Update-Optimal Authenticated Structures Based on Lattices

Charalampos Papamanthou* Roberto Tamassia†

July 7, 2010

Abstract

We study the problem of authenticating a *dynamic table* with n entries in the authenticated data structures model, which is related to memory checking. We present the first dynamic authenticated table that is *update-optimal*, using a *lattice-based* construction. In particular, the update time is $O(1)$, improving in this way the "a priori" $O(\log n)$ update bounds for previous constructions, such as the Merkle tree. Moreover, the space used by our data structure is $O(n)$ and logarithmic bounds hold for the other complexity measures, such as *proof size*. To achieve this result, we exploit the *linearity* of lattice-based hash functions and show how the security of lattice-based digests can be guaranteed under updates. This is the first construction achieving constant update bounds without causing other time complexities to increase beyond logarithmic. All previous solutions enjoying constant update complexity have $\Omega(n^\epsilon)$ proof or query bounds. As an application of our lattice-based authenticated table, we provide the first construction of an authenticated Bloom filter, an update-intensive data structure that falls into our model.

*Department of Computer Science, Brown University. Email: cpap@cs.brown.edu.
†Department of Computer Science, Brown University. Email: rt@cs.brown.edu.

1 Introduction

Increasing interest in online data storage and processing has recently led to the establishment of the field of *cloud computing* [22]. Files can be outsourced to service providers that offer huge capacity and fast network connections (e.g., Amazon S3) as a means of mitigating maintenance and storage costs. In such settings, the ability to check the integrity of remotely stored data is an important security property, or otherwise a faulty or malicious server can lose or tamper with the client's data (e.g., deleting or modifying a file). In order to efficiently check the integrity of outsourced data, the model of *authenticated data structures* (see, e.g., [26, 40]) has been developed, which is related to memory checking [7]. In an authenticated data structure, untrusted servers answer queries on a data structure on behalf of a trusted source and provide a proof of validity of each answer to the user.

The authenticated data structures model involves three participating entities. The owner of the data, called *source*, outsources its data to one or more untrusted parties, called *servers*. *Clients* issue queries to the servers and wish to verify the answers received by the servers, based only on the trust they have in the source. This trust is conveyed through a time-stamped signature on a *digest* of the data structure, i.e., a collision resistant succinct representation of the data structure (e.g., the root hash of a Merkle tree). Moreover, updates are issued by the source and are performed both by the source and by the server, since the data structure is replicated at both those entities (see *source update time* and *server update time* in Table 1). Variations of this model include the *two-party model* [32], where the source does both the updates and the queries (and the respective verifications), and which is closely related to memory checking [7].

In the study of authenticated data structures, apart from achieving provable security (i.e., a server cannot—in polynomial time—produce a valid proof for a false answer) under a well-accepted assumption (e.g., strong RSA assumption), it is important to achieve efficient asymptotic bounds for the relevant complexity measures, listed in the first column of Table 1 (also explained in more detail at the end of Section 2). The cryptographic primitives used by an authenticated data structure can have a significant impact on its efficiency. Towards this goal, we develop the first efficient authenticated data structure based on *lattices*, a mathematical tool that was shown to have many applications in cryptography after Ajtai's seminal result [1].

Our construction is the first to achieve constant complexity bounds for the source update time and update information size while keeping all the remaining complexity bounds logarithmic.[1] For example, although in the method of [4] updates are performed in $O(1)$ time, the size of the proof is $O(n)$, i.e., to prove the membership of a given element in the table, all the other elements have to be communicated. Similarly, while the data structure of [33] achieves constant update (or query) bounds using cryptographic accumulators, either the query (or the update) time is $O(n^{\epsilon})$. A similar trade-off was also observed in [15]. Therefore, if one wishes to avoid $O(n^{\epsilon})$ complexities, one has to resort to the widely used Merkle tree [27] and related hierarchical hashing structures [7, 20]. However, all solutions based on hierarchical hashing use "generic collision resistance"[2] as a hardness assumption (see Table 1), thus inherently requiring logarithmic complexity measures [41].

In this paper, we combine the simplicity of a Merkle tree (a binary tree is used in our construction) with the "linearity" of lattice-based hash functions [18] towards constructing an authenticated structure with constant update time, while keeping other complexity measures logarithmic. Moreover, we base the security of our construction on a well-accepted cryptographic assumption (the hardness of the GAPSVP_{γ} problem in lattices), which has its own significance given recent attacks on collision-resistant functions such as MD-5 [39], a function widely used in practical deployments of authenticated structures.

In this work, we use a model similar to that of memory checking [7]. The structure we wish to authen-

[1]All asymptotic complexities presented in this paper (e.g., Table 1) refer to the size of the structure n, and not to the security parameter k. The security parameter in this line of research is considered to be a constant in relation to n, i.e., $k = O(1)$.

[2]We call *generic collision resistant functions* (Generic CR in Table 1) those functions that are believed to be collision resistant in practice (e.g., SHA-256).

Table 1: Asymptotic complexity of previous solutions and of our work for the problem of authenticating a dynamic table of size n. Parameter $0 < \epsilon < 1$ is a constant that can be arbitrarily chosen. Also, "D. Log" stands for "Discrete Logarithm", "Generic CR" stands for "Generic Collision Resistance", GAPSVP_γ is the gap version of the shortest vector problem in lattices (see Definition 1), where $\gamma = 14\pi nk\sqrt{k}$ and where k is the security parameter. In all constructions the space at the client is $O(1)$, both source space and server space are $O(n)$. Note that the $O(.)$ notation refers to the size of the structure n and not to the security parameter k, which is taken to be a constant in relation to n.

	[7, 29]	**[4]**	**[31]**	**[10, 38]**	**[19]**	**[33]**	**this work**
source update	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$	$O(n^\epsilon)$	$O(1)$	$O(1)$
server update	$O(\log n)$	$O(1)$	$O(n)$	$O(n \log n)$	$O(n^\epsilon)$	$O(1)$	$O(\log n)$
server query	$O(\log n)$	$O(n)$	$O(1)$	$O(1)$	$O(n^\epsilon)$	$O(n^\epsilon)$	$O(\log n)$
verification	$O(\log n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$
proof size	$O(\log n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$
update info.	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(n^\epsilon)$	$O(1)$	$O(1)$
assumption	Generic CR	D. Log	Strong DH	Strong RSA			GAPSVP_γ

ticate is a *dynamic table* of size n, accessed through indices $1, \dots, n$. Operations consist of reading and writing table entries. We assume that each entry of the table can take one of C different values, where C is a constant, i.e., C is not dependent on n (although it could be $\mathsf{poly}(k)$, where k is the security parameter). E.g., for $C = 2$, we have a boolean table.

Related work. Lattice-based cryptography began with Ajtai's construction of a one-way hash function based on hard lattice problems [1], which was shown to be collision resistant by Goldreich [18] and was further generalized in [28]. Other hash functions based on lattices with reduced public key size are due to Micciancio [25] and Peikert [34]. Recently, *trapdoor* functions based on lattices were introduced in [17].

Several authenticated data structures based on cryptographic hashing have been developed, including Merkle trees [7, 27, 29] and authenticated skip lists [20]. Lower bounds for hashing-based authenticated data structures are given in [41] and, in the context of memory checking, in [15, 30]. Authenticated data structures using other cryptographic primitives, such as *one-way accumulators* [2, 5, 10] are presented in [19], achieving $O(n^\epsilon)$ bounds. Authenticated hash tables proved secure under the strong RSA assumption or the strong Diffie-Hellman assumption are presented in [33], where, however, either the update time or the query time is $O(n^\epsilon)$. Finally, we note that constant complexities, but in the *parallel model of computation*, are achieved in [21]. We observe that all of the above constructions belong to one of the following two categories: either (1) they have logarithmic source update complexity, with all the other complexity measures being also logarithmic, e.g., [7, 20, 29]; or (2) they have sublogarithmic source update complexity (e.g., constant) but at least one of the other complexities is $\Omega(n^\epsilon)$, e.g., [4, 31, 33]. A summary and comparison of our work with previous constructions in the literature can be found in Table 1.

Contributions. Our main contribution is the construction of an update-optimal authenticated data structure for an n-index dynamic table based on lattices (Theorem 6). We note that, to our knowledge, our work provides the first construction with $O(1)$ source update time, $O(1)$ update information size, and logarithmic complexity for the remaining performance measures. We are able to achieve these bounds by exploiting the *linearity* of lattice-based hash functions, which other primitives such as *generic collision resistant functions* (used in [7, 20, 29]) and *exponentiation functions* (used in [4, 31, 33]) lack. We extend our solution to the two-party model, which is related to memory checking (Theorem 7), and we also give the first construction of an *authenticated Bloom Filter* (Theorem 8). We note here that however, achieving an $O(1)$ bound for updates comes at a *practical* cost: The constants involved (due to the use of lattices) are rather high, making our result interesting mainly from a theoretical point of view (see Table 2 in the Appendix).

Overview of our solution. Our solution can be seen as a generalization of the Merkle tree and related hierarchical hashing constructions [7, 20, 29]. It exploits a special feature of lattice-based hash functions, i.e., their linearity, in the following sense. Collision-resistant hash functions used in hierarchical hashing (e.g., MD-5 or SHA-256) have been previously viewed as a black box for computing the digest of a node from the digests of its children. Instead, we employ a function that satisfies a powerful algebraic property in addition to maintaining the interface of this black box. Namely, the digest of a node v can be expressed as the "sum" of well-defined functions applied to data stored at the leaves of the subtree rooted at v (see Figure 1 in the Appendix). The systematic application of this property—which lies at the crux of our construction—has many algorithmic and security implications and eventually enables constant-time updates.

2 Preliminaries

We start with some preliminary notions that are important in our construction. In the following, we use k to denote the security parameter (we do not use n as is usually done in lattice-related bibliography) and n to denote the size of the table to be authenticated (i.e., the dimension of the problem). We use upper case bold letters to denote matrices, e.g., $\mathbf{B}$, lower case bold letters to denote vectors, e.g., $\mathbf{b}$, and lower case italic letters to denote scalars. Finally, for a vector $\mathbf{x} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_k]^T$, $\|\mathbf{x}\|$ denotes the Euclidean norm of $\mathbf{x}$.

Lattices. Given the security parameter k, a full-rank k-dimensional lattice is the infinite-sized set of all vectors produced as the integer combinations $\{\sum_{i=1}^{k} x_i \mathbf{b}_i \;:\; x_i \in \mathbb{Z},\; 1 \le i \le k\}$, where $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k\}$ is the *basis* of the lattice and $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k$ are linearly independent, all belonging to $\mathbb{R}^k$. We denote the lattice produced by $\mathbf{B}$ (i.e., the set of vectors) with $L(\mathbf{B})$.

A well-known difficult problem in lattices is the approximation within a polynomial factor of the *shortest* vector in a lattice (SVP problem). Namely, given a lattice $L(\mathbf{B})$ produced by a basis $\mathbf{B}$, approximate up to a polynomial factor in k the shortest (in an Euclidean sense) vector in $L(\mathbf{B})$, the length of which we denote with $\lambda(\mathbf{B})$. A similar problem in lattices is the "gap" version of the shortest vector problem (GAPSVP_γ), the difficulty of which is going to be useful in our context.

Definition 1 (Problem GAPSVP_γ) *Let k be the security parameter. An input to GAPSVP_γ is a k-dimensional lattice basis $\mathbf{B}$ and a number d. In* YES *inputs $\lambda(\mathbf{B}) \le d$ and in* NO *inputs $\lambda(\mathbf{B}) > \gamma \times d$, where $\gamma \ge 1$.*

We note that, for exponential values of γ, i.e, $\gamma = 2^{O(k)}$, one can use the LLL algorithm [24] and decide the above problem in polynomial time. The difficult version of the problem arises for polynomial γ, for which no efficient algorithm is known to date, even for factors slightly smaller than exponential [35], i.e., very big polynomials. Moreover, for polynomial factors, there is no proof that this problem is NP-hard[3], which makes the polynomial approximation cryptographically interesting as well.

Reductions. After Ajtai's seminal work [1] where an one-way function based on hard lattices problem is presented, Goldreich et al. [18] presented a variation of the function, providing at the same time collision resistance. Based on this collision resistant hash function, Micciancio [28] described a generalized version of it, a modification of which we are using in our construction. The security of the hash function is based on the difficulty of the *small integer solution* problem (SIS):

Definition 2 (Problem $\mathsf{SIS}_{p,m,\beta}$) *Let k be the security parameter. Given an integer p, a matrix $\mathbf{F} \in \mathbb{Z}_p^{k\times m}$ and a real β, find a non-zero integer vector $\mathbf{z} \in \mathbb{Z}^m \backslash \{\mathbf{0}\}$ such that $\mathbf{F}\mathbf{z} = \mathbf{0} \mod p$ and $\|\mathbf{z}\| \le \beta$.*

Note that at least one solution to the above problem exists when $\beta \ge \sqrt{m}p^{k/m}$ and $m > k$ [28]. Moreover, if $p \ge 4\sqrt{m}k^{1.5}\beta$, we will see that such a solution is difficult to find. We continue with the definition of SIS', where the solution vector is required to have at least one odd coordinate:

[3]In specific, as outlined in [35], the current state of knowledge indicates that for factors beyond $\sqrt{k/\log k}$, it is unlikely that this problem is NP-hard and no efficient algorithm is known to date.

Definition 3 (Problem $\mathsf{SIS}'_{p,m,\beta}$) *Let k be the security parameter. Given an integer p, a matrix $\mathbf{F} \in \mathbb{Z}_p^{k\times m}$ and a real β, find an integer vector $\mathbf{z} \in \mathbb{Z}^m \backslash 2\mathbb{Z}^m$ such that $\mathbf{Fz} = \mathbf{0} \mod p$ and $\|\mathbf{z}\| \leq \beta$.*

Micciancio [28] showed that if p is odd, there is a polynomial time reduction from $\mathsf{SIS}'_{p,m,\beta}$ to $\mathsf{SIS}_{p,m,\beta}$:

Lemma 1 (Reduction from $\mathsf{SIS}'_{p,m,\beta}$ to $\mathsf{SIS}_{p,m,\beta}$ [28]) *For any odd integer $p \in 2\mathbb{Z}+1$, and SIS' instance $I = (p, \mathbf{F}, \beta)$, if I has a solution as an instance of SIS, then it also has a solution as an instance of SIS'. Moreover, there is a polynomial time algorithm that on input a solution to a SIS instance I, outputs a solution to the same SIS' instance I.*

As proved by Micciancio [28], under a certain choice of parameters, GAPSVP_γ can be reduced to SIS' (this can be derived as a combination of Lemma 5.22 and Theorem 5.23 of [28]):

Lemma 2 (Reduction from GAPSVP_γ to $\mathsf{SIS}'_{p,m,\beta}$ [28]) *For any polynomially bounded $\beta, m, p = k^{O(1)}$, with $p \geq 4\sqrt{m}k^{1.5}\beta$ and $\gamma = 14\pi\sqrt{k}\beta$, there is a probabilistic polynomial time reduction from solving GAPSVP_γ in the worst case to solving $\mathsf{SIS}'_{p,m,\beta}$ on the average with non-negligible probability.*

A direct application of Lemma 1 and Lemma 2 gives the following result.

Theorem 1 *Let $p = k^{O(1)}$ be an odd positive integer. For any polynomially bounded $\beta, m = k^{O(1)}$, with $p \geq 4\sqrt{m}k^{1.5}\beta$ and $\gamma = 14\pi\sqrt{k}\beta$, there is a probabilistic polynomial time reduction from solving GAPSVP_γ in the worst case to solving $\mathsf{SIS}_{p,m,\beta}$ on the average with non-negligible probability.*

Theorem 1 states that if there is an algorithm that solves an average instance of $\mathsf{SIS}_{p,m,\beta}$ ("average" refers to the fact that the matrix $\mathbf{F} \in \mathbb{Z}_p^{k\times m}$ is chosen uniformly at random), for an odd p, $p \geq 4\sqrt{m}k^{1.5}\beta$ and $\gamma = 14\pi\sqrt{k}\beta$, then, this algorithm can be used to produce a solution to any instance of GAPSVP_γ.

Lattice-based hash function. Let $m = 2k^2$ and $\beta = \delta\sqrt{m}$ where δ is $\mathsf{poly}(k)$ and p be a polynomially bounded odd integer such that $p \geq 4\sqrt{m}k^{1.5}\beta$. It is easy to see that given k and δ there is always a $p = O(k^{3.5}\delta)$ to satisfy the above constraints. The collision resistant hash function that we are using is a generalization of the function presented in [28], where $\delta = O(1)$ (in the security parameter) is used instead. In our construction we use bigger values for δ. Namely the value that we use to bound the norm of the solution vector can be up to $\mathsf{poly}(k)$. This was observed in the original definition of Ajtai's one-way function [1], i.e., that the input vector can contain larger values (but not so large), and was also noted in its extension that achieves collision resistance [18]. This remark is very useful in our context and implies that, the larger value one picks for β, the larger the modulus p should be so that security is guaranteed.

Our hash function construction, however, uses a different modulus q (not p) that has k bits instead (note that that p has $O(\log k \log \delta)$ bits). This is because we require the complexity of computing the function not be dependent on δ. Let q be a k-bit modulus that is divided by p, i.e., $q = \Theta(2^k)$ and $p|q$. Let also λ be a value satisfying $\lambda = q/p = \Theta(2^k/k^{3.5}\delta)$. We sample a matrix $\mathbf{F} \in \mathbb{Z}_p^{k\times m}$ uniformly at random. After that we compute the matrix $\mathbf{M} = \lambda\mathbf{F}$. Note that the elements of matrix $\mathbf{M}$ have entries in $\mathbb{Z}_q$. Also note that λ defines an *injective homomorphism* from $\mathbb{Z}_p$ to $\mathbb{Z}_q$. We can now define the function $h_{\mathbf{M}} : \mathbb{Z}^m \rightarrow \mathbb{Z}_q^k$ as $h_{\mathbf{M}}(\mathbf{x}) = \mathbf{Mx} \mod q$, where $\|\mathbf{x}\| \leq \beta$ and the modulo operation is taken component-wise. The above function is collision resistant (for a constrained input) based on the difficulty of $\mathsf{GAPSVP}_{14\pi\sqrt{k}\beta}$:

Theorem 2 (Strong collision resistance) *Let $m = 2k^2$, $\beta = \delta\sqrt{m}$ and $p \geq 4\sqrt{m}k^{1.5}\beta$ be an odd positive integer. Let also $\mathbf{F} \in \mathbb{Z}_p^{k\times m}$ be a $k \times m$ matrix that is chosen uniformly at random and $\mathbf{M} = \lambda\mathbf{F} \in \mathbb{Z}_q^{k\times m}$ where q and λ are defined above. If there is an algorithm that finds two vectors $\mathbf{x}, \mathbf{y} \in \{0, 1, \ldots, \delta\}^m$ and $\mathbf{x} \neq \mathbf{y}$ such that $\mathbf{Mx} = \mathbf{My} \mod q$, then there is an algorithm to solve any instance of $\mathsf{GAPSVP}_{14\pi\delta\sqrt{km}}$.*

Proof: Suppose there is an algorithm that finds $\mathbf{x}, \mathbf{y} \in \{0, 1, \ldots, \delta\}^m$ with $\mathbf{x} \neq \mathbf{y}$ such that $\mathbf{Mx} = \mathbf{My} \mod q \Rightarrow \mathbf{M}(\mathbf{x} - \mathbf{y}) = \mathbf{0} \mod q$. By definition of q and $\mathbf{M}$ it is $\lambda\mathbf{F}(\mathbf{x} - \mathbf{y}) = \mathbf{0} \mod \lambda p \Rightarrow \exists \mathbf{r} \in \mathbb{Z}^k : \lambda\mathbf{F}(\mathbf{x} - \mathbf{y}) = \mathbf{r}\lambda p \Rightarrow \mathbf{F}(\mathbf{x} - \mathbf{y}) = \mathbf{r}p \Rightarrow \mathbf{F}(\mathbf{x} - \mathbf{y}) = \mathbf{0} \mod p$. Therefore the non-zero vector $\mathbf{z} = \mathbf{x} - \mathbf{y}$, which also has norm $\|\mathbf{z}\| \leq \beta$, since its coordinates are between $-\delta$ and $+\delta$, comprises a solution to the problem $\mathsf{SIS}_{p,m,\beta}$ (note that matrix $\mathbf{F}$ by construction is chosen uniformly at random). By Theorem 1, this can be used to solve GAPSVP_γ for $\gamma = 14\pi\sqrt{k}\beta$. Setting $\beta = \delta\sqrt{m}$ we get the desired result. $\square$

Since $\delta = \mathsf{poly}(k)$, γ is also $\mathsf{poly}(k)$ and therefore the presented hash function is secure since for polynomial γ (even for γ slightly smaller than exponential), no efficient algorithm to solve GAPSVP_γ is known to date [35]. We can now extend the function h to accept two inputs as follows: Denote with $\mathbb{T}^{\delta,+}$ the set of all $m \times 1$ ($m = 2k^2$) vectors such that their last k^2 entries are zero and the remaining entries are in $\{0, 1, \ldots, \delta\}$ and analogously with $\mathbb{T}^{\delta,-}$ the set of all $m \times 1$ vectors such that their first k^2 entries are zero and the remaining entries are in $\{0, 1, \ldots, \delta\}$. We can now define the function $h : \mathbb{T}^{\delta,+} \times \mathbb{T}^{\delta,-} \rightarrow \mathbb{Z}_q^k$

$$h_{\mathbf{M}}(\mathbf{x}, \mathbf{y}) = \mathbf{M}(\mathbf{x} + \mathbf{y}) \mod q\,, \tag{1}$$

where $\mathbf{x}, \mathbf{y} \in \{0, 1, \ldots, \delta\}^m$. Similarly as in Theorem 2, this function is strong collision resistant, i.e., if someone can find $(\mathbf{x}_1, \mathbf{y}_1) \in (\mathbb{T}^{\delta,+} \times \mathbb{T}^{\delta,-})$ and $(\mathbf{x}_2, \mathbf{y}_2) \in (\mathbb{T}^{\delta,+} \times \mathbb{T}^{\delta,-})$ with $(\mathbf{x}_1, \mathbf{y}_1) \neq (\mathbf{x}_2, \mathbf{y}_2)$ such that $\mathbf{M}(\mathbf{x}_1 + \mathbf{y}_1) = \mathbf{M}(\mathbf{x}_2 + \mathbf{y}_2) \mod q$ then one can solve the problem GAPSVP_γ for polynomial γ. To see that, note that the vector $\mathbf{x}_1 - \mathbf{x}_2 + \mathbf{y}_1 - \mathbf{y}_2$ has coordinates in $\{0, 1, \ldots, \delta\}$, since, by the definition of $\mathbb{T}^{\delta,+}$ and $\mathbb{T}^{\delta,-}$, the entries of $\mathbf{x}_1 - \mathbf{x}_2$ and $\mathbf{y}_1 - \mathbf{y}_2$ do not overlap.

Complexity of hash function. Due to the way the k-bit modulus q of our construction is defined, the complexity of our hash function does not depend on δ at all, but only on k. As we show below, the complexity is polynomial in k, but still, in our framework a *constant* quantity (i.e., independent of n). First of all, our hash function is described with a $k \times 2k^2$ matrix of k-bit entries. Therefore the space complexity is $O(k^4)$. Given now an input $x \in \{0, 1, \ldots, \delta\}^{2k^2}$, we can compute $h_{\mathbf{M}}(x)$ in $O(k^4 \log^2 k)$ time. To see that, an application of the hash function requires the computation of k internal products between vectors of $2k^2$ entries, and each multiplication in the internal product is a multiplication in $\mathbb{Z}_q$, which can be computed in $O(k \log^2 k)$ time using FFT [12]. This makes the total time equal to $O(k^4 \log^2 k)$. In the following we present some necessary definitions for authenticated data structures.

Authenticated data structures. As we mentioned in the introduction, there are three entities participating in the authenticated data structures computational model [26, 40]. 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 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 $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{verify}(q, \Pi(q), d_i)$, where q is a query on data structure D_i and $\Pi(q)$ is a proof provided by the server. Note that the digest d_i, the digest of D_i, is an input as well. All these complexity measures are listed in Table 1.

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[4] to a state of his liking, and which applies to any authenticated data structure:

Definition 4 (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 $i = 0, \ldots, h = \mathsf{poly}(k)$*

[4] The data structure considered in this paper is a dynamic table (array) of n indices.

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* $\Pi(q)$*. The authenticated data structure is secure if*

$$\Pr\left[\{q, \Pi(q), \alpha(q)\} \leftarrow \mathsf{Adv}(1^k, \mathsf{pk}); \mathsf{accept} \leftarrow \mathsf{verify}(q, \Pi(q), d_h); \mathsf{reject} = \mathsf{check}(q, \alpha(q), D_h)\right] \leq \nu(k)\,,$$

where $\nu(k)$ *is negligible*[5] *in the security parameter* k*.*

3 Main construction

Suppose we are given a table that consists of n indices $1, 2, \ldots, n$. In each index i we can store a value $\mathbf{x}_i$ from the set $S = \{0, 1, \ldots, C\}$, where $|S| = O(1)$. In this section we describe how we can build an authenticated structure on top of this table that uses the lattice-based hash function introduced in Section 2 and also supports updates in constant time.

Without loss of generality assume that n is a power of two so that we can build a complete binary tree on top of the table. Let T be that tree and let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be the values stored in the table. Assume each of the elements in $\{0, 1, \ldots, C\}$ can be represented with a vector of size k that has entries in $\mathbb{Z}_q$. Namely $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$. We are going to use the hash function $h_{\mathbf{M}}(\mathbf{x}, \mathbf{y})$ defined in Equation 1 in a recursive way to define the digest of the structure. We recall that k is the security parameter, $\mathbf{M}$ is a $k \times m$ matrix with elements sampled uniformly at random from $\mathbb{Z}_p$ and then multiplied with λ, $m = 2k^2$, $\beta = \delta\sqrt{m}$, $p \geq 4\sqrt{m}k^{1.5}\beta$ and $q = \lambda p$. Finally we also set $\delta = n$, i.e., we allow the inputs of our hash function to be in $\{0, 1, \ldots, n\}^{2k^2}$, where n is the size of our structure (table). We now continue with some definitions:

Definition 5 (Binary representation) *Let* $x \in \mathbb{Z}_q$*. We define* $f(x) \in \{0, 1\}^k$ *to be the binary representation of* x*. Namely if* $f(x) = [\mathbf{f}_0\ \mathbf{f}_1\ \ldots\ \mathbf{f}_{k-1}]^T$ *then it holds* $x = \sum_{i=0}^{k-1} \mathbf{f}_i 2^i \mod q$*.*

Definition 6 (Radix-2 representation) *Let* $x \in \mathbb{Z}_q$*. We define* $g(x) \in \mathbb{Z}_q^k$ *to be* some *radix-2 representation of* x*. Namely if* $g(x) = [\mathbf{f}_0\ \mathbf{f}_1\ \ldots\ \mathbf{f}_{k-1}]^T$ *then it holds* $x = \sum_{i=0}^{k-1} \mathbf{f}_i 2^i \mod q$*.*

By "some" radix-2 representation we mean that the function $g : \mathbb{Z}_q \rightarrow \mathbb{Z}_q^k$ is "one-to-many". For example, for $q = 16$, $x = 7$, possible values for $f(x)$ can be $[1\ 1\ 1\ 0]^T$ (the usual binary representation), $[-1\ 0\ 2\ 0]^T$ or $[-5\ \ -2\ 0\ 4]^T$ (and many more). We can give an important result for our construction:

Lemma 3 *For any* $x_1, x_2, \ldots, x_t \in \mathbb{Z}_q$ *there exist a radix-2 representation* $g(.)$ *such that* $g(x_1 + x_2 + \ldots + x_t \mod q) = f(x_1) + f(x_2) + \ldots + f(x_t) \mod q$*. Moreover it is* $g(x_1 + x_2 + \ldots + x_t \mod q) \in \{0, \ldots, t\}^k$*.*

Proof: (sketch) We note that componentwise addition of radix-2 representation vectors results (by vector addition properties) in a vector whose entries are the sum of entries that correspond to the same powers of 2, which translates into number addition too (detailed proof in the Appendix). □

Lemma 3 is useful in the following sense: Given two binary representations of x_1 and x_2, namely f_1 and f_2 respectively, a radix-2 representation of $x_1 + x_2$ is $f_1 + f_2$. Definitions 5 and 6 and also Lemma 3 (see Corollary 1) can now be naturally extended for vectors:

Definition 7 *Let* $\mathbf{x} \in \mathbb{Z}_q^k$*. We define* $f(\mathbf{x}) \in \{0, 1\}^{k^2}$ *to be the binary representation of* $\mathbf{x}$*. Namely every* $\mathbf{x}_i$*, for* $i = 1, \ldots, k$*, is mapped to the respective* k *entries* $f(\mathbf{x}_i)$ *in the resulting vector* $f(\mathbf{x})$*.*

Definition 8 *Let* $\mathbf{x} \in \mathbb{Z}_q^k$*. We define* $g(\mathbf{x}) \in \mathbb{Z}_q^{k^2}$ *to be* some *radix-2 representation of* $\mathbf{x}$*. Namely every* $\mathbf{x}_i$*, for* $i = 1, \ldots, k$*, is mapped to the respective* k *entries* $g(\mathbf{x}_i)$ *in the resulting vector* $g(\mathbf{x})$*.*

[5] We say that a real-valued function $\nu(k)$ over natural numbers is negligible if for any positive polynomial p, there exists integer m such that $\forall k > m$, $|\nu(k)| < \frac{1}{p(k)}$. We refer to a negligible function $\nu(k)$ also by saying that $\nu(k)$ is $\mathsf{neg}(k)$.

Corollary 1 *For any* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t \in \mathbb{Z}_q^k$ *there exist a radix-2 representation* $g(.)$ *such that* $g(\mathbf{x}_1 + \mathbf{x}_2 + \ldots + \mathbf{x}_t \mod q) = f(\mathbf{x}_1) + f(\mathbf{x}_2) + \ldots + f(\mathbf{x}_t) \mod q$. *Moreover it is* $g(\mathbf{x}_1 + \mathbf{x}_2 + \ldots + \mathbf{x}_t \mod q) \in \{0, \ldots, t\}^{k^2}$.

Let now $\mathbf{U} = [\mathbb{I}_{k^2} \ \mathbb{O}_{k^2}]^T$ (**U** stands for "up") and $\mathbf{D} = [\mathbb{O}_{k^2} \ \mathbb{I}_{k^2}]^T$ (**D** stands for "down") be $m \times k^2$ matrices, where $\mathbb{I}_t$ denotes the square unit matrix of dimension t and $\mathbb{O}_t$ denotes the square zero matrix of dimension t. It easy to see that for all $\mathbf{x} \in \{0, 1, \ldots, n\}^{k^2}$ it is $\mathbf{Ux} \in \mathbb{T}^{n,+}$ and $\mathbf{Dx} \in \mathbb{T}^{n,-}$. Namely multiplying matrices **U** and **D** with a vector in $\{0, 1, \ldots, n\}^{k^2}$ doubles the dimension of the vector by shifting its entries accordingly and by filling the vacant entries with zeros. This operation will be used to prepare the vectors in the appropriate input format for the hash function.

Digest definition. As we mentioned in the beginning of Section 3, we build a binary tree of ℓ levels on top of our n-index table. For each node v of the tree, we are going to define a collision resistant digest $d(v)$, based on the lattice-based hash function we introduce in Section 2. The digest of the root will serve as the digest of the whole structure. To begin, for every leaf node v_i of the tree, $i = 1, \ldots, n$ (note that at node v_i we store the value $\mathbf{x}_i$) we define the *leaf digest* $d(v_i)$ simply as $d(v_i) = \mathbf{x}_i \mod q$. For an internal node u, with left child $\mathsf{left}(u)$ and right child $\mathsf{right}(u)$, we define the *internal digest* as

$$d(u) = \mathbf{M}\left[\mathbf{U}g(d(\mathsf{left}(u))) + \mathbf{D}g(d(\mathsf{right}(u)))\right] \mod q\,, \tag{2}$$

where, by the constraint of the inputs in the definition of the hash function in Equation 1, it must be (we recall that we have set $\delta = n$)

$$g(d(\mathsf{left}(u))), g(d(\mathsf{right}(u))) \in \{0, 1, \ldots, \delta\}^{m/2} = \{0, 1, \ldots, n\}^{m/2}\,. \tag{3}$$

To formalize the properties of the inputs to the hash function, we give the following definition:

Definition 9 *Let* $x \in \mathbb{Z}_q^k$. *We say that* $g(x) \in \mathbb{Z}_q^{k^2}$ *is an* admissible *radix-2 representation of* x *if and only if* $g(x)$ *is a radix-2 representation of* x *that has entries in* $\{0, 1, \ldots, n\}$.

The flow of the computation in Equation 2 is as follows (see Figure 1): Suppose we are given an internal node u, with children $\mathsf{left}(u)$ and $\mathsf{right}(u)$, digests $d(\mathsf{left}(u)), d(\mathsf{right}(u)) \in \mathbb{Z}_q^k$. By applying $g(.)$ we transform them into vectors of k^2 "small" entries, i.e., into two *admissible* radix-2 representations. By multiplying with **U** and **D** we "prepare" them to be input to the hash function, as defined in Equation 1. We note here that the radix-2 representation $g(\mathbf{z})$ used in Equation 2 is some specific radix-2 representation of **z** that is *admissible*. It is actually the sum of a series of *binary representations* and is computed according to Definition 10. In the following definitions and theorems we denote with $\mathsf{bin}(x)$ the binary representation (string) of $x - 1$, with $\mathsf{bin}(x)_i$ the i-th bit of $\mathsf{bin}(x)$ and with $\mathsf{range}(z)$ the range of successive indices contained in the leaves of the subtree of T that is rooted on z (e.g., in Figure 1, it is $\mathsf{range}(r_{11}) = \{1, 2, 3, 4\}$):

Definition 10 *Let* $n = 2^\ell$, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ *be the values of the table that is to be authenticated and* T *be the complete binary tree of* ℓ *levels that is built on top of the table. Let* z *be an internal node of* T *at level* $0 \leq t < \ell$. *Then the* $g(.)$ *representation of* $d(z)$ *is computed as the sum of* $|\mathsf{range}(z)|$ *binary representations, i.e.,* $g(d(z)) = \sum_{i \in \mathsf{range}(z)} f(\mathbf{MA}_{i(t+1)} f(\mathbf{MA}_{i(t+2)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots))) \mod q$, *where* $\mathbf{A}_{ij} = \mathbf{U}$ *if* $\mathsf{bin}(i)_j = 0$ *and* $\mathbf{A}_{ij} = \mathbf{D}$ *otherwise.*

Although unintuitive, we are going to show later (Corollary 2) that $g(d(z))$, as defined in Definition 10, is indeed an admissible radix-2 representation of $d(z)$ (see Figure 1). Note for example that its entry constraints (from Definition 9) are indeed satisfied, since $|\mathsf{range}(z)| \leq \frac{n}{2}$. We can now define the *lattice digest* of a table $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ as follows:

Definition 11 *Let* $n = 2^\ell$, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ *be the values of the table that is to be authenticated and* T *be the complete binary tree of root* r *and height* ℓ *that is built on top of the table. Suppose we compute the digests* $d(u)$ *of the nodes* u *of the tree as above (Equation 2). We define the* lattice digest *of a node* u *to be the value* $d(u)$ *and the* lattice digest *of the table to be the value* $d(r)$.

We now present the main result of this section, namely the fact that the lattice digest can be expressed as a sum of n terms, which will eventually allow for more efficient updates:

Theorem 3 *Let $n = 2^\ell$, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ be the values of the table that is to be authenticated and T be the complete binary tree of ℓ levels that is built on top of the table. Let z be an internal node of T at level $0 \leq t < \ell$. Then, for every internal node z of the tree T, the* lattice digest *$d(z)$ of z can be expressed as $d(z) = \sum_{i \in \mathsf{range}(z)} \mathbf{MA}_{i(t+1)} f(\mathbf{MA}_{i(t+2)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots)) \mod q$, where $\mathbf{A}_{ij} = \mathbf{U}$ if $\mathsf{bin}(i)_j = 0$ and $\mathbf{A}_{ij} = \mathbf{D}$ otherwise.*

Proof: (sketch) By induction on the levels of the tree: We use the definition of the digest (Equation 2) recursively on all the nodes of the tree and start applying Corollary 1. Then we can express the digest as a sum of terms that are functions of the specific values stored in the table (full proof in the Appendix). □

Putting together Theorem 3 and Definition 10 we can prove the following:

Corollary 2 *Let z be an internal node of tree T. The $g(.)$ representation of $d(z)$ defined in Definition 10 is an admissible radix-2 representation of $d(z)$.*

To get some intuition about the expression of the lattice digest in Theorem 3, suppose we have a table of eight values $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_8$. Root r lies at level 0 and the leaves lie at level ℓ, as in Figure 1. Let r_{ij} be the j-th node at level i for $i = 1, \ldots, \ell$, with the numbering going also from the left to the right. According to Theorem 3, the *lattice digest* of the table $d(r)$ can be expressed as the following sum mod q:

$$\mathbf{MU}f(\mathbf{MU}f(\mathbf{MU}f(\mathbf{x}_1))) + \mathbf{MU}f(\mathbf{MU}f(\mathbf{MD}f(\mathbf{x}_2))) + \mathbf{MU}f(\mathbf{MD}f(\mathbf{MU}f(\mathbf{x}_3))) + \mathbf{MU}f(\mathbf{MD}f(\mathbf{MD}f(\mathbf{x}_4))) +$$
$$\mathbf{MD}f(\mathbf{MU}f(\mathbf{MU}f(\mathbf{x}_5))) + \mathbf{MD}f(\mathbf{MU}f(\mathbf{MD}f(\mathbf{x}_6))) + \mathbf{MD}f(\mathbf{MD}f(\mathbf{MU}f(\mathbf{x}_7))) + \mathbf{MD}f(\mathbf{MD}f(\mathbf{MD}f(\mathbf{x}_8)))\,.$$

Digest security. We now give the main security claim for the strong collision resistance of the *lattice digest*, given the results from Merkle [27] and Naor and Nissim [29]. In fact, Naor and Nissim [29] and Merkle [27] used exactly the same algorithmic construction (i.e., a binary tree) to provide a solution for an authenticated dictionary, generalizing their result for every strong collision resistant hash function h:

Remark 1 (Naor and Nissim [29]) *Possible choices for h include the more efficient MD4 [36], MD5 [37] or SHA [42] (collisions for MD4 and for the compress function of MD5 were found by Dobbertin [13, 14]) and functions based on a computational hardness assumption such as the hardness of discrete log [3, 8, 11] and subset-sum [18, 23] (these are much less efficient).*

The importance of the above remark is that essentially, one can use any strong collision resistant hash function $h(x, y)$ for a Merkle tree construction, given the hash function $h(x, y)$ is secure according to a widely acceptable computational assumption. Namely, it should be difficult (i.e., it should happen with negligible probability $\nu(k)$) for a computationally bounded adversary to find $(x, y) \neq (x', y')$ such that $h(x, y) = h(x', y')$. We therefore have the following result:

Theorem 4 (Strong collision resistance of the lattice digest) *Let k be the security parameter, $m = 2k^2$, $\beta = n\sqrt{m}$ and $p \geq 4\sqrt{m}k^{1.5}\beta$ be an odd positive integer. Let also $\mathbf{F} \in \mathbb{Z}_p^{k \times m}$ be a $k \times m$ matrix that is chosen uniformly at random and $\mathbf{M} = \lambda\mathbf{F} \in \mathbb{Z}_q^{k \times m}$, where q and λ are defined in Section 2. Let also $n = 2^\ell$, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ be the values of the table that is to be authenticated, having a* lattice digest *equal to d. It is computationally infeasible, i.e., it happens with negligible probability $\nu(k)$, for a computationally bounded adversary to find a different table $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n \in \mathbb{Z}_q^k$ of lattice digest equal to d, unless there is a polynomial-time algorithm for any instance of the problem GAPSVP_γ for $\gamma = 14\pi n\sqrt{km}$.*

Proof: By Remark 1 we can use any strong collision resistant hash function to recursively define a digest of a Merkle tree. Here we are using the function of Equation 1 which is strong collision resistant according to Theorem 2, unless there is a polynomial-time algorithm for any instance of the problem GAPSVP_γ for $\gamma = 14\pi n\sqrt{km} = \mathsf{poly}(k)$, since n is a polynomial of the security parameter (computational model). No polynomial algorithm is known to date that approximates GAPSVP_γ for $\gamma = \mathsf{poly}(k)$ [35]. □

Digest update. Suppose now that $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ are the values of the table and that the *lattice digests* have been computed. Let d be the *lattice digest* of the table. The objective of the update is to compute the new *lattice digest* of the table, in constant time, whenever the content of some index changes. We show how an update at index $1 \leq w \leq n$ can be performed, which applies for all indices. Note that for index w, where the value $\mathbf{x}_w$ is stored, the additive term from Theorem 3 is $\mathsf{term}(\mathbf{x}_w) = \mathbf{MA}_{w1} f(\mathbf{MA}_{w2} f(\ldots f(\mathbf{MA}_{w\ell} f(\mathbf{x}_w)) \ldots)) \mod q$. Note also that $\mathbf{x}_w$ does not appear in any other additive term $\mathsf{term}(\mathbf{x}_j)$ for all $j \neq w$ (see Theorem 3). Suppose now we update index w and we replace $\mathbf{x}_w$ with $\mathbf{y}_w$. The new digest, by Theorem 3, can be computed as $d' = d - \mathsf{term}(\mathbf{x}_w) + \mathsf{term}(\mathbf{y}_w) \mod q$, where $\mathsf{term}(\mathbf{y}_w) = \mathbf{MA}_{w1} f(\mathbf{MA}_{w2} f(\ldots f(\mathbf{MA}_{w\ell} f(\mathbf{y}_w)) \ldots))) \mod q$. If all the quantities $\mathsf{term}(.)$ for index w have been precomputed (one for each possible value that can be assigned to index w—and there is a constant number of such values—), the update can be performed in **constant** time, since it only involves two additions in $\mathbb{Z}_q$. We show now that this update does not violate any security requirement for the digest of any internal node and therefore the updated digest is a secure (collision-resistant) representation of our table:

Theorem 5 *Let $n = 2^\ell$, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ be the values of the table that is to be authenticated and T be the complete binary tree of ℓ levels that is built on top of the table. For $i = \ell, \ldots, 1$, let $\{v_i\}$ be the logarithmic-sized path from some index w to the root's child v_1, $d(v_i)$ be the respective* lattice digests *and $g(d(v_i)) \in \{0, 1, \ldots, n\}^{m/2}$ be the admissible radix-2 representations of them. An update is issued and the value of index w changes to $\mathbf{y}_w$. If $g(d'(v_i))$, $i = \ell, \ldots, 1$ are the updated $g(.)$ representations of the path nodes, then for every $i = \ell, \ldots, 1$, after the update, $g(d'(v_i))$ is also an admissible radix-2 representation.*

4 Authenticated data structures

We begin with describing how exactly the lattice-based construction is used in a three-party authenticated data structure model. We now have the following result:

Theorem 6 *Let k be the security parameter. Then there exists a three-party authenticated data structure for authenticating a dynamic table of n indices such that: (1) It is secure according to Definition 4 and assuming the hardness of GAPSVP_γ for $\gamma = O(nk\sqrt{k})$; (2) The source update time is $O(1)$; (3) The server update time is $O(\log n)$; (4) The source space is $O(n)$; (5) The server space is $O(n)$; (6) The client space is $O(1)$; (7) The server query time is $O(\log n)$; (8) The client verification time is $O(\log n)$; (9) The proof has size $O(\log n)$; (10) The update authentication information has size $O(1)$.*

Proof: (sketch) We recall that in each index in $\{1, \ldots, n\}$ the source can store one of the values of the set $S = \{0, 1, \ldots, C\}$. Each element of the set S is represented with a distinct element of $\mathbb{Z}_q^k$ and $|S| = O(1)$. Suppose now that $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ are the values of the table and that the *lattice digests* have been computed using Equation 2 and Definition 10. The source, for each index $w \in \{1, \ldots, n\}$ does the following precomputations: For each value $y_{wj} \in S - \{x_w\}$ $(j = 0, 1, \ldots, C)$ it computes and *stores* $\mathsf{term}(y_{wj})$ as defined above. Using these precomputations, the digest update can be performed with only two additions of vectors in $\mathbb{Z}_q$, which takes **constant** time (not dependent on n). Note that the source updates *only* the lattice digest of the whole structure, therefore it does not store the tree and the internal nodes digests. The servers, whenever an update at index w is issued by the source, have to update the digests of the internal nodes (the nodes belonging to the logarithmic-sized path from index w to the root of the tree) that are influenced by the update and as a result the *servers* update time is $O(\log n)$. Finally, the verification at the client side can be performed in logarithmic time (e.g., see Merkle tree verification in [27]). $\square$

Relation to memory checking. We note here that our model is different than the memory checking model [7]. If our lattice-based contruction was to be applied in a memory checking scenario, there would be no gains in terms of complexity—compared with known solutions [7]—, i.e., the *query complexity* (sum of reads and writes) would be $O(\log n)$[6]. However, the memory checking scenario assumes that the mem-

[6]Note that the best one can hope for in memory checking is $O(\log n / \log \log n)$ query complexity [15].

ory has no computational power, i.e., that it is a *passive* device. If we assume that the memory has some computational power, then this scenario can be realized with the *two-party authenticated data structures* model [32], where the untrusted server (untrusted memory) can execute computations (e.g., compute proofs) and is physically detached from the verification device (therefore communication complexity does matter). Intuitively, we can derive the two-party authenticated data structure model from the three-party one, by having the source executing both updates *and* queries (and the respective verification). Thus the same security definition (Definition 4) applies. Moreover, in this model, there is also the notion of "update proof", i.e., the server has to provide some proof that will enable the source to do the update (note that the source completely outsources his data structure to the server, unlike the three-party model where the source keeps the data structure locally). In this scenario we can use our lattice-based construction and improve the update complexity and the update proof of the source from $O(\log n)$ to $O(1)$ (as opposed to [32]):

Theorem 7 *Let k be the security parameter. Then there exists a two-party authenticated data structure for authenticating a dynamic table of n indices such that: (1) It is secure according to Definition 4 and assuming the hardness of GAPSVP_γ for $\gamma = O(nk\sqrt{k})$; (2) The source update time is $O(1)$; (3) The server update time is $O(\log n)$; (4) The source space is $O(1)$; (5) The server space is $O(n)$; (6) The server query time is $O(\log n)$; (7) The source verification time is $O(\log n)$; (8) The proof has size $O(\log n)$; (9) The update proof has size $O(1)$.*

Authenticated Bloom filters and discussion. In this section we show how we can use the lattice-based hash function to authenticate the Bloom filter functionality, a space-efficient dictionary, originally introduced in [6]. The Bloom filter consists of an array (table) $A[0 \ldots n-1]$ storing n bits. All the bits are initially set to 0. Suppose one needs to store a set S of r elements. Then K hash functions $h_i(.)$ with range $\{0, \ldots, n-1\}$ are used (these are not lattice-based hash functions) and for each element $s \in S$ we set the bits $A[h_i(s)]$ to 1, for $i = 1, \ldots, K$. In this way, false positives can occur, i.e., an element that is not present might be represented in A. The probability of a false positive can be proved to be $(1-p)^K$, where $p = e^{-Kr/n}$, which is minimized for $K = \ln 2(n/r)$ [6].

The Bloom filter above supports only insertions though. A deletion (i.e., setting some bits to 0) can cause the undesired deletion of many elements. To deal with this problem, *counting Bloom filters* were introduced by Fan et al. [16]. In this solution, by keeping a counter for each index of A (instead of just 0 or 1), we can tolerate deletions by incrementing the counter during insertions and decrementing the counter during deletions. However, the problem of *overflow* exists. As observed in [9], the overflow (at least one counter goes over some value C) occurs with probability $n(e \ln 2/C)^C$, for a certain set of r elements. Setting $C = O(1)$ (e.g., $C = 16$) is suitable for most of the applications [9].

By the above description, it is clear that we can use our lattice-based construction to authenticate the Bloom filter functionality: Each index of our table can take values from the set $\{0, \ldots, C\}$, where $C = O(1)$. Note that constant update complexity in this application is very important given that a Bloom filter is an *update-intensive* data structure (i.e., an insertion or deletion of an element involves K operations):

Theorem 8 *Let k be the security parameter. Then there exists a three-party authenticated data structure for authenticating a Bloom filter of size n, storing r elements and using K hash functions such that: (1) It is secure according to Definition 4 and assuming the hardness of GAPSVP_γ for $\gamma = O(nk\sqrt{k})$; (2) The source update time is $O(K)$; (3) The server update time is $O(K \log n)$; (4) The source space is $O(n)$; (5) The server space is $O(n)$; (6) The client space is $O(1)$; (7) The server query time is $O(K \log n)$; (8) The client verification time is $O(K \log n)$; (9) The proof has size $O(K \log n)$; (10) The update authentication information has size $O(1)$.*

Note that the above result can easily be generalized for the two-party model. For future work we envision applying lattices to more authenticated data structures problems, e.g., deriving a lattice-based cryptographic accumulator and also extending this solution to more complicated data structures while maintaining the optimal update complexity, e.g., deriving an update-optimal, lattice-based authenticated red-black tree.

References

[1] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.

[2] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology: Proc. EUROCRYPT*, volume 1233 of *LNCS*, pages 480–494. Springer-Verlag, 1997.

[3] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *LNCS*, pages 216–233. Springer-Verlag, 1994.

[4] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT, LNCS 1233*, pages 163–192. Springer-Verlag, 1997.

[5] J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology—EUROCRYPT 93*, volume 765 of *LNCS*, pages 274–285. Springer-Verlag, 1993.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

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

[8] S. Brands. An efficient off-line electronic cash system based on the representation problem. CWI Technical Report, CS-R9323, 1993.

[9] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2005.

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

[11] D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In *CRYPTO*, pages 470–484, 1991.

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

[13] H. Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption — FSE 1996*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer Verlag, 1996.

[14] H. Dobbertin. Cryptanalysis of MD5 compress. Presented at the Rump-session of EUROCRYPT '96, may 1996.

[15] C. Dwork, M. Naor, G. N. Rothblum, and V. Vaikuntanathan. How efficient can memory checking be? In *TCC*, pages 503–520, 2009.

[16] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Networking*, 8(3):281–293, 2000.

[17] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 197–206, New York, NY, USA, 2008. ACM.

[18] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems, 1996.

[19] 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.

[20] 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.

[21] E. Hall and C. S. Julta. Parallelizable authentication trees. In *Proc. Seclected Areas in Cryptography (SAC)*, pages 95–109, 2005.

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

[23] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996.

[24] A. K. Lenstra, H. W. L. Jr, and L. Lovasz. Factoring polynomials with rational coefficients. *Math.Ann.*, (261):515–534, 1982.

[25] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP (2)*, pages 144–155, 2006.

[26] 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.

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

[28] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

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

[30] M. Naor and G. N. Rothblum. The complexity of online memory checking. *J. ACM*, 56(1), 2009.

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

[32] C. Papamanthou and R. Tamassia. Time and space efficient algorithms for two-party authenticated data structures. In *Proc. Int. Conference on Information and Communications Security (ICICS)*, volume 4861 of *LNCS*, pages 1–15. Springer, 2007.

[33] 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.

[34] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. *Electronic Colloquium on Computational Complexity (ECCC)*, (158), 2005.

[35] O. Regev. On the complexity of lattice problems with polynomial approximation factors. *The LLL algorithm*, pages 475–496, 2010.

[36] R. Rivest. The MD4 message-digest algorithm. RFC 1320, Apr. 1992.

[37] R. Rivest. The MD5 message-digest algorithm. RFC 1321, Apr. 1992.

[38] T. Sander, A. Ta-Shma, and M. Yung. Blind, auditable membership proofs. In *Proc. Financial Cryptography (FC 2000)*, volume 1962 of *LNCS*. Springer-Verlag, 2001.

[39] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 55–69, Berlin, Heidelberg, 2009. Springer-Verlag.

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

[41] R. Tamassia and N. Triandopoulos. Computational bounds on hierarchical data processing with applications to information security. In *Proc. Int. Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 153–165. Springer-Verlag, 2005.

[42] U.S. National Institute of Standards and Technology. Secure hash standard. Federal Information Processing Standards Publication 180, 1992.

5 Appendix

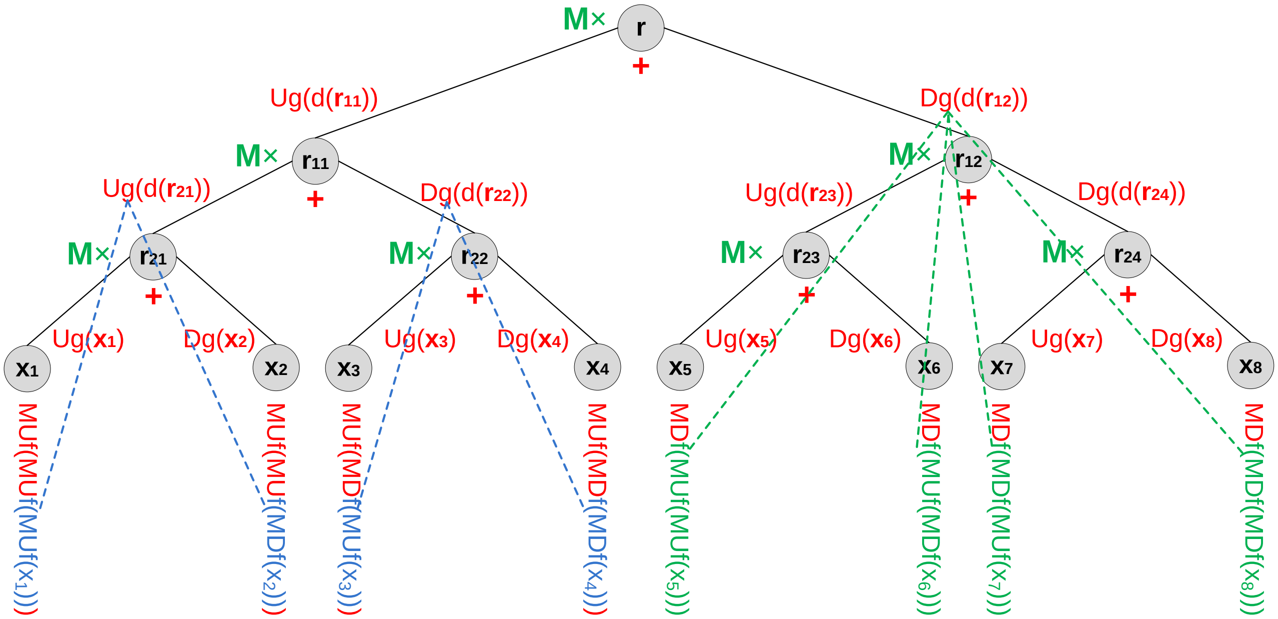

Figure 1: Tree T built on top of a table with 8 values $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_8$. After producing an *admissible* $g(.)$ representation of the child digests, we multiply with either $\mathbf{U}$ or $\mathbf{D}$, then we add the two resulting digests and we compute the hash function on them by multiplying with $\mathbf{M}$. At the leaves of the tree we show the terms that correspond to each index, as computed by Theorem 3. The relation between specific $f(.)$ representations of the additive terms computed by Theorem 3 and the $g(.)$ representation of the internal nodes are indicated with dashed lines. Note that the $g(.)$ representations of the internal nodes are the sum of specific $f(.)$ representations of the leaves, for example, $g(d(r_{12})) = f(\mathbf{MU}f(\mathbf{MU}f(\mathbf{x}_5))) + f(\mathbf{MU}f(\mathbf{MD}f(\mathbf{x}_6))) + f(\mathbf{MD}f(\mathbf{MU}f(\mathbf{x}_7))) + f(\mathbf{MD}f(\mathbf{MD}f(\mathbf{x}_8))) \mod q$.

Table 2: Exact complexity measures of our authenticated structure for a boolean table (i.e., $C = 2$), including constants. We recall that n is the size of the table to be authenticated and k is the security parameter. The local space needed by the client is constant. All constants in this table have been derived in the proof of Theorem 6 in the Appendix. Space complexity is in bits and time complexity is in number of group ($\mathbb{Z}_q$) operations.

source update	**server update**	**server query**	**verification**	**proof size**	**update info.**	**source space**	**server space**
k^2	$k^4 \log^2 k \log n$	$k^3 \log n$	$k^4 \log^2 k \log n$	$k^3 \log n$	k^2	$k^4 + k^2 n$	$k^4 + k^3(2n-1)$

5.1 Proof of Lemma 3

Let $\mathbf{x}_i = f(x_i)$ be the binary representation of x_i for $i = 1, \dots, t$. Then

$$\sum_{i=1}^{t} \mathbf{x}_i = \left[\sum_{i=1}^{t} \mathbf{x}_{i0} \ \sum_{i=1}^{t} \mathbf{x}_{i1} \dots \sum_{i=1}^{t} \mathbf{x}_{i(k-1)} \right]^T \mod q\,.$$

The resulting vector is a radix-2 representation of

$$\left(\sum_{i=1}^{t} \mathbf{x}_{i0}\right) \times 2^0 + \left(\sum_{i=1}^{t} \mathbf{x}_{i1}\right) \times 2^1 + \ldots + \left(\sum_{i=1}^{t} \mathbf{x}_{i(k-1)}\right) \times 2^{k-1} \mod q\,,$$

which can be written as

$$\sum_{j=0}^{k-1} \mathbf{x}_{1j} \times 2^j + \sum_{j=0}^{k-1} \mathbf{x}_{2j} \times 2^j + \ldots + \sum_{j=0}^{k-1} \mathbf{x}_{tj} \times 2^j = x_1 + x_2 + \ldots + x_t \mod q.$$

Therefore there exists a radix-2 representation g such that $g(x_1 + x_2 + \ldots + x_t \mod q) = f(x_1) + f(x_2) + \ldots + f(x_t) \mod q$. Finally note that since $g(.)$ is the sum of t binary representations, it cannot contain a entry that is greater than t. $\square$

5.2 Proof of Theorem 3

We prove the claim by induction on the levels of the tree T. For any internal node u that lies at level $\ell - 1$, there are only two nodes (that store for example values $\mathbf{x}_i$ (left child and odd index i) and $\mathbf{x}_j$ (right child and even index j) and belong to $\mathsf{range}(u)$) in the subtree rooted on u. It is

$$\mathbf{MU}f(\mathbf{x}_i) + \mathbf{MD}f(\mathbf{x}_j) = \mathbf{M}\left[\mathbf{U}g(\mathbf{x}_i) + \mathbf{D}g(\mathbf{x}_j)\right] = d(u)\,.$$

This is due to Equation 2 and also due to the fact that $g(.)$ coincides with $f(.)$, therefore satisfying the security requirement of Equation 3. Also $\mathbf{A}_{i1} = \mathbf{U}$ and $\mathbf{A}_{j1} = \mathbf{D}$, since i is odd and j is even. Hence the base case holds. Assume the theorem holds for any internal node v that lies at level $0 < t+1 \leq \ell$. Therefore

$$d(v) = \sum_{i \in \mathsf{range}(v)} \mathbf{MA}_{i(t+2)} f(\mathbf{MA}_{i(t+3)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots)) \mod q\,.$$

where $\mathbf{A}_{ij} = \mathbf{U}$ if $\mathsf{bin}(i)_j = 0$ and $\mathbf{A}_{ij} = \mathbf{D}$ if $\mathsf{bin}(i)_j = 1$. For any internal node z that lies at level t it should be

$$\begin{aligned} d(z) &= \sum_{i \in \mathsf{range}(z)} \mathbf{MA}_{i(t+1)} f(\mathbf{MA}_{i(t+2)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots)) \\ &= \mathbf{MU}\left(\sum_{i \in \mathsf{range}(\mathsf{left}(z))} f(\mathbf{MA}_{i(t+2)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots))\right) \\ &+ \mathbf{MD}\left(\sum_{i \in \mathsf{range}(\mathsf{right}(z))} f(\mathbf{MA}_{i(t+2)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots))\right) \mod q\,. \end{aligned}$$

By Corollary 1 there exist $g(.)$ representations of entries at most $\max\{|\mathsf{range}(\mathsf{left}(z))|, |\mathsf{range}(\mathsf{left}(z))|\} \leq \frac{n}{2}$ such that

$$\begin{aligned} d(z) &= \mathbf{MU}g\left(\sum_{i \in \mathsf{range}(\mathsf{left}(z))} \mathbf{MA}_{i(t+2)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots)\right) \\ &+ \mathbf{MD}g\left(\sum_{i \in \mathsf{range}(\mathsf{right}(z))} \mathbf{MA}_{i(t+2)} f(\ldots f(\mathbf{MA}_{i\ell} f(\mathbf{x}_i)) \ldots)\right) \mod q\,. \end{aligned}$$

By the inductive step this can be written as

$$d(z) = \mathbf{M}[\mathbf{U}g(d(\mathsf{left}(z))) + \mathbf{D}g(d(\mathsf{right}(z)))] \mod q\,,$$

where $g(.)$ are radix-2 representations that indeed satisfy the security requirement of Equation 3. Therefore this satisfies Definition 2 and $d(z)$ is indeed the correct digest of the internal node z. This completes the proof. □

5.3 Proof of Theorem 5

By Definition 10 and by the way updates are performed, at every time the $g(.)$ representations of the internal nodes are the sum of at most $\frac{n}{2}$ binary representations. Therefore the entries of the updated $g(.)$ representations cannot be greater than n, and therefore all the $g(.)$ representations of the internal nodes remain admissible after any update. □

5.4 Proof of Theorem 6

Security. We fix the parameters that we are using in our construction as follows: We recall that k is the security parameter, $\mathbf{M}$ is a $k \times m$ matrix with elements sampled uniformly at random from $\mathbb{Z}_q$, $m = 2k^2$, $\beta = \delta\sqrt{m}$, $p \geq 4\sqrt{m}k^{1.5}\beta$, q is a k-bit modulus and $\lambda = q/p$. We recall that elements of matrix $\mathbf{M}$ are computed as a product of random elements of $\mathbb{Z}_p$ and λ, so that to maintain an injective homomorphism from $\mathbb{Z}_p$ to $\mathbb{Z}_q$. Let's set $p = \lceil c_1 k^{3.5}\delta \rceil + 1$ or $p = \lceil c_1 k^{3.5}\delta \rceil$ such that p is an odd positive integer, as required by Theorem 1, for some suitable constant c_1. Finally we set $\delta = n$, where n is the size of our structure, which is a polynomially bounded value (we are in the computational model). This setup, by Theorem 2, will give a construction that is secure based on the difficulty of GAPSVP_γ for $\gamma = 14\pi\delta\sqrt{km}$. In specific, since $m = 2k^2$ and $\delta = n$ we have that $\gamma = O(nk\sqrt{k}) = O(k^c)$ for some $c = O(1)$.

Source. Suppose the initial state of the table is $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$ and that the initial digest of the table is d. As we showed in Section 3, for each index $w \in \{1, \ldots, n\}$ the source does the following precomputations: For each value $y_{wj} \in S - \{x_w\}$ ($j = 0, 1, \ldots, C$) it computes and stores $\mathsf{term}(y_{wj})$ as

$$\mathsf{term}(\mathbf{y}_{wj}) = \mathbf{MA}_{w1} f(\mathbf{MA}_{w2} f(\ldots f(\mathbf{MA}_{w\ell} f(\mathbf{y}_{wj}))\ldots)) \mod q\,,$$

where $S = \{0, \ldots, C\}$. Each term $\mathsf{term}(y_{wj})$ is an element in $\mathbb{Z}_q^k$ and therefore the source needs $O(k^2) \times O(|S|)$ bits for each index w (plus the matrix $\mathbf{M}$ that needs $O(k^4)$ bits). Therefore the space needed is $O(k^4 + k^2 n) = O(n)$. The source issues an update that changes the value of index w from $\mathbf{x}_w$ to $\mathbf{y}_w$. Then the updated digest d' is computed by setting

$$d' = d - \mathsf{term}(\mathbf{x}_w) + \mathsf{term}(\mathbf{y}_w) \mod q\,,$$

where $\mathsf{term}(\mathbf{x}_w)$ and $\mathsf{term}(\mathbf{y}_w)$ are defined as

$$\mathsf{term}(\mathbf{x}_w) = \mathbf{MA}_{w1} f(\mathbf{MA}_{w2} f(\ldots f(\mathbf{MA}_{w\ell} f(\mathbf{x}_w))\ldots)) \mod q$$

and

$$\mathsf{term}(\mathbf{y}_w) = \mathbf{MA}_{w1} f(\mathbf{MA}_{w2} f(\ldots f(\mathbf{MA}_{w\ell} f(\mathbf{y}_w))\ldots)) \mod q\,.$$

This operation requires two additions (i.e., $O(1)$ operations) in $\mathbb{Z}_q^k$, which take time $O(k^2) = O(1)$ (k is a constant). Finally note, that, by Theorem 5, there is no internal node of the tree whose $g(.)$ representation is not *admissible*, as a result of the described update. Therefore during all the updates, secure digests are being produced. As for the update authentication information, this is a signature of the lattice digest, which is $O(k^2) = O(1)$ bits long and therefore the signature is also $O(1)$ bits.

Servers. Suppose the table is at state $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{Z}_q^k$. The server stores the binary tree on top of the table, and at each internal node v of the binary tree, apart from the lattice digest $d(v)$ it also stores the admissible $g(.)$ representation of it, i.e., $g(d(v))$. The lattice digest $d(v)$ requires $O(k^2)$ bits and the $g(.)$ representation requires $O(k^3)$ bits (we recall that each $g(.)$ representation has k^2 entries in $\mathbb{Z}_q$ and therefore $O(k^3)$ bits are needed). Since the tree has $O(n)$ nodes in total (the server also stores matrix **M**), the server needs space $O(k^4 + k^3 n) = O(n)$. Suppose now an update is issued, that changes the value of the index w from $\mathbf{x}_w$ to $\mathbf{y}_w$. Let now

$$\mathsf{term}(\mathbf{x}_w) = \mathbf{MA}_{w1} f_{w1}(\mathbf{MA}_{w2} f_{w2}(\ldots f_{w(\ell-1)}(\mathbf{MA}_{w\ell} f_{w\ell}(\mathbf{x}_w))\ldots)) \mod q\,,$$

and

$$\mathsf{term}(\mathbf{y}_w) = \mathbf{MA}_{w1} f'_{w1}(\mathbf{MA}_{w2} f'_{w2}(\ldots f'_{w(\ell-1)}(\mathbf{MA}_{w\ell} f'_{w\ell}(\mathbf{y}_w))\ldots)) \mod q\,,$$

where $f_{wi}(.)$ and $f'_{wi}(.)$ are the respective binary representations, for $i = 1, \ldots, \ell$. The server computes the representations $f_{wi}(.)$ and $f'_{wi}(.)$ by using the recursive relations:

$$\begin{aligned} f_{w\ell} &= f(\mathbf{x}_w) \mod q \\ f_{wi} &= f(\mathbf{MA}_{w1} f_{w(i+1)}) \mod q\,, \end{aligned}$$

for $i = \ell - 1, \ldots, 1$ and

$$\begin{aligned} f'_{w\ell} &= f'(\mathbf{x}_w) \mod q \\ f'_{wi} &= f'(\mathbf{MA}_{w1} f'_{w(i+1)}) \mod q\,, \end{aligned}$$

for $i = \ell - 1, \ldots, 1$. This task is performed in $O(k^4 \log^2 k \log n) = O(\log n)$ time since it involves one application of the hash function (requiring $O(k^4 \log^2 k)$ time) and one binary representation computation of a k^2-bit number (taking time $O(k^3)$ time since the arithmetic is in $\mathbb{Z}_q$), per level (for a total of $\log n$ levels).

Let now $v_\ell, v_{\ell-1}, \ldots, v_1$ be the path from the node of index w to the child v_1 of the root of the tree. The server now is going to use the computed $f(.)$ representations from above to update $d(v_i)$ to $d'(v_i)$ and $g(d(v_i))$ to $g(d'(v_i))$ (i.e., the new admissible $g(.)$ representations) as follows. By Definition 10 we can set

$$g(d'(v_i)) = g(d(v_i)) - f_{wi} + f'_{wi} \mod q\,,$$

for $i = \ell, \ldots, 1$. This operation takes time $O(k^3)$ (the arithmetic is in $\mathbb{Z}_q$) and is performed $\log n$ times, therefore the total time is $O(k^3 \log n)$. Finally after the new admissible $g(.)$ representations are computed the lattice digests can be updated by applying the hash function per node (an operation that is also parrarelizable) which takes time $O(k^4 \log^2 k \log n) = O(\log n)$. Therefore the update time $O(\log n)$.

The query time involves the computation of the proof, basically computing the collection of the $g(.)$ admissible representations along the path of the queried index. The proof is going to be the following logarithmic-sized tuple:

$$\{g(d(v_\ell)), g(d(\mathsf{sib}(v_\ell))), g(d(v_{\ell-1})), g(d(\mathsf{sib}(v_{\ell-1}))), \ldots, g(d(v_1)), g(d(\mathsf{sib}(v_1)))\}\,,$$

exactly as is done in the computation of a Merkle tree proof. This takes $O(k^3 \log n) = O(\log n)$ time to compute, since we have to collect $O(\log n)$ vectors of $O(k^3)$ bits each, which makes the proof size also $O(k^3 \log n) = O(\log n)$.

Clients. Suppose the client queries for index w. Let $v_\ell, v_{\ell-1}, \ldots, v_1$ be the path from the node of index w to the child v_1 of the root of the tree. The server, as we showed above, computes the following proof

$$\{g(d(v_\ell)), g(d(\mathsf{sib}(v_\ell))), g(d(v_{\ell-1})), g(d(\mathsf{sib}(v_{\ell-1}))), \ldots, g(d(v_1)), g(d(\mathsf{sib}(v_1)))\}$$

and also sends the answer "the value of index w is $\mathbf{r}_w$". The client checks to see if $g(d(v_\ell)) = f(\mathbf{r}_w)$ and accordingly performs the following checks:

$$f(\mathbf{M}[\mathbf{A}_{i1}g(d(v_i)) + \mathbf{A}_{i2}g(d(\mathsf{sib}(v_i)))]) = g(d(v_{i-1}))\,?$$

for $i = \ell, \ldots, 2$ and where $\mathbf{A}_{i1}$ and $\mathbf{A}_{i2}$ are either $\mathbf{U}$ or $\mathbf{D}$ depending on the binary representation of w. During these computations the client should also check to see that the coordinates of the $g(.)$ representations are in $\{0, 1, \ldots, n\}$, i.e., that the $g(.)$ representations used are *admissible*. Finally, if d is the authentic digest received by the source the client performs the final verification, i.e., he checks to see if $\mathbf{M}[\mathbf{A}_{11}g(d(v_1)) + \mathbf{A}_{12}g(d(\mathsf{sib}(v_1)))] = d$? If all the checks succeed, then the client accepts the answer, otherwise the client rejects. Since the client has to do $O(\log n)$ checks, each one taking time $O(k^4 \log^2 k)$, since matrix multiplications are involved, the verification time is $O(k^4 \log^2 k \log n) = O(\log n)$. Finally, the client needs only to locally store the public key of the source and the matrix $\mathbf{M}$, in order to run the verification algorithm. Therefore the local space needed is $O(k^4 + k) = O(1)$. □

5.5 Proof of Theorem 7

The two-party authenticated data structure works mostly the same with the three-party one but with the following differences: The source, instead of keeping the values $\mathsf{term}(.)$ locally, signs and outsources all these values. Therefore, whenever the source issues an update, it can update the lattice digest by retrieving the appropriate signatures that correspond to a specific index (threfore the update proof is of constant size since there are $O(1)$ $\mathsf{term}(.)$ values that correspond to a specific index) and execute the lattice digest update as before (i.e., with two vector additions). All other algorithms, such as query, verification and server update remain unchanged and therefore they run in $O(\log n)$ time. □