aligned}
$$

This means that the server can break the q-strong Diffie-Hellmann assumption in polynomial time for the target group $\mathbb{G}_M$ of the bilinear-map. That completes the proof.

Proof of Theorem 4.6. Suppose all the verification tests have succeeded. That means that before the verification of the last test (Equation 8) all the values $y_i \in \mathsf{U}$ belong to some S_i (unless the q-strong Diffie-Hellman assumption has been broken). Therefore the reported union cannot contain extra elements. However, the reported union can contain less elements. Suppose not all the elements are reported in U and therefore U is not the correct union. Then there should be an S_j that contains an r such that $r \notin \mathsf{U}$. Therefore we can find $P(s)$, $Q(s)$ and α such that (and since Equation 8 verifies)

$$
\begin{aligned}
& e(\mathsf{acc}(S_j), \mathsf{W}_{S_j}) = e(\mathsf{acc}(\mathsf{U}), g) \Leftrightarrow \\
& e(g, \mathsf{W}_{S_j})^{(s+r)P(s)} = e(g,g)^{(s+r)Q(s)+\alpha} \Leftrightarrow \\
& e(g, \mathsf{W}_{S_j})^{(s+r)P(s)} = e(g,g)^{(s+r)Q(s)} e(g,g)^{\alpha} \Leftrightarrow \\
& e(g,g)^{\frac{1}{s+r}} = e(g, \mathsf{W}_{S_j})^{P(s)/\alpha} e(g,g)^{-Q(s)/\alpha} .
\end{aligned}
$$

Therefore the server can break the q-strong Diffie-Hellmann assumption in polynomial time for the target group $\mathbb{G}_M$ of the bilinear-map. That completes the proof.