Zero-Knowledge Authenticated Order Queries and Order Statistics on a List

Esha Ghosh[1(✉)], Olga Ohrimenko[2], and Roberto Tamassia[1]

[1] Department of Computer Science, Brown University, Providence, USA
{esha_ghosh,roberto_tamassia}@brown.edu
[2] Microsoft Research, Cambridge, UK
oohrim@microsoft.com

Abstract. An order query takes as input a set of elements from a list (ordered sequence) $\mathcal{L}$, and asks for this set to be ordered using the total order induced by $\mathcal{L}$. We introduce two formal models for answering order queries on a list in a verifiable and private manner. Our first model, called *zero-knowledge list* (ZKL), generalizes the standard two-party model of membership queries on a set to order queries on a list in zero-knowledge. We present a construction of ZKL based on zero-knowledge sets and a homomorphic integer commitment. Our second model, *privacy-preserving authenticated list* (PPAL), extends authenticated data structures by adding a zero-knowledge privacy requirement. This is a three-party model, where a list is outsourced by a trusted owner to an untrusted cloud server, which answers order queries issued by clients and returns proofs of the answers. PPAL supports data integrity against a malicious server and privacy protection against a malicious client. Though PPAL can be implemented using our ZKL construction, this construction is not as efficient as desired in cloud applications. We present an *efficient* PPAL construction based on our novel technique of blinded bilinear accumulators and bilinear maps. Both our models are provably secure in the Random Oracle model and are zero-knowledge (e.g., hiding even the size of the list). We also show that the ZKL and PPAL frameworks can be extended to support fundamental statistical queries efficiently and in zero-knowledge.

1 Introduction

Releasing verifiable partial information while maintaining privacy is a requirement in many practical scenarios where the data being dealt with is sensitive. A basic case is releasing a subset of a set and proving its authenticity in a privacy-preserving way (referred to as zero-knowledge property) [10,12,26,29]. However, in many other cases, the information is stored in data structures to support richer type of queries. In this paper, we consider *order queries* on two or more elements of a list, where the answer to the query returns the elements rearranged according to their order in the list. Order queries lie at the heart of many practical applications where the order between queried elements is revealed and proved but the rank of the queried elements in the list and information about other elements in the list should be protected.

T. Malkin et al. (Eds.): ACNS 2015, LNCS 9092, pp. 149–171, 2015.
DOI: 10.1007/978-3-319-28166-7_8

In an auction with a single winner (e.g., online ad auction for a single ad spot) every participant submits her secret bid to the auction organizer. After the top bidder is announced, a participant wishes to verify that her bid was inferior. The organizer would then provide a proof without revealing the amount of the top bid, the rank of the participant's bid, or any information about other bids.

Lenders often require an individual or a couple to prove eligibility for a loan by providing a bank statement and a pay stub. Such documents contain sensitive information beyond what the lender is looking for: whether the bank account balance and salary are above given thresholds. A desirable alternative would be to provide a proof from the bank and employer that these thresholds are met without revealing exact figures and even hiding who of the two spouses earns more.

The above examples can be generalized using order queries on an ordered set, aka list, that return the order of the queried elements as well as a proof of this order but without revealing anything more than the answer itself. We address this problem by introducing two different models: *zero knowledge lists* (ZKL) and *privacy-preserving authenticated lists* (PPAL).

ZKL considers two party model and extends zero knowledge sets [12,29] to lists. In ZKL a prover commits to a list and a verifier queries the prover to learn the order of a subset of list elements. The verifier should be able to verify the answer but learn no information about the rest of the list, e.g., the size of the list, the order of other elements of the list or the rank of the queried element(s). Here both the prover and the verifier can act as malicious adversaries. While the prover may want to give answers inconsistent with the initial list he committed to, the verifier may try to learn information beyond the query answer or arbitrarily deviate from the protocol.

PPAL considers three parties: the owner of the list, the server who answers list queries on behalf of the owner, and the client who queries the server. The privacy guarantee of PPAL is the same as in ZKL. For authenticity, PPAL assumes that the owner is trusted while the server and the client could be malicious. This trust model allows for a much more efficient construction than ZKL, as we will see later in the paper. PPAL has direct applications to outsourced services where the server is modeling the cloud service that the owner uses to interact with her clients.

We note that PPAL can be viewed as a privacy-preserving extension of authenticated data structures (ADS) (see, e.g., [19,20,28,36]), which also operate in a three party model: the server stores the owner's data and proves to the client the answer to a query. However, privacy properties have not been studied in this model and as a consequence, known ADS constructions leak information about the rest of the data through their proofs of authenticity. For example, the classic Merkle hash tree [28] on a set of n elements proves membership of an element via a proof of size $\log n$, thus leaking information about the size of the set. Also, if the elements are stored at the leaves in sorted order, the proof of *membership* of an element reveals its rank.

In this paper, we define the security properties for ZKL and PPAL and provide efficient constructions for them. The privacy property against the verifier in ZKL and the client in PPAL is *zero knowledge*. That is, the answers and the proofs are indistinguishable from those that are generated by a simulator

that knows nothing except the previous and current queries and answers and, hence, cannot possibly leak any information beyond that. While we show that PPAL can be implemented using our ZKL construction, we also provide a direct PPAL construction that is considerably more efficient thanks to the trust that clients put in the list owner. Let n be the size of the list and m be the size of the query, i.e., the number of list elements whose order is sought. Our PPAL construction uses proofs of $O(m)$ size and allows the client to verify a proof in $O(m)$ time. The owner executes the setup in $O(n)$ time and space. The server uses $O(n)$ space to store the list and related authentication information, and takes $O(\min(m \log n, n))$ time to answer a query and generate a proof. In contrast, in the ZKL construction, the time and storage requirements have an overhead that linearly depends on the security parameter. Note that ZKL also supports (non-) membership queries. The client in PPAL and the verifier in ZKL require only one round of communication for each query. Our ZKL construction is based on zero knowledge sets and homomorphic integer commitments. Our PPAL construction uses a novel technique of blinding of accumulators along with bilinear aggregate signatures. Both are secure in the random oracle model.

2 Problem Statement, Models, Related Work, and Contributions

In this section, we state our problem, outline our models, review related work, and summarize our contributions. Formal definitions and constructions are in the rest of the paper. Detailed proofs and construction that are omitted due to space restrictions are available in the full version [17].

2.1 Problem Statement and Models

Let $\mathcal{L}$ be a totally ordered list of distinct elements. An *order query* on $\mathcal{L}$ is defined as follows: given a set of elements of $\mathcal{L}$, return these elements rearranged according to their order in $\mathcal{L}$ and a proof of this order. Both models we introduce, PPAL and ZKL, support this query. ZKL, in addition to order queries, supports provable membership and non-membership queries. Beside providing authenticity, the proofs are required not to leak any information beyond the answer.

ZKL: This model has two parties: prover and verifier. The prover initially computes a commitment to a list $\mathcal{L}$ and reveals the commitment to the verifier. Later the verifier asks membership and order queries on $\mathcal{L}$ and the prover responds with a proof. Both the prover and the verifier can be malicious:

- The prover may try to give answers inconsistent with the initial commitment.
- The verifier may try to learn from the proofs additional information about $\mathcal{L}$ beyond what he has inferred from the answers. E.g., if the verifier has performed two order queries with answers $x < y$ and $x < z$, he may want to find out whether $y < z$ or $z < y$.

The security properties of ZKL, *completeness, soundness* and *zero-knowledge*, guarantee security against malicious prover and verifier. Completeness mandates that honestly generated proofs always satisfy the verification test. Soundness states that the prover should not be able to come up with a query, and corresponding inconsistent (with the initial commitment) answers and convincing proofs. Finally, zero-knowledge means that each proof reveals the answer and nothing else. In other words, there must exist a simulator, that given only oracle access to $\mathcal{L}$, can simulate proofs for membership and order queries that are indistinguishable from real proofs.

PPAL: This model has three parties: owner, server and client. The owner generates list $\mathcal{L}$ and outsources it to the server. The owner also sends (possibly different) digest information with respect to $\mathcal{L}$ to the server and the client. Given an order query from the client, the server, using the server digest, builds and returns to the client the answer and its proof, which is verified by the client using the client digest. Both the server and the client can be malicious:

- The server may try to forge proofs for incorrect answers to (order) queries, e.g., prove an incorrect ordering of a pair of elements of $\mathcal{L}$.
- The client, similar to the verifier in ZKL, may try to learn from the proofs additional information about list $\mathcal{L}$ beyond what he has inferred from the answers.

Note that in typical cloud database applications, the client is allowed to have only a restricted view of the data structure and the server enforces an access control policy that prevents the client from getting answers to unauthorized queries. This motivates the curious, possibly malicious, behavior from the client where he tries to ask ill-formed queries or queries violating the access control policy. However, we assume that the server enforces client's legitimate behavior by refusing to answer illegal queries. Hence, the security model for PPAL is defined as follows.

The properties of PPAL, *Completeness, Soundness* and *Zero-Knowledge*, guarantee security against malicious server and client. They are close to the ones of ZKL except for soundness. For PPAL it enforces that the client does not accept proofs forged by the server for incorrect answers w.r.t. owner's list. PPAL's owner and server together can be thought of as a single party in ZKL, the prover. Hence, ZKL soundness protects against the prover who tries to give answers inconsistent with her own initial commitment. In the PPAL model, the owner and the server are separate parties where the owner is trusted and soundness protects against a malicious server only.

To understand the strength of the zero-knowledge property, let us illustrate to what extent the proofs are non-revealing. This property guarantees that a client, who adaptively queries a static list, does not learn anything about ranks of the queried elements, the distance between them or even the size of $\mathcal{L}$. The client is not able to infer any relative order information that is not inferable by the rule of transitivity from the previously queried orders. It is worth noting that in the context of leakage-free redactable signature schemes, privacy property has been

defined using game-based definitions in *transparency* [6,34] and *privacy* [11,23]. However, our definition of simulatability of the query responses, or the zero-knowledge property, is a simpler and more intuitive way to capture the property of leakage-freeness.

Efficiency: We characterize the ideal efficiency goals of our models as follows, where $\mathcal{L}$ is a list of n items and m is the query size. The space for storing list $\mathcal{L}$ and the auxiliary information for generating proofs should be $O(n)$. As in related work, a multiplicative factor for element size of $O(\mathsf{poly}(\mathsf{k}))$, where k is the security parameter, is not shown in $O(\cdot)$. The setup to preprocess list $\mathcal{L}$ should take $O(n)$ time. The proof of the answer to a query should have $O(m)$ size. Processing a query to generate the answer and its proof should take $O(m)$ time. Verifying the proof of an answer should take $O(m)$ time.

Applications of Order Queries to Order Statistics: Our PPAL order queries can be used as a building block to answer efficiently and in zero knowledge (i.e., the returned proofs should be simulatable) many interesting statistical queries about a list $\mathcal{L}$ with n elements. Let a *pair order proof* denote the proof of the order of two elements from $\mathcal{L}$. Then a PPAL client can send the server a subset $\mathcal{S}$ of m list elements and request the server to return the *maximum*, *minimum*, or the *median* element of $\mathcal{S}$ w.r.t. the order of the elements in the list. This can be done by providing m pair order proofs. Order queries also can be extended to return the *top* t elements of $\mathcal{S}$ by means of $t(m-t)$ pair order proofs, or only $m-1$ pair order proofs if the order between the top t elements can be revealed, where $t < m$. Finally, given an element a in $\mathcal{L}$, the server can return the elements of $\mathcal{S}$ that are above (or below) the *threshold* value a by means of m pair order proofs. It is important to note that neither of these queries reveal anything more than the answer itself. Moreover, the size of the proof returned for each query is proportional to the query size and is optimal for the threshold query where the proof size is proportional to the answer size. We note that these statistical queries are also supported by ZKL.

2.2 Related Work

First, we discuss work on data structures that answer queries in zero knowledge. Our ZKL is the first extension of this work to lists and order queries. We then mention signature schemes that can be used to instantiate outsourced data structures that require privacy and integrity to be maintained. However, such instantiations are not efficient since they are based on different models of usage and underlying data. Finally, we outline leakage-free redactable signature schemes for ordered lists and other structured data. These signature schemes are not as efficient as our construction and their definitions are game-based as opposed to our intuitive zero-knowledge definition. Finally we discuss follow-up work on PPAL.

Zero Knowledge Data Structures: Zero-knowledge dictionary and range queries have received considerable attention in literature [10,12,26,29,32]. Our proposed ZKL model is the first generalization of this line of work that supports order queries.

The model of *zero knowledge set* (ZKS) was introduced by Micali *et al.* [29] where a prover commits to a finite set S in such a way that, later on, she will be able to efficiently (and non-interactively) prove statements of the form $x \in S$ or $x \notin S$ without leaking any information about S beyond what has been queried for, not even the size of S. The prover should not be able to prove contradictory statements about an element. Chase *et al.* [12] abstracted the above solution and described it in terms of a *mercurial commitment*, which was later generalized to q-trapdoor mercurial commitments in [10,26] and a closely related notion of *vector commitments* was proposed in [9]. Kate *et al.* [22] suggested a weaker primitive called *nearly-zero knowledge set* where the set size is not private. Ostrovsky *et al.* [32] generalized (non-)membership queries to orthogonal range queries on multidimensional dataset and considered adding privacy to their protocol. However, the use of NP-reductions and probabilistically checkable proofs makes their generic construction expensive.

We note that a recent work on DNSSEC zone enumeration by Goldberg *et al.* [18] uses a model related to our PPAL model and is independently developed. The framework supports only set (non-)membership queries and answers them in f-zero knowledge. This property ensures that the information leaked to the verifier is in terms of a function f on the set, e.g., f is the set size in [18].

Signature Schemes: A three party model where the owner digitally signs a data document and outsources it to the server and the server discloses to the client only part of the signed document along with a legitimately derived signature on it (without the owner's involvement), can be instantiated with a collection of signature schemes, namely, *content extraction, quotable, arithmetic, redactable, homomorphic, sanitizable and transitive signatures* [7,21,30,31,35,38]. Additionally, if the signatures reveal no information about the parent document, then this approach can be used to add privacy. However the generic instantiation, with signature schemes that do not specifically address structured data, is inefficient for most practical purposes.

Ahn *et al.* [1] present a unified framework for computing on authenticated data where a third party can derive a signature on an object x' from a signature on a parent object x as long as $P(x, x') = 1$ for some predicate P that captures the *authenticatable relationship* between x and x'. Additionally, a derived signature reveals no extra information about the parent x. This line of work was later refined in [2,37]. The authors in [1] propose a computationally expensive scheme based on the RSA accumulator and predicates for specific data structures are not considered. A related notion of malleable signature scheme was proposed in [13], where given a signature σ on a message x, it is possible to efficiently derive a signature σ' on a message x' such that $x' = T(x)$ for an *allowable* transformation T without access to the secret key. The privacy definition of [13] (simulation context hiding) is stronger than that of [1] as it allows for adversarially-generated

keys and signatures. However, the owner is a trusted party in our PPAL setting and therefore the stronger notion of simulation context hiding is not relevant in this framework. Moreover, in our PPAL model, given a quote from a document and a proof of the quote, the client *should* be able to verify that the quote is indeed in the document, this is inverse of the notion of unlinkability in [13].

Leakage-Free Signature Schemes for Structural Data: A *leakage-free redactable signature scheme (LRSS)* allows a third party to remove, or *redact*, parts of a signed document without signer's involvement. The verifier only sees the remaining redacted document and is able to verify that it is valid and authentic. Leakage-freeness property ensures that the redacted document and its signature do not reveal anything about the content or position of the removed parts. We discuss LRSSs that specifically look at structural data and ordered lists. In Table 1 we show that PPAL outperforms known LRSS constructions. Another significant difference of our work is the definition of privacy. The zero-knowledge property is more intuitive and simple in capturing the leakage-freeness property compared to the game based definitions in the LRSS literature [6,34].

Kundu and Bertino [24] introduced the idea of structural signatures for ordered trees (subsuming ordered lists) that support public redaction of subtrees by third-parties. This work was later extended to undirected graphs and DAGs [25]. The notion was later formalized as *LRSS* for ordered trees in [6] and subsequently several attacks on [24] were also proposed in [6,33]. The basic idea of the LRSS scheme presented in [6] is to sign *all possible ordered pairs* of elements of an ordered list. So both the computation cost and the storage space are quadratic in the number of elements of the list.

Building on the work of [6,34] proposed a LRSS for lists that has quadratic time and space complexity. Poehls *et al.* [33] presented a LRSS scheme for a list that has linear time and space complexity but assumes an associative non-abelian hash function, whose existence has not been formally proved. Kundu *et al.* [23], presented a construction that uses quadratic space at the server. Chang *et al.* [11] presented a leakage-free redactable signature scheme for a string (which can be viewed as a list) that hides the location of the redacted or deleted portions of the string at the expense of quadratic verification cost. None of the constructions of [11,23,24] satisfy our definition of zero-knowledge.

Follow-up Work: Finally we note that in recent work [16], Ghosh *et al.* have generalized the models introduced in this paper to general abstract data types that support both query and update operations. Also, they have presented efficient constructions for dynamic lists and partially-ordered sets of bounded dimension.

2.3 Contributions and Organization of the Paper

Our contributions are novel models and efficient constructions. After reviewing preliminary concepts and cryptographic primitives, in Sect. 3, we introduce the zero-knowledge list (ZKL) model. We describe our ZKL construction, its security

Table 1. Comparison of our ZKL and PPAL constructions with previous work. All the time and space complexities are asymptotic. Notation: n is the list size, m is the query size, k is the security parameter. WLOG we assume list elements are k bit long. Following the standard convention, we omit a multiplicative factor of $O(k)$ for element size in every cell. Assumptions: Strong RSA Assumption (SRSA); Existential Unforgeability under Chosen Message Attack (EUCMA) of the underlying signature scheme; Random Oracle Model (ROM); n-Element Aggregate Extraction Assumption (nEAE); Associative non-abelian hash function (AnAHF) [**non-standard**]; Collision Resistant Hash Function (CRHF); Discrete Log Assumption (DL); Factoring a composite (FC); n-Bilinear Diffie Hellman Inversion Assumption (nBDHI).

	[35]	[21]	[11]	[6]	[34]	[33]	[23]	[12]	ZKL	PPAL
Zero-knowledge				✓	✓	✓		✓	✓	✓
Setup time	$n \log n$	n	n	n^2	n^2	n	n	nk	nk	n
Storage space	n	n	n	n^2	n^2	n	n^2	nk	nk	n
Order query time	m	$n \log n$	n	mn	m	n	n		mk	$\min(m \log n, n)$
(Non)-Member query time								mk	mk	
Order verification time	$m \log n \log m$	$m \log n$	n^2	m^2	m^2	m	m		mk	m
(Non)-Member verification time								mk	mk	
Proof size	m	$m \log n$	n	m^2	m^2	m	n	mk	mk	m
Assumption	RSA	RSA	SRSA, Division	EUCMA	ROM, nEAE	**AnAHF**	ROM, RSA	CRHF, DL	ROM, CRHF, SRSA, FC	ROM, nBDHI

and efficiency in Sect. 4. In Sect. 5, we introduce the privacy-preserving authenticated list (PPAL) model. An efficient PPAL construction based on bilinear maps, its performance and security properties are given in Sect. 6. In Table 1, we compare our ZKL and PPAL construction with previous work in terms of performance and assumptions. We specifically indicate which constructions satisfy the zero-knowledge property. Our PPAL construction outperforms all previous work based on widely accepted assumptions [6,34] (the construction of [33] is based on a non-standard assumption).

3 Preliminaries

3.1 Data Type

We consider a *linearly ordered list* $\mathcal{L}$ as a data structure that the owner wishes to store with the server. A list is an ordered set of elements $\mathcal{L} = \{x_1, x_2, \ldots, x_n\}$, where each $x_i \in \{0,1\}^*, \forall x_1, x_2 \in \mathcal{L}, x_1 \neq x_2$ and either $x_1 < x_2$ or $x_2 < x_1$. Hence, $<$ is a strict order on elements of $\mathcal{L}$ that is irreflexive, asymmetric and transitive.

We denote the set of elements of the list $\mathcal{L}$ as $\mathsf{Elements}(\mathcal{L})$. A sublist of $\mathcal{L}$, δ, is defined as: $\delta = \{x \mid x \in \mathsf{Elements}(\mathcal{L})\}$. Note that the order of elements in δ may not follow the order of $\mathcal{L}$. We denote with $\pi_{\mathcal{L}}(\delta)$ the permutation of the elements of δ under the order of $\mathcal{L}$. $\mathcal{L}(x_i)$ denotes the membership of element x_i in $\mathcal{L}$, i.e., $\mathcal{L}(x_i) = \mathsf{true}$ if $x_i \in \mathcal{L}$ and $\mathcal{L}(x_i) = \mathsf{false}$ if $x_i \notin \mathcal{L}$. For all x_i such that $\mathcal{L}(x_i) = \mathsf{true}$, $\mathsf{rank}(\mathcal{L}, x_i)$ denotes the rank of element x_i in the list, $\mathcal{L}$.

3.2 Cryptographic Primitives

We now describe the cryptographic primitives that are used in our construction and cryptographic assumptions that underlie the security of our method. In particular, our zero knowledge list construction relies on homomorphic integer commitments, zero knowledge protocol to prove a number is non-negative and zero knowledge sets, while the construction for privacy preserving lists relies on bilinear aggregate signatures and n-Bilinear Diffie Hellman Inversion assumption.

Homomorphic Integer Commitment Scheme: We use a homomorphic integer commitment scheme $\mathsf{HomIntCom}$ that is statistically hiding and computationally binding [5,14]. The latter implies the existence of a trapdoor and, hence, can be used to "equivocate" a commitment (i.e., open the commitment to any message using the trapdoor). We denote a commitment to x as $C(x;r)$ where r is the randomness used for the commitment. For simplicity, we sometimes drop r from the notation and use $C(x)$ to denote the commitment to x. The *homomorphism* of the scheme is defined as $C(x+y) = C(x) \times C(y)$.

Proving an Integer is Non-negative in Zero-Knowledge: We use the following (interactive) protocol between a prover and a verifier: the prover sends a commitment c to an integer $x \geq 0$ to the verifier and proves in zero-knowledge that the committed integer is non-negative, without opening c. We denote this protocol as $\mathsf{P} \leftrightarrow \mathsf{V}(x, r : c = C(x; r) \wedge x \geq 0)$. As a concrete construction we use the protocol of [27] which is a Σ protocol, i.e., *honest verifier zero knowledge* and can be made non-interactive zero-knowledge (NIZK) in the random oracle model using Fiat-Shamir heuristic [15].

Zero Knowledge Set Scheme: Let D be a set of key value pairs. If (x, v) is a key, value pair of D, then we write $D(x) = v$ to denote v is the value corresponding to the key x. For the keys that are not present in D, $x \notin D$, we write $D(x) = \perp$. A Zero Knowledge Set scheme (ZKS) [29] consists of three probabilistic polynomial time algorithms, $\mathsf{ZKS} = (\mathsf{ZKSSetup}, \mathsf{ZKSProver} = (\mathsf{ZKSP}_1, \mathsf{ZKSP}_2), \mathsf{ZKSVerifier})$, and queries are of the form "is key x in D?". The $\mathsf{ZKSSetup}$ algorithm takes the security parameter as input and produces a public key for the scheme that both the prover ($\mathsf{ZKSProver}$) and the verifier ($\mathsf{ZKSVerifier}$) take as input. The prover, Prover, is a tuple of two algorithms: ZKSP_1 takes the security parameter, the public key, and the set D and produces a short digest commitment com for D. ZKSP_2 takes a query x and produces the value $v = D(x)$, and the corresponding proof of (non-)membership, proof_x. The verifier, $\mathsf{ZKSVerifier}$, takes the security parameter, the public key, com, a query x, an answer $D(x)$, and proof_x and returns a bit b, where $b = \mathsf{ACCEPT}/\mathsf{REJECT}$. For our construction of zero knowledge lists we pick a ZKS construction of [12] that is based on mercurial commitments.

Bilinear Aggregate Signature Scheme: Our PPAL scheme relies on bilinear aggregate signature scheme of Boneh *et al.* [4]. Given signatures $\sigma_1, \ldots, \sigma_n$ on *distinct* messages $M_1, \ldots, M_n$ from n distinct users $u_1, \ldots, u_n$, it is possible to aggregate these signatures into a single short signature σ such that it (and the n messages) convince the verifier that the n users indeed signed the n original messages (i.e., user i signed message M_i). We use the special case where a single user signs n *distinct* messages $M_1, \ldots, M_n$. The security requirement of an aggregate signature scheme guarantees that the aggregate signature σ is valid if and only if the aggregator used all σ_i's to construct it.

3.3 Hardness Assumption

Let p be a large k-bit prime where $k \in \mathbb{N}$ is a security parameter. Let $n \in \mathbb{N}$ be polynomial in k, $n = \mathsf{poly}(\mathsf{k})$. Let $e : G \times G \rightarrow G_1$ be a bilinear map where G and G_1 are groups of prime order p and g be a random generator of G. We denote a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ as an adversary who is running in time $\mathsf{poly}(\mathsf{k})$. We use $\mathcal{A}^{\mathsf{alg}(\mathsf{input},\ldots)}$ to show that an adversary $\mathcal{A}$ has an oracle access to an instantiation of an algorithm alg with first argument set to input and ... denoting that $\mathcal{A}$ can give arbitrary input for the rest of the arguments.

Definition 1 (n-Bilinear Diffie Hellman Inversion (n-BDHI) [3]). *Let s be a random element of $\mathbb{Z}_p^*$ and n be a positive integer. Then, for every PPT adversary $\mathcal{A}$ there exists a negligible function $\nu(.)$ such that:*
$Pr[s \xleftarrow{\$} \mathbb{Z}_p^*; y \leftarrow \mathcal{A}(\langle g, g^s, g^{s^2}, \ldots, g^{s^n}\rangle) : y = e(g,g)^{\frac{1}{s}}] \leq \nu(k)$.

4 Zero Knowledge List (ZKL)

We generalize the idea of consistent set membership queries [12,29] to support membership and order queries in *zero-knowledge* on a list with *no repeated elements*. More specifically, given a totally ordered list of unique elements $\mathcal{L} = \{y_1, y_2, \ldots, y_n\}$, we want to support non-interactively and in zero-knowledge, (proofs reveal nothing beyond the query answer, not even the size of the list) queries of the following form:

- Is $y_i \in \mathcal{L}$ or $y_i \notin \mathcal{L}$, i.e., $\mathcal{L}(y_i) = \mathsf{true}$ or $\mathcal{L}(y_i) = \mathsf{false}$?
- For two elements $y_i, y_j \in \mathcal{L}$, what is their relative order, i.e., $y_i < y_j$ or $y_j < y_i$ in $\mathcal{L}$?

We adopt the same adversarial model as in [12,29,32]. There are two parties: the *prover* and the *verifier*. The *prover* initially commits to a list of elements and makes the commitment public. We now formally describe the model and the security properties.

4.1 Model

A Zero Knowledge List scheme (ZKL) consists of three probabilistic polynomial time algorithms: $(\mathsf{Setup}, \mathsf{Prover} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{Verifier})$. The queries are of the form (δ, flag) where $\delta = \{z_1, \ldots, z_m\}, z_i \in \{0,1\}^*$, is a collection of elements, $\mathsf{flag} = 0$ denotes a (non-)membership query and $\mathsf{flag} = 1$ denotes an order query. In the following sections, we will use state to represent a variable that saves the current state of the algorithm (when it finishes execution).

$\mathsf{PK} \leftarrow \mathsf{Setup}(1^k)$ The Setup algorithm takes the security parameter as input and produces a public key PK for the scheme. The prover and the verifier both take as input the string PK that can be a random string (in which case, the protocol is in the common random string model) or have a specific structure (in which case the protocol is in the trusted parameters model).

$(\mathsf{com}, \mathsf{state}) \leftarrow \mathsf{P}_1(1^k, \mathsf{PK}, \mathcal{L})$ P_1 takes the security parameter, the public key PK and the list $\mathcal{L}$, and produces a short digest commitment com for the list.

$(\mathsf{member}, \mathsf{proof}_M, \mathsf{order}, \mathsf{proof}_O) \leftarrow \mathsf{P}_2(\mathsf{PK}, \mathsf{state}, \delta, \mathsf{flag})$ where $\delta = \{z_1, \ldots, z_m\}$ and flag denotes the type of query. P_2 produces the membership information of the queried elements, $\mathsf{member} = \{\mathcal{L}(z_1), \ldots, \mathcal{L}(z_m)\}$ and the proof of membership (and non-membership), proof_M. proof_O is set depending on flag:
$\mathsf{flag} = 0$: P_2 sets order and proof_O to $\perp$ and returns $(\mathsf{member}, \mathsf{proof}_M, \perp, \perp)$.
$\mathsf{flag} = 1$: Let $\tilde{\delta} = \{z_i \mid i \in [1,m] \wedge \mathcal{L}(z_i) = \mathsf{true}\}$. P_2 produces the correct list order among the elements of $\tilde{\delta}$, $\mathsf{order} = \pi_{\mathcal{L}}(\tilde{\delta})$, and the proof of the order, proof_O.

$b \leftarrow \mathsf{Verifier}(1^k, \mathsf{PK}, \mathsf{com}, \delta, \mathsf{flag}, \mathsf{member}, \mathsf{proof}_M, \mathsf{order}, \mathsf{proof}_O)$ Verifier takes the security parameter, the public key PK, the commitment com and a query (δ, flag) and member, proof_M, order, proof_O and returns a bit b, where $b =$ ACCEPT/REJECT.

Example: Let us illustrate the above functionality with a small example. Let $\mathcal{L} = \{A, B, C\}$ and $(\delta, \mathsf{flag}) = (\{B, D, A\}, 1)$ be the query. Given this query P_2 returns $\mathsf{member} = \{\mathcal{L}(B), \mathcal{L}(D), \mathcal{L}(A)\} = \{\mathsf{true}, \mathsf{false}, \mathsf{true}\}$, the corresponding proofs of membership and non-membership in proof_M, $\mathsf{order} = \{A, B\}$ and the corresponding proof of order between A and B in proof_O.

4.2 Security Properties

Recall that the security properties of ZKL, *Completeness, Soundness* and *Zero-Knowledge*, guarantee security against malicious prover and verifier. Completeness mandates that honestly generated proofs always satisfy the verification test. Soundness states that the prover should not be able to come up with a query, and corresponding inconsistent (with the initial commitment) answers and convincing proofs. Finally, zero-knowledge ensures that each proof reveals the answer and nothing else.

Definition 2 (Completeness). *For every list* $\mathcal{L}$*, every query* δ *and every* flag*,*

$$\begin{aligned}&\Pr[\mathsf{PK} \leftarrow \mathsf{Setup}(1^k);\\ &(\mathsf{com}, \mathsf{state}) \leftarrow \mathsf{P}_1(1^k, \mathsf{PK}, \mathcal{L});\\ &(\mathsf{member}, \mathsf{proof}_M, \mathsf{order}, \mathsf{proof}_O) \leftarrow \mathsf{P}_2(\mathsf{PK}, \mathsf{state}, \delta, \mathsf{flag}) :\\ &\mathsf{Verifier}(1^k, \mathsf{PK}, \mathsf{com}, \delta, \mathsf{flag}, \mathsf{member}, \mathsf{proof}_M, \mathsf{order}, \mathsf{proof}_O) = \mathsf{ACCEPT}] = 1\end{aligned}$$

Definition 3 (Soundness). *For every PPT malicious prover algorithm,* Adv*, for every query* δ *and for every* flag *there exists a negligible function* $\nu(.)$ *such that:*

$$\begin{aligned}&\Pr[\mathsf{PK} \leftarrow \mathsf{Setup}(1^k);\\ &(\mathsf{com}, \mathsf{member}^1, \mathsf{proof}_M^1, \mathsf{order}^1, \mathsf{proof}_O^1, \mathsf{member}^2,\\ &\mathsf{proof}_M^2, \mathsf{order}^2, \mathsf{proof}_O^2) \leftarrow \mathsf{Adv}(1^k, \mathsf{PK}) :\\ &\mathsf{Verifier}(1^k, \mathsf{PK}, \mathsf{com}, \delta, \mathsf{flag}, \mathsf{member}^1, \mathsf{proof}_M^1, \mathsf{order}^1, \mathsf{proof}_O^1) = \mathsf{ACCEPT} \wedge\\ &\mathsf{Verifier}(1^k, \mathsf{PK}, \mathsf{com}, \delta, \mathsf{flag}, \mathsf{member}^2, \mathsf{proof}_M^2, \mathsf{order}^2, \mathsf{proof}_O^2) = \mathsf{ACCEPT} \wedge\\ &((\mathsf{member}^1 \neq \mathsf{member}^2) \vee (\mathsf{order}^1 \neq \mathsf{order}^2))] \leq \nu(k)\end{aligned}$$

Definition 4 (Zero-Knowledge). *There exists a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ *such that for every PPT malicious verifier* $\mathsf{Adv} =$

$(\mathsf{Adv}_1, \mathsf{Adv}_2)$, *there exists a negligible function* $\nu(.)$ *such that:*

$$
\begin{aligned}
|\Pr[&\mathsf{PK} \leftarrow \mathsf{Setup}(1^k); (\mathcal{L}, \mathsf{state}_A) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{PK});\\
&(\mathsf{com}, \mathsf{state}_P) \leftarrow \mathsf{P}_1(1^k, \mathsf{PK}, \mathcal{L}) :\\
&\mathsf{Adv}_2^{\mathsf{P}_2(\mathsf{PK}, \mathsf{state}_P, \cdot)}(\mathsf{com}, \mathsf{state}_A) = 1] -\\
\Pr[&(\mathsf{PK}, \mathsf{state}_S) \leftarrow \mathsf{Sim}_1(1^k); (\mathcal{L}, \mathsf{state}_A) \leftarrow \mathsf{Adv}_1(1^k, \mathsf{PK});\\
&(\mathsf{com}, \mathsf{state}_S) \leftarrow \mathsf{Sim}_2(1^k, \mathsf{state}_S) :\\
&\mathsf{Adv}_2^{\mathsf{Sim}_3^{\mathcal{L}}(1^k, \mathsf{state}_S)}(\mathsf{com}, \mathsf{state}_A) = 1]| \leq \nu(k),
\end{aligned}
$$

where Sim_3 *has oracle access to* $\mathcal{L}$*, that is, given a query* (δ, flag)*,* Sim_3 *can query the list* $\mathcal{L}$ *to learn only the membership/non-membership of elements in* δ *and, if* $\mathsf{flag} = 1$*, learn the list order of the elements of* δ *in* $\mathcal{L}$*.*

4.3 ZKL Construction

The construction uses zero knowledge set scheme, homomorphic integer commitment scheme, zero-knowledge protocol to prove non-negativity of an integer and a collision resistant hash function $\mathbb{H} : \{0,1\}^* \rightarrow \{0,1\}^l$, if the elements of the list $\mathcal{L}$ are larger that l bits. In particular, given an input list $\mathcal{L}$ the prover P_1 creates a set D where for every element $y_j \in \mathcal{L}$ it adds a (key,value) pair $(\mathbb{H}(y_j), C(j))$. $\mathbb{H}(y_j)$ is a hash of y_j and $C(j)$ is a homomorphic integer commitment of $\mathsf{rank}(\mathcal{L}, y_j)$ (assuming $\mathsf{rank}(\mathcal{L}, y_j) = j$, wlog). P_1 sets up a zero knowledge set on D using ZKSP_1 from a zero-knowledge set scheme $\mathsf{ZKS} = (\mathsf{ZKSSetup}, \mathsf{ZKSProver} = (\mathsf{ZKSP}_1, \mathsf{ZKSP}_2), \mathsf{ZKSVerifier})$ [12]. The output of ZKSP_1 is a commitment to D, com, that P_1 sends to the verifier.

P_2 operates as follows. Membership and non-membership queries of the form $(\delta, 0)$ are replied in the same fashion as in zero knowledge set, by invoking ZKSP_2 on the hash of every element of sublist δ. Recall that as a response to a membership query for a key, ZKSP_2 returns the value corresponding to this key. In our case, the queried key is $\mathbb{H}(y_j)$ and the value returned by $\mathsf{ZKSP}_2, D(\mathbb{H}(y_j))$, is the commitment $C(j)$ where j is the rank of element y_j in the list $\mathcal{L}$, if $y_j \in \mathcal{L}$. If $y_j \notin \mathcal{L}$, the value returned is $\perp$. Hence, the verifier receives the commitments to ranks for queried member elements. These commitments are never opened but are used as part of order proofs.

For a given order query $(\delta, 1)$, for every adjacent pair of elements in the returned order, order, P_2 gives a proof of order. Recall that order contains the member elements of δ, arranged according to their order in the list $\mathcal{L}$. P_2 proves the order between two elements y_i and y_j as follows. Let $\mathsf{rank}(\mathcal{L}, y_i) = i, \mathsf{rank}(\mathcal{L}, y_j) = j$, and $C(i)$, $C(j)$ be the corresponding commitments and, wlog, let $i < j$. As noted above, $C(i)$ and $C(j)$ are already returned by P_2 as part of the membership proof. Additionally, P_2 returns a commitment to 1, $C(1)$, and its opening information ρ. Note that, the verifier can compute $C(1)$ himself, but then the prover needs $C(1)$ computed by the verifier, to be able to generate proof

for non-negativity of $C(j-i-1)$. To avoid this interaction, we make the prover send $C(1)$ and its opening.

The verification of the query answer proceeds as follows. Verifier computes $C(j-i-1) := C(j)/(C(i)C(1))$ using the homomorphic property of the integer commitment scheme. P_2 uses the zero knowledge protocol $\mathsf{P} \leftrightarrow \mathsf{V}(x, r : c = C(x;r) \wedge x \geq 0)$ to convince Verifier that $C(j-i-1)$ is a commitment to value ≥ 0. Note that we use the non-interactive general zero-knowledge version of the protocol as discussed in Sect. 3. Hence, the query phase proceeds in a single round.

We note that we require Verifier to verify that $j-i-1 \geq 0$ and not $j-i \geq 0$ since otherwise a cheating prover Adv can do the following: store the same arbitrary non-negative integer as a rank for every element in the list, hence, $C(j-i)$ and $C(i-j)$ are commitments to 0, and Adv can always succeed in proving an arbitrary order. However, an honest prover can always prove the non-negativity of $C(j-i-1)$ as $|j-i| \geq 1$ for any rank i, j of the list.

Also, we note that the commitments to ranks can be replaced by commitments to a strictly monotonic sequence as long as there is a 1:1 correspondence with the rank sequence. In this case, the distance between two elements will also be positive and, hence, the above protocol still holds.

Theorem 1. *The zero-knowledge list (ZKL) construction of Sect. 4.3 is a non-interactive two-party protocol that satisfies the security properties of completeness (Definition 2), soundness (Definition 3) and zero-knowledge (Definition 4) in the random oracle model (inherited from NIZK). The construction has the following performance, where n is the list size, m is the query size, each element of the list is a k-bit (if not, we can use a hash function to reduce every element to a k-bit string, as shown in the construction).*

- *The prover executes the commitment phase in $O(nk)$ time and space, where the multiplicative factor k is inherited from the height of the tree.*
- *In the query phase, the prover computes the proof of the answer in $O(mk)$ time.*
- *The verifier verifies the proof in $O(mk)$ time and space.*

The soundness of the ZKL scheme follows from the soundness of the ZKS scheme, the binding property of the commitment scheme, and the correctness of protocol $\mathsf{P} \leftrightarrow \mathsf{V}(x, r : c = C(x;r) \wedge x \geq 0)$ (see Sect. 3.2). For the zero-knowledge property, we write a simulator that uses the ZKS simulator and the trapdoor of the commitment scheme to equivocate commitments. The formal proof of Theorem 1 is omitted due to space restrictions and is presented in [17].

5 Privacy Preserving Authenticated List (PPAL)

In the previous section we presented a model and a construction for a new primitive called zero knowledge lists. As we noticed earlier, ZKL model gives the desired functionality to verify order queries on lists. However, the corresponding

construction does not provide the efficiency one may desire in cloud computing setting where the verifier (client) has limited memory resources as we discuss in Sect. 5.3. In this section we address this setting and define a model for privacy preserving authenticated lists, PPAL, that is executed between three parties. This model, arguably, fits cloud scenario better and, as we will see, our construction is also more efficient.

5.1 Model

PPAL is a tuple of three probabilistic polynomial time algorithms $(\mathsf{Setup}, \mathsf{Query}, \mathsf{Verify})$ executed between the owner of the data list $\mathcal{L}$, the server who stores $\mathcal{L}$ and answers queries from the client and the client who issues queries on the elements of the list and verifies corresponding answers. We note that this model assumes that the query is on the member elements of the list, i.e., for any query, δ, $\mathsf{Elements}(\delta) \subseteq \mathsf{Elements}(\mathcal{L})$. In other words, this model does not support proofs of non-membership, similar to other data structures that support only positive membership proofs, e.g., [6,8,9,11,23,24,33].

$(\mathsf{digest}_C, \mathsf{digest}_S) \leftarrow \mathsf{Setup}(1^k, \mathcal{L})$ This algorithm takes the security parameter and the source list $\mathcal{L}$ as input and produces two digests digest_C and digest_S for the list. This algorithm is run by the owner. digest_C is sent to the client and digest_S is sent to the server.

$(\mathsf{order}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S, \mathcal{L}, \delta)$ This algorithm takes the server digest generated by the owner, digest_S, the source list, $\mathcal{L}$, and a queried sublist, δ, as input, where a sublist of a list $\mathcal{L}$ is defined as: $\mathsf{Elements}(\delta) \subseteq \mathsf{Elements}(\mathcal{L})$. The algorithm produces the list order of the elements of $\mathcal{L}$, $\mathsf{order} = \pi_{\mathcal{L}}(\delta)$, and a proof, proof, of the answer. This algorithm is run by the server. Wlog, we assume $|\delta| > 1$. In the trivial case of $|\delta| = 1$, the server returns an empty proof, i.e., $(\mathsf{order} = \delta, \mathsf{proof} = \perp)$.

$b \leftarrow \mathsf{Verify}(\mathsf{digest}_C, \delta, \mathsf{order}, \mathsf{proof})$ This algorithm takes digest_C, a queried sublist δ, order and proof and returns a bit b, where $b = \mathsf{ACCEPT}$ iff $\mathsf{Elements}(\delta) \subseteq \mathsf{Elements}(\mathcal{L})$ and $\mathsf{order} = \pi_{\mathcal{L}}(\delta)$. Otherwise, $b = \mathsf{REJECT}$. This algorithm is run by the client.

5.2 Security Properties

A PPAL has three important security properties. Recall that the properties of PPAL, *Completeness, Soundness* and *Zero-Knowledge*, guarantee security against malicious server and client. They are close to the ones of ZKL except for soundness. For PPAL it enforces that the client does not accept proofs forged by the server for incorrect answers w.r.t. owner's list. We describe each security definition formally below.

The first property is *Completeness*. This property ensures that for any list $\mathcal{L}$ and for any sublist δ of $\mathcal{L}$, if $\mathsf{digest}_C, \mathsf{digest}_S, \mathsf{order}, \mathsf{proof}$ are generated honestly, i.e., the owner and the server honestly execute the protocol, then the client will be always convinced about the correct list order of δ.

Definition 5 (Completeness). *For all lists $\mathcal{L}$ and all sublists δ of $\mathcal{L}$*

$$\Pr[(\mathsf{digest}_C, \mathsf{digest}_S) \leftarrow \mathsf{Setup}(1^k, \mathcal{L}); (\mathsf{order}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S, \mathcal{L}, \delta) : \mathsf{Verify}(\mathsf{digest}_C, \delta, \mathsf{order}, \mathsf{proof}) = \mathsf{ACCEPT} \wedge \mathsf{order} = \pi_{\mathcal{L}}(\delta)] = 1$$

The second security property is *Soundness*. This property ensures that once an honest owner generates a pair $(\mathsf{digest}_C, \mathsf{digest}_S)$ for a list $\mathcal{L}$, even a malicious server will not be able to convince the client of an incorrect order of elements belonging to the list $\mathcal{L}$. This property ensures integrity of the scheme.

Definition 6 (Soundness). *For all PPT malicious query algorithms* Adv*, for all lists $\mathcal{L}$ and all query sublists δ of $\mathcal{L}$, there exists a negligible function $\nu(.)$ such that:*

$$\Pr[(\mathsf{digest}_C, \mathsf{digest}_S) \leftarrow \mathsf{Setup}(1^k, \mathcal{L}); (\mathsf{order}, \mathsf{proof}) \leftarrow \mathsf{Adv}(\mathsf{digest}_S, \mathcal{L}) : \mathsf{Verify}(\mathsf{digest}_C, \delta, \mathsf{order}, \mathsf{proof}) = \mathsf{ACCEPT} \wedge \mathsf{order} \neq \pi_{\mathcal{L}}(\delta)] \leq \nu(k)$$

The last property is *Zero-Knowledge*. This property captures that even a malicious client cannot learn anything about the list (and its size) beyond what the client has queried for. Informally, this property involves showing that there exists a simulator such that even for adversarially chosen list $\mathcal{L}$, no adversarial client (verifier) can tell if it is talking to a honest owner and honest server who know $\mathcal{L}$ and answer w.r.t. $\mathcal{L}$, or to the simulator that only has oracle access to the list $\mathcal{L}$.

Definition 7 (Zero-Knowledge). *There exists a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that for all PPT malicious verifiers* $\mathsf{Adv} = (\mathsf{Adv}_1, \mathsf{Adv}_2)$*, there exists a negligible function $\nu(.)$ such that:*

$$\begin{aligned} |\Pr[(\mathcal{L}, \mathsf{state}_A) \leftarrow \mathsf{Adv}_1(1^k); (\mathsf{digest}_C, \mathsf{digest}_S) \leftarrow \mathsf{Setup}(1^k, \mathcal{L}) : \\ \mathsf{Adv}_2^{\mathsf{Query}(\mathsf{digest}_S, \mathcal{L}, .)}(\mathsf{digest}_C, \mathsf{state}_A) = 1] - \\ \Pr[(\mathcal{L}, \mathsf{state}_A) \leftarrow \mathsf{Adv}_1(1^k); (\mathsf{digest}_C, \mathsf{state}_S) \leftarrow \mathsf{Sim}_1(1^k) : \\ \mathsf{Adv}_2^{\mathsf{Sim}_2^{\mathcal{L}}(1^k, \mathsf{state}_S)}(\mathsf{digest}_C, \mathsf{state}_A) = 1]| \leq \nu(k) \end{aligned}$$

Here Sim_2 has oracle access to $\mathcal{L}$, that is given a sublist δ of $\mathcal{L}$, Sim_2 can query the list $\mathcal{L}$ to learn only the correct list order of the sublist δ and cannot look at $\mathcal{L}$.

5.3 Construction of PPAL via ZKL

We show how a PPAL can be instantiated via a ZKL in Theorems 2 and 3 and then discuss that the resulting construction does not yield the desired efficiency.

Theorem 2. *Given a non-interactive ZKL scheme* $\mathsf{ZKL} = (\mathsf{Setup}, \mathsf{Prover} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{Verifier})$*, which supports queries of the form* (δ, flag) *on a list $\mathcal{L}$, we can instantiate a PPAL scheme for the list $\mathcal{L}$,* $\mathsf{PPAL} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Verify})$*, which supports queries of the form δ, where δ is a sublist of $\mathcal{L}$, as follows:*

$\mathsf{PPAL.Setup}(1^k, \mathcal{L})$*: Invoke* $\mathsf{PK} \leftarrow \mathsf{ZKL.Setup}(1^k)$ *and* $(\mathsf{com}, \mathsf{state}) \leftarrow \mathsf{ZKL}.\mathsf{P}_1(1^k, \mathsf{PK}, \mathcal{L})$*. Return* $(\mathsf{digest}_C = (\mathsf{PK}, \mathsf{com}), \mathsf{digest}_S = (\mathsf{PK}, \mathsf{com}, \mathsf{state}))$.

$\mathsf{PPAL.Query}(\mathsf{digest}_S, \mathcal{L}, \delta)$*: Invoke* $(\mathsf{member}, \mathsf{proof}_M, \mathsf{order}, \mathsf{proof}_O) \leftarrow \mathsf{ZKL}.\mathsf{P}_2(\mathsf{PK}, \mathsf{state}, \delta, 1)$*. Return* $(\mathsf{order}, \mathsf{proof} = (\mathsf{proof}_M, \mathsf{proof}_O))$.

$\mathsf{PPAL.Verify}(\mathsf{digest}_C, \delta, \mathsf{order}, \mathsf{proof}_M, \mathsf{proof}_O)$*: Set* $\mathsf{member} = \{1, 1, \ldots, 1\}$ *such that* $|\mathsf{member}| = |\delta| = |\mathsf{order}|$*. Return bit* b *where* $b \leftarrow \mathsf{ZKL.Verifier}(1^k, \mathsf{PK}, \mathsf{com}, \delta, 1, \mathsf{member}, \mathsf{proof}_M, \mathsf{order}, \mathsf{proof}_O)$.

Theorem 3. *A PPAL scheme instantiated using a ZKL scheme,* $\mathsf{ZKL} = (\mathsf{Setup}, \mathsf{Prover} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{Verifier})$ *has the following performance:*

- *The owner's runtime and space are proportional to the runtime and space of* $\mathsf{ZKL.Setup}$ *and* $\mathsf{ZKL.P}_1$*, respectively.*
- *The server's runtime and space are proportional to the runtime and space of* $\mathsf{ZKL.P}_2$.
- *The client's runtime and space are proportional to the runtime and space of* $\mathsf{ZKL.Verifier}$.

The correctness of Theorems 2 and 3 follow from the definition of PPAL and ZKL models. In a PPAL instantiated with the ZKL construction of Sect. 4, the owner runs in time and space $O(kn)$ and the server requires space $O(kn)$, where n is the list size and each element of the list is k-bits long. To answer a query of size m, the server runs in time $O(km)$ and the verification time of the client is $O(km)$.

As we see, this generic construction is not very efficient due to the multiplicative factor $O(k)$ and heavy cryptographic primitives. In Sect. 6, we present a direct PPAL construction which is a factor of $O(k)$ more efficient in space and computation requirements as compared to an adaptation of our ZKL construction from Sect. 4.

6 PPAL Construction

We start by presenting the intuition behind our construction of a privacy preserving authenticated list (PPAL). Next, we give more details on the algorithms and analyze the security and efficiency of the construction.

Intuition: Every element of the list is associated with a member witness where a member witness is a blinded component of the bilinear accumulator public key. This allows us to encode the rank of the element in the member witness and then "blind" rank information with randomness. Every pair of (element, member witness) is signed by the owner and the signatures are aggregated using bilinear aggregate signature scheme [4], to compute the list digest signature. Signatures and digest are sent to the server, who can use them to prove authenticity when answering client queries. The owner also sends the list digest signature and the public key of the bilinear aggregate signature scheme to the client. The advantage of using an aggregate signature is for the server to be able to compute a valid

digest signature for any sublist of the source list by exploiting the homomorphic nature of aggregate signatures, that is without owner's involvement. Moreover, the client can verify the individual signatures in a single invocation to aggregate signature verification.

The owner also sends to the server linear (in the list size) number of random elements used in the encoding of member witnesses. These random elements allow the server to compute the order witnesses on queried elements, without the owner's involvement. The order witness encodes the distance between two elements, i.e., the difference between element ranks, without revealing anything about it. Together with member witnesses, the client can later use bilinear map to verify the order of the elements.

Construction: Our construction for PPAL is presented in Fig. 1. We denote *member witness* for $x_i \in \mathcal{L}$ as $t_{x_i \in \mathcal{L}}$. For two elements $x_i, x_j \in \mathcal{L}$, such that $x_i < x_j$ in $\mathcal{L}$, $t_{x_i < x_j}$ is an *order witness* for the order between x_i and x_j. The construction works as follows.

In the Setup phase, the owner generates secret key (v, s) and public key g^v, where v is used for signatures. The owner picks a distinct random element r_i from the group $\mathbb{Z}_p^*$ for each element x_i in the list $\mathcal{L}, i \in [1, n]$. The element r_i is used to compute the member witness $t_{x_i \in \mathcal{L}}$. Later in the protocol, together with r_j, it is also used by the server to compute the order witness $t_{x_i < x_j}$. The owner also computes individual signatures, σ_i's, for each element and aggregates them into a digest signature $\sigma_{\mathcal{L}}$ for the list. It returns the signatures and member witnesses for every element of $\mathcal{L}$ in $\Sigma_{\mathcal{L}}$ and the set of random numbers picked for each index to be used in order witnesses in $\Omega_{\mathcal{L}}$. The owner sends $\mathsf{digest}_C = (g^v, \sigma_{\mathcal{L}})$ to the client and $\mathsf{digest}_S = (g^v, \sigma_{\mathcal{L}}, \langle g, g^s, g^{s^2}, \ldots, g^{s^n} \rangle, \Sigma_{\mathcal{L}}, \Omega_{\mathcal{L}})$ and $\mathcal{L}$ to the server.

Given a query δ, the server returns a response list order that contains elements of δ in the order they appear in $\mathcal{L}$. The server uses information in $\Sigma_{\mathcal{L}}$ to compute the digest signature for the sublist, σ_{order}, and its membership verification unit $\lambda_{\mathcal{L}'}$ which are part of the Σ_{order} component of the proof. To compute the Ω_{order} component of the proof, the server uses corresponding blinding values in $\Omega_{\mathcal{L}}$ and elements g^{s^d} where d's correspond to distances between ranks of queried elements.

The client first checks that all the returned elements are signed by the owner using Σ_{order} and then verifies the order of the returned elements using Ω_{order}. Hence, the client uses bilinear map for two purposes: first for member verification and then to verify the order. The query phase has a single round of communication between client and server.

We now describe the preprocessing step at the server that reduces the query time for a query of size m on a list of size n from $O(n)$ to $O(\min\{m \log n, n\})$. Let $\psi_i = \mathcal{H}(t_{x_i \in \mathcal{L}} || x_i)$ for $x_i \in \mathcal{L}$. The server computes and stores a balanced binary tree over n leaves, where the ith leaf corresponds to x_i and stores ψ_i. Each internal node of the tree stores the product of the values at its children. When answering a query of size m, the server can compute $\lambda_{\mathcal{L}'}$ by using partial products that correspond to intervals between elements in the query. There are $m + 1$ such partial products. Since each partial product can be computed using

Notation: $k \in \mathbb{N}$ is the security parameter of the scheme; G, G_1 multiplicative cyclic groups of prime order p where p is large k-bit prime; g: a random generator of G; e: computable bilinear nondegenerate map $e : G \times G \to G_1$; $\mathcal{H} : \{0,1\}^* \to G$: full domain hash function (instantiated with a cryptographic hash function); all arithmetic operations are performed using mod p. $\mathcal{L}$ is the input list of size $n = \mathsf{poly}(\mathsf{k})$, where x_i's are distinct and $\mathsf{rank}(\mathcal{L}, x_i) = i$. System parameters are $(p, G, G_1, e, g, \mathcal{H})$.

$(\mathsf{digest}_C, \mathsf{digest}_S) \leftarrow \mathsf{Setup}(1^k, \mathcal{L})$, where

$\mathcal{L}$ is the input list of length n;

$\mathsf{digest}_C = (g^v, \sigma_{\mathcal{L}})$;

$\mathsf{digest}_S = (g^v, \sigma_{\mathcal{L}}, \langle g, g^s, g^{s^2}, \ldots, g^{s^n} \rangle, \Sigma_{\mathcal{L}}, \Omega_{\mathcal{L}})$ and

$\langle s \xleftarrow{\$} \mathbb{Z}_p^*, v \xleftarrow{\$} \mathbb{Z}_p^* \rangle$ is the secret key of the owner;

$\Sigma_{\mathcal{L}} = \langle \{t_{x_i \in \mathcal{L}}, \sigma_i\}_{1 \le i \le n}, \mathcal{H}(\omega) \rangle$ is member authentication information and ω is the list nonce;

$\Omega_{\mathcal{L}} = \langle r_1, r_2, \ldots, r_n \rangle, r_i \neq r_j$ for $i \neq j$, is order authentication information;

$\sigma_{\mathcal{L}}$ is the digest signature of the list $\mathcal{L}$.

These elements are computed as follows:

For every element x_i in $\mathcal{L} = \{x_1, \ldots, x_n\}$: Pick $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. Compute member witness for index i as $t_{x_i \in \mathcal{L}} \leftarrow (g^{s^i})^{r_i}$ and signature for element x_i as $\sigma_i \leftarrow \mathcal{H}(t_{x_i \in \mathcal{L}} || x_i)^v$.

Pick the nonce, $\omega \xleftarrow{\$} \{0,1\}^*$, which should be unique for each list.

Set $\mathsf{salt} \leftarrow (\mathcal{H}(\omega))^v$. salt is treated as a list identifier which protects against mix-and-match attack and also protects from the leakage that the queried result is the complete list.

The list digest signature is computed as: $\sigma_{\mathcal{L}} \leftarrow \mathsf{salt} \times \prod_{1 \le i \le n} \sigma_i$.

$(\mathsf{order}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S, \mathcal{L}, \delta)$, where

$\delta = \{z_1, \ldots, z_m\}$ s.t. $z_i \in \mathcal{L}$, $\forall i \in [1, m]$, is the queried sublist;

$\mathsf{order} = \pi_{\mathcal{L}}(\delta) = \{y_1, \ldots, y_m\}$;

$\mathsf{proof} = (\Sigma_{\mathsf{order}}, \Omega_{\mathsf{order}})$:

$\Sigma_{\mathsf{order}} = (\sigma_{\mathsf{order}}, T, \lambda_{\mathcal{L}'})$ where $\mathcal{L}' = \mathcal{L} \setminus \delta$;

$T = \{t_{y_1 \in \mathcal{L}}, \ldots, t_{y_m \in \mathcal{L}}\}$;

$\Omega_{\mathsf{order}} = \{t_{y_1 < y_2}, t_{y_2 < y_3}, \ldots, t_{y_{m-1} < y_m}\}$.

These elements are computed as follows:

The digest signature for the sublist: $\sigma_{\mathsf{order}} \leftarrow \prod_{y_j \in \mathsf{order}} \sigma_{\mathsf{rank}(\mathcal{L}, y_j)}$.

The member verification unit: $\lambda_{\mathcal{L}'} \leftarrow \mathcal{H}(\omega) \times \prod_{x \in \mathcal{L}'} \mathcal{H}(t_{x_{\mathsf{rank}(\mathcal{L}, x)} \in \mathcal{L}} || x)$.

For every $j \in [1, m-1]$: Let $i' = \mathsf{rank}(\mathcal{L}, y_j)$ and $i'' = \mathsf{rank}(\mathcal{L}, y_{j+1})$, and $r' = \Omega_{\mathcal{L}}[i']^{-1}$ and $r'' = \Omega_{\mathcal{L}}[i'']$. Compute $t_{y_j < y_{j+1}} \leftarrow (g^{s^d})^{r'r''}$ where $d = |i' - i''|$.

$b \leftarrow \mathsf{Verify}(\mathsf{digest}_C, \delta, \mathsf{order}, \mathsf{proof})$ where $\mathsf{digest}_C, \delta, \mathsf{order}, \mathsf{proof}$ are defined as above.

The algorithm checks the following:

- Compute $\xi \leftarrow \prod_{y_j \in \delta} \mathcal{H}(t_{y_j \in \mathcal{L}} || y_j)$ and check if $e(\sigma_{\mathsf{order}}, g) \stackrel{?}{=} e(\xi, g^v)$.
- Check if $e(\sigma_{\mathcal{L}}, g) \stackrel{?}{=} e(\sigma_{\mathsf{order}}, g) \times e(\lambda_{\mathcal{L}'}, g^v)$.
- For every $j \in [1, m-1]$, $e(t_{y_j \in \mathcal{L}}, t_{y_j < y_{j+1}}) \stackrel{?}{=} e(t_{y_{j+1} \in \mathcal{L}}, g)$.

Return ACCEPT iff all the equalities of the three steps verify, and REJECT, otherwise.

Fig. 1. Privacy-preserving authenticated list (PPAL) construction

$O(\log n)$ precomputed products in the tree, it takes $O(m \log n)$ time to compute the product of $m+1$ of them. The server takes $O(n)$ for preprocessing and the query time is reduced to $O(\min\{m \log n, n\})$.

We summarize the properties and efficiency of our PPAL construction in Theorem 4.

Theorem 4. *The privacy-preserving authenticated list (PPAL) construction of Fig. 1 satisfies the security properties of completeness (Definition 5), soundness (Definition 6) and zero-knowledge (Definition 7) in the random oracle model (inherited from [4]) and under the n-BDHI assumption (Definition 1). Also, the construction has the following performance, where n denotes the list size and m denotes the query size.*

- *The owner and the server use $O(n)$ space.*
- *The owner performs the setup phase in $O(n)$ time and goes offline.*
- *The server performs the preprocessing phase in $O(n)$ time.*
- *Query phase is a single-round protocol between the server and the client.*
- *The server computes the answer to a query and its proof in $O(\min\{m \log n, n\})$ time.*
- *The client verifies the proof in $O(m)$ time and space.*

The formal proof is omitted due to space restrictions and is available in [17]. Here we highlight the proof of soundness and zero knowledge. To prove soundness, we assume that there exists a malicious server Adv, which forges the order on a non-trivial sublist $\delta = \{x_1, \ldots, x_m\}$, where $m \geq 2$, for a list $\mathcal{L}$. Then there exists at least one inversion pair (x_i, x_j) whose order is flipped in Adv's forgery. Wlog assume that $u < v$ where $u = \mathsf{rank}(\mathcal{L}, x_i)$ and $v = \mathsf{rank}(\mathcal{L}, x_j)$. Then Adv must have forged the witness $t_{x_j < x_i} = (g^{s^{(u-v)}})^{r_1 r_2^{-1}}$ that passes the verification, where $r_1, r_2 \in \mathbb{Z}_p^*$ are the blinded components of elements x_i and x_j, respectively. We show that by invoking Adv and using its forged witness $t_{x_j < x_i}$, we can construct a PPT adversary that successfully breaks the n-BDHI hardness assumption [3] by outputting $e\left(t_{x_j < x_i}, (g^{s^{v-u-1}})^{r_1{}^{-1} r_2}\right) = e(g,g)^{\frac{1}{s}}$, where $g^{s^{v-u-1}}$ is part of the input to the n-BDHI problem.

For the zero knowledge property, we write a simulator that can produce witnesses identically distributed to real witnesses by giving it only oracle access to the list, and using the fact that our PPAL construction uses witnesses blinded in their exponents.

Acknowledgment. This research was supported in part by the National Science Foundation under grant CNS–1228485. Olga Ohrimenko worked on this project in part while at Brown University. We are grateful to Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Claire Mathieu for useful discussions and for their feedback on early drafts of this work. We would also like to thank Ashish Kundu for introducing us to his work on structural signatures and Jia Xu for sharing a paper through personal communication.

References

1. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 1–20. Springer, Heidelberg (2012)
2. Attrapadung, N., Libert, B., Peters, T.: Computing on authenticated data: new privacy definitions and constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 367–385. Springer, Heidelberg (2012)
3. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
4. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
5. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)
6. Brzuska, C., et al.: Redactable signatures for tree-structured data: definitions and constructions. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 87–104. Springer, Heidelberg (2010)
7. Camacho, P., Hevia, A.: Short transitive signatures for directed trees. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 35–50. Springer, Heidelberg (2012)
8. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 481–500. Springer, Heidelberg (2009)
9. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Hanaoka, G., Kurosawa, K. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Heidelberg (2013)
10. Catalano, D., Fiore, D., Messina, M.: Zero-knowledge sets with short proofs. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 433–450. Springer, Heidelberg (2008)
11. Chang, E.-C., Lim, C.L., Xu, J.: Short redactable signatures using random trees. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 133–147. Springer, Heidelberg (2009)
12. Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial commitments with applications to zero-knowledge sets. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 422–439. Springer, Heidelberg (2005)
13. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable signatures: complex unary transformations and delegatable anonymous credentials. IACR Cryptology ePrint Archive 2013/179 (2013)
14. Damgård, I.B., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002)
15. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
16. Ghosh, E., Goodrich, M.T., Ohrimenko, O., Tamassia, R.: Fully-dynamic verifiable zero-knowledge order queries for network data. IACR Cryptology ePrint Archive 2015/283 (2015)

17. Ghosh, E., Ohrimenko, O., Tamassia, R.: Verifiable member and order queries on a list in zero-knowledge. IACR Cryptology ePrint Archive 2014/632 (2014)
18. Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., Ziv, A.: NSEC5: provably preventing DNSSEC zone enumeration. Cryptology ePrint Archive, Report 2014/582 (2014)
19. Goodrich, M.T., Nguyen, D., Ohrimenko, O., Papamanthou, C., Tamassia, R., Triandopoulos, N., Lopes, C.V.: Efficient verification of web-content searching through authenticated web crawlers. PVLDB **5**(10), 920–931 (2012)
20. Goodrich, M.T., Tamassia, R., Schwerin, A.: Implementation of an authenticated dictionary with skip lists and commutative hashing. In: Proceedings of the DARPA Information Survivability Conference and Exposition II (2001)
21. Johnson, R., Molnar, D., Song, D., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002)
22. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010)
23. Kundu, A., Atallah, M.J., Bertino, E.: Leakage-free redactable signatures. In: Proceedings of the CODASPY (2012)
24. Kundu, A., Bertino, E.: Structural signatures for tree data structures. PVLDB **1**(1), 138–150 (2008)
25. Kundu, A., Bertino, E.: Privacy-preserving authentication of trees and graphs. Int. J. Inf. Sec. **12**(6), 467–494 (2013)
26. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 499–517. Springer, Heidelberg (2010)
27. Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)
28. Merkle, R.C.: Protocols for public key cryptosystems. In: Proceedings of the IEEE Symposium on Security and Privacy (1980)
29. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: Proceedings of the FOCS (2003)
30. Micali, S., Rivest, R.L.: Transitive signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 236–243. Springer, Heidelberg (2002)
31. Miyazaki, K., Hanaoka, G., Imai, H.: Digitally signed document sanitizing scheme based on bilinear maps. In: Proceedings of the ASIACCS (2006)
32. Ostrovsky, R., Rackoff, C., Smith, A.: Efficient consistency proofs for generalized queries on a committed database. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 1041–1053. Springer, Heidelberg (2004)
33. Poehls, H.C., Samelin, K., Posegga, J., De Meer, H.: Length-hiding redactable signatures from one-way accumulators in $O(n)$. Technical report MIP-1201, Faculty of Computer Science and Mathematics (FIM), University of Passau (2012)
34. Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: Redactable signatures for independent removal of structure and content. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 17–33. Springer, Heidelberg (2012)

35. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: Kim, K. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 285–304. Springer, Heidelberg (2002)
36. Tamassia, R.: Authenticated data structures. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 2–5. Springer, Heidelberg (2003)
37. Wang, Z.: Improvement on Ahn et al.'s RSA P-homomorphic signature scheme. In: Keromytis, A.D., Di Pietro, R. (eds.) SecureComm 2012. LNICST, vol. 106, pp. 19–28. Springer, Heidelberg (2013)
38. Yi, X.: Directed transitive signature scheme. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 129–144. Springer, Heidelberg (2006)