Optimal Authenticated Data Structures with Multilinear Forms

Charalampos Papamanthou[1], Roberto Tamassia[1], and Nikos Triandopoulos[2]

[1] Department of Computer Science, Brown University, Providence RI, USA
[2] RSA Laboratories, Cambridge MA, USA

Abstract. Cloud computing and cloud storage are becoming increasingly prevalent. In this paradigm, clients outsource their data and computations to third-party service providers. Data integrity in the cloud therefore becomes an important factor for the functionality of these web services. Authenticated data structures, implemented with various cryptographic primitives, have been widely studied as a means of providing efficient solutions to data integrity problems (e.g., Merkle trees). In this paper, we introduce a new authenticated dictionary data structure that employs *multilinear forms*, a cryptographic primitive proposed by Silverberg and Boneh in 2003 [10], the construction of which, however, remains an open problem to date. Our authenticated dictionary is *optimal*, that is, it does not add any extra asymptotic cost to the plain dictionary data structure, yielding proofs of *constant* size, i.e., asymptotically equal to the size of the answer, while maintaining other relevant complexities *logarithmic*. Instead, solutions based on cryptographic hashing (e.g., Merkle trees) require proofs of logarithmic size [40]. Because multilinear forms are not known to exist yet, our result can be viewed from a different angle: if one could prove that optimal authenticated dictionaries cannot exist in the computational model, irrespectively of cryptographic primitives, then our solution would imply that cryptographically interesting multilinear form generators cannot exist as well (i.e., it can be viewed as a reduction). Thus, we provide an alternative avenue towards proving the nonexistence of multilinear form generators in the context of general lower bounds for authenticated data structures [40] and for memory checking [18], a model similar to the authenticated data structures model.

Keywords: authenticated dictionary, multilinear forms.

1 Introduction

Recently, there has been an increasing interest in *remote storage* of information and data. People outsource their personal files at service providers that offer huge storage space and fast network connections (e.g., Amazon S3). In this way, clients create *virtual* hard drives consisting of online storage units that are operated by remote and geographically dispersed servers. In addition to a convenient solution to space-shortage, data-archiving or back-up issues, remote storage allows for load-balanced distributed data management (e.g., database outsourcing). In such settings, the ability to check the integrity of remotely stored data is an important security property, or otherwise a malicious server can easily tamper with the client's data. When this data is structured, we need to provide solutions for *authenticated data structures* [39], which offer a computational model where untrusted entities answer queries on a data structure on behalf of a trusted source and provide proof of validity of the answer to the user.

In this paper we study two-party authenticated data structures, a model closely related to memory checking [6], where a client decides to outsource his data, which is stored in a data structure, to an untrusted server. However, the client needs to make sure that whenever he retrieves his data back, he is able to verify its validity, i.e., that nobody has tampered with it. In existing literature, a variety of authenticated data structures offer solutions that use different cryptographic primitives, such as cryptographic hashing (e.g., [6]), accumulators (e.g., [35]) and lattices (e.g., [34]). The choice of the specific cryptographic primitive has a drastic impact on the efficiency of the authenticated data structure. For example it is known that, when using generic collision-resistant hash functions[1] the best one can hope for are logarithmic complexities (in the size of the data structure) [40]. On the other hand, using accumulators, which favor constant size proofs, has so far resulted only in sublinear solutions [35], i.e., of $O(n^\epsilon)$ complexities (see Table 1). Deriving for example logarithmic time query algorithms (time to construct the proof) that come with constant size proofs and constant time verification has been an open problem.

This work begins by defining the notion of "optimal" authenticated data structures (see Definition 7), i.e., authenticated data structures that do not add any asymptotic overhead to the respective "plain" non-authenticated data structures.[2] Then we present an authenticated dictionary data structure that is based on a new cryptographic primitive that was recently proposed by Silverberg and Boneh, namely *multilinear forms* [10], the construction of which remains however an open problem to date. The use of such a primitive gives an authenticated dictionary with constant communication and constant verification complexity, while maintaining all other complexities logarithmic. To the best of our knowledge, this is the *first* optimal authenticated dictionary to appear in the literature, as it exactly matches the respective complexities[3] (update time, query time, answer size) of the optimal dictionary data structure (e.g., implemented as a red-black tree).

The multilinear form cryptographic primitive that is used in our construction can be described as the "multi" version of the well-known *bilinear map*. Although initially used to attack elliptic curve systems [28], bilinear maps, being literally an efficient a tool for solving the decisional Diffie-Hellman problem, eventually proved to be a very useful tool in cryptography (e.g., [7,8,9]) after their first appearance in the literature for a "good purpose" [24]. However, the main limitation of bilinear maps is the fact that they cannot be applied twice, i.e., the output element cannot be fed back into the map $e(.,.)$ in an efficient way. Finding such maps, i.e., self-bilinear maps, which could be used in a recursive way to construct multilinear forms, was recently proved to be infeasible

[1] Generic collision-resistant (denoted with GCR in Table 1) hash functions are functions that are believed to be collision-resistant in practice, e.g., the SHA family of functions.

[2] We note here that for the dictionary problem, no such authenticated data structure is known to exist to date.

[3] In this line of work, the asymptotic complexities always refer to the *size* of the data structure n and not to the security parameter t, i.e., $t = O(1)$ (see Table 1).

for groups that are of interest in cryptography, i.e., groups where the computational Diffie-Hellman problem is hard [13].

However, since *cryptographically interesting* multilinear form generators[4] are not known to exist to date, one can view our work from a different (and more theoretical) angle: A proof through a complexity lower bound of the nonexistence of *optimal* authenticated dictionaries would imply the nonexistence of cryptographically interesting multilinear form generators (see Theorem 2). This reveals yet another important relation between two fields—combinatorics and cryptography—and becomes more promising (towards proving nonexistence of cryptographically interesting multilinear form generators) given recent advances in the derivation of general complexity lower bounds for memory checking [18] and authenticated data structures [40].

1.1 Related Work

Multilinear forms were proposed as a possible useful tool in cryptography in 2003 [10] by Silverberg and Boneh. Since then, no efficient construction of interest in cryptography has appeared. A work similar in nature with ours—where an efficient construction for a cryptographic application based on multilinear forms is proposed— appears in [25]. The impossibility of deriving multilinear forms through *self-bilinear* maps is investigated in [13].

In the context of *authenticated data structures*, several authenticated data structures based on cryptographic hashing have been developed, first being the well-known Merkle trees [29]. Blum et al. develop a similar solution [6], which is dynamized in sake of efficient certificate revocation by Naor and Nissim in [30]. *Authenticated skip lists* in the two-party model are presented in [33]. Authenticated multi-dimensional range-searching and external memory data structures (I/O efficient) are presented in [27]. Queries over distributed hash tables are efficiently authenticated in [41]. In the context of databases, authenticated inclusive and exclusive (join, projection) queries are discussed [17], where the size of the proof is linear in the size of the answer. Atallah et al. [2] present authenticated data structures for efficient 1D and 2D authenticated range search queries (query time and proof size are constant) and also for authenticating tree hierarchies [43]. Also, logarithmic lower bounds for hash-based methods are shown in [40]. Finally, in the context of memory checking, a linear lower bound on the product of reliable space and query time for online memory checkers in the information theoretic model is given [31], whereas in [18], an $\Omega(\log n/\log\log n)$ lower bound on query time for online memory checkers in the semantic security model has been shown.

Solutions for authenticated data structures in various settings using other cryptographic primitives, namely *one-way accumulators*, were introduced by Benaloh and de Mare [5]. Subsequently, refinements of the RSA accumulator [4,20,38] were shown to achieve collision resistance as well. Dynamic accumulators were introduced in [12], where, assuming an honest prover, the time to update the accumulated value or a witness is independent on the number of the accumulated elements. A first step towards a different direction, where we assume that we cannot trust the prover and therefore the trapdoor information is kept secret, was made in [21] (however this work is not

[4] I.e., multilinear form generators for groups where the discrete log problem is hard, e.g., elliptic curve groups. We call these generators *admissible* later in the paper.

applicable to the two-party model), achieving $O(n^\epsilon)$ bounds. An authenticated data structure that combines hierarchical hashing with the accumulation-based scheme of [21] is presented in [22] and accumulators using other cryptographic primitives (e.g., general groups with bilinear pairings) the security of which is based on other assumptions (e.g., hardness of strong Diffie-Hellman problem) are introduced in [11,32,42]. Non-membership proofs for accumulators are presented in [3,16,26]. Finally, *authenticated hash tables* that use the RSA accumulator are introduced in [35]. In particular, for *authenticated membership queries*, there has been a lot of work using different algorithmic and cryptographic approaches. A summary and qualitative comparison can be found in Table 1.

Table 1. Comparison of existing schemes for authenticating dictionary queries in the two-party model for a set of size n w.r.t. used techniques and various asymptotic complexity measures. Here, $0 < \epsilon < 1$ is a fixed constant, d can be any function of n ($d \leq n$), "GCR" stands for "generic collision-resistant", OWF for "one-way function", "q-SDH" for "q-strong Diffie-Hellman", "SRSA" for "strong RSA" and "q-SMDH" for "q-strong multilinear Diffie-Hellman". All complexity measures refer to n (not to the security parameter t, which is taken to be a constant) and are asymptotic values. In all schemes, the server space is $O(n)$ and the trusted space needed at the client is $O(1)$.

reference	assumption	verification proof size	consistency proof size	query time	update time	verification time
[6,27,30,33]	GCR	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$
[18]	OWF	$\log_d n$	$d \log_d n$	$\log_d n$	$d \log_d n$	$\log_d n$
[12,38]	SRSA	1	1	1	$n \log n$	1
[32]	q-SDH	1	1	1	n	1
[35]	SRSA	1	1	1	$n^\epsilon \log n$	1
[35]	SRSA	1	1	n^ϵ	1	1
[36]	q-SDH	1	1	1	n^ϵ	1
this work	q-SMDH, GCR	1	$\log n$	$\log n$	$\log n$	1

1.2 Contributions

The contributions of this paper are as follows:

1. We formally (and in a rather intuitive way) define the notion of *optimal authenticated data structures*;
2. We present the first two-party authenticated dictionary that is based on *multilinear forms*. Our construction has constant verification time and constant communication complexity, while keeping all the other complexities logarithmic in the size of the structure, n (see Table 1). To the best of our knowledge, this is the first *optimal* authenticated dictionary to appear in the literature (Theorem 1);
3. We identify an important connection between lower bounds for authenticated data structures and the existence of cryptographically interesting multilinear form generators, giving new directions and intuition for deciding this problem (Theorem 2).

2 Preliminaries

In this section we introduce the main cryptographic primitive that we use in our solution, the *k-multilinear form*. We also introduce the notion of *cryptographic accumulators*, an instantiation of which will lie at the heart of our solution. Finally we formally define two-party authenticated data structures. Before we proceed, we give the definition of *negligible* functions. If t denotes the security parameter, then we have the following:

Definition 1. *We say that a real-valued function $\nu(t)$ over natural numbers is $\mathsf{neg}(t)$ if for any nonzero polynomial p, there exists m such that $\forall n > m$, $|\nu(n)| < \frac{1}{p(n)}$.*

2.1 Multilinear Forms

Let t be the security parameter. Let now $\mathbb{G}$, $\mathbb{G}_\mathsf{T}$ be two cyclic groups of prime order p and g be a generator of $\mathbb{G}$. We let the bit-size of p (the order of both $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$) to be a polynomial of the security parameter t, i.e., $\log p = O(\mathsf{poly}(t))$. In our context, any polynomial in the security parameter is regarded as a constant, since the main dimension of our problem is the size of the authenticated data structure, n. We are now ready to define a k-multilinear form. The definition is similar to the one presented in [10]:

Definition 2. *We say that a map $e : \mathbb{G}^k \rightarrow \mathbb{G}_\mathsf{T}$ is a k-multilinear form if it satisfies the following properties:*

1. *$\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ are cyclic groups of the same prime order p;*
2. *For all $a_1, a_2, \ldots, a_k \in \mathbb{Z}_p^*$ and $x_1, x_2, \ldots, x_k \in \mathbb{G}$ it is*
$$e(x_1^{a_1}, x_2^{a_2}, \ldots, x_k^{a_k}) = e(x_1, x_2, \ldots, x_k)^{a_1 a_2 \ldots a_k} \in \mathbb{G}_\mathsf{T};$$
3. *The map is non-degenerate: If $g \in \mathbb{G}$ generates $\mathbb{G}$ then $e(g, g, \ldots, g) \in \mathbb{G}_\mathsf{T}$ generates $\mathbb{G}_\mathsf{T}$.*

Note now that since both $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ are cyclic groups of prime order p the discrete logarithm problem is hard for both $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$. Following we continue with the definition of an *admissible* k-mulitlinear form, that is going to be useful in our context:

Definition 3. *We say that a k-multilinear form $e : \mathbb{G}^k \rightarrow \mathbb{G}_\mathsf{T}$ is admissible if it satisfies the following properties:*

1. *The bit-size of the elements in $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ is independent of k, i.e., it is $O(\mathsf{poly}(t)) = O(1)$;*
2. *Group operations (exponentiation, multiplication, inversion) in $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ are independent of k, i.e., they take time $O(\mathsf{poly}(t)) = O(1)$;*
3. *The multilinear form computation $e(x_1, x_2, \ldots, x_k)$ for $x_1, x_2, \ldots, x_k \in \mathbb{G}$ takes time $O(\mathsf{poly}(t)k) = O(k)$;*
4. *The discrete logarithm problem is hard both in $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$.*

We call the groups $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ for which there exists an admissible multilinear form *admissible multilinear groups*. Finally we define the admissible multilinear form generator:

Definition 4. *An admissible multilinear form generator $\mathcal{G}(1^t, k)$ is an algorithm that runs in polynomial time and for any k, outputs a description of admissible multilinear groups $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ (along with algorithms for group operations) and the admissible k-multilinear form $e : \mathbb{G}^k \rightarrow \mathbb{G}_\mathsf{T}$.*

2.2 The Bilinear-Map Accumulator

Let $\mathbb{G}$ be a cyclic group of prime order p that has generator g. The bilinear map accumulator [32] is an efficient way to provide short proofs of membership for elements that belong to a set. The bilinear-map accumulator works as follows. It accumulates a set of elements $\mathcal{X}$ in $\mathbb{Z}_p^*$ and the accumulation value $\mathsf{acc}(\mathcal{X})$ is an element in $\mathbb{G}$. Given a set of n elements $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ the accumulation value $\mathsf{acc}(\mathcal{X})$ is defined as

$$\mathsf{acc}(\mathcal{X}) = g^{(x_1+s)(x_2+s)\ldots(x_n+s)} \in \mathbb{G}\,,$$

where $s \in \mathbb{Z}_p^*$ is a randomly chosen value that constitutes the trapdoor in the scheme, i.e., the security of the scheme is based on the fact that s is kept secret from the adversary. The proof of membership for an element x_i that belongs to set $\mathcal{X}$ will be the witness

$$\mathcal{W}_{x_i} = g^{\prod_{x_j \in \mathcal{X} : x_j \neq x_i} (x_j+s)} \in \mathbb{G}\,.$$

Accordingly, a verifier can test set membership for x_i by testing the relationship

$$\mathcal{W}_{x_i}^{(x_i+s)} \stackrel{?}{=} \mathsf{acc}(\mathcal{X})\,. \qquad (1)$$

If (1) holds, then a verifier outputs "accept", else he outputs "reject". For a proof of non-membership, instead of explicitly storing the elements $x_{j_1} < x_{j_2} < \ldots < x_{j_n}$, the respective *hashed intervals* $h(-\infty, x_{j_1}), h(x_{j_1}, x_{j_2}), \ldots, h(x_{j_n}, +\infty)$ can be stored: Then a proof of non-membership for element x is the proof of membership of the hashed interval $h(x_{j_k}, x_{j_{k+1}})$ such that $x_{j_k} < x < x_{j_{k+1}}$. Alternatively (and less efficiently), one can explicitly compute non-membership witnesses, as described in [3,16]. In the following we present the computational assumption on which the security of the bilinear-map accumulator is based, the q-strong Diffie-Hellman assumption. This assumption was introduced in [7] and holds over general cyclic prime-order groups:

Definition 5 (q-strong Diffie-Hellman assumption). *Let $\mathbb{G}$ be a cyclic group of prime order p, generated by an element $g \in \mathbb{G}$. Given the elements of $\mathbb{G}$ $g, g^s, g^{s^2}, \ldots, g^{s^q}$ for some s chosen at random from $\mathbb{Z}_p^*$, the probability that a computationally bounded adversary* Adv *finds $c \in \mathbb{Z}_p^*$ and outputs*

$$g^{\frac{1}{s+c}} \in \mathbb{G}$$

is $\mathsf{neg}(t)$*, where the probability is taken over the random choices of $s \in \mathbb{Z}_p^*$.*

The security proof argument [32] for the accumulator that has just been presented is as follows: The adversary is given the elements $g, g^s, g^{s^2}, \ldots, g^{s^q} \in \mathbb{G}$. Then, given a set of elements $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and the respective accumulation value $\mathsf{acc}(\mathcal{X})$, we can prove that if a computationally bounded adversary has an algorithm to find a witness $\mathcal{W}_{x_i}$ that passes the verification test of Equation 1 for an element $x_i \notin \mathcal{X}$, then the adversary can break the q-strong Diffie-Hellman assumption (see [32] for the proof).

2.3 Two-Party Authenticated Data Structures

In the two-party authenticated data structures model, a *client* fully *outsources* the data structure to an untrusted *server*, keeping locally only the data structure digest (e.g., digest of the Merkle tree). The digest (or succinct state) d of an authenticated data structure is computed by augmenting the data structure with an *authentication structure* that uses a certain cryptographic primitive. Every solution for an authenticated data structure needs to define d. Computed on the correct data, d will serve as a secure data structure description subject to which the answer to a data structure query will be verified at the client by means of a corresponding proof. The digest d, for a given authenticated data structure (e.g., for the well-known Merkle trees [29], the digest is the cryptographic hash of the root of the tree), should have the following, fundamental property: If t is the security parameter, a computationally bounded adversary should not be able to find two different instances of the authenticated data structure (say for example two different graphs) that have the same digest with probability more than 2^{-t}. In order for the client to verify a query on a data structure, he needs to make sure that the digest d he possesses is the correct one (i.e., it corresponds to the most fresh version of the data structure). Therefore the client has to make sure it is consistent with the history of updates.

We now continue with the security definition of a two-party authenticated data structure: Suppose D_h is the data structure (at time h) we wish to authenticate and let t be the security parameter. A two-party authenticated data structure is a collection of the following algorithms (we assume that the algorithms run by the client have always access to the secret key sk):

1. $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^t)$. It outputs the secret (known only to the client) and public key (information known to the adversary) on input the security parameter. This procedure is executed by the client;
2. $\{\mathsf{auth}(D_0), d_0\} \leftarrow \mathsf{setup}(D_0, \mathsf{pk})$: This algorithm outputs the authenticated data structure $\mathsf{auth}(D_0)$, and the respective digest d_0. This procedure is executed by either the client or the server;
3. $\{\Pi(o), \alpha(o), D_{h+1}, \mathsf{auth}(D_{h+1}), d_{h+1}\} \leftarrow \mathsf{operate}(o, D_h, \mathsf{auth}(D_h), d_h)$, where $\Pi(o)$ is the proof returned (to the client) concerning an operation o (issued by the client) and $\alpha(o)$ is the answer to the operation o. We distinguish two cases:
   - If o is a query, $\Pi(o)$ is called **verification proof** and $\alpha(o)$ is the answer to the query o;
   - If o is an update, $\Pi(o)$ is called **consistency proof** and $\alpha(o)$ is the "answer" to the update, i.e., the portion of the data structure that has changed due to the update.

   The quantities $D_{h+1}, d_{h+1}, \mathsf{auth}(D_{h+1})$ take values only in the case of updates (i.e., when the data structure changes). This procedure is executed by the server;
4. $\{\mathsf{accept}, \mathsf{reject}, d_{h+1}\} \leftarrow \mathsf{verify}(o, \alpha(o), \Pi(o), d_h, \mathsf{sk})$, where d_h is the current digest of D_h. If $\Pi(o)$ is a verification proof, then it outputs either accept or reject. If $\Pi(o)$ is a consistency proof it outputs either accept or reject and also outputs the new digest d_{h+1}. We say that the client *accepts* the update (see security definition) if this algorithm outputs "accept". Note that this method does not have access to the whole data structure but only to the proof $\Pi(o)$ and is run by the client;

5. $\{\mathsf{accept}, \mathsf{reject}\} = \mathsf{check}(q, \alpha(q), D_h)$. This method decides whether $\alpha(q)$ is a right answer for query q on data structure D_h. Note that here q is an actual query, i.e., an operation that does not change the state of the data structure.

We can now state the formal security definition:

Definition 6 (Security). *Let t be the security parameter and* Adv *be a computationally-bounded adversary that is given the public key* pk*, output by* $\mathsf{keygen}()$*. The adversary chooses the initial state of our data structure D_0 and the client computes the respective digest d_0. The adversary* Adv *is given access to D_0 and d_0. For $i = 0, \ldots, h-1 =$* $\mathsf{poly}(t)$ *the adversary* Adv *issues an update u_i in the data structure D_i and computes D_{i+1} and d_{i+1}. The client accepts updates u_i, for all $i = 0, \ldots, h-1$, by running algorithm* $\mathsf{verify}()$*. 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 for any computationally-bounded adversary* Adv*, for any query q and for any series of updates it is*

$$\Pr\begin{bmatrix} \{q, \Pi(q), \alpha(q)\} \leftarrow \mathsf{Adv}(1^t, \mathsf{pk}); \\ \mathsf{accept} \leftarrow \mathsf{verify}(q, \alpha(q), \Pi(q), d_h, \mathsf{sk}); \\ \mathsf{reject} = \mathsf{check}(q, \alpha(q), D_h); \\ \mathsf{digest}(D_h) = d_h. \end{bmatrix} \leq \nu(t),$$

where $\nu(t)$ is negligible in the security parameter t.

We note here that the authenticated data structures model is different (achieving stronger guarantees) than the *verifiable computation model* [1,14,19,23]. Important properties such as efficient updates of queried data, and unlimited queries—as opposed to one-time queries or many queries admitting well-formed (verifying) answers—are all supported in the authenticated data structures model.

2.4 Optimality in Authenticated Data Structures

In this paper we are concerned with the notion of optimality of authenticated data structures, and we indeed present one such construction (i.e., an optimal *authenticated dictionary*) that is based on *multilinear forms*. But what does it mean for an authenticated data structure to be "optimal"?

Let D_h be a plain (non-authenticated) data structure, designed for efficiently answering some type of query. Denote with $|D_h|$ the space needed by D_h and with $\{\alpha(o), D_{h+1}\} \leftarrow \mathsf{OPERATE}(o, D_h)$ the following procedure:

– If o is a query, then $\mathsf{OPERATE}(o, D_h)$ is the algorithm that produces the answer $\alpha(o)$ to the query o. In this case $D_{h+1} = D_h$;
– If o is an update, then $\mathsf{OPERATE}(o, D_h)$ is the algorithm that executes the update. The answer $\alpha(o)$ is defined in a similar way as before as the "answer" to the update, i.e., the portion of the data structure that has changed due to the update. For example, in a red-black tree data structure, the size of the "update" answer can be $O(\log n)$, since information along a logarithmic-sized path can change.

We are now ready to define an *optimal* authenticated data structure:

Definition 7. *Let D_h be a data structure and let* $\mathsf{auth}(D_h)$ *be a respective authenticated data structure, along with algorithms* $\{\mathsf{genkey}(), \mathsf{setup}(), \mathsf{operate}(), \mathsf{verify}()\}$*, as defined in Section 2.3. We say that* $\mathsf{auth}(D_h)$ *is* optimal *if and only if*

1. *The authenticated data structure* $\mathsf{auth}(D_h)$ *is secure according to Definition 6;*
2. *It is* $|\mathsf{auth}(D_h)| = O(|D_h|)$*;*
3. *For both queries and updates o, the asymptotic time complexity of the algorithm* $\{\Pi(o), \alpha(o), D_{h+1}, \mathsf{auth}(D_{h+1}), d_{h+1}\} \leftarrow \mathsf{operate}(o, D_h, \mathsf{auth}(D_h), d_h)$ *is no more than the asymptotic time complexity of the respective plain data structure algorithm* $\{\alpha(o), D_{h+1}\} \leftarrow \mathsf{OPERATE}(o, D_h)$*;*
4. *For both queries and updates o it holds:*
   (a) The size of the proof is asymptotically no more than the size of the answer, i.e., $|\Pi(o)| = O(|\alpha(o)|)$*;*
   (b) The asymptotic time complexity of the algorithm $\{\mathsf{accept}, \mathsf{reject}, d_{h+1}\} \leftarrow \mathsf{verify}(o, \alpha(o), \Pi(o), d_h, \mathsf{sk})$ *is* $O(|\alpha(o)|)$.

Note that Property 4 requires that both the size of the proof for a certain operation returned by $\mathsf{auth}(D_h)$ and the time to verify it are no more (asymptotically) than the size of the answer (computed by D_h) to the respective operation. This property greatly relates to recent work that has appeared on *super-efficient* authenticated data structures [22], where range search queries with proofs asymptotically *less* than the size of the answer are authenticated, and *operation-sensitive* authenticated data structures [37], where fundamental set operations with proofs asymptotically equal to the size of the answer are authenticated. However, none of the authenticated data structures in [22,37] is optimal, due to increased update costs.

The above definition is rather intuitive. It implies that, in order for an authenticated data structure to be *optimal*, it should not be adding any *extra* asymptotic overhead to the *plain* (non-authenticated) data structure. So far in the literature (see Table 1), and specifically for the *authenticated dictionary* problem, no *optimal* authenticated data structure has been constructed. For example, traditional hash-based methods built with Merkle trees (e.g., [6,27,30,33]) fail to achieve Property 4 from Definition 7. Indeed, although the answer is of constant size (i.e., either "yes, the element is contained" or "no, the element is not contained"), the proof for that answer is asymptotically larger than $O(1)$, i.e., it is $O(\log n)$. To achieve this property, other solutions (e.g., [32,35]) have used accumulators. However, accumulator-based solutions, although succeed in satisfying Property 4, violate Property 3 from Definition 7, since update or query time is not logarithmic (e.g., it is $O(\sqrt{n})$). In this paper, we show how to construct the first optimal authenticated dictionary, using a cryptographic primitive the construction of which is, however, still an open problem.

3 Multilinear Form Authenticated Structures

In this section we present a two-party authenticated dictionary based on multilinear forms. We begin with the main building block, the *multilnear form* accumulator.

3.1 A Multilinear Form Accumulator

Let t be the security parameter, $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be a set of elements and $\mathbb{G}$, $\mathbb{G}_\mathsf{T}$ be two cyclic groups of prime order p for which there exists an admissible multilinear form $e : \mathbb{G}^t \rightarrow \mathbb{G}_\mathsf{T}$ as defined in Definition 3. Note that we require the number of the inputs of the multilinear form to be equal to the security parameter t. Group $\mathbb{G}$ is generated by g and group $\mathbb{G}_\mathsf{T}$ is generated by $e(g, g, \ldots, g)$. As in Section 2.2, we can define a new accumulator that is similar to the bilinear-map accumulator with the difference that the base of exponentiation is the generator $e(g, g, \ldots, g)$ of the cyclic group $\mathbb{G}_\mathsf{T}$, i.e.,

$$\mathsf{acc}(\mathcal{X}) = e(g, g, \ldots, g)^{(x_1+s)(x_2+s)\ldots(x_n+s)} \in \mathbb{G}_\mathsf{T}\,, \tag{2}$$

where s is a randomly chosen element of $\mathbb{Z}_p^*$. The proof of membership for an element x_i that belongs to set $\mathcal{X}$ will be the witness

$$\mathcal{W}_{x_i} = e(g, g, \ldots, g)^{\prod_{x_j \in \mathcal{X} : x_j \neq x_i}(x_j+s)} \in \mathbb{G}_\mathsf{T}\,. \tag{3}$$

Accordingly, a verifier can test set membership for x_i by computing $\mathcal{W}_{x_i}^{(x_i+s)}$ and checking that this equals $\mathsf{acc}(\mathcal{X})$. The q-strong Diffie-Hellmann assumption can be adjusted to the multilinear form setting, leading to the q-strong multilinear Diffie-Hellmann assumption (see q-SMDH in Table 1), as follows:

Definition 8 **(q-strong multilinear Diffie-Hellman assumption).** *Let $\mathbb{G}$, $\mathbb{G}_\mathsf{T}$ be cyclic groups of prime order p such that there exists an admissible multilinear form $e : \mathbb{G}^t \rightarrow \mathbb{G}_\mathsf{T}$. Let g be the generator of $\mathbb{G}$. Given the elements of $\mathbb{G}$ $g, g^s, g^{s^2}, \ldots, g^{s^q}$ for some s chosen at random from $\mathbb{Z}_p^*$, the probability that a computationally bounded adversary* Adv *finds $c \in \mathbb{Z}_p^*$ and outputs*

$$e(g, g, \ldots, g)^{\frac{1}{c+s}} \in \mathbb{G}_\mathsf{T}$$

is $\mathsf{neg}(t)$*, where the probability is taken over the random choices of $s \in \mathbb{Z}_p^*$.*

We now prove security (similar to [32]) of the multilinear form accumulator based on the q-strong multilinear Diffie-Hellman assumption:

Lemma 1. *Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, t be the security parameter, $\mathbb{G}$, $\mathbb{G}_\mathsf{T}$ be cyclic groups of prime order p such that there exists an admissible multilinear form $e : \mathbb{G}^t \rightarrow \mathbb{G}_\mathsf{T}$ and g be the generator of $\mathbb{G}$. Under the q-strong multilinear Diffie-Hellman assumption, the probability that a computationally bounded adversary* Adv *that is given the elements $g, g^s, g^{s^2}, \ldots, g^{s^q} \in \mathbb{G}$ can find a valid witness $\mathcal{W}_x$ for an element $x \notin \mathcal{X}$ is* $\mathsf{neg}(t)$.

Proof. By Equation 3 a computationally bounded adversary Adv finds a witness $\mathcal{W}_x$ such that $\mathcal{W}_x^{(x+s)} = e(g, g, \ldots, g)^{(x_1+s)(x_2+s)\ldots(x_n+s)}$. Since $x \notin \{x_1, x_2, \ldots, x_n\}$ we can write $(x_1+s)(x_2+s)\ldots(x_n+s) = \mathcal{P}(x+s)+\lambda$, where the coefficients of polynomial $\mathcal{P}$ and quantity λ are computable in polynomial time in n (polynomial division). Therefore Adv can compute $e(g, g, \ldots, g)^{(x+s)^{-1}} = [\mathcal{W}_x[e(g, g, \ldots, g)^{\mathcal{P}}]^{-1}]^{\lambda^{-1}}$, since $e(g, g, \ldots, g)^{s^i} \in \mathbb{G}_\mathsf{T}$ can efficiently be computed from $g^{s^i} \in \mathbb{G}$ by using the admissible multilinear form $e : \mathbb{G}^k \rightarrow \mathbb{G}_\mathsf{T}$ for all $i = 0, \ldots, q$. This breaks the q-strong multilinear strong Diffie-Hellman assumption. □

3.2 A Two-Party Multilinear Form Authenticated Dictionary Construction

In this section we describe a two-party authenticated dictionary based on admissible multilinear forms that achieves constant communication complexity and constant verification complexity. Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be the elements contained in the dictionary, where $x_1 < x_2 < \ldots < x_n$. The actual set we are going to store, in order to also support efficient range search and non-membership queries, is the set of intervals, i.e., the set $\mathcal{A} = \{a_1, a_2, \ldots, a_{n-1}\}$, where $a_i = x_i || x_{i+1}$ is simply the concatenation of the binary strings x_i, x_{i+1} of bit length $2t$ (we use t bits for each x_i).

In our construction we use a red-black tree, with data at the leaves (i.e., internal nodes data navigates the searches and does not correspond to actual data) [15]. We store key-value pairs at the leaves and the ordering is according to the keys. For an interval $a_i = x_i || x_{i+1}$, the key is x_i and the value is x_{i+1}, where x_i and x_{i+1} are successive elements of our set. We recall that a red-black tree implementation of a dictionary supports operations in $O(\log n)$ time in the *worst* case.

Define now k, the number of the inputs to the admissible multilinear form that we are going to use to be t, i.e., equal to the security parameter. Note that since we are in the computational model, it is always the case that $t > \log n$, where n is the total number of elements that we store in the dictionary. In the construction that follows we refer to operations on intervals $a_i = x_i || x_{i+1}$ (i.e., insert an interval $x_i || x_{i+1}$), instead of explicit elements x_i and then show in the proof of Theorem 1 how, by supporting operations for intervals, we can support operations for distinct elements. We define all four algorithms as described in Section 2.3:

1. $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^t)$. The secret sk is the trapdoor $s \in \mathbb{Z}_p^*$, which is picked randomly. Procedure $\mathsf{genkey}(1^t)$ also calls $\mathcal{G}(1^t, t)$ from Definition 4 and outputs the public key pk which is the description of the groups $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$, the admissible multilinear form $e : \mathbb{G}^t \rightarrow \mathbb{G}_\mathsf{T}$ and the elements $g, g^s, \ldots, g^{s^q} \in \mathbb{G}$. It also outputs a description of a collision-resistant hash function h that takes three inputs and outputs a hash of t bits. We recall that the number of the inputs of the admissible multilinear form that we are using is t, equal to the security parameter.
2. $\{\mathsf{auth}(D_0), d_0\} \leftarrow \mathsf{setup}(D_0, \mathsf{pk})$. The digest d_0 of the authenticated data structure is defined as the tuple $\{\mathsf{acc}(\mathcal{A}), \mathsf{hash}(\mathcal{A})\}$. It is

$$\mathsf{acc}(\mathcal{A}) = e(g, g, \ldots, g)^{(a_1+s)(a_2+s)\ldots(a_{n-1}+s)},$$

while $\mathsf{hash}(\mathcal{A})$ is computed as the digest of the well-known Merkle tree (description follows). Both are stored locally by the client and used for verification and updates. Let now T be the red-black tree built on top of the intervals $a_i = x_i || x_{i+1}$ for $i = 1, \ldots, n-1$ and where $x_1 < x_2 < \ldots < x_n$. Let $v_1, v_2, \ldots, v_{n-1}$ be the leaves of the tree, storing the intervals $a_1, a_2, \ldots, a_{n-1}$ respectively. We define the *label* and the *hash* of v_i as

$$\mathsf{label}(v_i) = g^{a_i+s} \in \mathbb{G}, \tag{4}$$

$$\mathsf{hash}(v_i) = h(\mathsf{null}, \mathsf{label}(v_i), \mathsf{null}), \tag{5}$$

where h is the collision-resistant hash function. Let now v_A be the internal node of T that is the root of the subtree of T that contains the elements of some $A \subseteq \mathcal{A}$

(i.e., from v_A you can reach elements in A by following the two downward paths). Then

$$\mathsf{label}(v_A) = g^{\prod_{a \in A}(a+s)} \in \mathbb{G}\,, \tag{6}$$

$$\mathsf{hash}(v_A) = h(\mathsf{hash}(\mathsf{lchild}(v_A)), \mathsf{label}(v_A), \mathsf{hash}(\mathsf{rchild}(v_A)))\,, \tag{7}$$

where $\mathsf{lchild}(v)$ and $\mathsf{rchild}(v)$ are the left and the right child of node v in the tree. Note that all the labels of the internal nodes of T can be computed in polynomial time in n without the use of the trapdoor s, and only by using the public key. Also, as we will see later, in sake of maintaining constant verification and communication complexity, the hashes $\mathsf{hash}(.)$ are used only for updates. Finally, $\mathsf{hash}(\mathcal{A})$ is defined to be the $\mathsf{hash}(.)$ value of the root of the tree T, recursively computed by means of the above equations.

3. $\{\Pi(o), \alpha(o), D_{h+1}, \mathsf{auth}(D_{h+1}), d_{h+1}\} \leftarrow \mathsf{operate}(o, D_h, \mathsf{auth}(D_h), d_h)$.
   (a) **Verification proof case.** Suppose o is a query for the interval a_j, stored at node v_j. Let $\pi(a_j) = v_j, v_{j1}, v_{j2}, \ldots, v_{jl}$ be the path of T from node v_j that refers to interval a_j to the child of the root of T, where $l = O(\log n)$. The verification proof $\Pi(a_j)$ is defined as

$$e(\mathsf{label}(\mathsf{sib}(v_j)), \mathsf{label}(\mathsf{sib}(v_{j1})), \ldots, \mathsf{label}(\mathsf{sib}(v_{jl})), g, \ldots, g) \in \mathbb{G}_\mathsf{T}\,, \tag{8}$$

   where $\mathsf{sib}(v)$ defines the sibling of node v in the red-black tree T, and function $\mathsf{label}()$ is defined in Equations 4 and 6. The answer $\alpha(o)$ will be the element a_j. Note now that

$$\Pi(a_j) = \mathcal{W}_{a_j} = e(g, g, \ldots, g)^{\prod_{a \in \mathcal{A}: a \neq a_j}(a+s)} \in \mathbb{G}_\mathsf{T}\,,$$

   as required by Equation 3. Moreover, it is computable in time $O(\log n)$ since we have to collect $O(\log n)$ labels and feed them as input in the admissible multilinear form $e()$. Note that the remaining inputs of the admissible multilinear form (i.e., $t - \log n$) are set equal to g, the generator of group $\mathbb{G}$. Moreover the size of the witness is $O(1)$ (only one group element of $\mathbb{G}_\mathsf{T}$). The presented method is the first one to construct a witness for an accumulator in logarithmic time. The straightforward method takes linear time;
   (b) **Consistency proof case.** Suppose o is an update (either an insertion or a deletion) of an interval a_j. In accordance with path $\pi(a_j)$ in the verification proof, whenever there is an update, let $\pi(a_j)$ be the portion of the red-black tree (that also contains structure) that is accessed (and eventually changes) due to the update of the interval a_j. Denote with $\Pi_1(a_j)$ the set of those labels $\mathsf{label}(v)$ such that $v \in \pi(a_j)$ and with $\Pi_2(a_j)$ the set of those hashes $\mathsf{hash}(v)$ such that $v \in \pi(a_j)$. The consistency proof is denoted with $\Pi(a_j) = \{\Pi_1(a_j), \Pi_2(a_j)\}$. We finally note that the time for the construction of the consistency proof is $O(\log n)$ and its size is also $O(\log n)$, since a red-black tree insertion or deletion takes $O(\log n)$ time in the worst case.

   We note here that, although this algorithm outputs D_{h+1}, it does not output $\mathsf{auth}(D_{h+1})$. The *updated* authenticated data structure (i.e., the new $O(\log n)$-sized portion of it) will be sent by the client to the server during the execution of the $\mathsf{verify}()$ algorithm. This is done primarily for efficiency reasons, since $\mathsf{auth}(D_{h+1})$ could have been computed by the server anyways, by using the public key, but in a less efficient way[5].

4. $\{\mathsf{accept}, \mathsf{reject}, d_{h+1}\} \leftarrow \mathsf{verify}(o, \alpha(o), \Pi(o), d_h, \mathsf{sk})$.
   (a) **Verification proof case.** If $\Pi(o)$ is a verification proof, i.e., $\Pi(o) = \Pi(a_j)$ (Equation 8), then clearly the procedure outputs "accept" (see Lemma 1) if and only if
   $$\Pi(a_j)^{a_j+s} = \mathsf{acc}(\mathcal{A}),$$
   else it outputs "reject". The digest of the structure remains the same. Note that the client does not use the multilinear form for verification since he has access to the trapdoor s. The verification involves only one exponentiation in the group $\mathbb{G}_\mathsf{T}$ and therefore takes time $O(1)$.
   (b) **Consistency proof case.** If $\Pi(o)$ is a consistency proof, then the client has to update both $\mathsf{acc}(\mathcal{A})$ and $\mathsf{hash}(\mathcal{A})$ to $\mathsf{acc}(\mathcal{A}')$ and $\mathsf{hash}(\mathcal{A}')$ respectively. The digest $\mathsf{acc}(\mathcal{A})$ is updated in constant time by setting $\mathsf{acc}(\mathcal{A}') = \mathsf{acc}(\mathcal{A})^{a_j+s}$ if a_j is inserted and $\mathsf{acc}(\mathcal{A}') = \mathsf{acc}(\mathcal{A})^{(a_j+s)^{-1}}$ if a_j is deleted. For the update of $\mathsf{hash}(\mathcal{A})$ to $\mathsf{hash}(\mathcal{A}')$, the client performs the following step:
      i. Initially he verifies the correctness of the labels in $\Pi_1(a_j)$ by recomputing the digest $\mathsf{hash}(\mathcal{A})$ by means of elements in $\Pi_1(a_j)$ and $\Pi_2(a_j)$. If this computation succeeds (Merkle tree verification) then the client is assured with probability $1 - \mathsf{neg}(t)$, due to collision resistance, that the labels in $\Pi_1(a_j)$ belong to the correct portion of red-black tree T before the certain update, i.e., they belong to the portion that is accessed due to this update;

   If the above test succeeds then the procedure outputs "accept", else it outputs "reject". If it accepts, with probability $1 - \mathsf{neg}(t)$, $\Pi_1(a_j)$ is the set of labels that is accessed during the update and needs to be updated. Each label in this set can be updated in constant time, since the trapdoor s is known by the client. While the labels are updated, the new hashes are also computed and finally $\mathsf{hash}(\mathcal{A}')$ is updated. Basically the client performs a red-black tree insertion/deletion locally, doing the necessary rotations, updating at the same time the information $\mathsf{label}()$ and $\mathsf{hash}()$. We conclude that $\mathsf{hash}(\mathcal{A}')$ is the updated digest with probability $1 - \mathsf{neg}(t)$, since it is a function of $\Pi_1(a_j)$. After the procedure finishes, the client sends the updated labels, hashes and the updated digests ($\mathsf{acc}(\mathcal{A}')$ and $\mathsf{hash}(\mathcal{A}')$) back to the server (i.e., the new portion of the authenticated data structure $\mathsf{auth}(D_{h+1})$): The server therefore only has to perform a logarithmic time writing of the new information to the data structure. Note that processing the consistency proof takes logarithmic time. Similar solutions for two-party authenticated dictionaries using only collision-resistant hashing have been explored in the literature (e.g., [33]).

Observations. Before presenting the main result of this section we make an important observation. The hashing structure on top of the red-black tree (i.e., the additional $\mathsf{hash}()$ label in the construction) is used only for efficiency reasons and not for security

[5] It is an open problem—even with the use of multilinear forms—to construct an optimal authenticated dictionary that avoids this interaction.

reasons. Namely, if we had not used the hashing structure, the untrusted server, having access to the public information $g^s, g^{s^2}, \ldots, g^{s^q}$, could update all the labels $\mathsf{label}(v)$ for all affected nodes v. However, this would take $O(n \log n)$ time ($O(n)$ time for each one of the $O(\log n)$ nodes of the path) by using well-known methods by means of Vieta's formulas (see for example [36]). In this case however, the update time at the client would be constant, since all needed would be one exponentiation for the update of $\mathsf{acc}(\mathcal{A})$.

However the labels of the affected nodes v, can easily be updated (in $O(1)$ time per node) by knowing the trapdoor s, something that only the client has access to. Therefore, with the hashing structure, we authenticate the "affected paths" so that the client can verify which labels are affected. Then the client efficiently performs the updates locally, and sends back the new values to be used in the future. This does not violate security since the new information provided to the server by the client is computable by the server in polynomial time anyways.

3.3 Main Results

We now present the main results of this section.

Theorem 1. *Assume the existence of an admissible multilinear form generator and collision-resistant hash functions. Then there exists an optimal two-party authenticated dictionary storing n elements with the following properties:*

1. *It is secure under the q-strong multilinear Diffie-Hellman assumption and according to Definition 6;*
2. *The size of the verification proof is $O(1)$ both for a (non-)membership query and for a range search query of ℓ elements;*[6]
3. *The query time at the server is $O(\log n)$ for a (non-)membership query and $O(\log n + \ell)$ for a range search query of ℓ elements;*
4. *The verification time at the client is $O(1)$ for a (non-)membership query and $O(\ell)$ for a range search query of ℓ elements;*
5. *The size of the consistency proof is $O(\log n)$;*
6. *The update time at the server and at the client is $O(\log n)$;*
7. *The server uses $O(n)$ space;*
8. *The client uses $O(1)$ space.*

Proof. **(Security)** (1) We prove security according to Definition 6. Given the security parameter t, the client runs algorithm $\{\mathsf{sk}, \mathsf{pk}\} \leftarrow \mathsf{genkey}(1^t)$. Then the adversary picks a data structure D_0, runs $\{\mathsf{auth}(D_0), d_0\} \leftarrow \mathsf{setup}(D_0, \mathsf{pk})$ and produces an empty authenticated data structure. The adversary chooses a polynomial (in t) number of updates (say h) to the data structure and turns it into a data structure D_h, with d_h being the tuple of digests $\mathsf{acc}(\mathcal{A})$ and $\mathsf{hash}(\mathcal{A})$, as defined in the description of the algorithms and where $\mathcal{A}$ is the current set of elements. Let now $\nu(t)$ be a function that is $\mathsf{neg}(t)$. Since the client has accepted all the updates (see security definition), this means, by construction, that $d_h = \{\mathsf{hash}(\mathcal{A}), \mathsf{acc}(\mathcal{A})\}$ is the correct digest of the data structure. Specifically, $\mathsf{hash}(\mathcal{A})$ is correct with probability $1 - \nu(t)$ (by collision resistance) and $\mathsf{acc}(\mathcal{A})$

[6] Note that *super-efficiency* [22] is achieved for range search queries besides optimality.

is correct with probability 1 (since $\mathsf{acc}(\mathcal{A})$ is updated with only one exponentiation at every update). Therefore, if D_h is the data structure after the update phase and d_h the tuple of the updated digests, we have $\Pr[\mathsf{digest}(D_h) = d_h] = (1-\nu(t))\times 1 = 1-\nu(t)$. Therefore, the probability of Definition 6 is written

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

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

$$\Pr\left[\begin{array}{l}\{a, \Pi(a), x\} \leftarrow \mathsf{Adv}(1^t, \mathsf{pk});\\ \Pi(a)^{a+s} = e(g, g, \ldots, g)^{(a_1+s)(a_2+s)\ldots(a_n+s)};\\ a \notin \{a_1, a_2, \ldots, a_n\}.\end{array}\right] \times (1-\nu(t)).$$

The first term, by Lemma 1 is $\mathsf{neg}(t)$. Therefore the whole probability is $\mathsf{neg}(t) \times (1 - \nu(t))$, which is $\mathsf{neg}(t)$.

(**Complexity**) (2-4) First of all we show equivalence of (non-)membership proofs of elements with membership proofs of intervals, that are actually stored in the authenticated data structure: A (non-)membership proof for element x is equivalent with a membership proof of the interval $a_i = x_i||x_{i+1}$ such that $x_i \leq x \leq x_{i+1}$ (note that for non-membership proofs it is $x_i < x < x_{i+1}$). Additionally, by Equation 8, we have that a verification proof for an interval is only one element of $\mathbb{G}_\mathsf{T}$ and is computed by applying the admissible multilinear form $e()$. Therefore the size of the verification proof for an element is $O(1)$ and the time to compute it is $O(\log n)$, since $O(\log n)$ elements of $\mathbb{G}$ along the red-black tree path have to be collected and then be fed into the admissible multilinear form $e(.)$. The time to verify involves one exponentiation (see $\mathsf{verify}()$ algorithm) and therefore is $O(1)$. A range proof of ℓ elements consists of one membership proof of an interval (instead of one element in the exponent, we omit all the elements of the respective interval). Therefore its size is $O(1)$, it can be computed in $O(\log n + \ell)$ time and it can be verified in $O(\ell)$ time.

(5-6) We now show equivalence of elements updates with intervals updates. Suppose the client wants to insert x. Firstly the client verifies the non-membership of x by verifying the membership of the interval $a_i = x_i||x_{i+1}$ such that $x_i < x < x_{i+1}$. After interval a_i has been verified the client issues the following updates with this order: $\mathsf{delete}(a_i)$, $\mathsf{insert}(x_i||x)$, $\mathsf{insert}(x||x_{i+1})$. For deletion of element x, the client first verifies the membership of intervals $x_i||x$ and $x||x_{i+1}$ and then issues the following updates in this order: $\mathsf{delete}(x_i||x)$, $\mathsf{delete}(x||x_{i+1})$, $\mathsf{insert}(x_i||x_{i+1})$. Since the cost of these individual updates is $O(\log n)$, we conclude that any update will cost $O(\log n)$ in the worst case.

(7-8) Finally, the extra space needed to store the labels and the hashes on the red-black tree is $O(n)$, while the space needed by the client is $O(1)$, since the client only needs to store $\mathsf{acc}(\mathcal{A})$, $\mathsf{hash}(\mathcal{A})$ and the secret trapdoor, s.

(**Optimality**) The optimal plain dictionary data structure is the red-black tree, achieving worst-case complexities $O(\log n)$ [15] and answers of constant size.

Therefore our authenticated data structure satisfies Definition 7 since it has logarithmic complexities in the worst case, constant verification proof size and constant verification time. □

We now present the final result of our paper that relates optimality of an authenticated dictionary with the existence of an admissible multilinear form generator.

Theorem 2. *If optimal authenticated dictionaries do not exist, then admissible multilinear form generators do not exist either.*

Proof. Let's assume this is not the case and admissible multilinear form generators do exist in the absence (through a generic lower bound proof—see for example a similar result for memory checking in [18]) of optimal authenticated dictionaries. This is a contradiction since we can use the construction of Theorem 1—which will give us a secure construction since we can use an admissible multilinear form generator for $k = t$—to derive an optimal authenticated dictionary. □

Finally we need to make the following important observation. Theorem 2 does not exclude the existence of some instance of a multilinear form, even in the absence of optimal authenticated dictionaries (say for example an instance of a multilinear form for $k = 5$). The result holds for all admissible k-multilinear forms, i.e., for the existence of an admissible multilinear form generator, as defined in Definition 4.

4 Conclusions

In this paper, we have presented the first *optimal* authenticated dictionary with constant-size verification proof, constant-time verification, and logarithmic query/update costs. Its design is based on multilinear forms, a recently-proposed cryptographic primitive [10] whose construction remains an open problem to date.

However, since multilinear forms are not known to exist yet, this work can be viewed from a different angle: if one could prove that optimal authenticated dictionaries cannot exist in the computational model, irrespectively of cryptographic primitives, then our result would imply that cryptographically interesting multilinear form generators cannot exist as well (i.e., it can be viewed as a reduction). Thus, we provide an alternative avenue towards proving the nonexistence of multilinear form generators in the context of general lower bounds for authenticated data structures [40] and for memory checking [18].

Acknowledgments

This research was supported in part by the U.S. National Science Foundation under grants CNS–1012060 and CNS–1012798 and by the Center for Geometric Computing and the Kanellakis Fellowship at Brown University. We thank Alice Silverberg for many useful comments and discussions as well as the conference reviewers for providing us with very insightful feedback. This work was performed while the third author was with Boston University. The views in this paper do not necessarily reflect the views of the sponsors.

References

1. Applebaum, B., Ishai, Y., Kushilevitz, E.: From secrecy to soundness: Efficient verification via secure computation. In: Proc. Int. Colloquium on Automata, Languages and Programming (ICALP), pp. 152–163 (2010)
2. Atallah, M.J., Cho, Y., Kundu, A.: Efficient data authentication in an environment of untrusted third-party distributors. In: Proc. Int. Conference on Data Engineering (ICDE), pp. 696–704 (2008)
3. Au, M.H., Tsang, P.P., Susilo, W., Mu, Y.: Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 295–308. Springer, Heidelberg (2009)
4. Baric, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)
5. Benaloh, J., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1993)
6. Blum, M., Evans, W.S., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica 12(2/3), 225–244 (1994)
7. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptology 21(2), 149–177 (2008)
8. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
9. Boneh, D., Mironov, I., Shoup, V.: A secure signature scheme from bilinear maps. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 98–110. Springer, Heidelberg (2003)
10. Boneh, D., Silverberg, A.: Applications of multilinear forms to cryptography. Contemporary Mathematics 324(1), 71–90 (2003)
11. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: Proc. Public Key Cryptography (PKC), pp. 481–500 (2009)
12. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
13. Cheon, J.H., Lee, D.H.: A note on self-bilinear maps. Korean Mathematical Society 46(2), 303–309 (2009)
14. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)
15. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
16. Damgård, I., Triandopoulos, N.: Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538 (2008)
17. Devanbu, P., Gertz, M., Kwong, A., Martel, C., Nuckolls, G., Stubblebine, S.: Flexible authentication of XML documents. Journal of Computer Security 6, 841–864 (2004)
18. Dwork, C., Naor, M., Rothblum, G.N., Vaikuntanathan, V.: How efficient can memory checking be? In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 503–520. Springer, Heidelberg (2009)

19. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
20. Gennaro, R., Halevi, S., Rabin, T.: Secure hash-and-sign signatures without the random oracle. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 123–139. Springer, Heidelberg (1999)
21. Goodrich, M.T., Tamassia, R., Hasic, J.: An efficient dynamic and distributed cryptographic accumulator. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 372–388. Springer, Heidelberg (2002)
22. Goodrich, M.T., Tamassia, R., Triandopoulos, N.: Super-efficient verification of dynamic outsourced databases. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 407–424. Springer, Heidelberg (2008)
23. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005)
24. Joux, A.: A one-round protocol for tripartite Diffie-Hellman. J. Cryptology 17(4), 263–276 (2004)
25. Lee, H.-M., Ha, K.J., Ku, K.-M.: ID-based multi-party authenticated key agreement protocols from multilinear forms. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 104–117. Springer, Heidelberg (2005)
26. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 253–269. Springer, Heidelberg (2007)
27. Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., Stubblebine, S.G.: A general model for authenticated data structures. Algorithmica 39(1), 21–41 (2004)
28. Menezes, A., Vanstone, S., Okamoto, T.: Reducing elliptic curve logarithms to logarithms in a finite field. In: Proc. Symposium on Theory of Computing (STOC), pp. 80–89 (1991)
29. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1989)
30. Naor, M., Nissim, K.: Certificate revocation and certificate update. In: Proc. USENIX Security Symposium (USENIX), pp. 217–228 (1998)
31. Naor, M., Rothblum, G.N.: The complexity of online memory checking. J. ACM 56(1) (2009)
32. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 275–292. Springer, Heidelberg (2005)
33. Papamanthou, C., Tamassia, R.: Time and space efficient algorithms for two-party authenticated data structures. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 1–15. Springer, Heidelberg (2007)
34. Papamanthou, C., Tamassia, R.: Update-optimal authenticated structures based on lattices. Cryptology ePrint Archive, Report 2010/128 (2010)
35. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables. In: Proc. ACM Conference on Computer and Communications Security (CCS), pp. 437–448 (2008)
36. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Cryptographic accumulators for authenticated hash tables. Cryptology ePrint Archive, Report 2009/625 (2009)
37. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal authentication of set operations on dynamic sets. Cryptology ePrint Archive, Report 2010/455 (2010)
38. Sander, T., Ta-Shma, A., Yung, M.: Blind, auditable membership proofs. In: Proc. Financial Cryptography (FC), pp. 53–71 (2001)

39. Tamassia, R.: Authenticated data structures. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 2–5. Springer, Heidelberg (2003)
40. Tamassia, R., Triandopoulos, N.: Computational bounds on hierarchical data processing with applications to information security. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 153–165. Springer, Heidelberg (2005)
41. Tamassia, R., Triandopoulos, N.: Efficient content authentication in peer-to-peer networks. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 354–372. Springer, Heidelberg (2007)
42. Wang, P., Wang, H., Pieprzyk, J.: A new dynamic accumulator for batch updates. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 98–112. Springer, Heidelberg (2007)
43. Yuan, H., Atallah, M.J.: Efficient distributed third-party data authentication for tree hierarchies. In: Proc. Int. Conference on Distributed Computing Systems (ICDCS), pp. 184–193 (2008)