Esha Ghosh, Olga Ohrimenko, and Roberto Tamassia

Efficient Verifiable Range and Closest Point Queries in Zero-Knowledge

Abstract: We present an efficient method for answering one-dimensional range and closest-point queries in a verifiable and privacy-preserving manner. We consider a model where a data owner outsources a dataset of key-value pairs to a server, who answers range and closest-point queries issued by a client and provides proofs of the answers. The client verifies the correctness of the answers while learning nothing about the dataset besides the answers to the current and previous queries. Our work yields for the first time a zero-knowledge privacy assurance to authenticated range and closest-point queries. Previous work leaked the size of the dataset and used an inefficient proof protocol. Our construction is based on hierarchical identity-based encryption. We prove its security and analyze its efficiency both theoretically and with experiments on synthetic and real data (Enron email and Boston taxi datasets).

Keywords: cloud privacy, security, efficient outsourced computation, public verifiability, zero-knowledge authenticated range queries, hierarchical identity based encryption

DOI 10.1515/popets-2016-0045
Received 2016-02-29; revised 2016-06-02; accepted 2016-06-02.

1 Introduction

In this work, we consider the problem of verifiably answering range queries on a key-value store $\mathcal{D}$ while hiding the rest of $\mathcal{D}$'s content. That is, a range query $[a, b]$ on $\mathcal{D}$ has the following requirements:

- it returns the answer $\mathcal{D}'$ and a proof of its correctness, i.e., $\mathcal{D}' \subseteq \mathcal{D}$ and there is no $(\bar{k}, \bar{v}) \in \mathcal{D} \cap \overline{\mathcal{D}'}$ such that $\bar{k} \in [a, b]$; and
- it reveals nothing beyond $\mathcal{D}'$, i.e., proofs are zero-knowledge and reveal nothing about $\mathcal{D} \backslash \mathcal{D}'$ (e.g., not even its size).

Range queries are fundamental search queries that have applications in a variety of fields, including data analytics, network security, geographic information systems, and environmental sensing. A variety of privacy and integrity issues for range queries have been investigated in the literature in models that span data management systems (e.g., [27, 41]) and sensor networks (e.g., [13, 40]). We consider a few concrete practical scenarios where both integrity and privacy of range queries are crucial:

Audit Digital records are often subject to audit inspections or authorized investigations where an analyst is given partial access to the records (e.g., emails in case of a dispute or suspicious activity). Kamara [23] recently proposed the MetaCrypt model for handling such authorized checks via a query protocol between the owner of digital content and an analyst (i.e., an authorized party). In this model, it is essential that the protocol lets the analyst verify authenticity of answers about the records and at the same time reveals nothing about the content outside of the authorized region. (For example, an authorization could be given only to emails that were sent during a certain time period.)

Access Control Consider another scenario where a data owner uploads her data to a cloud server and delegates the processing of client queries to the server while enforcing access control policies on the data. Outsourcing of medical records indexed by patient's visit date/date of birth/dosage of some medication is one such example. It is likely that not all the data should be accessible to everyone among the medical staff.

Partial Release of Credentials A government agency or some other trusted entity certifies various records about an individual. At a later point in time, this person can reveal a subset of these authenticated records to a third party and provide a proof of them. For example, entries and exits into/from a country by border control can be used by a traveler to prove she was abroad in a certain time frame, e.g., during jury duty or elections, without having to reveal all the details of her trips.

Esha Ghosh: Dept. of Computer Science, Brown University, Providence RI, USA E-mail: esha_ghosh@brown.edu
Olga Ohrimenko: Microsoft Research, E-mail: oohrim@microsoft.com
Roberto Tamassia: Dept. of Computer Science, Brown University, Providence RI, USA, E-mail: rt@cs.brown.edu

These examples motivate the problem of performing range queries in a three-party model where a trusted owner uploads $\mathcal{D}$ to an untrusted server and clients interact with the server to execute range queries on $\mathcal{D}$. Besides providing security guarantees, we also want to devise a system with low overhead for each party.

The correctness of the server's answer can be trivially achieved using, for example, proofs from a Merkle Hash tree [29] built on top of the items of $\mathcal{D}$ ordered by keys, where root digest is signed by the owner. However, our zero-knowledge privacy requirement makes the problem challenging since proofs from a hash tree, by construction, reveal the rank of the keys and the size of $\mathcal{D}$. As another attempt, the owner could sign every key in the universe of keys, $\mathcal{U}$: namely sign (k, v) for every $(k, v) \in \mathcal{D}$ and $(k, \perp)$, otherwise. The proof for a range $[a, b]$ would then consist of $b - a + 1$ signed pairs. This solution, while providing the desired privacy guarantee, incurs a very high overhead in terms of server storage, proof size, and client verification time.

Previous work.

In 2004, Ostrovsky, Rackoff and Smith [33] explored the above problem in the two party setting: a prover commits to $\mathcal{D}$ and a verifier sends range queries to the prover. The authors also pointed out the hardness of this problem: "This seems to be a fundamental problem with privacy of Merkle-tree commitments: revealing the hash values reveals structural information about the tree, and not revealing them and instead proving consistency using generic zero-knowledge techniques kills efficiency." To this end, the authors relaxed the privacy guarantees by revealing the size of $\mathcal{D}$ as well as all previously queried ranges on $\mathcal{D}$ to the verifier. The resulting scheme uses cut-and-choose techniques to prove commitment consistency in a variant of the Merkle tree.

The solution of [33] can be directly employed to support queries in the three party model by letting a trusted owner execute the prover's setup algorithm and a malicious server execute the prover's query algorithm. We discuss it in detail in Section 2. Unfortunately, the resulting scheme does not meet the privacy and performance goals of a desired solution. First, this scheme does not have full privacy guarantees since it reveals information beyond answers to queries. Second, this scheme cannot be used by multiple mutually distrusting clients (verifiers) since clients need to learn all queries to the system to verify the correctness of their own results. Finally, the performance of this scheme suffers from a proof size quadratic in the security parameter that continues to grow with number of queries. A latency of 5 rounds of interaction between the client and server also makes this scheme unsuitable for most practical applications.

Our contributions.

In this paper, we show that there is an efficient solution to answer range queries and prove the correctness of the answers in zero-knowledge, where the proofs do not reveal anything beyond the query answers. We propose an efficient scheme where the proof size is independent of the number of previous queries (for a detailed comparison with [33] see Section 2). Our gain in performance is due to a proof technique based on identity based encryption and a relaxation of the model in [33]. The addition of the owner to the model lets us make use of a trusted setup phase with a digest that can later be used by clients (verifiers) to verify server's (prover's) proofs. Arguably, the three party model fits better in the setting where data is produced by an honest party who wishes to delegate query answering to another party.

As an application of range queries, we show how to verifiably answer closest point queries without revealing any information on proximity to other points.

Our constructions can be easily augmented with additional security properties such as data privacy and controlled disclosure. In particular, the data owner can encrypt the value field of every record in $\mathcal{D}$ and release decryption keys to clients selectively.

In summary, our contributions are the following:

- Formalizing the problem of zero-knowledge verifiable range queries in the same model as [32]. This threat model has received considerable attention because of applications to DNS security (see [20, 25, 26, 32] for more details). We emphasize that in this model, zero-knowledge is a property of the proofs (as in the context of zero-knowledge sets, lists [12, 19] and *primary secondary resolver* systems [32]) and not of the protocol.
- Providing an efficient and provably secure construction that significantly improves over the best known solution in terms of rounds of interaction, proof complexity, query and verification times. Moreover, our scheme is stateless and achieves stronger privacy guarantees.
- Implementing our construction and conducting experiments to evaluate the performance overhead in practice.
- Simulating the scenario of third party audits and measuring the cost of query and verification on two

real-world datasets, namely, *Enron email dataset* and *Boston city taxi dataset.*

2 Related Work

In this section we give an overview of existing techniques (both privacy-preserving and not) for data verification in general and then zoom into the literature for range queries specifically. We note that range query is not to be confused with range proof [8] where the goal is to prove that the committed value lies in a specified integer range without revealing it.

Authenticated data structures (ADS) [14, 21, 35, 36, 38, 42] are often set in the three party model with a trusted owner, a *trusted* client and a malicious server; the owner outsources the data to the server and later the client interacts with the server to run queries on the data. The security requirement of such constructions is data authenticity for the client against the server. Since the client is trusted, the privacy requirement of our model is usually violated by the ADS proofs. For example, authenticated set union in [37] lets a client learn information about the sets beyond the result of the union (e.g., content of each set).

Zero knowledge sets [10, 12, 28, 30] and zero knowledge lists [19] provide both privacy and integrity of the respective datasets in the following model. A malicious prover commits to a database and a malicious verifier queries it; the prover may try to give answers inconsistent with the committed database, while the verifier may try to learn information beyond query answers. Ostrovsky *et al.* [33] studied range queries in the same model and this is the closest to our work. We discuss [33] in detail later.

Constructions that guarantee privacy and integrity in the slightly relaxed three-party model (where the committer is "honest" and the (malicious) prover is different from the committer) considered in this paper have been studied for positive membership queries on dataset [5, 6, 43], dictionary queries on sets [18, 20, 32], order queries and statistics on lists [9, 11, 17, 19, 25, 26, 39] and set algebra [18].

Papadopoulos *et al.* [34] studied range queries in the traditional ADS setting where privacy is not considered. For example, completeness proof in [34] reveals elements of the database that are outside of the queried range. The integrity of range queries in conjunction with protecting data content from the server has been considered in [13, 40]. However, the proof of completeness (i.e., that no data has been withheld by the server) reveals information about elements of the database that are outside the queried range. Recall that in our model, we focus on protecting privacy of data against clients who should not have access to this data.

Attribute-based encryption for range queries [41] enforces access control by designing encryption schemes that lets one decrypt data only if it lies within a queried range. These mechanisms provide no way of proving if the output of a range search is correct and complete. This is particularly tricky when the search result is empty.

Comparison with Ostrovsky et al. [33].
As noted in the introduction the closest to our work is the scheme by Ostrovsky *et al.* We contrast the two schemes in terms of the models, privacy guarantees as well as performance.

The model of [33] consists of two parties, the prover and the verifier. The prover commits to a data set in the setup phase and then answers queries from the verifier in the query phase. The prover and the verifier are non-colluding. This adversarial model is stronger than our three party model since this supports an "untrusted" committer whereas we support a "trusted" committer. The relaxation of the model of [33] lets us use primitives with a trapdoor (signatures and identity based encryption) as opposed to trapdoorless hash and commitments and generic zero-knowledge proofs. As a result, we obtain better performance.

The protocol of [33] requires the prover to maintain a state between the queries. Furthermore, the transcript of all old queries has to be incorporated in every subsequent query response. Such a scheme, clearly, can be used only by a single client or in a trusted environment. That is, the owner cannot enforce different access control policies across clients.

In terms of privacy, the construction of [33] reveals the size of the database to the verifier whereas we offer perfect zero-knowledge (i.e., the interaction between the client and the server, can be simulated using only answers to client's queries).

In terms of performance, the protocol by Ostrovsky *et al.* requires 5 rounds of communication and has $O((t+m)\log(n)l\lambda^2)$ proof complexity, where n is the number of keys in the database $\mathcal{D}$, l is length of each key in $\mathcal{D}$, λ is the security parameter, t is the number of queries before the current query and m is the size of the result to the current query. This two party scheme can be used in our three party setting if prover's setup algorithm is executed by a trusted party. Then the schemes can be compared directly with each other; our protocol requires only one round of communication and has proof complexity of $O(ml^2)$. In Section 7, we experimentally show that our server's protocol is orders of magnitude faster than *even* a substep of the query protocol in [33].

3 Preliminaries

Adversarial Model.
Our three party model closely follows the model described in [19] and [32]. The *trusted* data owner uploads data $\mathcal{D}$ to the server and goes offline. The server answers range queries on $\mathcal{D}$ from the client(s) on behalf of the owner. Both the server and client(s) are malicious but non-colluding. The server may attempt to tamper with $\mathcal{D}$ and give incorrect answers i.e., answers inconsistent with the owner generated $\mathcal{D}$. On the other hand, clients may try to learn more about $\mathcal{D}$ than what they are allowed to learn, i.e., information about $\mathcal{D}$ beyond what can be inferred from answers to their queries. For example, they may collect and analyze authenticity proofs they have received so far and carefully choose their subsequent queries. Note that privacy is compromised if the server and the clients collude since the server knows the database.

Notation.
Let $\lambda \in \mathbb{N}$ be the security parameter of the scheme. Without loss of generality (wlog) we assume that all the keys in the database $\mathcal{D}$ are l-bits long where $l = \mathsf{poly}(\lambda)$[1]. (One can view l as revealing a trivial upper bound on n since the clients know the value of l and can deduce that there can be at most 2^l key-value pairs in $\mathcal{D}$[2].)

Hierarchical Identity Based Encryption.
HIBE allows one to efficiently derive encryption and decryption keys respecting access control based on a hierarchy of identities. Messages can be encrypted under a public key of any identity in the hierarchy but decrypted only by the following subset of these identities. Given a hierarchy of identities arranged in a tree, an identity that corresponds to some node in the tree should be able to decrypt the cipheretexts intended for it and its descendants only. This property is achieved through a key generation algorithm, that, given a node ID and its secret key, can derive decryption keys for its descendant identities ID*. HIBE consists of the following algorithms [7].

$\mathsf{Setup}(1^\lambda, l)$ takes in the security parameter λ and the depth of the hierarchy tree l. It outputs a master public key MPK and a master secret key MSK.

$\mathsf{KeyGen}(\mathsf{SK_{ID}}, \mathsf{ID}^*)$ derives a secret key $\mathsf{SK_{ID^*}}$ for a descendent of ID, ID*. The SK for ID* can be generated incrementally, given a SK for the parent identity. Notice that SK for any identity can be derived from the master secret key MSK, since it is the SK for the root of the tree.

$\mathsf{Encrypt}(\mathsf{MPK}, \mathsf{ID}, M)$ encrypts M intended for ID as cipherext C using MPK.

$\mathsf{Decrypt}(\mathsf{SK}, \mathsf{C})$ decrypts C and returns M where SK is a secret key for a prefix of C's ID. Note that MSK can decrypt ciphertexts intended for any ID in the hierarchy tree.

The selective-security of HIBE is defined as follows.

Definition 1 (HIBE Security [7, 16])**.**

$$\begin{aligned}&\Pr[\mathsf{ID}^* \leftarrow \mathsf{Adv}(1^\lambda); \mathsf{MPK}, \mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, l);\\ &\quad M_0, M_1 \leftarrow \mathsf{Adv}^{\mathsf{KeyGen}',(\mathsf{MSK},\cdot),\mathsf{Decrypt}'(\mathsf{MSK},\cdot)}(1^\lambda, \mathsf{MPK});\\ &\quad b \leftarrow \{0,1\}; \mathsf{C} \leftarrow \mathsf{Encrypt}(\mathsf{MPK}, \mathsf{ID}^*, M_b);\\ &\quad b' \leftarrow \mathsf{Adv}^{\mathsf{KeyGen}'(\mathsf{MSK},\cdot),\mathsf{Decrypt}'(\mathsf{MSK},\cdot)}(1^\lambda, \mathsf{MPK}, \mathsf{C}) :\\ &\quad b = b'] \le \nu(\lambda)\end{aligned}$$

where KeyGen$'$ *acts as* KeyGen *except that it does not accept any* ID *that is either equal to* ID* *or its prefix. Similarly,* Decrypt$'$ *does not decrypt under* ID* *or any prefix of it.*

For estimating the running time of our algorithms and in our experiments we use HIBE instantiation from [16] which relies on bilinear map pairing and a cryptographic hash function that maps bit strings to group elements; the construction is secure under the Bilinear Diffie-Hellman assumption. Setup runs in constant time; KeyGen requires $O(l)$ additions in the group and constant number of hashes; Encrypt requires $O(l)$ group

[1] Note that this is not a limiting assumption since keys shorter than l-bits can be padded up.

[2] While describing the scheme we assume each (key,value) pair is unique and hence the number of keys equals the number of elements in the database. In case of multiple values per key, a hash chain can be used per key. As we show in our experiments with real data sets in Section 8, the number of collisions is negligibly small compared to the database size.

multiplications and $O(l)$ hashes; and Decrypt needs $O(l)$ group multiplications and bilinear map computations and 4 hashes. As is standard, we assume each group action and hash function computation takes unit time for asymptotic analysis. We note that in a more recent work [7], a HIBE scheme is proposed where MPK is of size l and the ciphertext size is $O(1)$ as opposed to the $O(1)$ size MPK and $O(l)$ size ciphertexts in [16].

Hierarchical Identity Based Signature.
A HIBS scheme allows one to derive verification and signing keys based on a hierarchical relation. In terms of algorithms HIBS shares Setup and KeyGen with a HIBE scheme. However, instead of Encrypt and Decrypt it uses the following two algorithms.

$\mathsf{MSign}(\mathsf{MPK}, \mathsf{SK}, M)$ takes the master public key MPK, the secret key SK for the signer's ID and a message M and outputs a signature s on m.

$\mathsf{MVerify}(\mathsf{MPK}, \mathsf{ID}, M, s)$ takes the master public key MPK, the ID of the signer, a message and a signature and returns accept/reject.

As noted in [16], a HIBE scheme can be easily converted to a HIBS scheme.

Signature Scheme.
Our construction relies on a classical signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ where $\mathsf{KeyGen}(1^\lambda)$ returns signing key SigSK and verification key SigPK. $\mathsf{Sign}(\mathsf{SigSK}, M)$ returns a signature σ_M on a message M and $\mathsf{Verify}(\mathsf{SigPK}, M, \sigma_M)$ returns 0/1 depending on whether the signature on M is verified using the verification key or not.

4 Model

We assume the owner has a database $\mathcal{D}$ of key-value pairs of the form (k, v). A range query q consists of an interval $[a, b]$ and the answer $a_{q,\mathcal{D}}$ to this query consists of all the key-value pairs of $\mathcal{D}$ whose keys are enclosed in the given interval. More generally, let $(\mathbb{D}, \mathbb{Q}, Q)$ be a triple where $\mathbb{D}$ is a set of valid databases, $\mathbb{Q}$ is a set of valid queries and Q is a rule that associates an answer, $a_{q,\mathcal{D}} = Q(q, \mathcal{D})$ with every valid database query pair, $q \in \mathbb{Q}, \mathcal{D} \in \mathbb{D}$.

We propose a *privacy-preserving authenticated range query* system $\mathcal{RQ} = (\mathsf{Setup}, \mathsf{Query})$ that considers the adversarial model of Section 3. Our model is a generalization of the *primary-secondary resolver* model in [32]. The trusted owner prepares her data $\mathcal{D}$ and releases authenticated information σ_S for the server and a digest $\sigma_\mathcal{D}$ for the clients. The client later queries the server on $\mathcal{D}$. Since the server is a malicious party she needs to return an answer to a client's query and run a (possibly interactive) protocol with the client to convince her of the authenticity of the returned answer. We denote this protocol as Query. Wlog we assume the upper bound on the key length l is a public parameter. We represent our model using a simple diagram in Figure 1 for the ease of exposition.

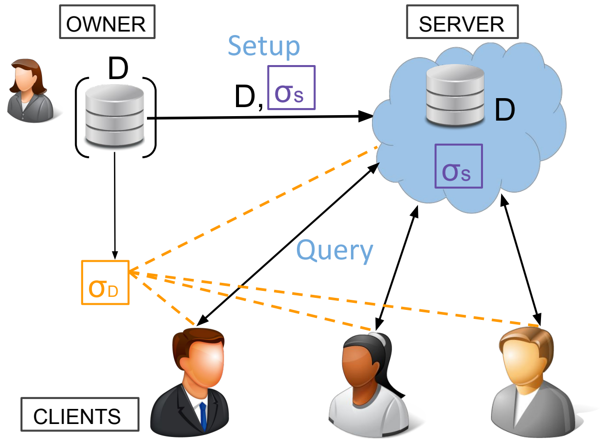

Fig. 1. Model $\mathcal{RQ}$: The owner runs Setup on database $\mathcal{D}$ to produce database digest signature, $\sigma_\mathcal{D}$, and a digest for the server, σ_S. Every party has access to $\sigma_\mathcal{D}$, but only the server has access to $\mathcal{D}$ and σ_S. Query protocol is executed between the client and the server.

$\sigma_\mathcal{D}, \sigma_S \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{D})$ This algorithm (run by the owner) takes the security parameter λ and a valid database $\mathcal{D}$ as input. It produces a short database digest signature, $\sigma_\mathcal{D}$, and a digest for the server, σ_S.

$\mathsf{answer} \leftarrow \mathsf{Query}(\mathcal{D}, q, \sigma_\mathcal{D}, \sigma_S)$ This is an interactive protocol executed between the client and the server. The client's input consists of $\sigma_\mathcal{D}$ and a query q. The server's input is $\mathcal{D}$, $\sigma_\mathcal{D}$ and σ_S. The protocol returns $\mathsf{answer} = Q(q, \mathcal{D})$ if the client is convinced by the server in its validity, $\mathsf{answer} = \perp$, otherwise.

4.1 Security Properties

A secure $\mathcal{RQ}$ scheme for answering queries has three security properties.

Completeness.
This property ensures that for any valid database $\mathcal{D}$ and for any valid query q, if the owner and the server honestly execute the protocol, then the client will always be convinced about the correctness of the answer.

Definition 2 (Completeness)**.** *For all $\mathcal{D} \in \mathbb{D}$ and all valid queries $q \in \mathbb{Q}$,*

$$\Pr[(\sigma_{\mathcal{D}}, \sigma_S) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{D}); \mathsf{answer} \leftarrow \mathsf{Query}(\mathcal{D}, q, \sigma_{\mathcal{D}}, \sigma_S) : \mathsf{answer} = Q(q, \mathcal{D})] = 1$$

Soundness.
This integrity property ensures that once an honest owner generates a pair $(\sigma_{\mathcal{D}}, \sigma_S)$ for a valid database $\mathcal{D}$, a malicious server can convince the client of an incorrect answer with at most negligible probability.

Definition 3 (Soundness)**.** *For all PPT algorithms* Adv*, for all databases $\mathcal{D} \in \mathbb{D}$ and all queries, $q \in \mathbb{Q}$, there exists a negligible function $\nu(.)$ such that:*

$$\Pr[(\mathcal{D}, q) \leftarrow \mathsf{Adv}(1^\lambda); (\sigma_{\mathcal{D}}, \sigma_S) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{D}); \mathsf{answer} \leftarrow \mathsf{Query}'(\mathcal{D}, q, \sigma_{\mathcal{D}}, \sigma_S) : \mathsf{answer} \neq Q(q, \mathcal{D})] \leq \nu(\lambda)$$

where Query′ *is an interactive query protocol executed between an honest client with input $\sigma_{\mathcal{D}}$ and q and the adversary* Adv *with input $\mathcal{D}, \sigma_{\mathcal{D}}, \sigma_S$. The honest verifier acts exactly as in* Query*, while the adversarial server* Adv *may deviate from the protocol arbitrarily.*

Zero-Knowledge.
This property captures that even a malicious client cannot learn anything about the database (and its size) beyond what she has queried for. Informally, this property involves showing that there exists a simulator that can mimic the behavior of the honest parties, i.e., a honest owner and a honest server who know $\mathcal{D}$, using only oracle access to $\mathcal{D}$. We model the indistinguishability based on the sequence of messages View that the malicious client Adv sends and receives while running $\mathcal{RQ}$ protocol on a database and queries of her choice.

Definition 4 (Zero-Knowledge)**.** *The real and ideal games are defined as:*

Game $\mathsf{Real}_{\mathsf{Adv}}(1^\lambda)$:

Setup: Adv *picks a database $\mathcal{D}$. The real challenger runs $(\sigma_{\mathcal{D}}, \sigma_S) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{D})$ and sends $\sigma_{\mathcal{D}}$ back to* Adv*.*

Query: Adv *runs the interactive query protocol* Query *with the challenger by adaptively choosing queries.*

Game $\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$:

Setup: Adv *first picks a database $\mathcal{D}$. The simulator creates a fake σ by running $(\sigma, \mathsf{state}_S) \leftarrow \mathsf{Sim}(1^\lambda, l)$ (state_S is the simulator's internal state) and returns σ to* Adv*.*

Query: Adv *runs the interactive query protocol with* Sim *which has oracle access to $\mathcal{D}$, i.e., it can only get the values that answer adversary's queries.*

Let $\mathsf{View}_{\mathsf{real}}$ and $\mathsf{View}_{\mathsf{ideal}}$ be the sequence of all messages that Adv *sends and receives in the real and ideal game. $\mathcal{RQ}$ is zero-knowledge if there exists a PPT algorithm* Sim *such that for all malicious stateful adversaries* Adv*, there exists a negligible function $\nu(.)$, such that: the probability that* Adv *distinguishes between $\mathsf{View}_{\mathsf{real}}$ and $\mathsf{View}_{\mathsf{ideal}}$ is at most $\nu(\lambda)$.*

5 Our Construction

We begin this section with some definitions that we use in our construction for range queries. We give our construction and show how to answer closest point queries efficiently using it.

Auxiliary Definitions.
Let $\mathcal{T}_l$ denote a full binary tree with $l + 1$ levels, where the root (level 0) is labeled $\perp$, the nodes at the ith level are i-bit strings from 0^i to $2^i - 1$, and the leaves are l-bit strings from 0^l to $2^l - 1$. A range w.r.t. $\mathcal{T}_l$ is a contiguous set of leaves and is represented using two leaves, the left end point and the right end point of the range. For example, the range represented by $[a, b]$ is $\{a, a + 1, \ldots, b - 1, b\}$.

The *canonical covering of a range* $[a, b]$ is the *minimal* set of nodes, P, of $\mathcal{T}_l$ such that (1) each node in the range $[a, b]$ is a descendant of one of the nodes in P and (2) for every node $x \in P$, the subtree rooted at x has it leftmost child x_{left} and rightmost child x_{right} inside the range $[a, b]$, i.e., $a \leq x_{\mathsf{left}}, x_{\mathsf{right}} \leq b$. Note that the canonical covering of a range wrt $\mathcal{T}_l$ is *unique*.

Example 1. *Let us consider $\mathcal{T}_3$ with the leaves $\{000, 001, \ldots, 111\}$. Given the range $[001, 100]$, its canonical covering in $\mathcal{T}_3$ is $P = \{001, 01, 100\}$. Note that $00 \notin P$ because its leftmost child $000 < 001$. Also, $P \neq \{001, 010, 011, 100\}$ since this covering is not minimal.*

Our construction uses method $\mathsf{GetRoots}(l, [a, b], \mathcal{K}_{[a,b]})$ to find the canonical covering of a set of ranges. This method takes as input the height l of tree $\mathcal{T}_l$, a range $[a, b]$, and a subset of keys from range $[a, b]$ denoted as $\mathcal{K}_{[a,b]} = \{k_1, \ldots, k_m\}$. It returns a set of nodes R that represent the union of the canonical coverings of the ranges $[a, k_1 - 1], [k_1 + 1, k_2 - 1], \ldots, [k_m + 1, b]$. Using a simple search procedure, this method takes time $O(ml)$ where m is the number of elements in $\mathcal{K}_{[a,b]}$.

5.1 Range Query Construction

Our construction for authenticated privacy-preserving range queries builds on a signature scheme and a HIBE scheme. Informally, it uses the signature scheme to prove that key-value pairs returned as an answer to a range query are indeed present in $\mathcal{D}$. It then uses HIBE key generation to prove that there are no other key-value pairs in $\mathcal{D}$ that belong to the queried interval.

Membership proofs using signatures are straightforward: the owner generates a signature for every key-value pair in $\mathcal{D}$ along with a nonce μ, sends them to the server and publishes the verification key of the signature scheme and μ to clients. The nonce μ is a unique identifier for $\mathcal{D}$ and the signature on (key, value, nonce) ties the (key, value) pair with a particular $\mathcal{D}$. Later, when a client queries for a range $[a, b]$ of $\mathcal{D}$, the server simply returns key-value pairs $(k_1, v_1), (k_2, v_2), \ldots, (k_m, v_m)$ of $\mathcal{D}$ lying in this interval along with the corresponding signatures. For every (k, v) in the answer, the client makes sure that $a \le k \le b$ and checks the signature on (k, v, μ).

Proving the completeness of the answer returned by the server is less trivial due to the privacy requirement (Definition 4). For example, traditional techniques for proving non-membership based on signing adjacent pairs of key-value pairs fail to preserve privacy.

Example 2. *Let $\{1, 2, 5, 6, 9\}$ be the keys present in $\mathcal{D}$ and $[3, 7]$ be the query. If privacy is not a requirement, a proof for the answer 5,6 would consist of signatures on pairs (2,5) and (6,9) showing that the keys 3,4 and 7,8 are not in $\mathcal{D}$. But this reveals that key 8 is not in $\mathcal{D}$ and key 9 is in $\mathcal{D}$, though these elements are outside the queried range.*

A secure but extremely inefficient way to solve the problem would be to accumulate *all* elements in $\mathcal{D}$, for example, using a zero-knowledge accumulator [18]. Then, for every element excluded from the answer to a range query the client is supplied with a non-membership proof. Unfortunately, in this approach the proof size grows proportionally to the size of the key domain and not query answer. That is, the range could cover exponential number of elements requiring as many proofs (e.g., the range query $[0, 2^l - 1]$ would require 2^l proofs).

Hence, our proof technique cannot directly rely on the elements present in $\mathcal{D}$. On the other hand, building a technique that relies on all the keys in the universe $\mathcal{U}$ that are not present in $\mathcal{D}$ is extremely inefficient for all the three parties (in fact, impossible for parties running in time polynomial in the security parameter λ since $|\mathcal{U}| \approx 2^\lambda$). Intuitively, we wish to develop a technique that succinctly captures ranges of keys non present in $\mathcal{D}$ and not each key individually (since $|\mathcal{D}| \ll |\mathcal{U}|$).

HIBE provides us exactly with the technique we need. Recall that a secret key at a given node of the hierarchy captures secret keys of all the nodes in the subtree rooted at this node, i.e., node's secret key can be used to decrypt a message encrypted using any of the keys in the subtree.

Setup.

Recall that $\mathcal{T}_l$ is the tree on the universe $\mathcal{U}$ of all l-bit strings ($\mathcal{T}_l$ is used for illustration purposes only and is never explicitly built). Let $\mathcal{D}$ be a database of size n whose keys are of length l and let $\mathcal{K} = \{k_1, \ldots, k_n\}$ be the set of keys present in $\mathcal{D}$, i.e., $|\mathcal{K}| = n$. The owner uses $\mathsf{GetRoots}(l, [0, 2^l - 1], \mathcal{K})$ procedure to get a set of nodes $\mathsf{root}_1, \mathsf{root}_2, \ldots, \mathsf{root}_t$. Recall that these nodes represent the canonical covering of the empty ranges in $\mathcal{T}_l$ created by $\mathcal{K}$ and that $t = O(nl)$ [31]. The owner generates a HIBE secret key for each of these roots and sends them to the server, while releasing the HIBE master public key. We describe the setup algorithm in more detail in Figure 2.

Query and Verification.

We now describe how the client verifies the completeness of the answer received from the server for query $[a, b]$. Given the answer $(k_1, v_1), (k_2, v_2), \ldots, (k_m, v_m)$, the client wants the server to prove to her that there are no elements of $\mathcal{D}$ in the ranges $[a, k_1 - 1]$, $[k_m + 1, b]$ and $[k_i + 1, k_{i+1} - 1]$ for $i = 1 \ldots m - 1$. Using HIBE, the server proves knowledge of the secret key of subtrees that cover the above ranges as follows. The client and the server independently run $\mathsf{GetRoots}(l, [a, b], \{k_1, \ldots, k_m\})$ and obtain a set of roots $R = \{\mathsf{root}^*_1, \ldots \mathsf{root}^*_{t'}\}$. The client's challenge is a sequence of ciphertexts that a server has

Fig. 2. Algorithm $(\sigma_{\mathcal{D}}, \sigma_S) \leftarrow$ **Setup**$(1^\lambda, \mathcal{D})$ run by the owner on input database $\mathcal{D}$ with n key-value pairs.

Step 1: Run $\mathsf{HIBE.Setup}(1^\lambda, l)$ to obtain pubic-secret key pair $(\mathsf{MPK}, \mathsf{MSK})$ and run $\mathsf{Sig.KeyGen}(1^\lambda)$ to get signing keys $(\mathsf{SigSK}, \mathsf{SigPK})$.

Step 2: $\mathcal{K}$ be a set of keys present in $\mathcal{D}$. The owner uses $\mathsf{GetRoots}(l, [0^l, 2^l - 1], \mathcal{K})$ to obtain the set of roots $\mathsf{root}_1, \ldots, \mathsf{root}_t$ of the forest $\mathcal{F}$ obtained from $\mathcal{T}_l$ by deleting the paths corresponding the the leaves in $\mathcal{K}$.

Step 3: For every root_i, generate its secret key: $\mathsf{SK}_{\mathsf{root}_i} \leftarrow \mathsf{HIBE.KeyGen}(\mathsf{MSK}, \mathsf{root}_i)$.

Step 4: Pick a random $\mu \xleftarrow{\$} \{0,1\}^{\mathsf{poly}(\lambda)}$, which is unique for $\mathcal{D}$. For every $(k, v) \in \mathcal{D}$, generate $\sigma_{(\mu,k,v)} \leftarrow \mathsf{Sig.Sign}(\mathsf{SigSK}, \mu, k, v)$.

Step 5: Set σ_S to $(\{\mathsf{root}_i, \mathsf{SK}_{\mathsf{root}_i}\}_{1 \le i \le t}, \{\sigma_{(\mu,k,v)}\}_{k \in \mathcal{D}})$.

Step 6: Set $\sigma_{\mathcal{D}} = (\mathsf{MPK}, \mathsf{SigPK}, \mu)$.

Step 7: Return $(\sigma_{\mathcal{D}}, \sigma_S)$.

Fig. 3. Interactive protocol answer $\leftarrow$ **Query**$(\mathcal{D}, q, \sigma_{\mathcal{D}}, \sigma_S)$ run by the client with input $q = [a, b]$ and $\sigma_{\mathcal{D}}$ and the server with input $\mathcal{D}$ and σ_S

$C \to S$**:** The client sends q to the server.

$C \leftarrow S$**:** The server returns $\{(k_1, v_1), \ldots, (k_m, v_m)\}$ and $\sigma_{(\mu,k_i,v_i)}$ for $i \in [1, m]$.

$C \to S$**:** The client verifies all signatures $\sigma_{(\mu,k_i,v_i)}$ using SigPK. If they do not verify, the client returns $\perp$. Otherwise, the client obtains the set $R = \{\mathsf{root}^*_1, \ldots \mathsf{root}^*_{t'}\}$ using $\mathsf{GetRoots}(l, [a, b], \{k_1, \ldots, k_m\})$. She then picks t' random messages $M_1, \ldots, M_t$, encrypts them using the HIBE scheme as $\mathsf{C}_{\mathsf{root}^*_j} \leftarrow \mathsf{HIBE.Encrypt}(\mathsf{MPK}, \mathsf{root}^*_j, M_j)$ for $1 \le j \le t'$ and sends $\mathsf{C}_{\mathsf{root}^*_1}, \ldots, \mathsf{C}_{\mathsf{root}^*_{t'}}$ to the server.

$C \leftarrow S$**:** The server independently runs $\mathsf{GetRoots}(l, [a, b], \{k_1, \ldots, k_m\})$ to obtain $R = \{\mathsf{root}_1, \ldots \mathsf{root}_{t'}\}$. For each root^*_j the server finds $\mathsf{SK}_{\mathsf{root}_i}$ s.t., root_i is a prefix of root^*_j. She then uses $\mathsf{HIBE.KeyGen}$ to generate a secret key for root^*_j and runs $\mathsf{HIBE.Decrypt}$ to decrypt $\mathsf{C}_{\mathsf{root}^*_j}$. She sends plaintexts for all ciphertext challenges back to the client.

C**:** The client outputs $\mathsf{answer} = [(k_1, v_1), \ldots, (k_m, v_m)]$ iff she receives a decryption of each of her challenge messages M_j. Otherwise, she outputs $\perp$.

to decrypt. The client chooses random messages and encrypts them for the roots in R.

The server can successfully decrypt the ciphertexts generated by the client by deriving the required secret key as long as the server did not cheat while returning the (k_i, v_i) pairs. That is, every range challenged by the client should be a subrange of the empty ranges generated by the owner during the setup (or equal to it). In other words, $\mathsf{root}'_j \in R$ is a node in the subtree rooted at one of root_i's. Hence the server should be able to derive secret key for each root in R. We describe this algorithm in more detail in Figure 3.

Non-interactive Query *protocol.*
During *Setup* the owner proceeds as before except she uses HIBS (instead of HIBE) to generate secret keys for the nodes in the set $\mathsf{GetRoots}(l, [0, 2^l - 1], \mathcal{K})$.

In the *Query* phase, upon receiving a query $[a, b]$ from the client, the server first retrieves signatures of every key-value pair in $\mathcal{D}$ whose key falls in the queried range (i.e., the first round in Figure 3). Then, she uses HIBS to derive the secret keys for the nodes in $R \leftarrow \mathsf{GetRoots}(l, [a, b], \{k_1, \ldots, k_m\})$ and signs each node id using the corresponding secret key. She sends back to the client (in a single round) signatures for the key-value pairs in the answer and HIBS signatures of the nodes in R.

The client uses signatures to verify membership of the received key-value pairs. For non-membership, she runs $R \leftarrow \mathsf{GetRoots}(l, [a, b], \{k_1, \ldots, k_m\})$ and verifies the signature of each node in R with the corresponding public key in HIBS. This eliminates the second round of interaction.

We note that Naor and Ziv [32] used HIBE (and HIBS) to prove non-membership queries on a set. Hence, our technique can be seen as a generalization of [32].

5.2 Closest Point Query

Using our construction (either interactive or non-interactive one), we can answer closest point query in zero-knowledge as follows. For a given query point k, let $\hat{k}$ be the closest point to k present in the database.

The server returns $\hat{k}$ and its signature. Wlog, let us assume $\hat{k}$ lies to the right of k. Then, the client challenges the server to prove that intervals $[k+1, \hat{k}-1]$ and $[2k-\hat{k}, k-1]$ are empty.

5.3 Complexity Analysis for $\mathcal{RQ}$

We analyze the complexity of each party involved in our instantiation of $\mathcal{RQ}$ using HIBE below. The asymptotic analysis of the HIBS construction is equivalent except query phase becomes non-interactive. Recall that l is the key size, n is the size of owner's dataset $\mathcal{D}$, and m is the size a query answer.

Owner. The owner's running time is dominated by HIBE.KeyGen in Setup which generates $O(nl)$ keys. As we described in Section 3, each encryption (and decryption) takes time $O(l)$. Hence, the run-time and space are proportional to $O(nl^2)$.

Server. The server requires $O(nl^2)$ space for storing the dataset and the secret keys obtained from the owner. During the query phase, in round 1, the server returns elements in the queried range and a signature of every element (which are precomputed by the owner). In round 2, the server receives $O(ml)$ challenge ciphertexts and generates a key for each ciphertext ID (to decrypt) using HIBE which in the worst case (when an empty range is just one element) takes $O(l)$ time. Hence, the total run time and space during the query phase is dominated by $O(ml^2)$.

Client. The client needs to verify m signatures and then encrypt and check equality of at most $O(ml)$ messages. As discussed in Section 3, the time required to encrypt each message is $O(l)$. So the time is upper bounded by $O(ml^2)$. The space requirement is $O(ml^2)$.

We summarize the security properties and asymptotic performance of our construction in Theorem 1. and present security proofs in the next section.

Theorem 1. *The construction of Section 5 satisfies the security properties of completeness (Definition 2), soundness (Definition 3), under the security of the* HIBE *scheme and unforgeability of the underlying signature scheme, and zero-knowledge (Definition 4). The construction has the following performance, where n is the number of elements of a database $\mathcal{D}$ of key-value pairs, with keys being l-bit values, and m is the size of the answer to a range query:*

- *The owner executes the setup phase in $O(nl^2)$ time and space.*
- *The server uses $O(nl^2)$ space and runs in $O(ml^2)$ time during the query protocol.*
- *The client runs in $O(ml^2)$ time and space during the query protocol.*
- *The query protocol has a single round of interaction in the* HIBE *instantiation and is non-interactive in the* HIBS *instantiation.*

Corollary 1. *The construction of Section 5 can be used to answer closest point queries with the same security properties as for range queries. In the resulting construction, the client and server each run in $O(l^2)$ time during the query protocol.*

Multiple values per key in a dataset.

Our construction can be easily extended to support datasets where a single key k is associated with more than one value. This can be done by mapping each key to a chain of values instead of a single value. In this case all values of the chain have to be signed and verified (additive cost). However, the proofs of empty ranges are not affected as they involve only the keys. (In our experiments on real datasets we observed small chain lengths that added a negligible overhead: a subset of the Boston taxi dataset, 54,152 records, had no collisions since location coordinates, used as a key, were all distinct, while the maximum chain length in the Enron dataset, 54,152 records, was 36 when email timestamp rounded to minutes was used as a key).

6 Security Analysis

In this section, we prove that $\mathcal{RQ}$ construction based on HIBE is secure according to definitions in Section 4.

6.1 Proof of Soundness

We prove that our construction in Section 5 is sound according to Definition 3.

Let Adv be the adversary that breaks our construction and outputs a forgery as per Definition 3. Given Adv, we construct an adversary $\mathcal{B}$ that either breaks HIBE selective security or the unforgeability of the underlying signature scheme.

Let $\mathcal{D}^*$ and $q^* = [a^*, b^*]$ be the arguments on which Adv will forge and $Q(q^*, \mathcal{D}^*) = \{(k_1, v_1), \ldots, (k_m, v_m)\}$, i.e., the keys present in $\mathcal{D}^*$ within $[a^*, b^*]$.

Adv can output two types of forgeries. First she could return answer with at least one $(k, v) \notin \mathcal{D}^*$. To do that, she needs to forge a signature on (k, v), thereby breaking the unforgeability of the underlying signature scheme. The second type of forgery is to omit an element from answer s.t. there exists at least one (k, v) s.t. $(k, v) \in \mathcal{D}^*$ and $a^* \leq k \leq b^*$ but $(k, v) \notin \mathsf{answer}$. We show that if Adv does the latter, using Adv we build $\mathcal{B}$ to break selective security of HIBE (Definition 1).

Given $\mathcal{D}^*$ and $q^* = [a^*, b^*]$, $\mathcal{B}$ picks one of $(k_i, v_i) \in Q(q^*, \mathcal{D}^*)$ randomly (i.e., she guesses that this is the key-value pair that Adv will omit in her forgery). $\mathcal{B}$ chooses a node ID^* w.r.t. k_i such that ID^* satisfies the following: (1) ID^* is a prefix of k_i but not of k_{i-1} and not of k_{i+1}, and (2) the leftmost and the rightmost child of the subtree (w.r.t. $\mathcal{T}_l$) rooted at ID^* are completely contained within $[a^*, b^*]$. $\mathcal{B}$ announces ID^* as the ID it will forge on, to the HIBE challenger.

The first condition ensures that if Adv forges a non-membership proof for k_i, then the forgery will be either w.r.t. ID^* or some prefix of it, but never its suffix. The second condition is to avoid picking an ID^* which Adv will never forge on.

$\mathcal{B}$ proceeds by computing a set of roots R using $\mathsf{GetRoots}(l, [0^l, 2^l - 1], \mathcal{D})$. She then runs HIBE.KeyGen to generate all the secret keys for the roots in R. HIBE.KeyGen will return $\perp$ for secret key queries for ID^* or its prefixes. By construction no node in R is a prefix of ID^*, hence, $\mathcal{B}$ can generate secret keys for all nodes in R using its HIBE.KeyGen oracle. $\mathcal{B}$ then generates nonce μ and forwards it, along with R and corresponding secret keys and the master public key from the HIBE challenger, to Adv.

Wlog we assume Adv outputs a forgery answer such that there exists a key-value pair in $\mathcal{D}^*$ that is not present in answer. (Recall that, if answer contains membership proof for some key-value pair not in $\mathcal{D}^*$, then the unforgeability of the underlying signature scheme can be broken.)

Given a forgery $[a^*, b^*]$ $\mathcal{B}$ does the following. She runs $R' \leftarrow \mathsf{GetRoots}(l, [a^*, b^*], \mathsf{answer})$ to check if $\mathsf{ID}^* \in R'$. If not, he aborts. Otherwise, she picks two random messages M_0, M_1 and sends them to the HIBE challenger. She receives back the challenge ciphertext C and uses it as a challenge for ID^* root. For encryption under the rest of the nodes in R', she picks random messages and encrypts them appropriately. Once Adv returns decryptions of these messages, $\mathcal{B}$ checks if C was decrypted to M_0 or M_1 and sends the corresponding bit to its HIBE challenger.

The number of nodes that can cover k_i is the number of prefixes of k_i, which can be at most the height of the tree l. Therefore the probability that the reduction does not abort, i.e., the reduction correctly guesses ID^* is $1/ml$: $\mathcal{B}$ first guesses the key k_i from m possible choices and then guesses its prefix from l possible choices. Hence, if Adv succeeds with probability ϵ, then the reduction succeeds in the HIBE game with probability ϵ/ml. □

6.2 Proof of Zero-Knowledge

In this section we show how to build a simulator for the protocol in Section 5 to show that it has the zero-knowledge property as defined in Definition 4.

Recall that View contains the client digest if $\mathcal{D}$ and all the messages exchanged between the client and the owner in the Setup phase and all messages between the client and the server during the query phase. In particular, during the Setup phase the client receives λ, l and $\sigma_{\mathcal{D}} = (\mathsf{MPK}, \mathsf{SigPK}, \mu)$. Hence, the view contains these four messages. During the query phase, for every query q, the view is augmented with the query parameter $[a, b]$, m elements returned as answer to q, m signatures, t ciphertexts and their decryptions.

We now show how to build Sim to create a view indistinguishable from the one the adversarial client (Adv in Definition 4) has when interacting with an owner and the server who have access to $\mathcal{D}$.

Setup phase.

Sim runs $\mathsf{HIBE.Setup}(1^\lambda, l)$, $\mathsf{Sig.KeyGen}(1^\lambda)$ and sets $\mu \xleftarrow{\$} \{0,1\}^{\mathsf{poly}(\lambda)}$. It sends $\sigma_{\mathcal{D}} = (\mathsf{MPK}, \mathsf{SigPK}, \mu)$ to Adv. and saves μ, MSK and SigSK.

Query phase.

On query $q = [a, b]$, Sim queries the oracle on $\mathcal{D}$ for an answer to q. Let $\{(k_1, v_1), \ldots, (k_m, v_m)\}$ be the elements in the answer it receives from the oracle. It signs every key-value pair as $\sigma_{(\mu, k_i, v_i)} \leftarrow \mathsf{Sig.Sign}_{\mathsf{SigSK}}(\mu, k_i, v_i)$ and sends $\sigma_{(\mu, k_1, v_1)}, \ldots, \sigma_{(\mu, k_m, v_m)}$ to Adv.

In the next round, Adv sends t ciphertexts $\mathsf{C}_{\mathsf{root}_1^*}, \ldots, \mathsf{C}_{\mathsf{root}_t^*}$. Sim first generates the roots itself by running GetRoots. For each root_t^* Sim uses HIBE.KeyGen to generate $\mathsf{SK}_{\mathsf{root}_i}$. (Recall that the root key MSK lets Sim generate a key for any node in the tree.) Sim decrypts the challenge ciphertexts and sends back the corresponding messages.

All messages generated by Sim are generated using the same algorithms as those by the owner and the server: the public keys of the signature and HIBE schemes, the random nonce μ and signatures on messages in the query answer. Hence, these messages in the View come from the same distribution as those in the real game. Sim can always decrypt ciphertexts in the challenge since it has the root key of HIBE. Since challenge ciphertexts are generated by Adv using the public key of HIBE, the ciphertexts and the corresponding plaintext values also come from the same distribution as in the real game.

7 Experiments on Synthetic Data

In this section, we measure the cost of each step of the protocol described in Section 5. The goal of our experiments is to determine how parameters such as database size, database clustering and range query size influence the storage overhead and the running times of the setup, query and verification algorithms. We also compare our results with our estimates of the query cost in the method of [33]. Finally, we measure the performance of our approach on closest point queries.

The experiments are run on a machine with 3 GHz Intel Core i7 processor and 16 GB 1600 MHz DDR3 memory, running OS X Yosemite version 10.10.5. Our implementation is in Java and builds on the HIBE library used in [22] that the authors made available to us. The security level in all our experiments is $\lambda = 512$ bits. The numbers reported here are averaged over 10 runs for each invocation unless indicated otherwise.

7.1 Data

We have generated and made publicly available several synthetic datasets [4] using the following approach. Given a parameter l and the corresponding 2^l upper bound on the database size, we choose several synthetic databases with sizes $n \leq 2^l$. Consider a range of keys from 0 to $2^l - 1$. (Recall that we represent the implicit full binary tree with leaves 0 to $2^l - 1$ as $\mathcal{T}_l$.) The value n determines the number of "non-empty" keys in this range. In order to capture how close or far these n keys are from each other in the range, we generate datasets with different key clustering levels. Low clustering level suggests many "empty" keys between the keys of $\mathcal{D}$, while high clustering level suggests that $\mathcal{D}$ has multiple subsequences of consecutive keys $k, k+1, k+2, \ldots$. Low clustering creates many empty ranges, which, in turn, requires more work for all three parties (e.g., the server has to prove that no keys were omitted in many ranges). As we will see in Section 8, real-world datasets often exhibit high clustering.

We use datasets with the following clustering levels:

$\mathcal{D}_\$$: the n keys are chosen at random;

$\mathcal{D}_x$: The keys in this dataset can be split into clusters of sequential keys with at most x keys in each cluster, such that in total, the dataset contains n keys. We generate the clusters as follows. Recall that l denotes the maximum bit length of the keys in the dataset. We create each cluster by choosing a seed and setting the first $l - \lceil \log x \rceil$ bits to random values and the remaining $\lceil \log x \rceil$ bits to 0's. Let $\texttt{seed}$ be the resulting integer. Then the cluster is populated with binary representation of the following x keys $\texttt{seed}, \texttt{seed}+1, \ldots, \texttt{seed}+x-1$. In our experiments we used datasets with x set to 16, 128, and 2048.
To give a concrete example, let $n = 10$, $l = 6$ and $x = 8$. There can be at most two clusters in $\mathcal{D}_8$. Let 0 and 32 be the two seeds. Then we generate two clusters: one with 8 keys and one with 2, and set $\mathcal{D}_8$ to their content. Hence, the resulting dataset contains keys $[000000, 000001, 000010, 000011, 000100, 000101,$ $000110, 000111, 100000, 100001]$.

7.2 Setup Phase

The setup phase is the preprocessing step run by the owner *once* in order to prepare data for uploading to the server. In Table 1, we consider 15-bit and 16-bit keys and show the impact of the database size, n, and the clustering parameter, x, on the setup cost.

Table 1. Setup time (in seconds) for the owner on datasets of varying size n (from 10^2 to 10^4), key length l (15 bits and 16 bits) and clustering level.

n	Clustering level				
	l	$\mathcal{D}_\$$	$\mathcal{D}_{16}$	$\mathcal{D}_{128}$	$\mathcal{D}_n$
$\mathbf{10^2}$	**15**	**80**	**1.2**	**0.8**	**0.8**
	16	**95**	**1.4**	**0.9**	**0.9**
$\mathbf{10^3}$	**15**	**519**	**1.7**	**1.3**	**0.4**
	16	**1172**	**1.9**	**1.4**	**0.6**
$\mathbf{10^4}$	**15**	**1742**	**6**	**2.7**	**0.5**
	16	**5200**	**16.6**	**5.9**	**0.7**

As expected, the setup time grows proportionally to nl^2 which matches the theoretical bounds.

Now let us look at the dependency on with effect of clustering. We observe that higher proportion of clustered keys incur a smaller cost since there are fewer empty ranges for which the owner has to derive the master secret keys using HIBE.

7.3 Range Query Phase

In this section, we analyze the query cost for the client and the server. The numbers reported here are averaged over 100 runs for each invocation. Additionally, we report the standard deviations (SD) for each invocation.

Recall that the query protocol is interactive; The client sends a range query, receives the answer, sends ciphertext challenges, gets back their decryptions and verifies that they are correct. The server receives a query, sends elements that answer the query and signatures of each of these elements (a simple lookup). Given a ciphertext, the server derives HIBE key and decrypts it.

In Table 2 (left), we report the client's time to create challenge ciphertexts. We note that the rest of the client's time consists of running m equality checks, which is a cheap operation, and two roundtrips to the server. In Table 2 (right), we report the server's time to decrypt challenge ciphertexts. This table does not include the cost of looking up m signatures, which is relatively cheap (the server simply uses the signatures provided by the owner).

The running times for each setting vary across 100 runs as can be observed from standard deviation (reported as a percentage of the mean within "( )"). High variance is explained by the different sizes of the canonical cover resulting from each range query. In particular, consider dataset $\mathcal{D}_n$, which has the highest variation. Since $\mathcal{D}_n$ consists of a single cluster, a random queried range can either completely fall within this cluster or fall partially/completely outside. In the former case, no covering is needed and the computation is very fast.

With observations similar to those made for the setup phase, we note that the number of empty ranges in the query answer as well as in $\mathcal{D}$ w.r.t. $\mathcal{T}_l$ influences the cost. Our approach is very efficient for datasets with key locality (either low or high); even queries returning 5000 elements require at most 4 seconds for the client to generate challenges for clustered datasets. In particular, if the answer to the query contains sequential keys (e.g., keys are clustered as $x, x+1, x+2, \ldots$) then the cost for the client and the server is smaller than in the case when there are missing keys (e.g., keys in the answer are $x, x+345, x+1741, \ldots$). The latter case requires the client to challenge the server (and the server to prove) on the canonical covering of every empty range created by the keys in the answer.

It is important to note that the difference in query execution time between datasets (e.g., between $\mathcal{D}_\$$ and $\mathcal{D}_n$) does not leak any additional information to the client beyond what the client can learn from the query answer, as, the client can determine the number of empty ranges to verify based on $\mathcal{T}_l$ (l is public) and the answer, which is not affected by the rest of the elements in $\mathcal{D}$.

Let us look closer at the cost of individual operations performed by the client and the server. The time at the client is split in (1) generating roots for the empty ranges to check (i.e., their canonical covering), and (2) encrypting challenge messages. The cost of (2) significantly dominates that of (1). For example, for a database of size $n = 10^4$ (with $l = 16$ and $\mathcal{D}_\$$), the cost of encryption for an answer of size 1000 is $262s$, whereas the cost of generating canonical coverings is $0.11s$. The time at the server is split in (1) generating roots for the empty ranges that need to be proved, (2) deriving keys for the roots from the stored secret keys, and (3) decrypting challenge messages. Similar to the client's case, server's cost is dominated by the decryption cost. In the same example, the typical cost of decryption is $262s$ and the cost of key derivation is $0.03s$. Note that since the canonical covering is unique, this cost of computing it using GetRoots is the same at the server and the client.

7.4 Server Storage Cost

We measure the total storage cost at the server to store the data ($\mathcal{D}$) and authentication information (σ_S) used for answering queries. Recall that σ_S contains a secret key of the HIBE scheme for the root of every subtree covering a range of empty keys (see Section 5). Since this cost again depends on the number of empty ranges, we report the number of empty ranges for $\mathcal{D}_\$$, $\mathcal{D}_{128}$ and $\mathcal{D}_n$ of varying sizes of n in Table 3. Since $\mathcal{D}_\$$ has the highest number of empty ranges, it also has the highest storage requirement. The storage cost is the size of each HIBE key times the number of ranges.

For $l = 20$, HIBE key size 2960 bytes is the worst case, i.e., when an empty range covers only one leaf of the tree. Hence, for $n = 10,000$ and $\mathcal{D}_\$$, the server

Table 2. Average ciphertext generation time (in seconds) for the client and average ciphertext decryption time (in seconds) for the server using HIBE during the query phase. We show in parentheses the standard deviation as a percentage of the average. Keys in the dataset have size $l = 16$ bits. We use '-' to denote missing table entries as $m > n$.

		Client: ciphertext generation						**Server: ciphertext decryption**					
		Query answer size m						**Query answer size m**					
		1	**10**	**100**	**500**	**1000**	**5000**	**1**	**10**	**100**	**500**	**1000**	**5000**
$n = 10^3$	$\mathcal{D}_\$$	0.6(40)	6.3 (10)	59.3(30)	226(50)	320(60)	-	0.6(40)	6.2 (10)	58.3(30)	223(50)	314(60)	-
	$\mathcal{D}_{16}$	0.7(70)	0.7(60)	0.7 (60)	0.7(70)	1.7(30)	-	0.7 (70)	0.7(60)	0.7(60)	0.7(70)	1.6(30)	-
	$\mathcal{D}_{128}$	0.7(40)	0.7(50)	0.7 (40)	1 (40)	1(30)	-	0.7(40)	0.6(50)	0.7(40)	1(40)	1(30)	-
	$\mathcal{D}_n$	0.3(110)	0.3(120)	0.3 (100)	0.3 (120)	0.4(100)	-	0.3 (110)	0.2(120)	0.3 (100)	0.3 (120)	0.3(100)	-
$n = 10^4$	$\mathcal{D}_\$$	0.3(70)	2.8(20)	28(10)	134(20)	262(20)	1387(10)	0.3(70)	2.8(20)	28(10)	134(20)	262(20)	1384(10)
	$\mathcal{D}_{16}$	0.6 (40)	0.6(50)	1 (60)	1(60)	1.7 (40)	2(50)	0.6(40)	0.6(50)	1 (60)	1(50)	1.6 (40)	2(50)
	$\mathcal{D}_{128}$	0.5(70)	0.6(60)	0.6(60)	0.9(40)	1.4(40)	3.9(20)	0.5(70)	0.6(60)	0.6(60)	0.8(40)	1.4(40)	3.8(20)
	$\mathcal{D}_n$	0.3(110)	0.3(110)	0.3(100)	0.3(120)	0.3(110)	0.4(100)	0.3(110)	0.3(110)	0.3(100)	0.3(120)	0.3(110)	0.4(100)

storage is 62Mb while for $n = 10,000$ and $\mathcal{D}_{128}$ it is only 0.3Mb.

7.5 Comparison with Ostrovsky *et al.* [33]

The protocol in [33] is based on a cut-and-choose technique. In particular, the protocol requires the server to commit to the database during the setup phase, then for every query, the server re-randomizes and permutes these commitments. Hence, for each step, the server has to also prove that the re-randomization and permutation are consistent with the original data. The protocol also involves proving the equality and comparison of two commitments, which requires bit level commitments.

Table 3. Number of HIBE secret keys by database size, n, and clustering level. Keys have $l = 20$ bits.

n	**Clustering level**		
	$\mathcal{D}_\$$	$\mathcal{D}_{128}$	$\mathcal{D}_n$
10^2	845	16	16
10^3	5222	32	12
10^4	21004	107	12
10^5	271089	814	10

Given the complexity of the above approach, we chose to estimate only some operations that are performed during the cut-and-choose protocol. We set the cut-and-choose parameter $\lambda' = 80$. The protocol of [33] requires a homomorphic commitment scheme, for which we use Pedersen commitments. Using the same library we use for our experiments, we measured the cost of re-randomizing a Pedersen commitment (CRR) as $0.2ms$. Though not expensive on its own, the number of times it is invoked in the protocol is at least $2m\lambda' \log n$ for a query on a dataset $\mathcal{D}$ of size n that returns an answer of size m. The cut-and-choose protocol also involves random permutation sampling, permuting the commitments and oblivious evaluation of the circuit for hash function. Oblivious hash evaluation alone makes our method at least an order of magnitude faster than [33] when using the timing result from [24]. The cost of one invocation of hash evaluation is estimated to be between $0.85ms$ and $3.5ms$ in [24]. We use the fastest time of $0.85ms$ in our estimate below.

In Table 4, we show that the server's cost of our protocol is orders of magnitude faster than the estimate of just one step of [33]. We note that the performance advantage of our protocol grows with the query answer size and clustering level. The clustering does not affect [33], while our approach makes use of it. Moreover, our performance does not degrade as the number of queries grows (recall that [33] is stateful).

7.6 Closest Point Query

We experiment with closest point query (Table 5) for key size $l = 24$ by varying the database size of $\mathcal{D}_{2048}$. We see that the encryption (client) and decryption (server) time drops as the database size grows. As we have discussed before, this is due to the number of empty ranges decreasing as the database size grows for a fixed l.

8 Experiments on Real-World Data

In this section, we measure the cost of query and verification on two real-world datasets, *Enron email dataset*

Table 4. Total time (in seconds) for the server in our approach vs. time of one step of the server (CRR) in [33] for $l = 16$, $n = 10^4$.

Answer	Our scheme			CRR
size m	$\mathcal{D}_\$$	$\mathcal{D}_{128}$	$\mathcal{D}_n$	[33]
100	28	0.4	0.4	206.5
1000	237	1.3	0.5	2064.7
5000	1317	3.6	0.2	10323.6

Table 5. Closest point query time (in seconds) for the server and the client for datasets of varying size, n. The key size is $l = 24$ bits the clustering is $\mathcal{D}_{2048}$.

n	10^2	10^3	10^4	10^5	10^6
Server	1.4	0.9	1.0	1.3	0.5
Client	1.6	1.0	1.3	1.5	0.6

and *Boston city taxi dataset*. The goal of these experiments is to measure the performance of our protocol on real world data. We describe our experimental set up for each dataset first and then report the performance of our method on various range query sizes in Table 6. We note that we ran experiments on the same machine and using the same parameters as in Section 7.

8.1 Enron Data

The Enron dataset [2] consists of 54,152 emails generated by 158 employees (including incoming and sent-mails). The emails were sent between 1999 and 2002. The first two lines of each email contain the fields `Message-ID` and `Date`, for example:

> **`Message-ID:`** ⟨6233160.1075840045559.
> JavaMail.evans@thyme⟩
> **`Date:`** Wed, 6 Feb 2002 14:51:43 -0800 (PST)

We created a database consisting of all the emails in the inbox and sent-mail folder of each employee and used ⟨`Date`, `Message-ID`⟩ as the the key-value pair for the database. We first converted all the dates into UTC time and kept precision up to minutes. The emails were then sorted by the timestamp (date) value. In case of collision, i.e., if there were more than one `Message-ID` associated with a timestamp, we created a chain for that timestamp. In particular, we found collisions for 6331 timestamps, where the average number of emails per timestamp (i.e., the length of a chain) was 1.8 and the maximum number of emails per timestamp was 36.

For this dataset we consider an audit query that requests emails sent between timestamp A and B (e.g., an authorized time interval). Our protocol then returns all message-ID's sent and received in the time period between A and B. It also proves that the answer is complete, i.e., no email has been omitted, and reveals nothing else about the email corpus.

8.2 Boston Taxi Data

As our second example, we chose the Boston taxi dataset, which contains data about taxi rides in Boston from November 2012 [1], sorted by pickup date. For our experiments we use the first 54,152 taxi rides of the original dataset in order to match the size of the Enron dataset.

We build a database of the original records using record fields `PICKUPADDRESS`, `PICKUPLONG` and `PICKUPLAT` as follows. Using the GeoHash library [3], we create a binary hash value, `GeoHash`, from the longitude and latitude of each pick up location (fields `PICKUPLONG` and `PICKUPLAT`). A geohash is a short binary string representation of a (latitude, longitude) point. It is computed via a hierarchical spatial data structure that subdivides space into buckets of grid shape, using the Z-order curve, or Hilbert curve — more generally known as space-filling curves [15]. Geohashes offer arbitrary precision and the possibility of gradually removing characters from the end of the code to reduce its size (and gradually lose precision). The longer a shared prefix of two geohashes is, the closer the two places are. We use the ⟨`GeoHash`, `PICKUPADDRESS`⟩ as the key-value pair of the dataset.

The range query on this dataset requests all addresses between two pickup locations, C and D. Our protocol returns all the addresses between these pick up locations and proves that the answer is complete, i.e., no address has been omitted, without revealing anything else about the other records in the dataset.

8.3 Performance

The results of our experiments are shown in Table 6 where we report averages and standard deviation over 100 runs.

As we have already discussed in Section 7, low clustering creates many empty ranges, which, in turn, requires more work for all three parties, namely, owner, server and client. In the Enron email dataset, the emails

Table 6. Average ciphertext generation time (in seconds) for the client and average ciphertext decryption time for the server using HIBE during the query phase. We show in parentheses the standard deviation as a percentage of the average.

	Client: ciphertext generation					**Server: ciphertext decryption**				
	Query answer size m					**Query answer size m**				
	1	**10**	**10^2**	**10^3**	**10^4**	**1**	**10**	**10^2**	**10^3**	**10^4**
Enron	**1.2(60)**	**12.9(30)**	**53(30)**	**504(20)**	**3904(30)**	**1.2(60)**	**12.9(30)**	**51(30)**	**505 (20)**	**3915(30)**
Boston	**1.7(50)**	**8.1(40)**	**29(20)**	**192(30)**	**795(60)**	**1.6(50)**	**8(40)**	**28(20)**	**191(30)**	**793(60)**

clustering is much lower compared to the Boston taxi dataset, where data is clustered around certain neighborhoods. Note that although we did not use the full Boston taxi dataset, our choice of records was based on the timestamp and not the location, as we chose first 54K records of the original dataset sorted by date. To give concrete numbers, the number of empty intervals for the Enron dataset is 22,650 whereas for the Boston taxi dataset it is only 3,176. As expected, the effect of clustering is evident in Table 6. For the same query size, we observe that the performance on the Boston taxi dataset is much better than that on the Enron dataset.

Here, we also recall that the time taken to answer a query of size m varies depending on the size of the covering forest and depth of each root of the covering forest needed to construct a proof for the returned answers. This dependency is evident in the standard deviation across the 100 random queries performed. For example, we observed the following variation in size and depth of the covering forest for queries of size 10. In the Enron dataset, the size of the covering forest varied from 4 (each of depth 20) to 86 (depth varying between 4 and 20), which resulted in query times between $1.26s$ and $18.54s$, respectively. In the Boston dataset, the size of the forest varied from 15 (depth varying between 13 and 21) to 55 (depth varying between 4 and 21), which resulted in query times between $4.1s$ and $13.8s$, respectively.

We also note that, as expected from the asymptotic analysis of Section 5, the query time grows linearly with the size of query answer.

9 Conclusion and Future Directions

In this paper we presented and experimentally evaluated an efficient solution for answering 1-D range queries while providing both integrity and privacy (zero-knowledge). Our technique extends directly to multi-dimensional data only with an exponential (in the security parameter) blow up in the number of secret keys stored by the server. Finding an efficient solution that maintains integrity and zero-knowledge for multi-dimensional range queries is left as an open problem. Another interesting research direction is to support range queries on dynamic datasets efficiently without compromising on privacy and security.

10 Acknowledgment

This research was supported in part by the National Science Foundation under grants CNS–1525044 and IIS–1212508.

References

[1] Boston taxi dataset, https://data.cityofboston.gov/Transportation/Boston-Taxi-Data/ypqb-henq, accessed on 22 Feb 2016

[2] Enron email dataset, https://www.cs.cmu.edu/~./enron/, accessed on 22 Feb 2016

[3] GeoHash library, https://github.com/kungfoo/geohash-java, accessed on 22 Feb 2016

[4] Synthetic datasets for range queries used in this paper, https://repository.library.brown.edu/studio/item/bdr:653992/

[5] Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. In: Theory of Cryptography - Proc. TCC (2012)

[6] Attrapadung, N., Libert, B., Peters, T.: Computing on authenticated data: New privacy definitions and constructions. In: Advances in Cryptology – Proc. ASIACRYPT (2012)

[7] Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: Proc. Annual Int. Conf. on Theory and Applications of Cryptographic Techniques. pp. 440–456. EUROCRYPT (2005)

[8] Brickell, E.F., Chaum, D., Damgaard, I.B., van de Graaf, J.: Gradual and verifiable release of a secret. In: Advances in Cryptology CRYPTO. LNCS, vol. 293 (1988)

[9] Brzuska, C., Busch, H., Dagdelen, Ö., Fischlin, M., Franz, M., Katzenbeisser, S., Manulis, M., Onete, C., Peter, A.,

Poettering, B., Schröder, D.: Redactable signatures for tree-structured data: Definitions and constructions. In: Proc. Applied Cryptography and Network Security ACNS (2010)

[10] Catalano, D., Fiore, D., Messina, M.: Zero-knowledge sets with short proofs. In: Advances in Cryptology – Proc. EUROCRYPT (2008)

[11] Chang, E.C., Lim, C.L., Xu, J.: Short redactable signatures using random trees. In: Topics in Cryptology — Proc. Cryptographer's Track at the RSA Conference CT-RSA (2009)

[12] Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial commitments with applications to zero-knowledge sets. In: Advances in Cryptology – Proc. EUROCRYPT (2005)

[13] Chen, F., Liu, A.X.: Privacy- and integrity-preserving range queries in sensor networks. IEEE/ACM Trans. Netw. 20(6), 1774–1787 (Dec 2012)

[14] Devanbu, P.T., Gertz, M., Martel, C.U., Stubblebine, S.G.: Authentic third-party data publication. In: Conference on Data and Applications Security and Privacy DBSec. pp. 101–112 (2000)

[15] Gaede, V., Günther, O.: Multidimensional access methods. ACM Comput. Surv. 30(2), 170–231 (Jun 1998)

[16] Gentry, C., Silverberg, A.: Hierarchical ID-based cryptography. In: Advances in Cryptology – Proc. ASIACRYPT, vol. 2501, pp. 548–566 (2002)

[17] Ghosh, E., Goodrich, M.T., Ohrimenko, O., Tamassia, R.: Fully-dynamic verifiable zero-knowledge order queries for network data. Cryptology ePrint Archive, Report 2015/283 (2015)

[18] Ghosh, E., Ohrimenko, O., Papadopoulos, D., Tamassia, R., Triandopoulos, N.: Zero-knowledge accumulators and set operations. Cryptology ePrint Archive, Report 2015/404 (2015)

[19] Ghosh, E., Ohrimenko, O., Tamassia, R.: Zero-knowledge authenticated order queries and order statistics on a list. In: Proc. Applied Cryptography and Network Security ACNS (2015)

[20] Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., Ziv, A.: NSEC5: Provably preventing DNSSEC zone enumeration. In: The Network and Distributed System Security Symposium NDSS (2015)

[21] 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. Proceedings of Very Large Data Bases PVLDB 5(10), 920–931 (2012)

[22] Hengartner, U., Steenkiste, P.: Exploiting hierarchical identity-based encryption for access control to pervasive computing information. In: Security and Privacy for Emerging Areas in Communications Networks, SecureComm. pp. 384–396 (2005)

[23] Kamara, S.: Restructuring the NSA metadata program. In: Financial Cryptography and Data Security – FC Workshops. pp. 235–247 (2014)

[24] Kerschbaum, F.: Oblivious outsourcing of garbled circuit generation. In: Proc. ACM Symp. on Applied Computing. pp. 2134–2140 (2015)

[25] Kundu, A., Atallah, M.J., Bertino, E.: Leakage-free redactable signatures. In: .ACM conference on Data and application security and privacy CODASPY (2012)

[26] Kundu, A., Bertino, E.: Structural signatures for tree data structures. Proceedings of Very Large Data Bases PVLDB 1(1) (2008)

[27] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proc. of ACM SIGMOD International Conference on Management of Data. pp. 121–132 (2006)

[28] Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Theory of Cryptography - Proc. TCC (2010)

[29] Merkle, R.C.: A certified digital signature. In: Advances in Cryptology CRYPTO. pp. 218–238 (1989)

[30] Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: Symposium on Foundations of Computer Science FOCS, Proceedings. pp. 80–91 (2003)

[31] Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. Electronic Colloquium on Computational Complexity ECCC (043) (2002)

[32] Naor, M., Ziv, A.: Primary-secondary-resolver membership proof systems. In: Theory of Cryptography - Proc. TCC, vol. 9015, pp. 199–228 (2015)

[33] Ostrovsky, R., Rackoff, C., Smith, A.: Efficient consistency proofs for generalized queries on a committed database. In: Automata, Languages and Programming: International Colloquium, ICALP, Proceedings. pp. 1041–1053 (2004)

[34] Papadopoulos, D., Papadopoulos, S., Triandopoulos, N.: Taking authenticated range queries to arbitrary dimensions. In: Proc.ACM Conf. on Computer and Communications Security. pp. 819–830. CCS (2014)

[35] Papadopoulos, D., Papamanthou, C., Tamassia, R., Triandopoulos, N.: Practical authenticated pattern matching with optimal proof size. Proceedings of Very Large Data Bases PVLDB 8(7), 750–761 (2015)

[36] Papamanthou, C., Shi, E., Tamassia, R., Yi, K.: Streaming authenticated data structures. In: Advances in Cryptology - Proc. EUROCRYPT. pp. 353–370 (2013)

[37] Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: Advances in Cryptology – Proc. CRYPTO. pp. 91–110 (2011)

[38] Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables based on cryptographic accumulators. Algorithmica 74(2), 664–712 (2016)

[39] Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: On structural signatures for tree data structures. In: Proc. Applied Cryptography and Network Security ACNS (2012)

[40] Sheng, B., Li, Q.: Verifiable privacy-preserving range query in two-tiered sensor networks. In: Proc. IEEE Int. Conf. on Computer Communications INFOCOM. pp. 46–50 (2008)

[41] Shi, E., Bethencourt, J., Chan, T.H.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: Proc. IEEE Symp. on Security and Privacy. pp. 350–364 (2007)

[42] Tamassia, R.: Authenticated data structures. In: Proc. European Symp. on Algorithms ESA. LNCS, vol. 2832, pp. 2–5 (2003)

[43] Wang, Z.: Improvement on Ahn et al.'s RSA P-homomorphic signature scheme. In: Security and Privacy for Emerging Areas in Communications Networks, SecureComm (2012)