Abstract of “Integrity and Privacy in the Cloud:
Efficient algorithms for secure and privacy-preserving processing of outsourced data” by Esha Ghosh, Ph.D., Brown University, May 2018.

An integral component of the modern computing era is the ability to outsource data and computation to remote cloud service providers or CSPs. The advent of cloud services, however, raises important challenges of in terms of integrity and privacy of data and computation. As soon as users delegate computation to cloud platforms (such as Microsoft Azure or Amazon AWS), concerns related to integrity of the results arise. For example, have all correct inputs been used in the computation? Were all the computation steps applied in the correct order? Have the results been delivered untampered? Moreover, in the face of an alarming number of data breaches and massive surveillance programs around the globe, the privacy of outsourced data is becoming more important than ever.

This thesis focuses on designing efficient privacy-preserving and verifiable data processing queries for a rich class of data structures along with prototype implementation and experimental validation. In particular, we focus on the following setting: how can a trusted data owner outsource her data to an untrusted server, such that the server will not be able to cheat while answering queries on the stored data. In other words, we require the server to produce a cryptographic proof for each answer it produces. Moreover, we require the proofs to be privacy-preserving, i.e., they should not leak any information about the data structure or the updates on it besides what can be inferred from the answers.

We also consider another dimension of privacy for verifiable outsourced data-processing, namely, encrypting the outsourced data. More concretely, we consider the setting where the data structure is encrypted before outsourcing using a customized encryption scheme that allows the server to compute queries on the encrypted data. Furthermore, the client can efficiently check if the server has correctly computed the answer.

In this thesis, we focus on range queries, closest point queries, dictionary queries, set algebraic queries and reachability and shortest path queries on general graphs.

Integrity and Privacy in the Cloud:
Efficient algorithms for secure and privacy-preserving processing of outsourced data

by
Esha Ghosh

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2018

This dissertation by Esha Ghosh is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Roberto Tamassia, Director

Recommended to the Graduate Council

Date _____ _____

Anna Lysyanskaya, Reader

Date _____ _____

Seny Kamara, Reader

Approved by the Graduate Council

Date _____ _____

Andrew G. Campbell
Dean of the Graduate School

Acknowledgements

It is a great pleasure to thank everybody without whose continuous support, this thesis would not have materialized. First and foremost, I thank my adviser Roberto Tamassia. Roberto has been a truly supportive adviser, an inspiring mentor and a great role model. He has always made himself available for me, be it for research meetings or for ethical advice. Roberto's kind and graceful advising style and his approach towards inclusiveness and diversity is a great source of inspiration for me. I feel truly honored to be his student.

I am very thankful to my committee members Anna Lysyanskaya and Seny Kamara who followed my research closely and have mentored me throughout my Ph.D journey. I had the wonderful opportunity to collaborate with Seny towards the end of my Ph.D. I thank Sorin Istrail and John Savage for following my research from time to time and for their great words of encouragement. My heartfelt thanks to Ugur Cetintemel for his insightful career advice and support.

I had the pleasure to collaborate with Michael Goodrich in the earlier half of my PhD. Michael's enthusiasm and self-critical approach to research greatly shaped my perspective on research. Olya Ohrimenko has been a wonderful mentor, my internship host (at Microsoft Research) and collaborator on multiple projects. She has been a source of unwavering support throughout these years.

I am particularly thankful to the security group at Brown; it is a remarkable place for exchanging ideas and sharing constructive criticism. The CS department at Brown could not have fostered such a great academic environment without tremendous contributions from the TStaff and AStaff members. My sincere thanks to Lauren Clarke, Genie DeGouveia, Donald Johwa, Jane McIlmail, Jesse Polhemus, Dawn Reed, Frank Pari and to all other staff members.

I am grateful to all my other collaborators, namely, Christian Cachin, Jan Camenisch, Melissa Chase, Angelo De Caro, Apoorvaa Deshpande, Dario Fiore, Cédric Fournet, Markulf Kohlweiss, Dimitrios Papadopoulos, Bryan Parno, Alessandro Sorniotti, Björn Tackman and Nikos Triandopoulos. My collaboration with them has been a great learning experience. A special thanks to my internship mentors Alessandro and Angelo for hosting me at IBM Zürich. I tremendously enjoyed my time at IBM. Thanks to Roberto and the support from my Microsoft Research Dissertation Grant, I also had the opportunity to work with a group of bright undergraduate and masters students at Brown, Talley Amir, Nick Cunningham and Harjasleen Malvai. I thank Donghoon Chang, Melissa Chase, Jonathan Katz, Satya Lokam, Dahlia Malkhi, Charalampos Papamanthou and Sushmita Ruj for hosting me for short visits during my Ph.D.

My heartfelt thanks to my officemates and friends, Anubhav, Apoorvaa, Cindy, Elizabeth, Evgenios,

Jasper, Krishna, Prannath, Yizhen and Zhiyu. They have been truly supportive and made my stay in Providence, fond. A very special thanks to Chaya for always being there, every step of the way and for making New York, *home*.

Sajin has been a tremendous source of strength and support during the final years of my Ph.D. I could not have undertaken this journey, far away from home, if not for the unconditional support of my family. Throughout my life, their love and support has given me the courage to pursue my interests unapologetically and has made me the person I am today.

My heartfelt thanks to Microsoft Research for appreciating and supporting my dissertation work through a Microsoft Research Dissertation Grant. This dissertation was also supported by NSF grants, the Coline M. Makepeace Fellowship and the Paris C. Kanellakis Fellowship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An integral component of computing today is the ability to outsource data and computation to remote cloud service providers (CSP) like Amazon AWS, Mircosoft Azure, Google Cloud. The huge benefit of using CSP's comes in terms of efficiency, availibilty and scalability. But on the flip side, there are serious threats it terms of integrity and privacy of personal data. This is particularly important, since users of the cloud services completely lose control of their data and processing, once it is uploaded to the the cloud servers. For example, have all correct data been used in the computation? Were all the computation steps applied in the correct order? Have the results been delivered untampered?

With the popularity of could service adoptions, cloud breaches are also becoming increasingly common. In the face of the alarming number of data breaches and massive surveillance programs around the globe, providing integrity and privacy assurances to users of cloud providers is becoming increasingly important. This thesis addresses some of the challenges in developing efficient and secure solutions to protect integrity and privacy of data stored in the cloud.

A cryptographic solution to provide integrity in the cloud, requires the server to prove the correctness of each query or computation it runs for its clients. More concretely, each computation result needs to be accompanied with a cryptographic proof-of-correctness that is very efficient to verify, i.e., verification takes much less time than the computation. This cryptographic primitive is known as *Verifiable Computation* (VC).

A popular approach to design practically deployable VC schemes, is to design VC schemes for specific data structure queries. This approach allows for customized protocols that gain efficiency at the cost of losing generality. Typically, such VC schemes are known as *Authenticated Data Structures* (ADS) [149]. There has been a long line of work on designing ADS schemes for various data structures. Due to its performance benefits ADS have been adopted in many other applications beyond cloud services. DNSSEC is such an example.

However, in all the traditional ADS schemes, the proof-of-integrity of the result, leaks information about the database that is not deducible from the result itself. This can be particularly harmful when third party service providers are allowed selective access to the remotely stored data.To give a concrete example, the Domain Name System (DNS) zone enumeration attack is a vulnerability in the framework of outsourced database queries. The (trusted) owner of a DNS zone outsources the DNS table to DNS nameservers to

respond to queries issued by (third party) DNS resolvers. DNSSEC mandates providing proofs-of-integrity along with the answers to the DNS queries using a traditional ADS. However, the proofs leak more information than the answers. A malicious DNS resolver can exploit this leakage to learn the IP addresses of all hosts in a zone (including routers and other devices). The recovered information can be used to launch more complex attacks, some of which are mentioned in RFC 5155.

A significant part of my thesis focuses on designing efficient privacy-preserving ADS for a large class of data queries along with prototype implementation and experimental validation [154, 76]. We formalized privacy-preserving ADS in a three party model [150] where a trusted owner outsources a data structure to an untrusted server who answers queries issued by untrusted clients. The owner can update the data structure at any point. The server answers queries in such a way that the clients (1) can verify the correctness of the answers but (2) do not learn anything about the data structure or the updates besides what can be inferred from the answers.

In the rest of my thesis, I discuss my work on designing efficient and verifiable, privacy-preserving schemes for queries on outsourced data structures. I mainly focus on range queries, set algebraic queries and shortest path and reachability queries on graphs. My thesis is divided in four chapters as follows.

Range and Closest Point Queries: Range queries are fundamental searches that have numerous applications in data management and analytics, network security, geographic information systems, and environmental sensing. In Chapter 2 we address the problem of answering verifiable range-queries on one-dimensional data while maintaining zero-knowledge privacy and efficiency. As an application, we show how to efficiently answer closest point queries while maintaining the same level of privacy and integrity. We validated the efficiency of our scheme by implementing a prototype and running experiments on two real world data sets (Enron email data set [3] and the Boston taxi data set [2]). Compared to just one step of the previous scheme in the literature, the server's cost of our protocol is orders of magnitude faster. The full version for this work appeared in [154] and an extended abstract is available on [77].

Set Algebraic Queries: Privacy-preserving membership/non-membership queries over a dynamic set have several interesting applications including certificate revocation, anonymous credentials, DNS queries and NoSQL database queries(like *MongoDB, CouchDB*). Cryptographic accumulators are traditionally used for such authentication mechanisms. However, traditional accumulators do not provide any privacy and hence fall short of applications that need privacy. Keeping this in mind, in Chapter 3, we propose a generic framework and construction of *zero-knowledge accumulators*. We measure the concrete cost for our zero-knowledge accumulator and show that it adds almost no overhead to the previous non-private version. We further generalize it to provide efficient privacy-preserving succinct proofs for set queries over a dynamically evolving collection of sets to support more complex computations like equi-joins and time-stamped keyword search. Our new constructions add almost no overhead to the previous non-private constructions in the literature. An extended abstract of this work appeared in [152] and the full version of this work is available on [75].

Graph Structure Queries: Verifiable, private queries on graph data are common in a wide range of application domains, namely, online ad auctions, medical health records (stored in XML format) and access

control policies expressed as graphs and online social networks (OSN). Graph databases have received significant interest across various disciplines of research and industry due to their vast applicability. This has resulted in a large number of expressive graph database systems such as Pregel, Horton, GraphChiDB, Neoj, DEX, Titan etc. However, these systems do not have support for privacy.

Motivated by such applications, we investigate the problem of verifiable reachability and shortest path queries in outsourced graphs, while maintaining privacy. We investigate the problem from two different privacy dimensions.

In Chapter 4, we provide verifiable shortest path queries on outsourced graphs that are encrypted. More concretely, our scheme allows encrypting a graph in a way such that it can later be queried for a pair of encrypted vertices. The server computes the encrypted shortest path between those two vertices and returns it. Furthermore, the client can efficiently check if the server has correctly computed the encrypted shortest path. We implement our scheme and experimentally validate its performance on 9 graph data sets obtained from [105, 74]. The size of our encrypted graph is 4X the size of the underlying unencrypted search structure. The search token (consisting of two encrypted vertices) is small (just 32 Bytes in our experiments) irrespective of the length of the searched path. We also show that the size of the encrypted path, the time to generate it at the server and the time to decrypt it at the client, are proportional to the length of the returned path alone. In other words, the size and the cost of computing the encrypted path are optimal; those are not affected by other graph parameters, like density or size of a graph.

In Chapter 5, we investigate another dimension of privacy, namely, privacy of the proofs. More concretely, we design protocols for efficiently answering queries of the form *is vertex v reachable from vertex u* in outsourced *static and dynamic* graphs. While in case of static graphs, zero-knowledge privacy is required only for queries, for dynamic graphs, we also require zero-knowledge privacy for updates. In the context of cloud services, there has been some attempt to look at verifiable graph queries [162, 83], but not much work has been done in answering recahability queries on both *static and dynamic graphs* while maintaining privacy. The privacy property is crucial in the context of cloud security. For example, consider the case of cloud VM isolation. While cloud services have significantly reduced the cost of computations for organizations, many are still hesitant to adopt them due to security and privacy concerns. There are well known ways to prevent attacks due to colocation on the cloud, in particular, *security groups*[21, 156], if implemented correctly, can help prevent information flow to VMs with high security requirement. Note that security groups can be represented as directed graphs where reachability of node u from node v represents a possibility for information flow from the entity represented by u to the entity represented by v. This motivates the scenario where an organization can obtain proof of sufficient network isolation from different groups using zero-knowledge authenticated graph reachability queries.

Chapter 2

Efficient Verifiable Range and Closest Point Queries in Zero-Knowledge

2.1 Introduction

The full version for this work appeared in [154] and an extended abstract is available on [77]. In this work, we consider the problem of verifiably answering range queries on a key-value store DS while hiding the rest of DS's content. That is, a range query $[a,b]$ on DS has the following requirements:

- it returns the answer DS' and a proof of its correctness, i.e., $\mathsf{DS}' \subseteq \mathsf{DS}$ and there is no $(\bar{k},\bar{v}) \in \mathsf{DS} \cap \overline{\mathsf{DS}'}$ such that $\bar{k} \in [a,b]$; and
- it reveals nothing beyond DS', i.e., proofs are zero-knowledge and reveal nothing about $\mathsf{DS} \setminus \mathsf{DS}'$ (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., [106, 146]) and sensor networks (e.g., [45, 145]). 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 [92] 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.

These examples motivate the problem of performing range queries in a three-party model where a trusted owner uploads DS to an untrusted server and clients interact with the server to execute range queries on DS. 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 [114] built on top of the items of DS 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 DS. As another attempt, the owner could sign every key in the universe of keys, $\mathcal{U}$: namely sign (k,v) for every $(k,v) \in \mathsf{DS}$ 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 [127] explored the above problem in the two party setting: a prover commits to DS 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 DS as well as all previously queried ranges on DS 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 [127] 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 2.2. 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 [127] see Section 2.2). Our gain in performance is due to a proof technique based on identity based encryption and a relaxation of the model in [127]. 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 DS 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 [123]. This threat model has received considerable attention because of applications to DNS security (see [123, 80, 103, 102] 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 [42, 153] and *primary secondary resolver* systems [123]) 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*.

2.2 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 [27] where the goal is to prove that the committed value lies in a specified integer range without revealing it.

Authenticated data structures (ADS) [58, 149, 133, 129, 130, 82] 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 [131] lets a client learn information about the sets beyond the result of the union (e.g., content of each set).

Zero knowledge sets [116, 42, 39, 108] and zero knowledge lists [153] 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.* [127] studied range queries in the same model and this is the closest to our work. We discuss [127] 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 [12, 157, 14], dictionary queries on sets [123, 80, 152], order queries and statistics on lists [103, 40, 28, 141, 102, 153, 150] and set algebra [152].

Papadopoulos *et al.* [128] studied range queries in the traditional ADS setting where privacy is not considered. For example, completeness proof in [128] 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 [145, 45]. 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 [146] 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.* [127]. 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 [127] 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 [127] 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 [127] 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 [127] 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 DS, l is length of each key in DS, 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 2.7, we experimentally show that our server's protocol is orders of magnitude faster than *even* a substep of the query protocol in [127].

2.3 Preliminaries

Adversarial Model. Our three party model closely follows the model described in [153] and [123]. The *trusted* data owner uploads data DS to the server and goes offline. The server answers range queries on DS 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 DS and give incorrect answers i.e., answers inconsistent with the owner generated DS. On the other hand, clients may try to learn more about DS than what they are allowed to learn, i.e., information about DS 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 DS 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 DS[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 [23].

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

$\mathsf{KeyGen}(\mathsf{SK}_{\mathsf{ID}}, \mathsf{ID}^*)$ derives a secret key $\mathsf{SK}_{\mathsf{ID}^*}$ 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 $\mathsf{M}_G\mathsf{SK}$, since it is the SK for the root of the tree.

[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 2.8, the number of collisions is negligibly small compared to the database size.

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

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

The selective-security of HIBE is defined as follows.

Definition 1 (HIBE Security [73, 23])**.**

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

where KeyGen' *acts as* KeyGen *except that it does not accept any* ID *that is either equal to* ID^* *or its prefix. Similarly,* $\mathsf{Decrypt}'$ *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 [73] 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 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 [23], a HIBE scheme is proposed where $\mathsf{M}_G\mathsf{PK}$ is of size l and the ciphertext size is $O(1)$ as opposed to the $O(1)$ size $\mathsf{M}_G\mathsf{PK}$ and $O(l)$ size ciphertexts in [73].

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.

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

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

As noted in [73], 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.

2.4 Model

We assume the owner has a database DS of key-value pairs of the form $(\mathsf{key}, \mathsf{val})$. A range query q consists of an interval $[a,b]$ and the answer $a_{q,\mathsf{DS}}$ to this query consists of all the key-value pairs of DS 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, $a_{q,\mathsf{DS}} = Q(q, \mathsf{DS})$ with every valid database query pair, $q \in \mathbb{Q}, \mathsf{DS} \in \mathbb{D}$.

We propose a *privacy-preserving authenticated range query* system $\mathcal{RQ} = (\mathsf{Setup}, \mathsf{Query})$ that considers the adversarial model of Section 2.3. Our model is a generalization of the *primary-secondary resolver* model in [123]. The trusted owner prepares her data DS and releases authenticated information σ_S for the server and a digest σ_{DS} for the clients. The client later queries the server on DS. 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 2.1 for the ease of exposition.

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

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

2.4.1 Security Properties

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

Completeness. This property ensures that for any valid database DS 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* $\mathsf{DS} \in \mathbb{D}$ *and all valid queries* $q \in \mathbb{Q}$*,*

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

Soundness. This integrity property ensures that once an honest owner generates a pair $(\sigma_{\mathsf{DS}}, \sigma_S)$ for a valid database DS, 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* $\mathsf{DS} \in \mathbb{D}$ *and all queries,* $q \in \mathbb{Q}$*,*

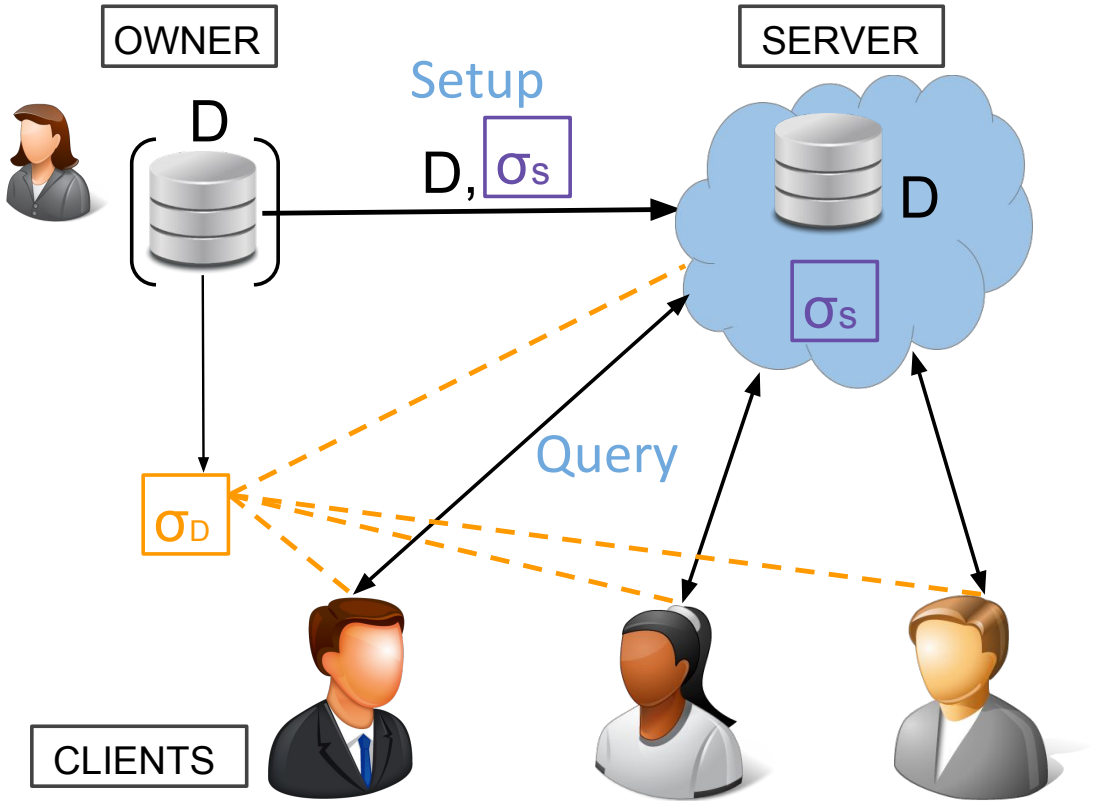


Figure 2.1: *Model $\mathcal{RQ}$: The owner runs* Setup *on database* DS *to produce database digest signature,* σ_{DS}*, and a digest for the server,* σ_S*. Every party has access to* σ_{DS}*, but only the server has access to* DS *and* σ_S*.* Query *protocol is executed between the client and the server.*

there exists a negligible function $\nu(.)$ *such that:*

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

where Query' *is an interactive query protocol executed between an honest client with input* σ_{DS} *and* q *and the adversary* Adv *with input* $\mathsf{DS}, \sigma_{\mathsf{DS}}, \sigma_S$*. 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 DS, using only oracle access to DS. 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* DS*. The real challenger runs* $(\sigma_{\mathsf{DS}}, \sigma_S) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{DS})$ *and sends* σ_{DS} *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* DS*. The simulator creates a fake* σ *by running* $(\sigma, \mathsf{st}_S) \leftarrow \mathsf{Sim}(1^\lambda, l)$ *(*st_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* DS*, 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* $\nu(\lambda)$*.*

2.5 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 $\mathcal{K}_{[a,b]}$.

2.5.1 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 DS. It then uses HIBE key generation to prove that there are no other key-value pairs in DS that belong to the queried interval.

Membership proofs using signatures are straightforward: the owner generates a signature for every key-value pair in DS 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 DS and the signature on (key, value, nonce) ties the (key, value) pair with a particular DS. Later, when a client queries for a range $[a,b]$ of DS, the server simply returns key-value pairs $(k_1, v_1), (k_2, v_2), \ldots, (k_m, v_m)$ of DS lying in this interval along with the corresponding signatures. For every (k,v) in the answer, the client makes sure that $a \leq k \leq b$ 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* DS *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* DS*. But this reveals that key 8 is not in* DS *and key 9 is in* DS*, 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 DS, for example, using a zero-knowledge accumulator [152]. 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

Figure 2.2: *Algorithm* $(\sigma_{\mathsf{DS}}, \sigma_S) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{DS})$ *run by the owner on input database* DS *with n key-value pairs.*

Step 1: Run $\mathsf{HIBE.Setup}(1^\lambda, l)$ to obtain pubic-secret key pair $(\mathsf{M}_G\mathsf{PK}, \mathsf{M}_G\mathsf{SK})$ 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 DS. 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: $\mathsf{SK}_{\mathsf{root}_i} \leftarrow \mathsf{HIBE.KeyGen}(\mathsf{M}_G\mathsf{SK}, \mathsf{root}_i)$.
Step 4: Pick a random $\mu \xleftarrow{\$} \{0,1\}^{\mathsf{poly}(\lambda)}$, which is unique for DS. For every $(k, v) \in \mathsf{DS}$, generate $\sigma_{(\mu,k,v)} \leftarrow \mathsf{Sig.Sign}(\mathsf{SigSK}, \mu, k, v)$.
Step 5: Set σ_S to $(\{\mathsf{root}_i, \mathsf{SK}_{\mathsf{root}_i}\}_{1 \le i \le t}, \{\sigma_{(\mu,\mathsf{key},\mathsf{val})}\}_{\mathsf{key} \in \mathsf{DS}})$.
Step 6: Set $\sigma_{\mathsf{DS}} = (\mathsf{M}_G\mathsf{PK}, \mathsf{SigPK}, \mu)$.
Step 7: Return $(\sigma_{\mathsf{DS}}, \sigma_S)$.

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 DS. On the other hand, building a technique that relies on all the keys in the universe $\mathcal{U}$ that are not present in DS 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 DS and not each key individually (since $|\mathsf{DS}| \ll |\mathcal{U}|$).

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 DS 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 DS, 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 $t = O(nl)$ [120]. 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 2.2.

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 DS 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

Figure 2.3: *Interactive protocol* $\mathsf{answer} \leftarrow \mathsf{Query}(\mathsf{DS}, q, \sigma_{\mathsf{DS}}, \sigma_S)$ *run by the client with input* $q = [a,b]$ *and* σ_{DS} *and the server with input* DS *and* σ_S

$C \rightarrow S$**:** The client sends q to the server.
$C \leftarrow S$**:** The server returns $\{(\mathsf{key}_1, \mathsf{val}_1), \ldots, (\mathsf{key}_m, \mathsf{val}_m)\}$ and $\sigma_{(\mu, \mathsf{key}_i, \mathsf{val}_i)}$ for $i \in [1,m]$.
$C \rightarrow S$**:** The client verifies all signatures $\sigma_{(\mu, \mathsf{key}_i, \mathsf{val}_i)}$ 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 $\mathsf{C}_{\mathsf{root}_j^*} \leftarrow \mathsf{HIBE.Encrypt}(\mathsf{M}_G\mathsf{PK}, \mathsf{root}_j^*, M_j)$ 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 $\mathsf{answer} = [(\mathsf{key}_1, \mathsf{val}_1), \ldots, (\mathsf{key}_m, \mathsf{val}_m)]$ iff she receives a decryption of each of her challenge messages M_j. Otherwise, she outputs $\perp$.

set of roots $R = \{\mathsf{root}_1^*, \ldots \mathsf{root}_{t'}^*\}$. The client's challenge is a sequence of ciphertexts that a server has 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 2.3.

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 DS whose key falls in the queried range (i.e., the first round in Figure 2.3). 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 [123] used HIBE (and HIBS) to prove non-membership queries on a set. Hence, our technique can be seen as a generalization of [123].

2.5.2 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.

2.5.3 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 DS, 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 2.3, 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 2.3, 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 2.5 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* DS *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 2.5 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 key 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).

2.6 Security Analysis

In this section, we prove that $\mathcal{RQ}$ construction based on HIBE is secure according to definitions in Section 2.4.

2.6.1 Proof of Soundness

We prove that our construction in Section 2.5 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 $\mathcal{A}$ that either breaks HIBE selective security or the unforgeability of the underlying signature scheme.

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

Adv can output two types of forgeries. First she could return answer with at least one $(k, v) \notin \mathsf{DS}^*$. 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. $(k, v) \in \mathsf{DS}^*$ and $a^* \leq k \leq b^*$ but $(k, v) \notin$ answer. We show that if Adv does the latter, using Adv we build $\mathcal{A}$ to break selective security of HIBE (Definition 1).

Given DS^* and $q^* = [a^*, b^*]$, $\mathcal{A}$ picks one of $(k_i, v_i) \in Q(q^*, \mathsf{DS}^*)$ randomly (i.e., she guesses that this is the key-value pair that Adv will omit in her forgery). $\mathcal{A}$ 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 $[a^*, b^*]$. $\mathcal{A}$ 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.

$\mathcal{A}$ proceeds by computing a set of roots R using $\mathsf{GetRoots}(l, [0^l, 2^l - 1], \mathsf{DS})$. 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, $\mathcal{A}$ can generate secret keys for all nodes in R using its HIBE.KeyGen oracle. $\mathcal{A}$ 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 DS^* that is not present in answer. (Recall that, if answer contains membership proof for some key-value pair not in DS^*,

then the unforgeability of the underlying signature scheme can be broken.)

Given a forgery $[a^*, b^*]$ $\mathcal{A}$ 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, $\mathcal{A}$ 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 $1/ml$: $\mathcal{A}$ 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. □

2.6.2 Proof of Zero-Knowledge

In this section we show how to build a simulator for the protocol in Section 2.5 to show that it has the zero-knowledge property as defined in Definition 4.

Recall that View contains the client digest if DS 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 $\sigma_{\mathsf{DS}} = (\mathsf{M}_G\mathsf{PK}, \mathsf{SigPK}, \mu)$. 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 DS.

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 $\sigma_{\mathsf{DS}} = (\mathsf{M}_G\mathsf{PK}, \mathsf{SigPK}, \mu)$ to Adv. and saves $\mu, \mathsf{M}_G\mathsf{SK}$ and SigSK.

Query phase. On query $q = [a,b]$, Sim queries the oracle on DS for an answer to q. Let $\{(\mathsf{key}_1, \mathsf{val}_1), \ldots, (\mathsf{key}_m, \mathsf{val}_m)\}$ 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 $\mathsf{M}_G\mathsf{SK}$ 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.

2.7 Experiments on Synthetic Data

In this section, we measure the cost of each step of the protocol described in Section 2.5. 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 [127]. 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 [86] 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.

2.7.1 Data

We have generated and made publicly available several synthetic datasets [8] 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 DS, while high clustering level suggests that DS 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 2.8, real-world datasets often exhibit high clustering.

We use datasets with the following clustering levels:

$\mathsf{DS}_\$$: the n keys are chosen at random;

DS_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 `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 DS_8. Let 0 and 32 be the two seeds. Then we generate two clusters: one with 8 keys and one with 2, and set DS_8 to their content. Hence, the resulting dataset contains keys $[000000, 000001, 000010, 000011, 000100, 000101,$
$000110, 000111, 100000, 100001]$.

2.7.2 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 2.1, 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 2.1: *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	$\mathsf{DS}_\$$	DS_{16}	DS_{128}	DS_n
10^2	15	80	1.2	0.8	0.8
	16	95	1.4	0.9	0.9
10^3	15	519	1.7	1.3	0.4
	16	1172	1.9	1.4	0.6
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.

2.7.3 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 2.2 (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 2.2 (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 DS_n, which has the highest variation. Since DS_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.

Table 2.2: *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
$n = 10^3$	$\mathsf{DS}_\$$	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)	-
	DS_{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)	-
	DS_{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)	-
	DS_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)	-
$n = 10^4$	$\mathsf{DS}_\$$	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)
	DS_{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)
	DS_{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)
	DS_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)

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 DS 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 $\mathsf{DS}_\$$ and DS_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 DS.

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 $\mathsf{DS}_\$$), 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.

2.7.4 Server Storage Cost

We measure the total storage cost at the server to store the data (DS) 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 2.5). Since this cost again depends on the number of empty ranges, we report the number of empty ranges for $\mathsf{DS}_{\$}$, DS_{128} and DS_n of varying sizes of n in Table 2.3. Since $\mathsf{DS}_{\$}$ 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 $\mathsf{DS}_{\$}$, the server storage is 62Mb while for $n = 10,000$ and DS_{128} it is only 0.3Mb.

2.7.5 Comparison with Ostrovsky *et al.* [127]

The protocol in [127] 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 2.3: *Number of* HIBE *secret keys by database size, n, and clustering level. Keys have* $l = 20$ *bits.*

n	Clustering level		
	$\mathsf{DS}_{\$}$	DS_{128}	DS_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 [127] 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 DS 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 [127] when using the timing result from [96]. The cost of one invocation of hash evaluation is estimated to be between $0.85ms$ and $3.5ms$ in [96]. We use the fastest time of $0.85ms$ in our estimate below.

In Table 2.4, we show that the server's cost of our protocol is orders of magnitude faster than the estimate of just one step of [127]. We note that the performance advantage of our protocol grows with the query answer size and clustering level. The clustering does not affect [127], while our approach makes use of it. Moreover, our performance does not degrade as the number of queries grows (recall that [127] is stateful).

2.7.6 Closest Point Query

We experiment with closest point query (Table 2.5) for key size $l = 24$ by varying the database size of DS_{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.

Table 2.4: *Total time (in seconds) for the server in our approach vs. time of one step of the server (CRR) in [127] for $l = 16$, $n = 10^4$.*

Answer size m	Our scheme			CRR [127]
	$\mathsf{DS}_{\$}$	DS_{128}	DS_n	
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 2.5: *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 DS_{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

2.8 Experiments on Real-World Data

In this section, we measure the cost of query and verification on two real-world datasets, *Enron email dataset* 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 2.6. We note that we ran experiments on the same machine and using the same parameters as in Section 2.7.

2.8.1 Enron Data

The Enron dataset [3] 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.

2.8.2 Boston Taxi Data

As our second example, we chose the Boston taxi dataset, which contains data about taxi rides in Boston from November 2012 [2], 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 [4], 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 [70]. 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 $\langle$`GeoHash`, `PICKUPADDRESS`$\rangle$ 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.

2.8.3 Performance

The results of our experiments are shown in Table 2.6 where we report averages and standard deviation over 100 runs.

Table 2.6: *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)

As we have already discussed in Section 2.7, 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 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 2.6. 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 2.5, the query time grows linearly with the size of query answer.

2.9 Conclusion

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.

Chapter 3

Zero-Knowledge Accumulators and Set Operations

3.1 Introduction

An extended abstract of this work appeared in [152] and the full version of this work is available on [75].

A cryptographic accumulator is a primitive that offers a way to succinctly represent a set $\mathcal{X}$ of elements by a single value acc referred to as the *accumulation value*. Moreover, it provides a method to efficiently and succinctly prove (to a party that only holds acc) that a candidate element x belongs to the set, by computing a constant-size proof proof, referred to as *witness*. The interaction is in a three-party model, where the *owner* of the set runs the initial key generation and setup process to publish the accumulation value. Later an untrusted *server* handles queries regarding the set issued by a number of *clients*, providing membership answers with corresponding witnesses.

Accumulators were originally introduced by Benaloh and del Mare in [19]. Since then, the relevant literature has grown significantly (see for example, [125, 17, 29, 34, 31, 53, 33, 15, 109, 36, 57][1]) with constructions in various models. At the same time, accumulators have found numerous other applications in the context of public-key infrastructure, certificate management and revocation, time-stamping, authenticated dictionaries, set operations, anonymous credentials, and more.

Traditionally in the literature, the security property associated with accumulators was *soundness* (or *collision-freeness*), expressed as the inability to compute a witness for an element $x \notin \mathcal{X}$. Subsequently, accumulators were extended to *universal accumulators* [107, 53, 15] that support non-membership proofs as well. Soundness for universal accumulators expresses the inability to forge a witness for an element, i.e., if $x \in \mathcal{X}$, it should be hard to prove $x \notin \mathcal{X}$ and vice-versa. No notion of privacy was considered, e.g., "what does an adversary that observes the client-server communication learn about the set $\mathcal{X}$" or "does the accumulation acc reveal anything about the elements of $\mathcal{X}$". It is clear to us that such a property would be attractive, if not—depending on the application—crucial. For example, in the context of securing the Domain Name

[1] We refer interested readers to [57] for a comprehensive review of existing schemes.

System (DNS) protocol by accumulating the set of records in a zone, it is crucial to not leak any information about values in the accumulated set while responding to queries.[2] As an additional example, recently Miers et al. [118] developed a privacy enhancement for Bitcoin, that utilizes the accumulator of [34]. In such a context, it is very important to minimize what is leaked by accumulation values and witnesses in order to achieve anonymity (for individuals and transactions).

Quite recently, de Meer et al. [54] and Derler et al. [57] suggested the introduction of an *indistinguishability* property for cryptographic accumulators, in order to provide some notion of privacy. Unfortunately, the definition of the former was inherently flawed, as noted in [57][3], whereas the later, while meant to serve cryptographic accumulators that support changes in the accumulated set (i.e., element insertion and deletion), did not protect the privacy of theses changes, as any adversary suspecting a particular modification can check the correctness of his guess.

In this work, we propose the notion of *zero-knowledge* for cryptographic accumulators. We define this property via an extensive real/ideal game, similar to that of standard zero-knowledge [81]. In the real setting, an adversary is allowed to choose his challenge set and to receive the corresponding accumulation. He is then given oracle access to the querying algorithm as well as an update algorithm that allows him to request updates in the set (receiving the updated accumulation value every time). In the ideal setting, the adversary interacts with a simulator that does not know anything about the set or the nature of the updates, other than the fact that an update occurred. Zero-knowledge is then defined as the inability of the adversary to distinguish between the two settings. Our notion of zero-knowledge differs from the privacy notion of [57], by protecting not only the originally accumulated set but also all subsequent updates. In fact, we formally prove that zero-knowledge is a strictly stronger property than indistinguishability in the context of cryptographic accumulators.

We provide the first zero-knowledge accumulator construction and prove its security. Our construction builds upon the *bilinear accumulator* of Nguyen [124] and achieves *perfect* zero-knowledge. Our scheme falls within the category of *dynamic universal* cryptographic accumulators: It allows to not only prove membership, but also non-membership statements (i.e., one can compute a witness for the fact that $x \notin \mathcal{X}$), and supports efficient changes in the accumulation value due to insertions and deletions in the set. It is secure under the q-Strong Bilinear Diffie-Hellman assumption, introduced in [22]. In order to provide non-membership witness computation in zero-knowledge, we had to deviate from existing non-membership proof techniques for the bilinear accumulator ([53, 15]). We instead used the disjointness technique of [131], appropriately enhanced for privacy. From an efficiency perspective, we show that the introduction of zero-knowledge to the bilinear accumulator comes at an insignificant cost: Asymptotically all computational overheads are either the same or within a poly-logarithmic factor of the construction of [124] that offers no privacy.

In general, a cryptographic accumulator can be viewed as special case of an *authenticated data structure* (ADS) [115, 114, 117, 149], where the supported data type is a set, and the queries are set membership/non-membership for elements of this set. As a result, our zero-knowledge accumulator has the same functionality as an authenticated set but with additional privacy property. Moreover, it falls within the general framework of *zero-knowledge authenticated data structures* (ZKADS) introduced recently in [150], where an underlying

[2] See for example, `https://tools.ietf.org/html/rfc5155`.

[3] Subsequently, the definition was strengthened in [140], but it is still subsumed by that of [57].

data structure supports queries such that the response to a query is verifiable and leaks nothing other than the answer itself.

Beyond set-membership One natural question is how to build a ZKADS with an expanded supported functionality that goes beyond set-membership. In particular, given multiple sets, we are interested in accommodating more elaborate set-operations (set union, intersection and difference).[4] We propose *zero-knowledge authenticated set collection* for the following setting. A party that owns a database of sets of elements outsources it to an untrusted server that is subsequently charged with handling queries, expressed as set operations among the database sets, issued by multiple clients. We provide the first scheme that provides not only integrity of set operations but also privacy with respect to clients (i.e., the provided proofs leak nothing beyond the answer). The fundamental building block for this construction is our zero-knowledge accumulator construction, together with a carefully deployed *accumulation tree* [132]. We note that if we restrict the security properties only to soundness—as is the case in the traditional literature of authenticated data structures—there are existing schemes (specifically for set-operations) by Papamanthou et al. [131] for the single-operation case, and by Canetti et al. [36] and Kosba et al. [101] for the case of multiple (nested) operations. However, none of these constructions offer any notion of privacy, thus our construction offers a natural strengthening of their security guarantees.

Contributions. Our contributions can be summarized as follows:

- We define the property of zero-knowledge for cryptographic accumulators and show that it is strictly stronger than existing privacy notions for accumulators.
- We describe the complex relations between cryptographic primitives in the area. Specifically, we show that zero-knowledge sets can be used in a black-box manner to construct zero-knowledge accumulators (with or without trapdoors). We also show that zero-knowledge accumulators imply primary-secondary-resolver membership proof systems [123].
- We provide the first construction of a zero-knowledge dynamic universal accumulator (with trapdoor). Our scheme is secure under the q-SBDH assumption and is perfect zero-knowledge.
- Using our zero-knowledge accumulator as a building block, we construct the first protocol for zero-knowledge outsourced set algebra operations. Our scheme offers secure and efficient intersection, union and set-difference operations under the q-SBDH assumption. We instantiate the set-difference operation in the random oracle model to achieve efficiency and discuss how it can be instantiated in the standard model with some efficiency overhead. Our construction (except for the update cost) is asymptotically as efficient as the previous state-of-the-art construction from [131], that offered no privacy guarantees.

Other related work. Our privacy notion is reminiscent of that of zero-knowledge sets [116, 42, 39, 108] where set membership and non-membership queries can be answered without revealing anything else about

[4] We stress that these operations form a complete set-operations algebra.

the set. Zero-knowledge sets are a stronger primitive since they assume no trusted owner: Server and owner are the same (untrusted) entity. On the other hand, accumulators (typically) yield more lightweight constructions with faster verification and constant-size proofs, albeit in the three-party model[5].

Very recently, Naor et al. [123] introduced primary-secondary-resolver membership proof systems, a primitive that is also a relaxation of zero-knowledge sets in the three-party model, and showed applications in network protocols in [80]. Our definitions are quite similar, however since they define non-adaptive security our zero-knowledge accumulators imply their definition. Moreover, their privacy notion is functional zero-knowledge, i.e., they tolerate some function of the set to be leaked, e.g., its cardinality. Finally, they only cater for the static case and have to rely on external mechanisms (e.g., time-to-live cookies) to handle changes in the set. In contrast, our zero-knowledge definition also protects updates and our construction has built-in mechanisms to handle them.

Existing works for cryptographic accumulators (e.g., [34, 124, 15, 107]) equip the primitive with zero-knowledge proof-of-knowledge protocols, such that a client that knows his value x is (or is not) in $\mathcal{X}$, can efficiently prove to a third-party arbitrator that indeed his value is (resp. is not) in the set, without revealing x. We stress that this privacy goal is very different from ours. Here we are ensuring that the entire protocol execution (as observed by a curious client or an external attacker) leaks nothing.

Regarding related work for set operations, the focus in the cryptographic literature has been on the privacy aspect with a very long line of works (see for example, [68, 99, 20, 85, 87]), some of which focus specifically on set-intersection (e.g., [90, 52, 50, 60]). The above works fit in the secure two-party computation model and most are secure (or can be made with some loss in efficiency) also against malicious adversaries, thus guaranteeing the authenticity of the result. However, typically this approach requires multi-round interaction, and larger communication cost than our construction. On the other hand, here our two security properties are "one-sided": Only the server may cheat with respect to soundness and only the client with respect to privacy; in this setting we achieve non-interactive solutions with optimal proof-size. There also exist works that deal exclusively with the integrity of set operations, such as [119] that achieves linear verification and proof cost, and [163] that only focuses on set-intersection but can be combined with an encryption scheme to achieve privacy versus the server.

Another work that is related to ours is that of Fauzi et al. [63] where the authors present an efficient non-interactive zero-knowledge argument for proving relations between committed sets. Conceptually, this work is close to zero-knowledge sets, allowing also for more general set operation queries. From a security viewpoint, this work is in the stronger two-party model hence it can accommodate our three-party setting as well. Also, from a functionality viewpoint, their construction works for (more general) multi-set operations. However, they rely on non-falsifiable knowledge-type assumptions to prove their scheme secure, and their scheme trivially leaks an upper-bound on the committed sets. Moreover, their construction cannot be efficiently generalized for operations on more than two sets at a time, and they do not explicitly consider efficient modifications in the sets.

We also note that recently other instantiations of zero-knowledge authenticated data structures have been proposed, including lists, trees and partially-ordered sets of bounded dimension [78, 150].

[5] See however the discussion of accumulators versus trapdoorless accumulators in Section 3.3.

3.2 Preliminaries

In this section we introduce notation and cryptographic tools that we will be using for the rest of the paper.

We denote with λ the security parameter and with $\nu(\lambda)$ a negligible function. A function $f(\lambda)$ is negligible if for each polynomial function $poly(\lambda)$ and all large enough values of λ, $f(\lambda) < 1/(poly(\lambda))$. We say that an event can occur with negligible probability if its occurrence probability can be upper bounded by a negligible function. Respectively, an event takes place with overwhelming probability if its complement takes place with negligible probability. The symbol $\xleftarrow{\$} \mathbb{X}$ denotes uniform sampling from domain $\mathbb{X}$. We denote the fact that a Turing machine Adv is probabilistic, polynomial-time by writing PPT Adv.

Bilinear pairings. Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p, generated by g. Let also $\mathbb{G}_T$ be a cyclic multiplicative group with the same order p and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear pairing with the following properties: (1) Bilinearity: $e(P^a, Q^b) = e(P,Q)^{ab}$ for all $P, Q \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$; (2) Non-degeneracy: $e(g,g) \neq 1_{\mathbb{G}_T}$; (3) Computability: There is an efficient algorithm to compute $e(P,Q)$ for all $P, Q \in \mathbb{G}$. We denote with $pub := (p, \mathbb{G}, \mathbb{G}_T, e, g)$ the bilinear pairings parameters, output by a randomized polynomial-time algorithm $\mathsf{GenParams}$ on input 1^λ. For clarity of presentation, we assume for the rest of the paper a symmetric (Type 1) pairing e. We note though that both our constructions can be securely implemented in the (more efficient) asymmetric pairing case, with straight-forward modifications (see [44] for a general discussion on pairings).

Our security proofs makes use of the q-Strong Bilinear Diffie-Hellman (q-SBDH) assumption over groups with bilinear pairings introduced by Boneh and Boyen in [22].

Assumption 1 (q-Strong Bilinear Diffie-Hellman)**.** *For any PPT adversary* Adv *and for q being a parameter of size polynomial in* λ*, there exists negligible function* $\nu(\lambda)$ *such that the following holds:*

$$\Pr[pub \leftarrow \mathsf{GenParams}(1^\lambda); s \leftarrow_R \mathbb{Z}_p^*; (z,\gamma) \in \mathbb{Z}_p^* \times \mathbb{G}_T \leftarrow \mathsf{Adv}(pub, (g^s, ..., g^{s^q})) : \gamma = e(g,g)^{1/(z+s)})] \leq \nu(\lambda)]$$

Non Interactive Zero Knowledge proof of Knowledge Protocols (NIZKPoK). A non-interactive zero-knowledge proof of knowledge protocol (NIZKPoK) for any NP language membership L proof system operates in the public random string model, and consists of polynomial-time algorithms P and V that work as follows: The algorithm P takes the common reference string σ and values (x,w) such that $x \in L$ and w is a witness to this. P outputs a proof π. The algorithm V takes (σ, x, π) as input and outputs accept or reject. The security properties of a NIZKPoK are the following:

Competeness: For all $x \in L$, for all witnesses w for x, for all values of random string σ and for all outputs π of $P(\sigma, x, w)$, $V(\sigma, x, \pi) = \mathsf{accept}$.

Soundness: For all adversarial prover algorithms P^*, for a randomly chosen σ, the probability that P^* can produce (x, π) such that $x \notin L$ but $V(\sigma, x, \pi) = \mathsf{accept}$ is negligible.

Knowledge Soundness with error δ*:* A zero-knowledge proof of knowledge protocol for any NP language membership L has a stronger form of soundness that says that if a cheating prover P^* convinces the verifier that $x \in L$ with noticeable probability (i.e., more than the soundness error), then not only this means that $x \in L$ but it actually means that P^* "knows" a witness in the sense that it could obtain a witness by running some algorithm. To put more formally, for every possibly cheating prover P^*, and

every x, if P^* produces π such that $Pr[V(\sigma,x,\pi) = \mathsf{accept}] > \delta + \rho$, ($\delta$ is the soundness error) then there's a algorithm E (called a knowledge extractor) with running time polynomial in $1/\rho$ and the running time of P^*, that on input x outputs a witness w for x with probability at least $1/2$.

Zero-Knowledge: There exist algorithms ZKSim-Setup and ZKSim-Prove, such that the following holds. ZKSim-Setup takes the security parameter as input and outputs (σ, s). For all x, ZKSim-Prove takes (σ, s, x) as input and outputs simulated proof π^S. Even for a sequence of adaptively and adversarially picked $(x_i, \ldots, x_m)$ (where m is polynomial in the security parameter), if $x_i \in L$ for $i \in [1, m]$, then the simulated proofs ${\pi_1}^S, \ldots {\pi_m}^S$ are distributed indistinguishably from proofs $\pi_1, \ldots, \pi_m$ that are computed by running $P(\sigma, x_i, w_i)$ where w_i is some witness that $x_i \in L$.

NIZKPoK protocol for Discrete Log (DL). Here we describe a NIZKPoK protocol based on DL assumption, following the style of Schnorr protocols [10]. The construction is non-interactive and uses the Fiat-Shamir transformation [65] for efficiency and its security is provable in the random-oracle (RO) model [18]. Informally, in the following protocol, the prover P proves to the verifier V that it knows the discrete log of a given value in zero-knowledge. We succinctly represent this protocol as $\mathsf{PK} = \{(h,x) : h = g'^x\}$. Let us denote the proof units sent by the prover P to the verifier V as $\mathsf{PKproof}$. We describe the protocol in the RO model in Figure 3.1, where $\mathcal{H}$ is a cryptographic hash function viewed as a RO.

The protocol proceeds as follows:
- P picks a random $u \in \mathbb{Z}_p^*$, computes $b \leftarrow g'^u$.
- Then P computes $c \leftarrow \mathcal{H}(b)$.
- P computes $r \leftarrow u + cx$ and sets $\mathsf{PKproof} := (b, c, r)$.
- Finally P sends $\mathsf{PKproof}$ to the verifier.

The verification proceeds as follows:
- Parse $\mathsf{PKproof}$ as (b, c, r).
- Verify if $c = \mathcal{H}(b)$. If not, **return** reject. Else proceed to next step.
- Verify if $g'^r = bh^c$. If the verification fails, **return** reject. Else **return** accept.

Figure 3.1: $\mathsf{PK} = \{(h,x) : h = g'^x\}$

Characteristic Polynomial. A set $\mathcal{X} = \{x_1, \ldots, x_n\}$ with elements $x_i \in \mathbb{Z}_p$ can be represented by a polynomial following an idea introduced in [68]. The polynomial $\mathsf{Ch}_\mathcal{X}(z) = \prod_{i=1}^{n}(x_i + z)$ from $\mathbb{Z}_p[z]$, where z is a formal variable, is called the *characteristic polynomial* of $\mathcal{X}$. In what follows, we will denote this polynomial simply by $\mathsf{Ch}_\mathcal{X}$ and its evaluation at a point y as $\mathsf{Ch}_\mathcal{X}(y)$.

The following lemma characterizes the efficiency of computing the characteristic polynomial of a set and of Extended Euclidian algorithm.

Lemma 1 ([138]). *Given a set $\mathcal{X} = x_1, ..., x_n \in \mathbb{Z}_p^n$, its characteristic polynomial $\mathsf{Ch}_\mathcal{X} := \sum_{i=0}^{n} c_i z^i \in \mathbb{Z}_p[z]$ can be computed with $O(n \log n)$ operations by FFT interpolation.*

We will use the following Lemma while proving correctness of coefficients of a polynomial:

Lemma 2 (Schwartz–Zippel). *Let $p[z], q[z]$ be two d-degree polynomials from $\mathbb{Z}_p[z]$. Then for $w \xleftarrow{\$} \mathbb{Z}_p$, the probability that $p(w) = z(w)$ is at most d/p, and the equality can be tested in time $O(d)$.*

If $p \in O(2^{\lambda})$, it follows that the above probability is negligible, if d is $poly(\lambda)$.

Complexity Model. To explicitly measure complexity with respect to the number of primitive cryptographic operations, without considering the dependency on the security parameter, we adopt the complexity model used in [131]. The *access complexity* of an algorithm is defined as the number of memory accesses this algorithm performs on the authenticated data structure stored in an indexed memory of n cells, in order for the algorithm to complete its execution. We require that each memory cell can store up to $O(poly(\log n))$ bits. The *group complexity* of data collection is defined as the number of elementary data objects contained in that object. Whenever it is clear from the context, we omit the terms "access" and "group".

Given a collection of sets $\mathcal{X}_{i_1}, \ldots \mathcal{X}_{i_k}$ and their characteristic polynomial representation, we summarize a characterization of the intersection of the sets in the following lemma.

Lemma 3 ([131])**.** *Set* answer *is the intersection of the sets $\mathcal{X}_{i_1}, \ldots \mathcal{X}_{i_k}$ if and only if there exists polynomials $q_1[z], \ldots q_k[z]$ such that $\sum_{j \in [i_1, i_k]} q_1[z] P_1[z] = 1$ where $P_j[z] = \mathsf{Ch}_{\mathcal{X}_j \setminus \mathsf{answer}}[z]$. Moreover, computing polynomials $q_j[z]$ where $j \in [i_1, i_k]$ has $O(N \log^2 N \log\log N)$ complexity where $N = \sum_{j \in [i_1, i_k]} n_j$ and $n_j = |\mathcal{X}_j|$.*

Accumulation tree: Given a collection of sets $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, let $\mathsf{acc}(\mathcal{X}_i)$ be a succinct representation (constant size) of $\mathcal{X}_i$ using its characteristic polynomial. We describe an authentication mechanism that does the following. A trusted party computes m hash values $h_i := h(\mathsf{acc}(\mathcal{X}_i))$ (using collision resistant cryptographic hash function) of the m sets of $\mathbb{S}$. Then given a short public digest information of the current set collection $\mathbb{S}$, the authentication mechanism provides publicly verifiable proofs of the form "h_i is the hash of the i^{th} set of the current set collection $\mathbb{S}$".

A popular authentication mechanism for proofs of this form are Merkle hash trees that based on a single value digest can provide logarithmic size proofs and support updates. An alternative authentication mechanism to Merkle trees, (specifically in the bilinear group setting) are *accumulation trees* [132]. Intuitively, an accumulation tree can be seen as a "flat" version of Merkle trees.

An accumulation tree (AT) is a tree with $\lceil \frac{1}{\varepsilon} \rceil$ levels, where $0 < \varepsilon < 1$ is a parameter chosen upon setup, and m leaves. Each internal node of T has degree $O(m^{\varepsilon})$ and T has constant height for a fixed ε. Each leaf node contains the h_i and each internal node contains the hash of the values of its children. We will describe the setup, query, update and verification of AT in the following algorithms. For simplicity, we skip an explicit key generation phase and describe the keys needed for each algorithm as its input. We have 3 kinds of keys for an AT: the secret key (sk), an evaluation key (ek) derivable from the secret key, which is used by the ATQuery algorithm for generating authentication paths, and finally a verification key (vk) corresponding to the secret key, which is used for verification of an authentication path. An accumulation tree scheme AT is defined as a tuple of 4 PPT algorithms: $\mathsf{AT} = (\mathsf{ATSetup}, \mathsf{ATQuery}, \mathsf{ATUpdate}, \mathsf{ATVerify})$. We describe the input and output of each of the algorithms here. To capture the notion of the most recent set collection, we use subscript t, i.e., $\mathbb{S}_t$ denotes the set collection at time t. Though this subscript is not necessary to describe these algorithms by themselves, we would need this index when using AT as a subroutine for our zero-knowledge authenticated set collection scheme (ZKASC). A detailed construction from [131] is given below. Additionally, we discuss how to extend the update algoithm to support a batch of updates efficiently.

Setup: $(\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0) \leftarrow \mathsf{ATSetup}(\mathsf{sk}, (\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m)))$ Recall that ATSetup takes a secret key

(sk) and a set of accumulation values $(\mathsf{acc}(\mathcal{X}_1),\ldots,\mathsf{acc}(\mathcal{X}_m))$ for a set collection $(\mathbb{S}_t)$ and builds an AT on top of it. This algorithm returns the authentication information for the set collection $(\mathsf{auth}(\mathbb{S}_t))$ and the root of the AT as the digest digest_t. The algorithm works as follows:

1. Pick a constant $0 < \varepsilon < 1$ and build an *accumulation tree* T as follows:
2. If v is a leaf corresponding to set $\mathcal{X}_i$, then set $d(v) = \mathsf{acc}(\mathcal{X}_i)^{\mathsf{sk}+i}$. Otherwise, set $d(v) = g^{\prod_{w\in C(v)}(\mathsf{sk}+h(d(w)))}$ where $C(v)$ denotes the children of node v.
3. Set $\mathsf{auth}(\mathbb{S}_t) = \{d(v)|v \in V(T)\}$
4. Let root be the root of T. Set $\mathsf{digest}_t := d(\mathsf{root})$
5. Return $(\mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$

Query: $(\Pi_i, \alpha_i) \leftarrow \mathsf{ATQuery}(\mathsf{ek}_t, i, \mathsf{auth}(\mathbb{S}_t))$ Recall that ATQuery takes the evaluation key ek_t, authentication information for the set collection $\mathsf{auth}(\mathbb{S}_t)$ and a particular index of the set collection and returns the authentication path for that set, denoted as Π_i and the accumulation value of that set as α_i. The algorithm works as follows:

1. Let leaf $v_0 = \mathsf{acc}(\mathcal{X}_i)$ and $v_0,\ldots,v_l$ be the leaf to root path in T.
2. Let $\mathsf{proof}_{j-1,j} = g^{\prod_{w\in\{C(v_j)-v_{j-1}\}}(\mathsf{sk}+h(d(w)))}$ for $j \in [1,l]$.
3. Let $\pi_j = (d(v_{j-1}), \mathsf{proof}_{j-1,j})$
4. Set $\Pi_i := \{\pi_j\}_{j\in[1,l]}$
5. Set $\alpha_i := \mathsf{acc}(\mathcal{X}_i)$.
6. (Π_i, α_i)

Update: $(\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}_i) \leftarrow \mathsf{ATUpdate}(\mathsf{sk}, i, \mathsf{acc}'(\mathcal{X}_i), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$ Recall that ATUpdate is the update algorithm that updates the accumulation value for a particular set $\mathcal{X}_i$, the authentication information for the set collection $\mathsf{auth}(\mathbb{S})$, and the root digest digest_t. This algorithm outputs the updated authentication information auth', the updated digest digest' and the update authentication information (in updinfo). The algorithm works as follows:

1. Let v_0 be the node corresponding to $\mathcal{X}_i$ and let $v_0, v_1, \ldots, v_l$ be the leaf to root path in T. Set $d'(v_0) := \mathsf{acc}(\mathcal{X}_i)$
2. For each node v_j on the path (i.e., $j = 1,\ldots,l$), set $d'(v_j) = g^{(\mathsf{sk}+d(v_{j-1}))^{-1}(\mathsf{sk}+d'(v_{j-1}))}$ where $d(v_{j-1})$ is the old value and $d'(v_{j-1})$ is the updated value.
3. Set $\mathsf{updinfo}_i = \{d'(v_{j-1})\}_{j\in[1,l]}$
4. Set $\mathsf{digest}' := d'(v_l)$.
5. Set $\mathsf{auth}' := \mathsf{auth}(\mathbb{S}_t) - \{d(v_{j-1})\}_{j\in[1,l]} + \{d'(v_{j-1})\}_{j\in[1,l]}$.
6. $(\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}_i)$.

Batch Update: $(\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}) \leftarrow \mathsf{ATUpdateBatch}(\mathsf{sk}, i_1, \mathsf{acc}'(\mathcal{X}_{i_1}), \ldots, i_k, \mathsf{acc}'(\mathcal{X}_{i_k}), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$ This algorithm is an extension of ATUpdate to support batch updates efficiently. Note this extension is our contribution.

1. The accumulation tree needs to be updated bottom-up starting at the leaf level. All the updates on a level have to be completed before proceeding to the next level. The update procedure is the

same as ATUpdate with the difference that instead of updating one leaf, here all the leaves are updated together.

2. At the leaf level, update all the leaves with $\mathsf{acc}'(\mathcal{X}_j)$ where $\mathsf{acc}'(\mathcal{X}_j)$ is the new accumulated value for $\mathcal{X}_j$, for $j \in [i_1, i_k]$.
3. Proceed to update the levels bottom up as described in ATUpdate.
4. At the end of update, return the updated authentication information auth', the updated root digest' and updinfo that contains the updated authentication path for each updated leaf $\mathcal{X}_j$.

Verify: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, i, \Pi_i, \alpha_i)$ Recall that ATVerify is the verification algorithm that takes the verification key of the scheme vk, digest of the set collection digest_t and a particular set index i along with its authentication path (Π_i) and accumulation value (α_i) as input and returns accept if the α_i is indeed the accumulation value of the i^{th} set of the collection. It returns reject otherwise. The algorithm works as follows:

1. Parse Π_j as $\{\pi_1, \ldots, \pi_l\}$.
2. Parse as π_j as (β_j, γ_j).
3. reject if any of the following is true:
   (a) $e(\beta_1, g) \neq e(\alpha_i, (\mathsf{vk})g^i)$
   (b) For some $j \in [2, l]$: $(\beta_j, g) \neq e(\gamma_{j-1}, (\mathsf{vk})g^{h(\beta_{j-1})})$
   (c) $e(\mathsf{digest}_t, g) \neq e(\gamma_l, (\mathsf{vk})g^{h(\beta_l)})$
4. accept otherwise.

The following lemma summarizes its security and efficiency.

Lemma 4. *[131] Under the q-SBDH assumption, for any adversarially chosen authentication path Π_i for y against an honestly generated digest,* digest*, such that* $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}, i, \Pi_i, y)$ *returns* accept*, it must be that y is the i^{th} element of the tree except for negligible probability. Algorithm* ATQuery *takes $O(m^{\varepsilon} \log m)$ and outputs a proof of $O(1)$ group elements and algorithm* ATVerify *has access complexity $O(1)$ and algorithm* ATUpdate *has access complexity $O(1)$.*

3.3 Zero-Knowledge Universal Accumulators

A cryptographic accumulator is a primitive that allows one to succinctly represent a set $\mathcal{X}$ of elements from a domain $\mathbb{X}$, by a single value acc from a (possibly different) domain $\mathbb{A}$, known as the *accumulation value*. Moreover, it provides a way to efficiently and succinctly prove that a candidate element x belongs to the set, (to a party that only holds acc) by computing a constant-size proof proof, referred to as *witness*.

Accumulators were introduced by Benaloh and de Mare in [19] as an alternative to cryptographic signatures for timestamping purposes. The authors provided the first construction in the RSA setting. Later, Baric and Pfitzmann [17] refined this construction by strengthening the security notion and Camenisch and Lysyanskaya [34] added the property of efficiently updating the accumulation value. More recently, Nguyen [124] and Camenisch et al. [32] proposed constructions in the prime-order bilinear group setting. Li et al. [107],

and Damgård and Triandopoulos [53] extended the RSA and bilinear accumulator respectively, adding the property to prove non-membership as well. Accumulators that achieve this are referred to as *universal*.

Trusted/untrusted setup. All the above constructions –and the one we provide here– are in the trusted-setup model, i.e., the party that generates the scheme parameters originally, holds some trapdoor information that is not revealed to the adversary. E.g., for the RSA-based constructions, any adversary that knows the factorization of the modulo can trivially cheat. An alternative body of constructions aims to build trapdoorless accumulators (also referred to as *strong* accumulators) [125, 126, 142, 29, 31, 109], where the owner is entirely untrusted (effectively the owner and the server are the same entity). Unfortunately, the earlier of these works are quite inefficient for all practical purposes, while the more recent ones either yield witnesses that grow logarithmically with the size of $\mathcal{X}$ or rely on algebraic groups, the use of which is not yet common in cryptography. Alternatively, trapdoorless accumulators can be trivially constructed from zero-knowledge sets [116], a much stronger primitive. While a scheme without the need for a trusted setup is clearly more attractive in terms of security, it is safe to say that we do not yet have a practical scheme with constant-size proofs, based on standard security assumptions.

In this work we aim to construct a dynamic, universal accumulator that also achieves a very strong privacy property. We formalize the required privacy property by defining *Zero-Knowledge Universal Accumulators* (ZKUA). Informally, the zero-knowledge property ensures that an adversarial party that sees the accumulation value as well as all membership and non-membership witnesses exchanged during the protocol execution learns nothing about the set, *not even its size*. This property even holds for adversarial clients that issue queries hoping to learn additional information about the set. Zero-knowledge guarantees that nothing can be learned from the protocol except for the answer to a query itself. In other words, explicitly querying for an element is the only way to learn whether an element appears in the set or not.

Moreover, privacy should also hold with respect to updates (insertions or deletions) in the set. That is, an adversary that observes the accumulation and witnesses both before and after an update should learn nothing about the occurred changes, other than the fact that an update occurred –not even whether it was an insertion or deletion.

In the rest of the section, we provide the definition for ZKUA and the corresponding security properties.

Note on definitional style. Traditionally in the literature, a cryptographic accumulator is defined as an ensemble of families of functions, referring to the function f that takes as input a set and a seed accumulation value and outputs a new accumulation value. At a high level, the requirements included the existence of an efficient algorithm to sample from the ensemble and that f can be efficiently evaluated. Moreover, f had to be quasi-commutative (i.e., after a series of invocations the final output is order-independent) which yields an efficient way to produce witnesses. Informally, a witness for $x \in \mathcal{X}$ consists of the evaluation of f on all the other elements of the set (corresponding to the accumulation value of $\mathcal{X} \setminus x$). Since f is quasi-commutative, a client can evaluate it using the witness and x and check that it is equal to the set accumulation value produced by the trusted owner. However, in this work we follow the more general definitional style introduced in [64], and more recently by [57]. The cryptographic accumulator is described as a tuple of algorithms which is more meaningful in the context of the applications discussed here. The quasi-commutativity property (while satisfied by our construction) is purposely omitted, as there exist more recent constructions of cryptographic

accumulators that do not satisfy it but have alternative ways for witness generation (e.g., hash-tree based construction).

A *universal accumulator* (UA) consists of four algorithms $(\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Verify})$ and it supports queries of the following form: "is $x \in \mathcal{X}$?" for elements from a domain $\mathbb{X}$. The response is of the form (b, proof) where b is a boolean value indicating if the element is in the set, i.e., $b = 1$ if $x \in \mathcal{X}$, $b = 0$ if $x \notin \mathcal{X}$ and proof is the corresponding membership/non-membership witness for x.

Definition 5. ***(Universal Accumulator)*** *A universal accumulator is a tuple of four PPT algorithms* $(\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Verify})$ *defined as follows:*

$(sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda)$

This probabilistic algorithm takes as input the security parameter and outputs a (public) verification key vk that will be used by the client to verify query responses and a secret key sk that is kept by the owner.

$(\mathsf{acc}, ek, \mathsf{aux}) \leftarrow \mathsf{Setup}(sk, \mathcal{X})$

This probabilistic algorithm is run by the owner. It takes as input the source set $\mathcal{X}$ *and produces the accumulation value* acc *that will be published to both server and client, and an evaluation key ek as well as auxiliary information* aux *that will be sent only to the server in order to facilitate proof construction.*

$(b, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{acc}, \mathcal{X}, x, ek, \mathsf{aux})$

This algorithm is run by the server. It takes as input the evaluation key and the accumulation value ek, acc *generated by the owner, the source set* $\mathcal{X}$*, a queried element x, as input. It outputs a boolean value b indicating whether the element is in the set and a witness* proof *for the answer.*

$(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{Verify}(\mathsf{acc}, x, b, \mathsf{proof}, vk)$

This algorithm is run by the client. It takes as input the accumulation value acc *and the public key vk computed by the owner, a queried element x, a bit b, the witness* proof *and it outputs* accept/reject.

The above definition captures what is known in the literature as *static* accumulator, where Setup has to be executed again whenever change in $\mathcal{X}$ occurs. The following defines dynamic accumulators, by introducing an algorithm Update that takes the current accumulation value and the description of an update (e.g., "insert x" or "remove x") and outputs the appropriately modified accumulation value, together with a possibly modified ek' and aux'. Of course, this can always be achieved by re-running Setup, but we also require that the execution of Update is faster than that of Setup. Moreover, there must also exist a Upd algorithm that takes the accumulation value and witness before an update, together with the new accumulation value after the update and produces a corresponding new witness.

Definition 6. ***(Dynamic Universal Accumulator)*** *A dynamic universal accumulator is a tuple of five PPT algorithms,* $\mathsf{DUA} = (\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Verify}, \mathsf{Update})$ *defined as follows:*

$(\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Verify})$ *as in Definition 5.*

$(\mathsf{acc}', ek', \mathsf{aux}') \leftarrow \mathsf{Update}(\mathsf{acc}, \mathcal{X}, x, sk, \mathsf{aux}, \mathsf{upd})$

This algorithm takes as input the current set with its accumulation value and auxiliary information, as well as an element x to be inserted to $\mathcal{X}$ if $\mathsf{upd} = 1$ *or removed from $\mathcal{X}$ if* $\mathsf{upd} = 0$. *If* $\mathsf{upd} = 1$ *and* $x \in \mathcal{X}$, *(likewise if* $\mathsf{upd} = 1$ *and* $x \notin \mathcal{X}$*) the algorithm outputs $\perp$ and halts, indicating an invalid update. Otherwise, it outputs $(\mathsf{acc}', ek', aux')$ where acc' is the new accumulation value corresponding to set $\mathcal{X} \cup \{x\}$ or $\mathcal{X} \setminus \{x\}$ (to be published), ek' is the (possibly) modified evaluation key, and aux' is respective auxiliary information (both to be sent only to the server).*

$(\mathsf{upd}, \mathsf{proof}') \leftarrow \mathsf{Upd}(\mathsf{acc}, \mathsf{acc}', x, \mathsf{proof}, y, ek', \mathsf{aux}, \mathsf{aux}', \mathsf{upd})$

This algorithm is to be run after an invocation of Update. *It take as input the old and the new accumulation values and auxiliary informations, the evaluation key ek' output by* Update, *as well as the element x that was inserted or removed from the set, according to the binary value* upd *(the same as in the execution of* Update*). It also takes a different element y and its existing witness* proof *(that may be a membership or non-membership witness). It outputs a new witness* proof' *for y, with respect to the new set $\mathcal{X}'$ as computed after performing the update. The output must be the* same *as the one computable by running* $\mathsf{Query}(\mathsf{acc}', \mathcal{X}', y, ek', \mathsf{aux}')$.

In terms of efficiency, the focus is on the owner's ability to quickly compute the new accumulation value after a modification in the set (typically with most existing constructions, the presence of the trapdoor information makes this achievable with a constant number of group operations). Slightly more formally, the runtime of Update must be asymptotically smaller than that of Setup on the updated set. An even more attractive property is the ability to update existing witnesses efficiently (i.e., not recomputing them from scratch) after an update occurs, with Upd. As a last remark, we point out that the ability to do this for positive witnesses is inherently more important than that of non-membership witnesses. The former corresponds to the (polynomially many) values in the set whereas the latter will be exponentially many (or infinite). A server that wants to cache witness values and update them efficiently can thus benefit more from storing pre-computed positive witnesses than negative ones (that are less likely to be used again).

Early cryptographic accumulator constructions had deterministic algorithms; once the trapdoor was chosen each set had a uniquely defined accumulation value. As discussed at the end of this section, the consequent introduction of privacy requirements led to constructions where Setup and Update are randomized, which, in turn, introduced the natural distinction of accumulators into deterministic and randomized. In fact, any deterministic accumulator trivially fails to meet even weak privacy notions: An adversary that suspects that set $\mathcal{X}$ is the pre-image of a given accumulation value can test this himself. In order to achieve the wanted zero-knowledge property, our definition also refers to randomized schemes but, to simplify notation, we omit randomness from the input (unless otherwise noted).

Another important point is which parties can run each algorithm. The way we formulated our definition, Setup and Update require knowledge of sk to execute, Query requires ek and Verify takes only vk. From a practical point of view, the owner is the party that is responsible for maintaining the accumulation value at all times (e.g., signing it and posting it to a public log); all changes in $\mathcal{X}$ should, in a sense, be validated by him first. On the other had, in most popular existing schemes (e.g., the RSA construction of [34] and the bilinear

accumulator of [124]) setup and update processes can also be executed by the server (who does not know the trapdoor sk) and the only distinction is that the owner can achieve the same result much faster (utilizing sk). The same is true for our construction here, but in the following security definitions we adopt the more general framework where the adversary is given oracle access to these algorithms, that captures both cases.

3.3.1 Security Properties

The first property we require from a cryptographic accumulator is completeness, i.e., a witness output by any sequence of invocations of the scheme algorithms, for a valid statement (corresponding to the state of the set at the time of the witness generation) is verified correctly with all but negligible probability.

Definition 7 (Completeness)**.** *Let $\mathcal{X}_i$ denote the set constructed after i invocations of the* Update *algorithm (starting from a set $\mathcal{X}_0$) and likewise for ek_i, aux_i. A dynamic universal accumulator is secure if, for all sets $\mathcal{X}_0$ where $|\mathcal{X}_0|$ and and $l \geq 0$ polynomial in λ and all $x_i \in \mathbb{X}$, for $0 = 1, \ldots, l$, there exists a negligible function $\nu(\lambda)$ such that:*

$$\Pr\left[\begin{array}{c} (sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda); (ek_0, \mathsf{acc}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(sk, \mathcal{X}_0); \\ \{(\mathsf{acc}_{i+1}, ek_{i+1}, \mathsf{aux}_{i+1}) \leftarrow \mathsf{Update}(\mathsf{acc}_i, \mathcal{X}_i, x_i, sk, \mathsf{aux}_i, \mathsf{upd}_i)\}_{0 \leq i \leq l} \\ (b, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{acc}_l, \mathcal{X}_l, x, ek_l, \mathsf{aux}_l) : \mathsf{Verify}(\mathsf{acc}_l, x, b, \mathsf{proof}, vk) = \mathsf{accept} \end{array}\right] \geq 1 - \nu(\lambda)$$

where the probability is taken over the randomness of the algorithms.

In the above we purposely omitted the Upd algorithm that was introduced purely for efficiency gains at the server. In fact, recall that we restricted it to return the exact same output as Update (run for the corresponding set and element) hence the value proof in the above definition might as well have been computed during an earlier update and subsequently updated by (one or more) calls of Upd.

The second property is soundness which captures that fact that adversarial servers cannot provide accepting witnesses for incorrect statements. It is formulated as the inability of Adv to win a game during which he is given oracle access to all the algorithms of the scheme (except for those he can run on his own using ek, aux –see discussion on private versus public setup and updates above) and is required to output such a statement and a corresponding witness.

Definition 8 (Soundness)**.** *For all PPT adversaries* Adv *running on input 1^λ and all l polynomial in λ, the probability of winning the following game, taken over the randomness of the algorithms and the coins of* Adv *is negligible:*

Setup *The challenger runs $(sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda)$ and forwards vk to* Adv*. The latter responds with a set $\mathcal{X}_0$. The challenger runs $(ek_0, \mathsf{acc}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(sk, \mathcal{X}_0)$ and sends the output to the adversary.*

Updates *The challenger initiates a list $\mathcal{L}$ and inserts the tuple $(acc_0, \mathcal{X}_0)$. Following this, the adversary issues update x_i and receives the output of $\mathsf{Update}(\mathsf{acc}_i, \mathcal{X}_i, x_i, sk, \mathsf{aux}_i, \mathsf{upd}_i)$ from the challenger, for $i = 0, \ldots, l$. After each invocation of* Update*, if the output is not $\perp$, the challenger appends the returned $(\mathsf{acc}_{i+1}, \mathcal{X}_{i+1})$ to $\mathcal{L}$. Otherwise, he appends $(\mathsf{acc}_i, \mathcal{X}_i)$.*

Challenge *The adversary outputs an index j, and a triplet $(x^*, b^*, \mathsf{proof}^*)$. Let $\mathcal{L}[j]$ be $(\mathsf{acc}_j, \mathcal{X}_j)$. The adversary wins the game if:*

$$\mathsf{Verify}(\mathsf{acc}_j, x^*, b^*, \mathsf{proof}^*, vk) = \mathsf{accept} \wedge ((x^* \in \mathcal{X}_j \wedge b^* = 0) \vee (x^* \notin \mathcal{X}_j \wedge b^* = 1))$$

discussion on the winning conditions of the game is due here. This property (also referred to as collision-freeness) was introduced in this format in [107] and was more recently adapted in [57] with slight modifications. In particular, Adv outputs set $\mathcal{X}^*$ and accumulation value acc^* as well as the randomness used (possibly) to compute the latter (to cater for randomized accumulators). It is trivial to show that the two versions of the property are equivalent.

An alternative, more demanding, way to formulate the game is to require that the adversary wins if he outputs two accepting witnesses for the same element and with respect to the same accumulation value (without revealing the pre-image set): a membership and a non-membership one. This property, introduced in the context of accumulators in [29], is known as *undeniability* and is the same as the privacy property of zero-knowledge sets. Recently, Derler et al. [57] showed that undeniability is a stronger property than soundness. However, existing constructions for undeniable accumulators are in the trapdoor-less setting (with the limitations discussed above); since our construction is the trusted setup setting, we restrict our attention to soundness. This should come as no surprise, as undeniability allows an adversary to provide a candidate accumulation value, without explicitly giving a corresponding set. In a three-party setting (with trusted setup) the accumulation value is always maintained by the trusted owner; there is no need to question whether it was honestly computed (e.g., whether he knows a set pre-image or even whether there exists one) hence undeniability in this model is an "overkill" in terms of security.

The last property is zero-knowledge. As we already explained, this notion captures that even a malicious client cannot learn anything about the set beyond what he has queried for. We formalize this in a way that is very similar to zero-knowledge sets (e.g, see the definition of [42]) appropriately extended to handle not only queries but also updates issued by the adversary. We require that there exists a simulator such that no adversarial client can distinguish whether he is interacting with the algorithms of the scheme or with the simulator that has no knowledge of the set or the element updates that occur, other than whether a queried element is in the set and whether requested updates are valid.

Definition 9 (Zero-Knowledge)**.** *Let D be a binary function for checking the validity of queries and updates on a set. For queries, $D(\mathsf{query}, x, \mathcal{X})) = 1$ iff $x \in \mathcal{X}$. For updates $D(\mathsf{update}, x, c, \mathcal{X})) = 1$ iff $(c = 1 \wedge x \notin \mathcal{X})$ or $(c = 0 \wedge x \in \mathcal{X})$. Let $\mathsf{Real}_{\mathsf{Adv}}(1^\lambda), \mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$ be games between a challenger, an adversary* Adv *and a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$, defined as follows:*

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

Setup *The challenger runs $(sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda)$ and forwards vk to* Adv*. The latter chooses a set $\mathcal{X}_0$ with $|\mathcal{X}_0| \in poly(k)$ and sends it to the challenger who in turn responds with acc_0 output by running the algorithm $\mathsf{Setup}(sk, \mathcal{X}_0)$. Finally, the challenger sets $(\mathcal{X}, \mathsf{acc}, \mathsf{aux}) \leftarrow (\mathcal{X}_0, \mathsf{acc}_0, \mathsf{aux}_0)$.*

Query *For $i = 1, \ldots, l$, where $l \in poly(\lambda)$,* Adv *outputs (op, x_i, c_i) where $\mathsf{op} \in \{\mathsf{query}, \mathsf{update}\}$ and $c_i \in \{0, 1\}$:*

If $\mathsf{op} = \mathsf{query}$**:** *The challenger runs* $(b, \mathsf{proof}_i) \leftarrow \mathsf{Query}(\mathsf{acc}, \mathcal{X}, x_i, ek, \mathsf{aux})$ *and returns the output to* Adv.

If $\mathsf{op} = \mathsf{update}$**:** *The challenger runs* $\mathsf{Update}(\mathsf{acc}, \mathcal{X}, x_i, sk, \mathsf{aux}, c_i)$. *If the output is not* $\perp$ *he updates the set accordingly to get* $\mathcal{X}_i$*, sets* $(\mathcal{X}, \mathsf{acc}, ek, \mathsf{aux}) \leftarrow (\mathcal{X}_i, \mathsf{acc}_i, ek_i, \mathsf{aux}_i)$ *and forwards* acc *to* Adv. *Else, he responds with* $\perp$.

Response *The adversary outputs a bit d.*

$\mathsf{Ideal}_{\mathsf{Adv}}(1^\lambda)$**:**

Setup *The simulator* Sim_1*, on input* 1^λ*, forwards vk to* Adv. *The adversary chooses a set* $\mathcal{X}_0$ *with* $|\mathcal{X}_0| \in poly(\lambda)$. Sim_1 *(without seeing* $\mathcal{X}_0$*) responds with* acc_0 *and maintains state* st_S. *Finally, let* $(\mathcal{X}, \mathsf{acc}) \leftarrow (\mathcal{X}_0, \mathsf{acc}_0)$.

Query *For* $i = 1, \ldots, l$ Adv *outputs* (op, x_i, c_i) *where* $\mathsf{op} \in \{\mathsf{query}, \mathsf{update}\}$ *and* $c_i \in \{0, 1\}$*:*

If $\mathsf{op} = \mathsf{query}$**:** *The simulator runs* $(b, \mathsf{proof}_i) \leftarrow \mathsf{Sim}_2(\mathsf{acc}, x_i, \mathsf{st}_S, D(\mathsf{query}, x_i, \mathcal{X}))$ *and returns the output to* Adv.

If $\mathsf{op} = \mathsf{update}$**:** *The simulator runs* $\mathsf{Sim}_2(\mathsf{acc}, \mathsf{st}_S, D(\mathsf{update}, x_i, c_i, \mathcal{X}))$. *If the output of* $D(\mathsf{update}, x_i, c_i, \mathcal{X})$ *is 1, let* $\mathcal{X} \leftarrow \mathcal{X}_i \cup x_i$ *in the case* $c_1 = 1$ *and* $\mathcal{X} \leftarrow \mathcal{X}_i \setminus x_i$ *in the case* $c_1 = 0$ *–i.e.,* $\mathcal{X}$ *is a placeholder variable for the latest set version at all times according to valid updates, that is however never observed by the simulator. The simulator responds to* Adv *with* acc'*. If the response* acc' *is not* $\perp$ *then* $\mathsf{acc} \leftarrow \mathsf{acc}'$.

Response *The adversary outputs a bit d.*

A dynamic universal accumulator is zero-knowledge if there exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that for all adversaries* Adv *there exists negligible function* ν *such that:*

$$|\Pr[\mathsf{Real}_{\mathsf{Adv}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathsf{Adv}}(1^\lambda) = 1]| \leq \nu(\lambda).$$

If the above probabilities are equivalent, then the accumulator is perfect zero-knowledge. If the inequality only holds for PPT Adv*, then the accumulator is computational zero-knowledge.*

Observe that, even though Adv may be unbounded (in the case of statistical or perfect zero-knowledge) the size of the set is always polynomial in the security parameter; in fact it is upper bounded by $O(|\mathcal{X}_0| + l)$. This ensures that we will have polynomial-time simulation, matching the real-world execution where all parties run in polynomial-time. Having computationally unbounded adversaries is still meaningful; such a party may, after having requested polynomially many updates, spend unlimited computational resources trying to distinguish the two settings.

While trying to provide a formal notion of privacy for cryptographic accumulators, the fact that the accumulation value computation must be randomized becomes evident. If Setup is a deterministic algorithm, then each set has a uniquely defined accumulation value (subject to particular *sk*) that can be easily reproduced by any adversary even with oracle access to the algorithm. This observation has already been made in [54, 55, 57].

3.4 Relation to Other Definitions

In this section, we discuss the relation of zero-knowledge accumulators with the existing notion of indistinguishable accumulations as well as with other cryptographic primitives.

3.4.1 Zero-knowledge implies indistinguishability (for accumulators)

The notion of zero-knowledge defined here is a strengthening of the indistinguishability property introduced in [57]. There the authors introduce a notion similar to ours that also requires the accumulation value produced by Setup to be randomized. If we restrict our attention to static accumulators, the effect of both notions is the same, i.e., the clients see a randomized accumulation value and corresponding "blinded" witnesses.

However, while the indistiguishability game entails updates, it inherently does not offer any privacy for the elements inserted to or removed from the set, as the Update algorithm is deterministic. At a high level, that notion only protects the original accumulated set and not subsequent updates. We believe this is an important omission for a meaningful privacy definition for accumulators, as highlighted by the following example. Consider, for example, a third-party adversary that observes the protocol's execution before and after an insertion (or deletion) update. If the adversary has reasons to suspect that the inserted (or deleted) value may be y, he can always test that. A very realistic example of this behavior is a setting where the accumulator is used to implement a revocation list. In that case an adversary may want to know if his fake certificate (value y in the above case) has been "caught" yet.

We provide the following result[6]:

Theorem 2. *Every zero-knowledge dynamic universal accumulator is also indistinguishable under the definition of [57], while the opposite is not always true.*

Proof. We first show that every scheme that is zero-knowledge is also indistinguishable. Then we show that the construction of [57] is not zero-knowledge.

ZK $\Rightarrow$ IND: We prove this direction by contradiction. Assume there exists an accumulator that is zero-knowledge but not indistinguishable. Then, there exists PPT adversary Adv that wins the indistinguishability game. Adv gives two sets $\mathcal{X}_0, \mathcal{X}_1$ to a challenger who flips a coin b and provides oracle access to Adv for the algorithms with respect to $\mathcal{X}_b$. By assumption, Adv can output a bit b' correctly guessing b with non-negligible advantage ε over $1/2$. The (natural) constraint is that Adv cannot issue a query (or update request) that is trivially revealing the chosen set (e.g., if $x \in \mathcal{X}_0$ and $x \notin \mathcal{X}_1$, Adv is not allowed to query for x). We defer interested readers to [57] for a formal definition of the indistinguishability game.

We will now construct PPT adversary Adv$'$ that breaks the zero-knowledge property of the scheme as follows. Adv$'$ on input $1^\lambda, vk$ runs Adv with the same input and receives sets $\mathcal{X}_0, \mathcal{X}_1$. He then forwards

[6] In [57] the indistinguishability definition assumes that the adversary is also given access to the Setup algorithm arbitrarily many times. This makes sense in their model, since they explicitly require that Setup is randomized whereas Update is deterministic. Here this requirement is redundant since both processes may be randomized; any setup response can be emulated by a series of update calls that shape the required set. To simplify the process, we assume that the indistinguishability adversary only makes Update and Query calls. We stress that this is not a limitation of the reduction. We could alternatively have chosen to define our zero-knowledge game giving the adversary access to Setup and the result would still hold.

$\mathcal{X}_1$ as the challenge for the zero-knowledge game and receives accumulation value acc_0, which he forwards to Adv. Consequently, he responds to all messages of Adv (queries and updates) with calls to the zero-knowledge game interface and forwards all responses back to Adv. Finally, he outputs the output bit b' of Adv.

Firstly, observe that Adv$'$ is clearly PPT, since Adv is PPT. Now let us argue about his success probability in distinguishing between real and ideal interaction. Observe that, if Adv$'$ is interacting with the algorithms of the scheme (i.e., is playing the real game), the interface he is providing to Adv is a perfect simulation of the indistinguishability game for $b = 1$. On the other hand, if he is interacting with Sim, the view of the latter during this interaction is exactly the same independently of whether the set chosen by Adv$'$ is $\mathcal{X}_0$ or $\mathcal{X}_1$. Hence, the view offered to Adv is the same in both cases, and therefore $\Pr[b' = 1] = \Pr[b' = 0] = 1/2$. Let E be the event that the Adv$'$ is playing the real game (and likewise for the complement E^c). From the above analysis (recall that Adv$'$ outputs the bit b' returned by Adv), it holds that $\Pr[b' = 1|E] > 1/2 + \varepsilon$ and $\Pr[b' = 1|E^c] = 1/2$. This implies that Adv$'$ can distinguish between the two executions with non-negligible probability, breaking the zero-knowledge property of the scheme. The claim follows by contradiction.

IND $\not\Rightarrow$ ZK: In the construction of [57], given the accumulation acc of set $\mathcal{X}$, the accumulation value acc$'$ of set $\mathcal{X} \cup x$, for $x \notin \mathcal{X}$, is computed via a deterministic Update algorithm, that is executable in polynomial time even without access to the trapdoor (and similarly for a deletion).

Assume now an adversary Adv that simply observes query execution and the accumulation value throughout the protocol, and wants to deduce whether value x is added to the set after a given update. Adv proceeds as follows (assuming acc is the accumulation state pre-update and acc$'$ the one published afterwards). After each update, run Update on input x, acc to receive acc$''$. Check whether acc$'' =$ acc$'$. If so, deduce that x was inserted, otherwise not. This clearly violates zero-knowledge. Recall that in the ideal game the simulator does not get access to the inserted (or deleted) element. Since the update algorithm is deterministic, given acc and x there exists a unique output accumulation value, and the simulator thus has negligible probability to emulate the real interaction.

This concludes our proof. □

The indistinguishability property of [57] is a strengthening of a notion introduced in [54]. The latter was the first work to formally define a privacy property for cryptographic accumulators, however their definition had inherent problems, e.g., it was easy to prove that deterministic accumulators –that clearly were not private– satisfied it. Another technique for providing privacy to cryptographic accumulators was proposed earlier in [107], without a formalization. The idea is to simply produce a randomized accumulation value for a set $\mathcal{X}$ by choosing at random an element x from the elements universe during Setup and outputting the accumulation of set $\mathcal{X} \cup \{x\}$. This generic mechanism will work for any static accumulator, but will also not protect updates. Moreover it weakens soundness as an adversary could potentially produce a membership witness for the element $x \notin \mathcal{X}$. Out approach does not suffer from such issues as there is no additional element accumulated and the randomness r used to blind the accumulation value during Setup is explicitly given to

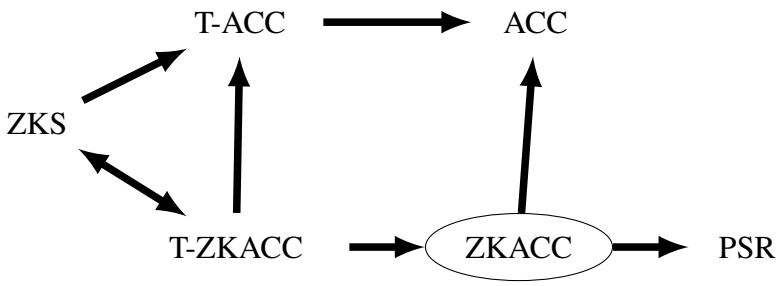


Figure 3.2: *Relations among cryptographic primitives for proof of membership and non-membership (static case). ZKS: zero-knowledge sets, T-ACC: trapdoorless accumulators, ACC: accumulators, T-ZKACC: trapdoorless zero-knowledge accumulators, ZKACC: our zero-knowledge accumulators (circled), PSR: primary-secondary-resolver membership proof systems.*

the server without compromising soundness.

Contrary to [57], our zero-knowledge property provably protects not only the original set but also all subsequent updates. In fact, the only thing that an adversary (client or third-party) learns is that an update happened; not even whether the update was an insertion or deletion! Liskov in [111] achieved a weaker notion of privacy for updates (called update transparency), in the model of zero-knowledge databases, that relies on assigning a pseudonym pattern $N(x)$ to each element x and leaks not only the fact that an update occurred but also an associated pseudonym. The author conjectures that the use of pseudonyms is unavoidable in order to achieve non-membership witnesses; here we show that this is not necessarily true, albeit in the more restricted three-party setting of cryptographic accumulators.

Finally, Theorem 2 implies that our construction from Section 3.5, is also the only known algebraic construction of a universal indistinguishable accumulator. The two schemes of [57] are a black-box reduction from the stronger primitive of zero-knowledge sets, and a construction similar to ours that only offers membership witnesses.

3.4.2 Relation to other primitives

Next we turn our attention to how zero-knowledge accumulators compare against other similar cryptographic primitives. We present a mapping of the research literature for the construction of cryptographic proofs for set-membership and non-membership, which has attracted significant attention lately. This is far from a complete presentation of results in the area; we focus on the relation between those primitives that are most closely related to the problem, avoiding general approaches (e.g., general-purpose zero-knowledge protocols) or related models that address similar problems (such as group signatures, e.g., [13]).

The overall picture for the static case (i.e., without assuming changes in the set) can be seen in Figure 3.2. Arrows denote implication, e.g., an arrow from A to B translates to "B can be built in black-box manner from A". Likewise, double-sided arrows denote equivalence of definitions, i.e., both can be constructed in a black-box manner from each other.

Zero-knowledge sets are a stronger primitive than accumulators; they satisfy the same soundness property with trapdoorless accumulators but they additionally offer privacy. Hence they are a starting point for our

mapping, since they can be used to build the other primitives.

In Section 3.3, we already discussed trapdoorless (or strong) accumulators. If a scheme is a trapdoorless accumulator it is secure with an untrusted setup execution, therefore (and quite trivially) it is also secure with a trusted setup, hence it is a also an accumulator.

As a mental exercise, let us now try to define the privacy-preserving counterparts of strong accumulators, i.e., *trapdoorless zero-knowledge accumulators*. Quite informally, the completeness and zero-knowledge definitions remain the same but the soundness property is replaced by the, strictly stronger, property of undeniability (see, e.g., [109] for a concrete definition), which is the same as the soundness property of zero-knowledge sets: By "merging" the existing soundness guarantee of trapdoorless accumulators with our zero-knowledge property (which, for the static case, is identical to that of zero-knowledge sets) we –quite unsurprisingly– ended up with zero-knowledge sets. We stress that latter exist in the common reference string model (or the trusted parameters model) hence this *must* also be true for trapdoorless zero-knowledge accumulators (e.g., a trusted authority runs the key-generation algorithm and publishes the result as a common reference string). On the contrary, this is not necessary for trapdoorless accumulators (without privacy) since the security game there is one-sided; the client can perform key-generation himself. As a final note, we point out, that zero knowledge (trapdoorless) accumulators imply (trapdoorless) accumulators since the former satisfy a strict superset of the security properties of the latter.

This equivalence of zero-knowledge sets and trapdoorless zero-knowledge accumulators can be useful in two ways: (i) more efficient (e.g., with smaller proof sizes) zero-knowledge sets may be achievable with techniques borrowed from the accumulators literature, and (ii) an impossibility result in one of the two models is translatable to the other. This holds, for example, in the case of the batch-update impossibility for accumulators of [30] and the lower bound for online public-key operation of [80]. We want to stress that our construction in Section 3.5 is not trapdoorless; to the best of our knowledge, the best known way to construct trapdoorless zero-knowledge accumulators is via a black-box reduction from zero-knowledge sets.

Another related primitive are primary-secondary-resolver membership proof systems (PSR) introduced in [123]. Their privacy notion a relaxation defined as functional zero-knowledge, i.e., the simulator is allowed to learn some function of the set (typically its size). Also, the games in the PSR definition are non-adaptive in the following sense: Adv needs to declare its cheating set before he even receives the corresponding keys (ek, vk for soundness and only vk for zero-knowledge –using our terminology)[7]. For the above reasons, while it is trivial that zero-knowledge accumulators imply PSR (where the leaked function is void), the other direction is generally not true. We stress that the above distinction between adaptive and selective security does not hold in the dynamic setting. There an adversary may declare a cheating set originally, receive the keys, and then modify his choice via a series of update calls (see, however, our discussion for this setting in the next paragraph).

Our results here are complementary to the relations proven in [123]. There, the authors prove that PSR systems exist, if and only if, one-way functions exist, which in turn implies that zero-knowledge sets cannot

[7] One could possibly modify the PSR model –and the security games– significantly to make them adaptive, by separating the key generation and setup algorithms. Indeed, to the best of our knowledge, the PSR construction of [80] would probably satisfy such a modified definition, assuming it was instantiated with an adaptively-secure signature scheme and an adaptively-secure verifiable random function.

be built in a black-box manner from PSR.

Dynamic setting. Once we move to the dynamic setting, where there exist efficient algorithms for modifications in the set, the relations are largely the same as in Figure 3.2, but some clarifications are in order.

Firstly, the only work addressing updatable zero-knowledge sets is [111], where two notions of privacy are introduced: opacity and transparency. The relations between definitions hold with respect to opacity. We defer the interested reader to that work for an in-detail discussion of the two properties. Here, we will only mention that an (efficient) construction for opaque zero-knowledge sets remains an open problem. On the other hand, when restricted to the three-party model (i.e., with trusted setup), it can be shown that our construction from Section 3.5 (with minor modifications) satisfies the opacity property.

Regarding the relation between zero-knowledge accumulators and PSR, matters are also straight-forward as the latter are explicitly defined only for the static case. In [123], the authors recommend the usage of techniques from certificate-revocation lists [121], as an additional external mechanism to accommodate updates. Contrary to this, our definitional approach is to make update-handling mechanisms explicitly part of the scheme. In this sense, zero-knowledge accumulators are a natural definitional extension of PSR in the dynamic setting. That said, we explicitly require that clients can at all times access the latest accumulation value, which would not be the case following the revocation scheme approach. We stress however that this does not necessitate authenticated channels between owner and clients; in practice it is achievable with a "timestamp-sign-and-publish" from the owner.

3.4.3 Relation to zero-knowledge authenticated data structures

Zero-knowledge accumulators can be seen as a straight-forward relaxation of zero-knowledge sets in the three-party model, i.e., in a honest-committer setting. The set is at all times maintained by a trusted party (the owner) that oversees insertions and deletions checking their validity. This is strongly reflected in the security property: Soundness in *zero-knowledge sets* (ZKS) does not require that the prover produces a commitment pre-image; indeed the prover may not even know such a set. On the other hand, soundness for zero-knowledge accumulators (trapdoorless) is defined as the inability of an adversary to produce a particular set and an element and a satisfying witness for a false statement. This stems from the fact that the trusted owner "authenticates" that the accumulation value known to clients (corresponding to the commitment in the case of ZKS) is indeed honestly computed, i.e., it corresponds to executing setup (and possibly update) on a known set. With that observation in mind, we can say that zero-knowledge accumulators are, in a sense, zero-knowledge authenticated sets. We note that a similar observation was made by [78] who addressed the problem of order queries on a list in both two-party and three-party settings. Their three-party model (*privacy-preserving authenticated list* (PPAL)) is also a similar relaxation of their two-party model (*zero-knowledge list* (ZKL)) .

This in turn, highlights the relation of zero-knowledge accumulators with the framework of zero-knowledge authenticated data structures (ZK-ADS), recently introduced in [151].[8] ZK-ADS extend the well-known primitive of authenticated data structures (ADS) adding an additional zero-knowledge property. The

[8]Though [151] uses the term Privacy-Preserving Authenticated Data Structures, we use ZK-ADS to fit our notation.

setting is the standard three-party model but now the supported type may be any kind of data structure. The choice of data structure defines the kind of data stored and the type of supported queries. In [151], the authors provided constructions for various types of data structures, in particular for a zero-knowledge authenticated list (i.e., a data structure that supports "insert-after", "delete" operations, as well as "order" queries), a tree, and a partially-ordered set (poset) of bounded dimension. Consequently, a zero-knowledge accumulator (or zero-knowledge authenticate set, as discussed above) is a type of ZK-ADS where the data structure is a set of elements supporting –unordered– insertions and deletions, and membership/non-membership queries.

The above constructions are the only ZK-ADS instantiations in the literature so far. One natural way to extend zero-knowledge authenticated sets to accommodate more elaborate query types is by allowing for set-operations beyond (non-)membership. In particular, consider a data structure, called set collection, that consists of a collection of sets and accommodates operations among (a subset of) them. We stress that a construction that accommodates set unions, intersection and differences, allows for a complete set-operation algebra (building any possible "circuit" of set-operations[9]). In Section 3.6 we provide a definition of *zero-knowledge authenticated set collection*, in the style of [151], and in Section 3.7 we provide the corresponding construction (which naturally uses our zero-knowledge authenticated set construction from Section 3.5 as a building block).

3.5 A Zero-knowledge accumulator from the q-Strong Bilinear Diffie-Hellman Assumption

In this section we present our construction for a zero-knowledge dynamic universal accumulator. It builds upon the bilinear accumulator of Nguyen [124], adopting some of the techniques of [57] that we further expand to achieve zero-knowledge. It supports sets with elements from $\mathbb{Z}_p \setminus \{s\}$ where p is prime and $p \in O(2^\lambda)$ and s is the scheme trapdoor. Note that, the fact that the elements must be of $\log p$ bits each, is not a strong limitation of the scheme; one can always apply a collision-resistant hash function that maps arbitrarily long strings to $\mathbb{Z}_p$. The description of the scheme can be seen in Figure 3.3.

Observe that the key vk published from the key-generating algorithm, reveals nothing for the set itself. The accumulation value produced by Setup is the standard bilinear accumulation value of [124] which is now blinded by a random value r, also revealed to the server. Witness generation for both cases utilizes this randomness r. For membership queries, the process is the same as in [124, 53] with one additional exponentiation with r for privacy purposes.

The major deviation occurs in the non-membership case. As previously discussed, there are existing works [53, 15] that enhance the bilinear accumulator to provide non-membership witnesses. Their technique is a complement of the one used for the membership case. At a high level, it entails proving that the degree-one polynomial $x+z$ does not divide $\mathsf{Ch}_\mathcal{X}[z]$, by revealing the scalar (i.e., zero-degree polynomial) remainder of their long division. Unfortunately, using this approach here entirely breaks the zero-knowledge property: It essentially reveals r (multiplied by an easily computable query-specific value) to any client. Instead, we adopt

[9] In the computationally-bounded setting, a negation operation is infeasible unless the element domain is of polynomial size in the security parameter. In that case, a negation can be instantiated as a set difference from the set that contains the entire domain.

an entirely different approach. Our scheme uses the set-disjointness test, first proposed by Papamanthou et al. [131], in order to prove non-membership of a queried element x. In order to prove that $x \notin \mathcal{X}$, the server proves the equivalent statement $\mathcal{X} \cap \{x\} = \emptyset$. The different nature of the proved statement allows us to use fresh query-specific randomness γ together with r to prove non-membership in zero-knowledge.

Verification is also different in the two cases, but always very efficient. Finally, the way updates are handled is especially important as it is another strong point of divergence from previous schemes that seek to provide privacy. After each update, a fresh randomness r' is used to blind the new accumulation value. Indeed this re-randomization technique that perfectly hides the nature of the change in $\mathcal{X}$ is what allows us to achieve our strong notion of zero-knowledge. Observe that, at all times, the owner maintains a variable N which is the maximum set-cardinality observed up to that point (through the original setup and subsequent insertions). When an insertion occurs that increases N (by one), then the owner provides to the server one additional ek component, that is necessary to the latter for subsequent witness generation. This is a slight deviation from our notation in Section 3.3 where the new key produced from Update replaces the previous ek. Instead the new evaluation key must be set to $ek \cup ek'$. This difference has no meaningful impact in the security of our scheme; we could always have Update output the entire old key together with the additional element. From an efficiency perspective though, that overly naive approach would require Update to run in time linear to N.

Regarding witness updates, observe that for the (more meaningful, as discussed in Section 3.3) case of membership witnesses there indeed exists a fast method. On the other hand, for non-membership witness updates, our scheme resorts to re-computation from scratch.

Efficiency. We discuss the asymptotic complexity of the algorithms of our construction. Let n be the cardinality of the set at any given time. GenKey runs in time $poly(\lambda)$. The algorithm Setup runs has access complexity $O(n)$ since a degree n polynomial can be evaluated in that many operations, and ek takes n consecutive exponentiations. From Lemma 1, computing the characteristic polynomial of a set of size n takes $O(n \log n)$, hence this is the overall complexity for membership witness generation. Computing a non-membership witness takes $O(n \log^2 n \log\log n)$, due to the complexity of the Extended Euclidean algorithm execution. The witnesses consist of $O(1)$ elements from $\mathbb{G}$ and verification requires can be done with $O(1)$ operations. Finally, Update has access complexity $O(1)$, membership witness updates can also be achieved in $O(1)$, and non-membership witness updates take as long as fresh witness generations, i.e., $O(n \log^2 n \log\log n)$.

One alternative way to run the scheme is to have the owner pre-compute all the membership witnesses at no additional asymptotic overhead during Setup. In that case, the server can just use the cached member witnesses and serve membership queries with $O(1)$ lookups. Whenever an update occurs, the server can take $O(n)$ independent witness updates to update all his cached positive witnesses. However, he would still need to compute the non-membership witnesses on the fly; pre-computation is impossible since there are exponentially many non-members of $\mathcal{X}$ and updating any cached ones in not faster than generating them.

In terms of storage requirements, the client only needs to store vk and acc, i.e., a constant number of bilinear group elements. The owner stores the set $\mathcal{X}$, hence $O(n)$ group complexity is trivially necessary.

From the description of $\mathsf{Setup}, \mathsf{Update}$, $|ek| = n$ at all times. Hence, the server's storage is also $O(n)$.

Overall, a comparison with the bilinear accumulator of [124] which achieves the exact same soundness property (under the same assumption), reveals that zero-knowledge is achieved by our construction with no asymptotic overhead at all for membership queries and a very small additional cost for non-membership queries. The above shows that the very strong privacy notion of zero-knowledge is achievable with minimal additional overhead.

Proving (non-)membership in batch. Another important property of our construction is that it allows the server to efficiently prove statements in batch. Assume a client holds an entire set $\mathcal{Y} = (y_1, \ldots, y_m)$ and wants to issue a query for each y_i. One way to achieve this would be to provide a separate membership/non-membership witness separately. This approach yields a proof that consists of $O(m)$ group elements.

However, with our construction the server can produce a single membership witness for all $y_i \in \mathcal{X}$ and a single non-membership witness for those $\notin \mathcal{X}$. The detailed construction for this case (as well as its practical benefits) is presented in Section 3.7. We can now present our main result:

Theorem 1. *The algorithms* $\{\mathsf{Gen}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Verify}, \mathsf{Update}, \mathsf{Upd}\}$ *constitute a zero-knowledge dynamic universal accumulator that: (i) has perfect completeness, (ii) is perfect zero-knowledge, (iii) is secure under the N-SBDH assumption, where N is the maximum set-size observed during the soundness game. Let n be the cardinality of the set. Then, the runtime of* GenKey *is* $O(poly(\lambda))$ *where* λ *is the security parameter, the access complexity of* Setup *is* $O(n)$*, that of* Query *is* $O(n \log n)$ *for membership witnesses and* $O(n \log^2 n \log\log n)$ *for non-membership witnesses, that of* Verify *is* $O(1)$*, that of* Update *is* $O(1)$*, and that of* Upd *is* $O(1)$ *for membership witnesses and* $O(n \log^2 n \log\log n)$ *for non-membership witnesses. Finally, witnesses consist of* $O(1)$ *bilinear group elements.*

Completeness follows by close inspection of the algorithms' execution. We proceed to prove soundness and zero-knowledge.

Proof of Soundness. Assume for contradiction that there exists PPT adversary Adv that on input 1^λ breaks the soundness of our scheme with non-negligible probability. We will construct a PPT adversary Adv' that breaks the N-SBDH assumption. Adv' runs as follows:

1. On input $(pub, (g^s, \ldots, g^{s^N}))$, run Adv on input $(g^s, pub, 1^\lambda)$.
2. Upon receiving set $\mathcal{X}_0$, choose $r_0 \xleftarrow{\$} \mathbb{Z}_p^*$. Use r_0 and $(g^s, \ldots, g^{s^N})$ to compute $\mathsf{acc}_0 = g^{r_0 \cdot \mathsf{Ch}_{\mathcal{X}_0}(s)} = g^{(\mathsf{Ch}_{\mathcal{X}_0}(s))^{r_0}}$ and respond with $(ek_0 = (g, g^s, \ldots, g^{s^{|\mathcal{X}_0|}}), \mathsf{acc}_0, r_0)$. Initiate list $\mathcal{L}$ and insert triplet $(\mathsf{acc}_0, \mathcal{X}_0, r_0)$ as $\mathcal{L}[0]$ (i.e., the first element of the list). The notation $\mathcal{L}[i]_j$ denotes the first part of the i-th element of the list (e.g., $\mathcal{L}[0]_0 = \mathsf{acc}_0$). Also set $n = |\mathcal{X}_0|$.
3. Initiate update counter $i = 0$. While $i \leq l$ proceed as follows. Upon receiving update upd_i, x_i, check whether this is a valid update for $\mathcal{X}_i = \mathcal{L}[i]_1$. If it is not, respond with $\perp$ and re-append $acc_i = \mathcal{L}[i]_0, \mathcal{X}_i, r_i$ to $\mathcal{L}$. Otherwise, pick $r' \xleftarrow{\$} \mathbb{Z}_p^*$ and set $r_{i+1} = r_i \cdot r'$. Update $\mathcal{X}_i$ according to upd_i, x_i to get $\mathcal{X}_{i+1}$. If $|\mathcal{X}_{i+1}| > n$, set $n = |\mathcal{X}_{i+1}|$ and $ek_{i+1} = g^n$. Else, $ek_{i+1} = \emptyset$. Use r_{i+1} and $(g^s, \ldots, g^{s^N})$ to compute $\mathsf{acc}_{i+1} = g^{r_{i+1} \cdot \mathsf{Ch}_{\mathcal{X}_{i+1}}(s)} = g^{(\mathsf{Ch}_{\mathcal{X}_{i+1}}(s))^{r_{i+1}}}$ and respond with $(ek_{i+1}, \mathsf{acc}_{i+1}, r_{i+1})$. Append triplet $(\mathsf{acc}_{i+1}, \mathcal{X}_{i+1}, r_{i+1})$ to $\mathcal{L}$. In both cases, increase i by 1.

4. Upon receiving challenge index j and challenge triplet $(x^*, b^*, \mathsf{proof}^*)$ proceed as follows:
   - If $b^* = 1$, then $x^* \notin \mathcal{X}_j$ yet $\mathsf{Verify}(\mathsf{acc}_j, x^*, 1, vk)$ accepts. Compute polynomial $q[z]$ and scalar c such that $\mathsf{Ch}_{\mathcal{X}_j}[z] = (x^* + z)q[z] + c$. Output $[x^*, (e(\mathsf{proof}^*, g^{(x^*+s)})^{r_j^{-1}} e(g, g^{-q(s)}))^{c^{-1}}]$.
   - If $b^* = 0$, then $x^* \in \mathcal{X}_j$ yet $\mathsf{Verify}(\mathsf{acc}_j, x^*, 0, vk)$ accepts. Parse proof^* as (W_1^*, W_2^*). Compute polynomial $q[z]$ such that $\mathsf{Ch}_{\mathcal{X}_j}[z] = (x^* + z)q[z]$. Output $[x^*, (e(W_1^*, g^{r_j \cdot q(s)}) e(W_2^*, g))]$.

First of all observe that Adv' perfectly emulates the challenger for the DUA security game to Adv. This holds since all accumulation values and witness are computable without access to trapdoor sk in polynomial time. All the necessary polynomial arithmetic can be also run efficiently hence Adv' is PPT.

Regarding its success probability, we argue for the two cases separately as follows:

$\mathbf{b}^* = \mathbf{1}$ Since $x^* \notin \mathcal{X}_j$, it follows that $(x^* + z) \nmid \mathsf{Ch}_{\mathcal{X}_j}[z]$ which guarantees the existence of $q[z], c$. Also observe that c is a scalar (zero-degree polynomial) since it is the remainder of the polynomial division and it must have degree less than that of $(x^* + z)$. Since verify accepts we can write:

$$\begin{aligned} e(\mathsf{proof}^*, g^{x^*} \cdot g^s) = e(\mathsf{proof}^*, g)^{(x^*+s)} &= e(\mathsf{acc}_j, g) \\ &= e(g^{r_j \cdot \mathsf{Ch}_{\mathcal{X}_j}(s)}, g) \\ &= e(g, g)^{r_j((x^*+s)q(s)+c)}, \end{aligned}$$

from which it follows that:

$$\begin{aligned} e(\mathsf{proof}^*, g)^{r^{-1}(x^*+s)} &= e(g, g)^{(x^*+s)q(s)+c} \\ e(\mathsf{proof}^*, g)^{r^{-1}} &= e(g, g)^{q(s)+c/(x^*+s)} \\ e(\mathsf{proof}^*, g)^{r^{-1}} e(g, g)^{-q(s)} &= e(g, g)^{c/(x^*+s)} \\ [e(\mathsf{proof}^*, g)^{r^{-1}} e(g, g)^{-q(s)}]^{c^{-1}} &= e(g, g)^{1/(x^*+s)}. \end{aligned}$$

$\mathbf{b}^* = \mathbf{0}$ Since $x^* \in \mathcal{X}_j$, it follows that $(x^* + z) | \mathsf{Ch}_{\mathcal{X}_j}[z]$ which guarantees the existence of $q[z]$. Since verify accepts we can write:

$$\begin{aligned} e(W_1^*, \mathsf{acc}_j) e(W_2^*, g^{x^*} \cdot g^s) &= e(g, g) \\ e(W_1^*, g^{r_j \cdot \mathsf{Ch}_{\mathcal{X}_j}(s)}) e(W_2^*, g^{(x^*+s)}) &= e(g, g) \\ e(W_1^*, g^{r_j(x^*+s)q(s)}) e(W_2^*, g^{(x^*+s)}) &= e(g, g) \\ [e(W_1^*, g^{r_j \cdot q(s)}) e(W_2^*, g)]^{(x^*+s)} &= e(g, g) \\ [e(W_1^*, g^{r_j \cdot q(s)}) e(W_2^*, g)] &= e(g, g)^{1/(x^*+s)} \end{aligned}$$

Observe that in both cases the left hand of the above equations is efficiently computable with access to $pub, (g^s, \ldots, g^{s^N}), r_j, \mathcal{X}_j, x^*, \mathsf{proof}^*$. Hence, whenever Adv' succeeds in breaking the soundness of our scheme, Adv' outputs a pair breaking the N-SBDH assumption. By assumption the latter can happen only with negligible probability, and our claim that our scheme has soundness follows by contradiction. □

Proof of Zero-Knowledge. We define simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ as follows. At all times, we assume st_S contains all variables seen by the simulator this far.

- Sim_1 runs $\mathsf{GenParams}$ to receive pub. He then picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and sends g, g^s, pub to Adv. After Adv has output his set choice $\mathcal{X}$, Sim_1 picks $r \xleftarrow{\$} \mathbb{Z}_p^*$ and responds with $\mathsf{acc} = g^r$. Finally, he stores r initiates empty list $\mathcal{C}$.
- For $i = 1, \ldots, l$ upon input (op, x_i, c_i):
  - If $\mathsf{op} = \mathsf{query}$, the simulator checks if $x_i \in \mathcal{C}$. If not, then if $D(\mathsf{query}, x_i, \mathcal{X}) = 1$, he computes $\kappa = r \cdot (x_i + s)^{-1}$ and responds with $(b = 1, \mathsf{proof} = g^{\kappa})$. Else, if $D(\mathsf{query}, x_i, c_i, \mathcal{X}) = 1$ he computes q_1, q_2 such that $q_1 \cdot r + q_2 \cdot (x_i + s) = 1$, picks $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and responds with $(b = 1, \mathsf{proof} = (W_1 = g^{q_1 + \gamma(x_i + s)}, W_2 = g^{q_2 - \gamma r}))$. In both cases, the simulator appends (x_i, b, proof) to $\mathcal{C}$. Finally, if $x_i \in \mathcal{C}$ he responds with the corresponding entries b, proof.
  - If $\mathsf{op} = \mathsf{update}$ then the simulator proceeds as follows. If $D(\mathsf{update}, x_i, c_i, \mathcal{X}) = 0$ then he responds with $\perp$. Else, he picks $r' \xleftarrow{\$} \mathbb{Z}_p^*$ and responds with $\mathsf{acc} = g^{r'}$. Finally he sets $r \leftarrow r$ and $\mathcal{C} \leftarrow \emptyset$.

The simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ produces a view that is identically distributed to that produced by the challenger during $\mathsf{Real}_{\mathsf{Adv}}$. Observe that random values r are chosen independently after each update (and initial setup) in both cases. Once s, r are fixed then for any possible choice of $\mathcal{X}$ there exists unique $r^* \in \mathbb{Z}_p^*$ such that $g^r = g^{r^* \cdot \mathsf{Ch}_{\mathcal{X}}(s)}$. It follows that the accumulation values in $\mathsf{Real}_{\mathsf{Adv}}$ are indistinguishable from the (truly random) ones produced by Sim. For fixed s, r, given a set-element combination $(\mathcal{X}, x_i)$ with $x_i \in \mathcal{X}$, in each game there exists a unique membership witness proof that satisfies the verifying equation. For negative witness $\mathsf{proof} = (W_1, W_2)$, given a set-element combination $(\mathcal{X}, x_i)$ with $x_i \notin \mathcal{X}$, for each possible independently chosen value of γ, in both games there exists only one distinct corresponding pair W_1, W_2 that satisfies the verifying equation.

It follows that the probabilities in Definition 9 are equivalent and our scheme is perfect zero-knowledge.

□

□

3.6 Zero-Knowledge Authenticated Set Collection (ZKASC)

Zero-knowledge accumulators presented so far provide a succinct representation for maintaining a dynamic set and replying (non-)member queries in zero-knowledge. As we already discussed, zero-knowledge accumulators (without trapdoor) can also be viewed as zero-knowledge authenticated sets (ZK-AS) where authenticated zero-knowledge membership/non-membership queries are supported on an outsourced set. In this section, we generalize this problem to a collection of sets and study verification of outsourced set algebra operations in zero-knowledge, which we refer to as *zero-knowledge authenticated set collection* (ZKASC). In particular, we consider a dynamic collection $\mathbb{S}$ of m sets $\mathcal{X}_1, \ldots, \mathcal{X}_m$ that is remotely stored on an untrusted server. We then develop mechanisms to answer primitive queries on these sets (is-subset, intersection, union and set difference) such that the answers to these queries can be verified publicly and in zero-knowledge. That is, the proofs of the queries should reveal nothing beyond the query answer. In addition, we require the

verification of any set operation to be operation-sensitive, i.e., the required complexity depends only on the (description and outcome of the) operation, and not on the sizes of the involved sets.

The *sets collection* data structure $\mathbb{S}$, consists of m sets, denoted with $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, each containing elements from a universe $\mathbb{X}$. A set does not contain duplicate elements, however an element can appear in more than one set. The *abstract data type* for set collection is defined as $\mathbb{S}$ with two types of operations defined on it: immutable operations $Q()$ and mutable operations $U()$. $Q(\mathbb{S}, q)$ takes a set algebra query q as input and returns an answer and a proof and it does not alter $\mathbb{S}$. The queries are defined with respect to the indices of a collection of sets $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$. $U(\mathbb{S}, u)$ takes as input an update request and changes $\mathbb{S}$ accordingly. It then outputs the modified set collection $\mathbb{S}'$. An update $u = (x, \mathsf{upd}, i)$ is either an insertion (if $\mathsf{upd} = 1$) of an element x into a set $\mathcal{X}_i$ or a deletion (if $\mathsf{upd} = 0$) of x from $\mathcal{X}_i$. The following queries are supported on $\mathbb{S}$:

Subset The query q takes a set of elements Δ and a set index i as input and returns answer where $\mathsf{answer} = 1$ if $\Delta \subseteq \mathcal{X}_i \in \mathbb{S}$, and $\mathsf{answer} = 0$, otherwise.

Set Difference The query q takes two set indices i_1 and i_2 and returns $\mathsf{answer} = \mathcal{X}_{i_1} \setminus \mathcal{X}_{i_2}$.

Intersection The query q takes a set of indices $(i_1, \ldots, i_k)$ as input and returns $\mathsf{answer} = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2} \cap \ldots \cap \mathcal{X}_{i_k}$.

Union The query q takes a set of indices $(i_1, \ldots, i_k)$ as input and returns $\mathsf{answer} = \mathcal{X}_{i_1} \cup \mathcal{X}_{i_2} \cup \ldots \cup \mathcal{X}_{i_k}$.

3.6.1 Model

ZKASC can be seen as the traditional *authenticated data structure* (ADS) model with the added requirement of privacy (zero-knowledge). Indeed, ZKASC follows the model of zero-knowledge authenticated data structure [150] instantiated for a set collection and set algebra queries. In particular, ZKASC is a tuple of six probabilistic polynomial time algorithms $\mathsf{ZKASC} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Update}, \mathsf{UpdateServer}, \mathsf{Query}, \mathsf{Verify})$. We note that zero-knowledge authenticated set also follows the zero-knowledge authenticated data structure [150] model where the data structure consists of a single set and the supported queries are membership and non-membership queries.

We first describe how the algorithms of ZKASC are used between the three parties of our model and then give their API. The owner uses Gen to generate the necessary keys. He then runs Setup to prepare $\mathbb{S}_0$ for outsourcing it to the server and to compute digest for the client and necessary auxiliary information for the server. The owner can update his set collection and make corresponding changes to digest using Update. Since the set collection and the information of the server need to be updated on the server as well, the owner generates an update string that is enough for the server to make the update herself using UpdateServer. The client can query the data structure by sending queries to the server. For a query, the server runs Query and generates answer. Using the auxiliary information, she also prepares a proof of the answer. The client then uses Verify to verify the query answer against proof and the digest he has received from the owner after the last update.

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda)$ where 1^λ is the security parameter. Gen outputs a secret key (for the owner) and the corresponding verification key vk.

$(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk}, \mathsf{vk}, \mathbb{S}_0)$ where $\mathbb{S}_0$ is the initial set collection and sk, vk are the

keys generated by Gen. Setup computes the authentication information $\mathsf{auth}(\mathbb{S}_0)$ for $\mathbb{S}_0$, a short digest digest_0 for $\mathbb{S}_0$, an evaluation key ek_0 and auxiliary information aux_0. digest_0 is public, while $\mathsf{auth}(\mathbb{S}_0)$, ek_0 and aux_0 are sent to the server. These units lets the server compute proofs of query answer of the clients.

$(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t) \leftarrow \mathsf{Update}(\mathsf{sk}, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, \mathsf{ek}_t, \mathsf{SID}_t, u_t)$
where $u_t = (x, \mathsf{upd}, i)$ is an update operation to be performed on $\mathbb{S}_t$. SID_t is set to the output of a function f on the queries invoked since the last update (Setup for the 0^{th} update). Update returns the updated set collection, $\mathbb{S}_{t+1} = U(\mathbb{S}_t, u)$, the corresponding the authentication information $\mathsf{auth}(\mathbb{S}_{t+1})$, the evaluation key ek_{t+1} and auxiliary information aux_{t+1}, the updated public digest digest_{t+1}, and an update string $\mathsf{updinfo}_t$ that is used by the server to update her information. Note that the evaluation key is a part of the information the server requires to compute proofs of answers to the client queries. The secret key and the verification key do not change throughout the scheme.

$(\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{ek}_t, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, u_t, \mathsf{updinfo}_t)$
where $\mathsf{updinfo}_t$ is used to update $\mathsf{auth}(\mathbb{S}_t)$, digest_t, aux_t and ek_t to $\mathsf{auth}(\mathbb{S}_{t+1})$, digest_{t+1}, aux_{t+1} and ek_{t+1} respectively. u_t is used to update $\mathbb{S}_t$ to $\mathbb{S}_{t+1}$.

$(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, q)$ where q depends on the exact set algebra operation. In particular, for subset query $q = \Delta, i$ where Δ denotes a set of elements and i denotes the set index of $\mathbb{S}_t$. For intersection and union queries $q = i_1, \ldots, i_k$ where $i_1, \ldots, i_k$ are set indices of $\mathbb{S}_t$ and for set difference $q = i_1, i_2$. where i_1, i_2 are indices of $\mathbb{S}_t$. The algorithm outputs the query answer answer, is its proof proof.

$(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{Verify}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$ where input arguments are as defined above. The output is accept if $\mathsf{answer} = Q(\mathbb{S}_t, q)$, and reject, otherwise.

We leave function f to be defined by a particular instantiation. An example could be making f return the cardinality of its input. Once defined, f remains fixed for the instantiation. Since the function is public, anybody, who has access to the (authentic) queries since the last update, can compute it.

3.6.2 Security Properties

A zero-knowledge authenticated data structure [150] has three security properties: completeness, soundness and zero-knowledge. Our ZKASC adapts these properties as follows.

Completeness dictates that if all three parties are honest, then for a set collection, the client will always accept an answer to his query from the server. Here honest behavior implies that whenever the owner updates the set collection and its public digest, the server updates the set collection and her authentication and auxiliary information accordingly and replies client's queries faithfully w.r.t. the latest set collection and digest.

Definition 10 (Completeness)**.** *For an ZKASC* $(\mathbb{S}_0, Q, U)$*, any sequence of updates* $u_0, u_1, \ldots, u_L$ *on the set*

collection $\mathbb{S}_0$, and for all queries q on $\mathbb{S}_L$:

$$\Pr[(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda); (\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{vk},\mathbb{S}_0);$$

$$\Big\{(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t) \leftarrow$$
$$\mathsf{Update}(\mathsf{sk}, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, \mathsf{ek}_t, \mathsf{SID}_t, u_t);$$
$$(\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{ek}_t, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, u_t, \mathsf{updinfo}_t); \Big\}_{0\le t\le L}$$

$$(\mathsf{answer},\mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{ek}_L, \mathsf{aux}_L, \mathsf{auth}(\mathbb{S}_L), \mathbb{S}_L, q) :$$
$$\mathsf{Verify}(\mathsf{vk}, \mathsf{digest}_L, \mathsf{answer}, \mathsf{proof}) = \mathsf{accept} \wedge \mathsf{answer} = Q(\mathbb{S}_L, q)] = 1.$$

where the probability is taken over the randomness of the algorithms.

Soundness protects the client against a malicious server. This property ensures that if the server forges the answer to a client's query, then the client will accept the answer with at most negligible probability. The definition considers adversarial server that picks the set collection and adaptively requests updates. After seeing all the replies from the owner, she can pick any point of time (w.r.t. updates) to create a forgery.

Since, given the authentication and auxiliary information to the server, the server can compute answers to queries herself, it is superfluous to give Adv explicit access to Query algorithm. Therefore, we set input of f to empty and SID to $\perp$, as a consequence, in algorithm Update.

Definition 11 (Soundness)**.** *For all PPT adversaries,* Adv *and for all possible valid queries q on the set collection $\mathbb{S}_j$ of an abstract data type $(\mathbb{S}_j, Q, U)$, there exists a negligible function $\nu(.)$ such that, the probability of winning the following game is negligible, where the probability is taken over the randomness of the algorithms and the coins of* Adv*:*

Setup Adv *receives* vk *where* $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda)$. *Given* vk, Adv *picks* $(\mathbb{S}_0, Q, U)$ *and receives* $\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0$ *for* $\mathbb{S}_0$*, where* $(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{vk},\mathbb{S}_0)$.

Query Adv *requests a series of updates $u_1, u_2, \ldots, u_L$, where $L = \mathsf{poly}(\mathsf{k})$, of its choice. For every update request* Adv *receives an update string. Let $\mathbb{S}_{i+1}$ denote the state of the set collection after the i^{th} update u_i and $\mathsf{updinfo}_i$ be the update string corresponding to u_i received by the adversary, i.e., $(\mathbb{S}_{i+1}, \mathsf{auth}(\mathbb{S}_{i+1}), \mathsf{digest}_{i+1}, \mathsf{aux}_{i+1}, \mathsf{ek}_{i+1}, \mathsf{updinfo}_i) \leftarrow \mathsf{Update}(\mathsf{sk}, \mathbb{S}_i, \mathsf{auth}(\mathbb{S}_i), \mathsf{digest}_i, \mathsf{aux}_i, \mathsf{ek}_i, \mathsf{SID}_i, u_i)$ where $\mathsf{SID}_i = \perp$.*

Response *Finally,* Adv *outputs $(\mathbb{S}_j, q, \mathsf{answer}, \mathsf{proof})$, $0 \le j \le L$, and wins the game if the following holds:*

$$\mathsf{answer} \neq Q(\mathbb{S}_j, q) \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{digest}_j, \mathsf{answer}, \mathsf{proof}) = \mathsf{accept}.$$

Zero-knowledge captures privacy guarantees about the set collection against a malicious client. Recall that the client receives a proof for every query answer. Periodically he also receives an updated digest, due to the owner making changes to the set collection. Informally, (1) the proofs should reveal nothing beyond the query answer, and (2) an updated digest should reveal nothing about update operations performed on the set

collection. This security property guarantees that the client does not learn which elements were updated, unless he queries for an updated element (deleted or replaced), before and after the update.

The definition of the zero-knowledge property captures the adversarial client's (Adv) view in two games. Let $\mathcal{E}$ be a ZKASC scheme. In the $\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda)$ game (λ is the security parameter), Adv interacts with the honest owner and the honest server (jointly called challenger), whereas in the $\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$ game, it interacts with a simulator, who mimics the behavior of the challenger with oracle access to the source set collection, i.e., it is allowed to query the set collection only with client's queries and does not know anything else about the set collection (except its cardinality, which is a public information) or the updates.

Adv picks $\mathbb{S}_0$ and adaptively asks for queries and updates on it. Its goal is to determine if it is talking to the real challenger or to the simulator, with non-negligible advantage over a random guess.

We note that here SID need not be used explicitly in the definition, since the challenger and the simulator know all the queries and can compute f themselves.

Let D be a binary function for checking the validity of queries and updates on a set collection. For queries, $D(\mathbb{S}_t, q) = 1$ iff the query indices in q are valid set indices of $\mathbb{S}_t$. For updates, $D(\mathbb{S}_t, u = (x, \mathsf{upd}, i)) = 1$ iff i is a valid set index of $\mathbb{S}_t$ and u is a valid update on $\mathcal{X}_i \in \mathbb{S}_t$.

Definition 12 (Zero-Knowledge)**.** *Let $\mathcal{E}$ be a* ZKASC *scheme.* $\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}$ *and* $\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}$ *be defined as follows where, the simulator always checks the validity of an update or query using oracle access to the function $D()$ as defined above.*

$\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda)$**:**

Setup *The challenger runs* $\mathsf{Gen}(1^\lambda)$ *to generate* sk, vk *and sends* vk *to* Adv_1*. Given* vk*,* Adv_1 *picks* $(\mathbb{S}_0, Q, U)$ *of its choice and receives* digest^0 *corresponding to* $\mathbb{S}_0$ *from the real challenger $\mathcal{C}$ who runs* $\mathsf{Setup}(\mathsf{sk}, \mathsf{vk}, \mathbb{S}_0)$ *to generate it.* Adv_1 *saves its state information in* st_A*.*

Query Adv_2 *has access to* st_A *and requests a series of queries* $\{\mathsf{op}_1, \mathsf{op}_2, \ldots, \mathsf{op}_M\}$*, for* $M = \mathsf{poly}(\mathsf{k})$*.*

If $\mathsf{op} = u$ **is an update request:** *$\mathcal{C}$ runs function $D()$ to check if the update request is valid. If not it returns $\perp$. Else, $\mathcal{C}$ runs* Update *algorithm. Let $\mathbb{S}_t$ be the most recent set collection and* digest_t *be the public digest on it generated by the* Update *algorithm.*
$\mathcal{C}$ returns digest_t *to* Adv_2*.*

If $\mathsf{op} = q_i$ **is a query:** *$\mathcal{C}$ runs function $D()$ to check if the query is valid. If not it returns $\perp$. Else, $\mathcal{C}$ runs* Query *algorithm for the query with the most recent set collection and the corresponding digest as its parameter.*
$\mathcal{C}$ returns answer *and* proof *to* Adv_2*.*

Response Adv_2 *outputs a bit b.*

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

Setup *Initially* Sim_1 *generates a public key, i.e.,* $(\mathsf{vk}, \mathsf{st}_S) \leftarrow \mathsf{Sim}_1(1^\lambda)$ *and sends it to* Adv_1*. Given* vk*,* Adv_1 *picks* $(\mathbb{S}_0, Q, U)$ *of its choice and receives* digest_0 *from the simulator* Sim_1*, i.e.,* $(\mathsf{digest}_0, \mathsf{st}_S) \leftarrow \mathsf{Sim}_1(\mathsf{st}_S)$*.* Adv_1 *saves its state information in* st_A*.*

Query Adv_2*, who has access to* st_A*, requests a series of queries* $\{\mathsf{op}_1, \mathsf{op}_2, \ldots, \mathsf{op}_M\}$*, for* $M = \mathsf{poly}(\mathsf{k})$*.* Sim_2 *is given oracle access to the the most recent set collection and is allowed to query the set*

collection oracle only for queries that are queried by Adv. *Let* $\mathbb{S}_{t-1}$ *denote the state of the data structure at the time of* op. *The simulator runs* $(\mathsf{st}_S, a) \leftarrow \mathsf{Sim}_2^{D,\mathbb{S}_{t-1}}(1^\lambda, \mathsf{st}_S, D(\mathbb{S}_{t-1}, \mathsf{op}_i))$ *and returns answer a to* Adv_2 *where:*

If $D(\mathbb{S}_{t-1}, \mathsf{op}_i) = 1$ **and** op_i **is an update request:** $a = \mathsf{digest}_t$, *the updated digest.*

If $D(\mathbb{S}_{t-1}, \mathsf{op}_i) = 1$ **and** op_i **is a query:** $a = (\mathsf{answer}, \mathsf{proof})$ *corresponding to the query* q_i.

Response Adv_2 *outputs a bit b.*

A ZKASC $\mathcal{E}$ *is zero-knowledge if there exists a PPT algorithm* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *s.t. for all malicious stateful adversaries* $\mathsf{Adv} = (\mathsf{Adv}_1, \mathsf{Adv}_2)$ *there exists a negligible function* $\nu(.)$ *s.t.*

$$|\Pr[\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}(1^\lambda) = 1]| \leq \nu(\lambda).$$

3.7 Zero-Knowledge Authenticated Set Collection Construction

In this section, we give an efficient construction for ZKASC. Our construction relies on two basic primitives: *zero-knowledge dynamic universal accumulator* introduced in Section 3.3 and *accumulation tree* described in Section 3.2. The sets collection data structure consists of m sets, denoted with $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, each containing elements from a universe $\mathbb{X}$[10]. We use our zero-knowledge accumulator construction from Section 3.5 to represent every set in the set collection succinctly using $\mathsf{acc}(\mathcal{X}_i)$. Our construction, similar to [131], relies on accumulation trees to verify that $\mathsf{acc}(\mathcal{X}_i)$ is indeed a representation of the i^{th} set of the current set collection. We note that accumulation tree presents an optimal choice when considering several parameters, including digest size, proof size, verification time, setup and update cost, and storage size.

Given the above representation of a set collection, we develop techniques for efficiently and authentically proving zero-knowledge set algebraic queries and performing updates on $\mathbb{S}$. We note that the authors of [131] also consider authenticated set algebra. Unfortunately, since privacy was not an objective of [131], their techniques for authentic set queries cannot be trivially extended to support our strong privacy notion (zero-knowledge).

We organize the description of our construction as follows. The setup and maintenance of the dynamic set collection $\mathbb{S}$ is given in the next section. These phases instantiate and update zero-knowledge accumulators and authentication tree used to store and update the sets in the collection. We then develop algorithms for verifiable zero-knowledge query and verification algorithms. The query and verify algorithms invoke a subroutine for each individual operation, namely, is-subset, intersection, union and set difference. We describe two algorithms for each set operation: an algorithm executed by the server to construct a proof of an answer on $\mathbb{S}$ and a verification algorithm executed by the client to verify the answer and the proof. For each algorithm we analyze its efficiency as we go. We then compare the asymptotic performance of the ZKASC algorithms with that of [131], which offers no privacy guarantee, in Figure 3.4. Finally, we prove that the construction described in this section meets the security properties of ZKASC.

[10]Without loss of generality we assume that our universe $\mathbb{X}$ is the set of nonnegative integers in the interval $[m+1, p-1]$ as in [131].

3.7.1 Setup and Update Algorithms

Here we describe and give pseudocode for the algorithms GenKey, Setup, Update, and UpdateServer. We recall that the first two algorithms are used by the owner to prepare keys, collection $\mathbb{S}$ and relevant data structures before outsourcing them to the untrusted server. The owner runs Update to make efficient updates to $\mathbb{S}$, while the server executes UpdateServer to propagate these updates. Each algorithm is described below.

The GenKey algorithm (Algorithm 1) takes the security parameter as input and generates the secret key sk and the corresponding verification key vk for the scheme. As part of the algorithm, it also initializes our zero-knowledge accumulator from Section 3.3. For simplicity, we refer to it as DUA.

The Setup algorithm (Algorithm 2) takes the secret key, the verification key and the set collection as input and generates the authentication information for the set collection, a short public digest, the evaluation key and some auxiliary information for the set collection. Among the output objects, only the digest is public and is accessible by the client. The rest of the output are only sent to the server. The server uses them to generate proofs for client queries.

Algorithm 1 $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{GenKey}(1^\lambda)$

1: Run $\mathsf{GenParams}(1^\lambda)$ to receive bilinear parameters $pub = (p,\mathbb{G},\mathbb{G}_T,e,g)$.
2: Pick a cryptographic hash function $\mathcal{H}$ which will be viewed as a Random Oracle and append pub with $\mathcal{H}$.
   $\%\mathcal{H}$ **will be viewed as a Random Oracle for set difference only.**
3: Invoke $\mathsf{DUA.GenKey}(1^\lambda)$. Let $\mathsf{DUA.GenKey}(1^\lambda) = (s, g^s)$.
4: **return** $(s,(g^s,pub))$.

Algorithm 2 $(\mathbb{S}_0,\mathsf{auth}(\mathbb{S}_0),\mathsf{digest}_0,\mathsf{ek}_0,\mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{vk},\mathbb{S}_0)$

1: Let $\mathbb{S}_0 = \{\mathcal{X}_1,\ldots,\mathcal{X}_m\}$. Invoke $\mathsf{DUA.Setup}(s,\mathcal{X}_i)$ for each $\mathcal{X}_i \in \mathbb{S}_0$. Let $\mathsf{DUA.Setup}(s,\mathcal{X}_i) = (\mathsf{acc}(\mathcal{X}_i),\mathsf{ek}_i,\mathsf{aux}_i)$.
2: Let $|\mathcal{X}_i| = n_i$. Set $\mathsf{ek}_0 := (g, g^s, \ldots, g^{s^N})$ where $N = \sum_{i\in[1,m]} n_i$
3: Parse aux_i corresponding to $\mathcal{X}_i$ as (r_i, n_i) for all $i \in [1,m]$.
4: Set $\mathsf{aux}_0 = ((r_i \mid \mathcal{X}_i \in \mathbb{S}_0), N)$.
5: Invoke $\mathsf{AT.ATSetup}(\mathsf{sk},(\mathsf{acc}(\mathcal{X}_1),\ldots,\mathsf{acc}(\mathcal{X}_m)))$ and let $\mathsf{AT.ATSetup}(\mathsf{sk},(\mathsf{acc}(\mathcal{X}_1),\ldots,\mathsf{acc}(\mathcal{X}_m))) = (\mathsf{auth}(\mathbb{S}_0),\mathsf{digest}_0)$.
6: **return** $(\mathbb{S}_0,\mathsf{auth}(\mathbb{S}_0),\mathsf{digest}_0,\mathsf{ek}_0,\mathsf{aux}_0)$

Update (Algorithm 3) algorithm takes as input an update string $u_t = (x,\mathsf{upd},i)$ where x denotes the element, upd, as in DUA, is a boolean indication if an insert or delete is to be preformed and i denotes the index of the set in which the update is to be performed. This algorithm updates the corresponding set in the set collection and accordingly updates the authentication information, the auxiliary information and the public digest.

As described so far, the accumulation value of the set updated due to u_t is regenerated with fresh randomness (in DUA.Update), which causes a subsequent change in the digest for $\mathbb{S}$. However, this is not enough to achieve zero-knowledge. If a client queries wrt some set $j \neq i$ before and after u_t was performed, and sees that $\mathsf{acc}(\mathcal{X}_j)$ has not changed, then he learns that $\mathcal{X}_j$ is not affected by the update. This will also imply that

the proofs that the client holds wrt $\mathcal{X}_j$ between updates are still valid, i.e, they are not ephemeral.

Zero-knowledge property implies that even an adversarial client does not learn anything beyond the fact that an update has occurred (unless his query explicitly returns some updated element) and none of the proofs that a client holds should not be valid after an update. We set f to be a function that takes the client queries since the last update and returns the indices of the sets accessed by them; therefore SID_t has the indices of the sets touched by queries since the $(t-1)^{th}$ update u_{t-1} (we consider setup as the 0^{th} update). To achieve zero-knowledge property, Update needs to use the subroutine Refresh, which picks fresh randomness to refresh accumulation values of all the set indices in SID_t. Update algorithm also produces a string $\mathsf{updinfo}_t$ that the server can use to update its authentication and auxiliary information. Note that, SID_t used for efficiency. The non-efficient way of achieving zero-knowledge would be to refresh accumulation values of *all* the sets in the collection.

Algorithm 3 $(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t) \leftarrow \mathsf{Update}(\mathsf{sk}, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, \mathsf{ek}_t, \mathsf{SID}_t, u_t)$ where $u_t = (x, \mathsf{upd}, i)$

1: Initialize set $W := \perp$.
2: $u_t = (x, \mathsf{upd}, i)$ to be done on set $\mathcal{X}_i$.
3: If $(\mathsf{upd} = 1 \wedge x \in \mathcal{X}_1)$ or $(\mathsf{upd} = 0 \wedge x \notin \mathcal{X}_i)$ **return** $\perp$. Else proceed to next step.
4: Parse aux_t as $(\{r_j \mid \mathcal{X}_j \in \mathbb{S}_t\}, N)$ and set $\mathsf{aux}_j = (r_j, N)$ for all $j \in [1, m]$.
5: Call $\mathsf{DUA.Update}(\mathsf{acc}(\mathcal{X}_i), \mathcal{X}_i, x, s, \mathsf{aux}_i, \mathsf{upd})$. Let $\mathsf{DUA.Update}(\mathsf{acc}(\mathcal{X}_i), \mathcal{X}_i, x, s, \mathsf{aux}_i, \mathsf{upd}) = (\mathsf{acc}'(\mathcal{X}_i), \mathsf{ek}'_i, \mathsf{aux}'_i)$
6: Replace old $\mathcal{X}_i$ with the updated $\mathcal{X}_i$ to obtain $\mathbb{S}_{t+1}$.
7: Let $M = \sum_{\mathcal{X}_j \in \mathbb{S}_{t+1}} |\mathcal{X}_j|$. Set $N := M$.
8: **If** $M > N$ do the following:
9: Set $N := M$.
10: Update N in aux'_i.
11: Set $\mathsf{ek}_{t+1} := \mathsf{ek}_t \cup g^{s^N}$ and $\mathsf{ekupdate} := g^{s^N}$.
12: **Else**
13: Set $\mathsf{ek}_{t+1} := \mathsf{ek}_t$ and $\mathsf{ekupdate} := \phi$
14: Update $W \leftarrow W \cup i$.
15: Call $\mathsf{Refresh}(s, \mathsf{SID}_t, (\forall j \in \mathsf{SID}_t : (\mathsf{acc}(\mathcal{X}_j), \mathsf{aux}_j))$. Let $\mathsf{Refresh}(s, \mathsf{SID}_t, (\forall j \in \mathsf{SID}_t : (\mathsf{acc}(\mathcal{X}_j), \mathsf{aux}_j)) = (\forall j \in \mathsf{SID}_t : (\mathsf{acc}'(\mathcal{X}_j), \mathsf{aux}'_j))$, where $\mathsf{aux}_j = (r_j, N)$
16: Update $W \leftarrow W \cup j$ for all $j \in \mathsf{SID}_t$.
17: Set $\mathsf{aux}_{t+1} := ((r'_j \cdot r_j \mid j \in W), N)$
18: Call ATUpdateBatch to update the accumulation tree for all leaves corresponding to $\mathcal{X}_j$ for $j \in W$.
19: Let auth' be the updated authentication information, digest' be the updated root and a list of $\mathsf{updinfo}_j$ containing the updated authentication paths for each set $\mathcal{X}_j$.
20: Set $\mathsf{auth}(\mathbb{S}_{t+1}) := \mathsf{auth}'$ and $\mathsf{digest}_{t+1} := \mathsf{digest}'$
21: Set $\mathsf{updinfo}_t := (W, (j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{aux}'_j, \mathsf{updinfo}_j) \mid j \in W, \mathsf{ekupdate})$
22: **return** $(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t)$.

Next we describe UpdateServer algorithm which is run by the server to propagate the update on the set collection and its authentication information using $\mathsf{updinfo}_t$ generated by the owner corresponding to the t^{th} update. Informally, this algorithm updates the relevant set and updates all the authentication paths corresponding to the sets whose accumulation value has been changed or refreshed by the owner. The algorithm also updates its evaluation key and the auxiliary information.

Algorithm 4 $(\forall i \in \mathsf{SID}_t : (\mathsf{acc}'(\mathcal{X}_i), \mathsf{aux}'_i)) \leftarrow \mathsf{Refresh}(\mathsf{sk}, \mathsf{SID}_t, (\forall i \in \mathsf{SID}_t : (\mathsf{acc}(\mathcal{X}_i), \mathsf{aux}_i))$

For every $i \in \mathsf{SID}_t$:
  Pick $r'_i \leftarrow \mathbb{Z}_p^*$.
  Compute $\mathsf{acc}'(\mathcal{X}_i) \leftarrow \mathsf{acc}(\mathcal{X}_i)^{r_i}$.
  Parse aux_i as (r_i, N) and compute $\mathsf{aux}'_i \leftarrow (r_i \cdot r'_i, N)$.
return $(\forall i \in \mathsf{SID}_t : (\mathsf{acc}'(\mathcal{X}_i), \mathsf{aux}'_i))$

Algorithm 5 $(\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{ek}_t, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, u_t, \mathsf{updinfo}_t)$ where $u_t = (x, \mathsf{upd}, i)$

Update $\mathcal{X}_i$ with u_t.
Replace old $\mathcal{X}_i$ in $\mathbb{S}_t$ with the updated $\mathcal{X}_i$ to obtain $\mathbb{S}_{t+1}$.
Parse $\mathsf{updinfo}_t$ as $(W, (j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{aux}'_j, \mathsf{updinfo}_j) \mid j \in W, \mathsf{ekupdate})$.
For all $j \in W$ do the following:
    Parse aux'_j as $(\tilde{r}_j, N)$.
    Replace r_j with $\tilde{r}_j$ and update N in aux_t to generate aux_{t+1}.
    Using $\mathsf{updinfo}_j$ update the authentication path of $\mathsf{acc}(\mathcal{X}_j)$ in the accumulation tree.
    Update the authentication path of $\mathsf{acc}(\mathcal{X}_j)$ in $\mathsf{auth}(\mathbb{S}_t)$ to obtain $\mathsf{auth}(\mathbb{S}_{t+1})$
Set the updated root of the accumulation tree as digest_{t+1}
If $\mathsf{ekupdate} = \phi$, then set $\mathsf{ek}_{t+1} := \mathsf{ek}_t$. **Else** set $\mathsf{ek}_{t+1} := \mathsf{ek}_t \cup \mathsf{ekupdate}$.
return $(\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1})$.

Efficiency We analyze the access complexity of each algorithm. Let $M = \sum_{i \in m} |\mathcal{X}_i|$ where $\mathbb{S} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ and let $L = |\mathsf{SID}_t|$ for update u_t.

GenKey**:** The complexity of this algorithm is $O(poly(\lambda))$.

Setup**:** For the Setup algorithm, DUA.Setup is called for all m sets. Therefore, the complexity of this step is $O(M)$. Setting up the AT has complexity $O(m)$. Therefore, the total complexity of Setup is $O(M+m)$.

Update**:** This algorithm takes constant time for the update and constant time for refreshing each accumulator value for the sets whose indices are in SID_t. From Lemma 4, updating each authentication path has complexity $O(1)$. Hence the total complexity of this algorithm is $O(L)$.

UpdateServer**:** This algorithm has access complexity similar to Update, i.e., $O(L)$.

3.7.2 Set Algebra Query and Verify Algorithms

We proceed with verifiable zero-knowledge set algebra operations that use data structures described in the previous section. Query and Verify algorithms let the server construct a proof of a response to a query and the client verify it, respectively. Since ZKASC supports several set operations, we describe each algorithm in terms of modular subroutines as follows. Algorithm Query takes the query arguments along with an indicator flag indicator that denotes the type of the input query (is-subset, intersection, union, difference) and invokes the appropriate subroutine that we describe subsequently. In particular, we annotate query q with indicator. If indicator $= S$, Query invokes subsetQ (Section 3.7.3), if indicator $= I$, Query invokes intersectionQ (Section 3.7.4), if indicator $= U$, Query invokes unionQ (Section 3.7.5) and if indicator $= D$, Query invokes differenceQ (Section 3.7.6). Similarly, we annotate the input argument of Verify with the flag indicating which verification algorithm should be used.

Query and Verify algorithms for each set algebra operation are given in the next four sections along with the analysis of their efficiency.

3.7.3 Subset Query

A subset query is parametrized by a set of elements Δ and an index i of a set collection, i.e., $q = (\Delta, i)$. Given q, the subset query returns a boolean as the answer. In particular, $\mathsf{answer} = 1$ if $\Delta \subseteq \mathcal{X}_i$ and $\mathsf{answer} = 0$ if $\Delta \nsubseteq \mathcal{X}_i$. This query can be seen as a generalization of Query algorithm for DUA where membership/non-membership query is supported for a batch of elements instead of a single element. The proof technique is similar to the membership and non-membership proof generation for a single element using Query algorithm. We give the pseudocode for subset query in Algorithm 6 and verification Algorithm 7.

Algorithm 6 *Subset*: $(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{SubsetQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, q = (\Delta, i))$

Let $\Delta = \{x_1, \ldots, x_k\}$ and $[a_0, \ldots, a_k]$ be the coefficients of the polynomial $\mathsf{Ch}_\Delta[z]$.
If $\Delta \subseteq \mathcal{X}_i$**:**
    Set $\mathsf{answer} = 1$
    Compute $W_{\Delta,\mathcal{X}_i} \leftarrow g^{r_i \mathsf{Ch}_{\mathcal{X}_i - \Delta}(s)}$.
Else when $\Delta \nsubseteq \mathcal{X}_i$**:**
    Set $\mathsf{answer} = 0$
    Using Extended Euclidian Algorithm, compute polynomials $q_1[z]$ and $q_2[z]$ such that $q_1[z]\mathsf{Ch}_\Delta[z] + q_2[z]\mathsf{Ch}_{\mathcal{X}_i}[z] = 1$.
    Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q_1'[z] = q_1[z] + \gamma \mathsf{Ch}_{\mathcal{X}_i}[z]$ and $q_2'[z] = (r_i^{-1})(q_2[z] - \gamma \mathsf{Ch}_\Delta[z])$.
    Compute $F_1 \leftarrow g^{q_1'(s)}$ and $F_2 \leftarrow g^{q_2'(s)}$.
    Set $W_{\Delta,\mathcal{X}_i} := (F_1, F_2)$.
Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, i, \mathsf{auth}(\mathbb{S}_t))$. Let $(\Pi_i, \alpha_i) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, i, \mathsf{auth}(\mathbb{S}_t))$.
Set $\mathsf{proof} = ([a_0, \ldots, a_k], \{g^s, \ldots, g^{s^k}\}, W_{\Delta,\mathcal{X}_i}, \Pi_i, \alpha_i)$.
return $(\mathsf{answer}, \mathsf{proof})$

Algorithm 7 *Subset Verification*: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{SubsetV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

Parse proof as $([a_0, \ldots, a_k], \{g^s, \ldots, g^{s^k}\}, W_{\Delta,\mathcal{X}_i}, \Pi_i, \alpha_i)$.
Certify the validity of $[a_0, \ldots, a_k]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i \in [0,\Delta]} a_i w^i = \prod_{i \in [0,\Delta]} (x_i + w)$ where $\Delta = \{x_1, \ldots, x_k\}$. If the verification fails, **return** reject. Else proceed to next step.
Invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, i, \Pi_i, \alpha_i)$ If verification fails **return** reject. Else proceed to next step.
If $\mathsf{answer} = 1$**:**
    Check if $e(g^{\mathsf{Ch}_\Delta(s)}, W_{(\Delta,\mathcal{X}_i)}) = e(\mathsf{acc}(\mathcal{X}_i), g)$. If verification fails **return** reject. Else **return** accept.
Else if $\mathsf{answer} = 0$**:**
    Parse $W_{(\Delta,\mathcal{X}_i)}$ as (F_1, F_2)
    Check if $e(F_1, g^{\mathsf{Ch}_\Delta(s)}) e(F_2, \mathsf{acc}(\mathcal{X}_i)) = e(g, g)$ If verification fails **return** reject. Else **return** accept.

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below. Note that $|\Delta| = k$ in Algorithm 6 and 7.

Query: Let us analyze the complexity of each of the steps. Step 1 has complexity $O(k\log k)$ by Lemma 1. Step 4 has complexity $O((|\mathcal{X}_i|-k)\log(|\mathcal{X}_i|-k))$ by the same lemma. In case of non-membership, in Step 7, by Lemma 3, this step has complexity $O(N\log^2 N\log\log N)$ where $N=|\mathcal{X}_i|+k$. The complexity of Step 9 is $O(N\log N)$. The complexity of Step 11 is $O(m^{\varepsilon}\log m)$ by Lemma 4, where $m=|\mathbb{S}|$. Hence the overall complexity of the query algorithm is $O(k\log k+(|\mathcal{X}_i|-k)\log(|\mathcal{X}_i|-k)+N\log^2 N\log\log N+N\log N+m^{\varepsilon}\log m)=O(N\log^2 N\log\log N+m^{\varepsilon}\log m)$.

Verification: Verification in Step 2 takes time $O(k)$ by Lemma 2. By Lemma 4, Step 3 has complexity $O(1)$. Computing $g^{\mathsf{Ch}_{\Delta}(s)}$ takes time $O(k)$. Therefore Step 5 has complexity $O(k)$. Finally Step 9 involves requires 5 bilinear map computations and one multiplication and hence has complexity $O(1)$. Therefore, the overall complexity of verification is $O(k)$.

Proof size: The proof size is $O(k)$.

3.7.4 Set Intersection Query

Set intersection query q is parameterized by a set of indices of the set collection, $q=(i_1,\ldots,i_k)$. The answer to an intersection query is a set of elements which we denote as answer and a simulatable proof of the correctness of the answer. If the intersection is computed correctly then $\mathsf{answer}=\mathcal{X}_{i_1}\cap\mathcal{X}_{i_2}\cap\ldots\cap\mathcal{X}_{i_k}$. We express the correctness of intersection with the two following conditions as in [131]:

Subset condition: $\mathsf{answer}\subseteq\mathcal{X}_{i_1}\wedge\mathsf{answer}\subseteq\mathcal{X}_{i_2}\wedge\ldots\wedge\mathsf{answer}\subseteq\mathcal{X}_{i_k}$. This condition ensures that the returned answer is a subset of all the queried set indices, i.e., every element of answer belongs to each of the sets $\mathcal{X}_j$ for $j\in[i_1,i_k]$;

Completeness condition: $(\mathcal{X}_{i_1}-\mathsf{answer})\cap(\mathcal{X}_{i_2}-\mathsf{answer})\cap\ldots\cap(\mathcal{X}_{i_k}-\mathsf{answer})=\emptyset$. This is the completeness condition, that ensures that answer indeed contains *all* the common elements of $\mathcal{X}_{i_1},\ldots,\mathcal{X}_{i_k}$, i.e., none of the elements have been omitted from answer.

To prove the first condition, we will use subset query as a subroutine. Proving the second condition is more tricky; it relies on the fact that the characteristic polynomials for the sets $\mathcal{X}_j-\mathsf{answer}$, for all $j\in[i_1,i_k]$, do not have common factors. In other words, these polynomials should be co-prime and their GCD should be 1 (Lemma 3). Since the proof units should be simulatable, we cannot directly use the technique as in [131]. To this end, we randomize the proof units by generalizing the randomization technique in Section 3.5 used to prove non-membership in a single set. We present the pseudocode for the set intersection query in Algorithm 8 and its verification in Algorithm 9.

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below.

Query: Let us analyze the complexity of each of the steps. Let $m=|\mathbb{S}|$, $N=\sum_{j\in[i_1,i_k]}n_j$, $n_j=|\mathcal{X}_j|$ and k is the number of indices in the query. Note that N is an upper bound on ρ. Let us denote $\rho=\rho$ Step 3 has complexity $O(\rho\log\rho)$ by Lemma 1. By Lemma 3, Step 5 has complexity $O(N\log^2 N\log\log N)$. Step 7 has complexity $O(N\log N)$ (by Lemma 1). Finally, Step 9 has complexity $O(km^{\varepsilon}\log m)$ by Lemma 4. Therefore, the overall complexity is $O(\rho\log\rho+N\log^2 N\log\log N+N\log N+km^{\varepsilon}\log m)=O(N\log^2 N\log\log N+km^{\varepsilon}logm)$

Algorithm 8 *Set Intersection*: $(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{IntersectionQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, q = (i_1, \ldots, i_k))$

1: Let $\mathsf{answer} = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2} \cap \ldots \cap \mathcal{X}_{i_k}$ and WLOG let us assume k is even.
2: Let $\mathsf{Ch}_{\mathsf{answer}}$ be the charactersitic polynomial for answer.
3: Let $[a_\delta, \ldots, a_0]$ be the coefficients of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
4: Let $P_j[z] = \mathsf{Ch}_{\mathcal{X}_j \setminus \mathsf{answer}}$ for all $j \in [i_1, i_k]$ where r_i is the random number used as the blinding factor for $\mathcal{X}_j$.
5: Using Extended Euclidian Algorithm, compute polynomials $q_j[z]$ such that $\sum_{j \in [i_1, i_k]} q_j[z] P_j[z] = 1$.
6: Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q'_{i_1}[z] := (r_{i_1}^{-1})(q_{i_1}[z] + \gamma P_{i_2}[z])$ and $q'_{i_2}[z] := q_{i_2}[z] - \gamma P_{i_1}[z]$, $q'_{i_3}[z] := (r_{i_3}^{-1})(q_{i_3}[z] + \gamma P_{i_4}[z])$ and $q'_{i_4}[z] := q_{i_4}[z] - \gamma P_{i_3}[z], \ldots, q'_{i_{k-1}}[z] := (r_{i_{k-1}}^{-1})(q_{i_{k-1}}[z] + \gamma P_{i_k}[z])$ and $q'_{i_k}[z] := q_{i_k}[z] - \gamma P_{i_{k-1}}[z]$.
7: Compute $F_j = g^{q'_j(s)}$ for all $j \in [i_1, i_k]$.
8: Invoke Subset query with $\mathsf{subsetQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, \delta, j)$ for all $j \in [i_1, i_k]$. Let $\mathsf{subsetQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, \delta, j) = (1, W_{\mathsf{answer}, \mathcal{X}_j})$
9: Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $(\Pi_j, \alpha_j) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$.
10: Set $\mathsf{proof} = ([a_\delta, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{F_j, W_{(\mathsf{answer}, \mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$, where $|\mathsf{answer}| = \rho$.
11: **return** $(\mathsf{answer}, \mathsf{proof})$

Algorithm 9 *Set Intersection Verification*: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{IntersectionV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

1: Parse proof as $([a_\delta, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{F_j, W_{(\mathsf{answer}, \mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$ where $|\mathsf{answer}| = \rho$.
2: Certify the validity of $[a_\delta, \ldots, a_0]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i \in [0, \delta]} a_i w^i = \prod_{i \in [0, \delta]} (x_i + w)$ where $\mathsf{answer} = \{x_0, \ldots, x_\delta\}$. If the verification fails, **return** reject. Else proceed to next step.
3: For all $j \in [i_1, i_k]$ invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, j, \Pi_j, \alpha_j)$ If verification fails for at least one α_j, **return** reject. Else proceed to next step.
4: For each $j \in [i_1, i_k]$ verify that: $e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, W_{(\mathsf{answer}, \mathcal{X}_j)}) = e(\mathsf{acc}(\mathsf{st}_j), g)$. If verification fails for at least one α_j, **return** reject. Else proceed to next step.
5: Check the completeness condition by checking if $\prod_{j \in [i_1, i_k]} e(F_j, W_{(\mathsf{answer}, \mathcal{X}_j)}) = e(g, g)$. If verification fails **return** reject. Else
6: **return** accept

Verification: For the verification algorithm, let us look at each of the verification steps individually. Step 2 has complexity $O(\rho)$ by Lemma 2. Step 3takes has complexity $O(k)$ by Lemma 4. In Step 4, computing $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ has complexity $O(\rho)$ and then there are $2k$ bilinear map computations. Therefor this step has complexity $O(\rho + k)$. Finally, Step 4 requires $k+1$ bilinear map computations, $k-1$ products and hence has complexity $O(k)$. Therefore, the overall complexity of verification is $O(\rho + k)$.

Proof size: The proof size is $O(\rho + k)$.

3.7.5 Set Union

Set union query, like intersection, is parameterized by a set of indices of the set collection, $q = (i_1, \ldots i_k)$. The answer to an union query contains a set of elements, denoted as $\mathsf{answer} = \mathcal{X}_{i_1} \cup \mathcal{X}_{i_2} \cup \ldots \cup \mathcal{X}_{i_k}$, and a simulatable proof of the correctness of the answer.

We introduce a technique for checking correctness of union operation based on the following conditions:

Superset condition: $\mathcal{X}_{i_1} \subseteq \mathsf{answer} \wedge \mathcal{X}_{i_2} \subseteq \mathsf{answer} \wedge \ldots \wedge \mathcal{X}_{i_k} \subseteq \mathsf{answer}$. This condition ensures that no element has been excluded from the returned answer.

Membership condition: $\mathsf{answer} \subseteq \tilde{U}$ where $\tilde{U} = \mathcal{X}_{i_1} \uplus \mathcal{X}_{i_2} \uplus \ldots \uplus \mathcal{X}_{i_k}$. $\uplus$ denotes multiset union of the queries sets, i.e., $\uplus$ preserves the multiplicity of every element in the union. This condition ensures that every element of answer belongs to *at least* one of the sets $\mathcal{X}_{i_1}, \ldots, \mathcal{X}_{i_k}$.

The query algorithm generates the query answer along with proofs for both the conditions mentioned here. The first condition can be checked by using the subset proof developed in Section 3.7.3. The second condition should be proved carefully and not reveal (1) whether an element belongs to more than one of the sets in the query, and (2) which set an element in the union comes from. For example, returning $\tilde{U}$ in the clear trivially reveals the multiplicity of every element in answer. Instead, the server returns $\mathsf{acc}(\tilde{U})$ which equals $g^{\mathsf{Ch}_{\tilde{U}}(s)}$ blinded with randomness in the exponent. In order to prove that the server computed $\mathsf{acc}(\tilde{U})$ correctly, we introduce a *union tree*.

A union tree (UT) is a binary tree computed as follows. Corresponding to the k queried indices, $\mathsf{acc}(\mathcal{X}_{i_1}), \ldots, \mathsf{acc}(\mathcal{X}_{i_k})$ are the leaves of UT. The leaves are computed bottom up. Every internal node is computed as follows: For each internal node v, let v_1 and v_2 be its two children. The (multi)set associated with v is the multiset $M = M_1 \uplus M_2$ where M_1 and M_2 are (multi)sets for v_1 and v_2 respectively. Let r_1 and r_2 be the corresponding blinding factors. Then the node v stores value $g^{r_1 r_2 \mathsf{Ch}_M(s)}$. Finally, the server constructs proof of subset for answer in $\tilde{U}$. We give the pseudocode for the set union query in Algorithm 10 and its verification in Algorithm 11.

Algorithm 10 *Set Union*: $(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{unionQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, (i_1, \ldots, i_k))$

1: Let $\mathsf{answer} = \mathcal{X}_{i_1} \cup \mathcal{X}_{i_2} \cup \ldots \cup \mathcal{X}_{i_k}$.
2: Let $[a_\delta, \ldots, a_0]$ be the coefficients of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
3: Let $\tilde{U} = \mathcal{X}_1 \uplus \mathcal{X}_2 \uplus \ldots \uplus \mathcal{X}_k$ where $\uplus$ denotes multiset union, i.e., $\uplus$ preserves the multiplicity of every element in the union.
4: For each index $j \in [i_1, i_k]$ compute $W_{(\mathcal{X}_j, \mathsf{answer})} \leftarrow g^{\frac{\mathsf{Ch}_{\mathsf{answer}}(s)}{r_j \mathsf{Ch}_{\mathcal{X}_j}(s)}}$.
5: Build a binary tree on k leaves (representing the k indices in the query) bottom-up as follows: For a leaf node, let $a(v) = \mathsf{acc}(\mathcal{X}_j) = g^{r_j \mathsf{Ch}_{\mathcal{X}_j}(s)}$. For each internal node v, let v_1 and v_2 be its two children. The (multi)set associated with v is the multiset $M = M_1 \uplus M_2$ where M_1 and M_2 are (multi)sets for v_1 and v_2 respectively. Let r_1 and r_2 be the corresponding blinding factors. Then $a(v) = g^{r_1 r_2 \mathsf{Ch}_M(s)}$.
6: Compute $W_{(\mathsf{answer}, \tilde{U})} \leftarrow g^{\frac{(\prod_{j \in [i_1, i_k]} r_j)\mathsf{Ch}_{\tilde{U}}(s)}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$
7: Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $(\Pi_j, \alpha_j) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$.
8: $\mathsf{proof} := ([a_\delta, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{a(v)\}_{v \in V(\mathsf{UT})}, W_{(\mathsf{answer}, \tilde{U})}, \{W_{(\mathcal{X}_j, \mathsf{answer})}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$, where $|\mathsf{answer}| = \rho$.
9: **return** $(\mathsf{answer}, \mathsf{proof})$

Note: Previously developed techniques for verifiable set union cannot be used for the following reasons. The proof of membership condition in [131] does not satisfy zero-knowledge property since it reveals which particular set an element in answer comes from. Moreover, it is inefficient as pointed out in [36]. On the other hand, the union proof in [36] relies on non-falsifiable knowledge assumption and it is not zero-knowledge as it requires set intersection of [131] as a subroutine.

Algorithm 11 *Set Union*: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{unionV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

1: Check that answer has no repeated elements. If not, **return** reject.
2: Parse proof as $([a_\delta, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{a(v)\}_{v \in V(\mathsf{UT})}, W_{(\mathsf{answer}, \tilde{U})}, \{W_{(\mathcal{X}_j, \mathsf{answer})}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$ where $|\mathsf{answer}| = \rho$.
3: Certify the validity of $[a_\delta, \ldots, a_0]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i \in [0,\delta]} a_i w^i = \prod_{i \in [0,\delta]} (x_i + w)$ where $\mathsf{answer} = \{x_1, \ldots, x_\delta\}$. If the verification fails, **return** reject. Else proceed to next step.
4: For all $j \in [i_1, i_k]$ invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, j, \Pi_j, \alpha_j)$ If verification fails for at least one α_j, **return** reject. Else proceed to next step.
5: For each $j \in [i_1, i_k]$ verify that: $e(\alpha_j, W_{(\mathcal{X}_j, \mathsf{answer})}) = e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, g)$. If verification fails for at least one α_j, **return** reject. Else proceed to next step.
6: For each given node v in UT, verify that $e(a(v), g) = e(a(v_1), a(v_2))$, where v_1, v_2 are children of v. If verification fails for at least one node, **return** reject. Else proceed to next step.
7: Let root of UT be denoted as $\mathsf{acc}(\tilde{U})$. Verify that $e(W_{(\mathsf{answer}, \tilde{U})}, g^{\mathsf{Ch}_{\mathsf{answer}}(s)}) = e(\mathsf{acc}(\tilde{U}), g)$. If the verification fails, **return** reject. Else
8: **return** accept.

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below.

Query: Let $m = |\mathbb{S}|$, $N = \sum_{j \in [i_1, i_k]} n_j$, $n_j = |\mathcal{X}_j|$ and k is the number of indices in the query. Note that $N = |\tilde{U}|$ and N is an upper bound on ρ. We will analyze the complexity in each step. By Lemma 1, Step 2 has complexity $O(\rho \log \rho)$. By the same lemma, Step 4 has complexity $O(k\rho \log \rho)$. Now let us analyze the complexity of computing the union tree in Step 5. Computing all the nodes at a level has complexity $O(N \log N)$ (by Lemma 1) and the tree has $\log k$ levels. So the overall complexity in computing the union tree is $O(N \log N \log k)$. In Step 6, computing the member witness takes time $O((N - \rho) \log(N - \rho))$ which is bounded by $O(N \log N)$. Finally, Step 7 takes time $O(km^\varepsilon \log m)$. So the overall complexity of unionQ is $O(\rho \log \rho + k\rho \log \rho + N \log N \log k + N \log N + km^\varepsilon \log m) = O(k\rho \log \rho + N \log N \log k + km^\varepsilon \log m)$.

Verification: For the verification algorithm, let us look at each of the verification steps individually. Step 1 takes time $O(\rho)$. Step 3 takes time $O(\rho)$ by Lemma 2. By Lemma 4, Step 4takes total time $O(k)$. In Step 5, computing $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ takes time $O(\rho)$ and then there are $2k$ bilinear map computations. Therefor this step runs in time $O(\rho + k)$. Verifying the correctness of the union tree in Step 6 requires 2 bilinear map computation for each internal node of the tree and therefore takes time $O(k)$, where k is the number of nodes in the union tree. Finally, Step 5 takes $O(1)$ time since $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ has already been computed. Therefore, the total time at verification is $O(\rho + 2k) = O(\rho + k)$.

Proof size: The proof size is $O(\rho + k) = O(\rho + k)$.

3.7.6 Set Difference Query

Set difference query q is parameterized by two set indices of the set collection $q = (i_1, i_2)$. The answer to a set difference query is $\mathsf{answer} = \mathcal{X}_{i_1} - \mathcal{X}_{i_2}$ and a proof of correctness of the answer. We express the correctness of the answer using the following condition $(\mathsf{answer} = \mathcal{X}_{i_1} - \mathcal{X}_{i_2}) \iff \mathcal{X}_{i_1} - (\mathcal{X}_{i_1} - \mathsf{answer}) = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$. This condition ensures that (1) all the elements of answer indeed belongs to $\mathcal{X}_{i_1}$ and (2) *all* the elements of $\mathcal{X}_{i_1}$ that

are not in $\mathcal{X}_{i_2}$ are contained in answer. In other words, the union of answer and the intersection $I = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$ equals $\mathcal{X}_{i_1}$.

The condition is tricky to prove for the following reasons. The server cannot reveal neither $\mathcal{X}_{i_1} - \mathsf{answer}$ nor $\mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$ to the client, since this reveals more than the set difference answer the client requested for (hence, breaking our zero-knowledge property).[11] Hence, we are required to provide blinded accumulators corresponding to these sets. Unfortunately, the blinded version of $\mathcal{X}_{i-1} - (\mathcal{X}_{i_1} - \mathsf{answer})$ and $\mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$, even if the server computed the answer correctly, would be different. This is caused by different blinding factors used for these accumulators, even though the exponent that corresponds to the elements of the sets is the same. We use this fact and let the server prove that the non-blinded exponents are the same.

Our solution relies on an NIZKPoK protocol as described in Section 3.2. We use the NIZK version of the Σ-protocol in the random oracle model (standard Fiat Shamir transformation) as described in Figure 3.1. We would like to note that there are alternative methods [71, 112] to make Σ-protocols NIZK that do not rely on the random oracle model. Unfortunately, these methods come with some performance penalty. We give the pseudocode for the set difference query in Algorithm 12 and its verification in Algorithm 13.

Algorithm 12 *Set Difference*: $(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{differenceQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, (i_1, i_2))$

]

1: Let $\mathsf{answer} = \mathcal{X}_{i_1} - \mathcal{X}_{i_2} = \{x_1, \ldots, x_\delta\}$.
2: Let $[a_\delta, \ldots, a_0]$ be the coefficients of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
3: Compute membership witness for answer in $\mathcal{X}_{i_1}$ using the subset algorithm. Let $W_{(\mathsf{answer}, \mathcal{X}_{i_1})} = g^{r_i \mathsf{Ch}_{\mathcal{X}_{i_1} - \mathsf{answer}}(s)}$.
4: Compute $I \leftarrow \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$
5: Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $\mathsf{acc}(I) \leftarrow g^{r_{i_1} r_{i_2} \gamma \mathsf{Ch}_I}$.
6: Let $P_j[z] = \mathsf{Ch}_{\mathcal{X}_j - I}$ for all $j \in [i_1, i_2]$.
7: Compute $W_{I, \mathcal{X} i_1} \leftarrow g^{\frac{P_{i_1}(s)}{r_{i_2} \gamma}}$
8: Compute $W_{I, \mathcal{X} i_2} \leftarrow g^{\frac{P_{i_2}(s)}{r_{i_1} \gamma}}$
9: Using Extended Euclidian Algorithm, compute polynomials $q_j[z]$ such that $\sum_{j \in [i_1, i_2]} q_j[z] P_j[z] = 1$.
10: Pick a random $\beta \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q'_{i_1}[z] = (r_{i_2}\gamma)(q_{i_1}[z] + \beta P_{i_2}[z])$ and $q'_{i_2}[z] = (r_{i_2}\gamma)(q_{i_2}[z] - \beta P_{i_1}[z])$.
11: Compute $F_j = g^{q'_j(s)}$ for all $j \in [i_1, i_k]$.
12: Compute PKproof by invoking PK with $h = acc(I)$, $g' = W_{\mathsf{answer}, \mathcal{X}_{i_1}}$ and $x = r_{i_2}\gamma$.
13: Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$. Let $(\Pi_j, \alpha_j) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for $j \in [i_1, i_2]$.
14: Set $\mathsf{proof} := ([a_\delta, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, W_{(\mathsf{answer}, \mathcal{X}_{i_1})}, \mathsf{acc}(I), \{F_j, W_{(I, \mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1, i_2]}, \mathsf{PKproof})$ where where $|\mathsf{answer}| = \rho$.
15: **return** $(\mathsf{answer}, \mathsf{proof})$

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below.

Query Let us analyze the complexity of each of the steps. Let $m = |\mathbb{S}|$, $N = \sum_{j \in [i_1, i_2]} n_j$, $n_j = |\mathcal{X}_j|$ and k is the number of indices in the query. Note that N is an upper bound on ρ. Step 2 has complexity

[11] We note that the sets are revealed to the client in [131] where privacy is not a concern.

Algorithm 13 *Set Difference Verification*: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{differenceV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

1: Parse $\mathsf{proof} := ([a_\delta, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, W_{(\mathsf{answer}, \mathcal{X}_{i_1})}, \mathsf{acc}(I), \{F_j, W_{(I,\mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j\in[i_1,i_2]}, \mathsf{PKproof})$ where $|\mathsf{answer}| = \rho$.
2: Certify the validity of $[a_\delta, \ldots, a_0]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i\in[0,\delta]} a_i w^i = \prod_{i\in[0,\delta]}(x_i + w)$ where $\mathsf{answer} = \{x_1, \ldots, x_\delta\}$. If the verification fails, **return** reject. Else proceed to next step.
3: For all $j \in [i_1, i_k]$ invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, j, \Pi_j, \alpha_j)$ If verification fails for at least one α_j, **return** reject. Else proceed to next step.
4: Verify $W_{(\mathsf{answer}, \mathcal{X}_{i_1})}$ by checking $e(W_{(\mathsf{answer}, \mathcal{X}_{i_1})}, g^{\mathsf{Ch}_{\mathsf{answer}}(s)}) = e(\alpha_i, g)$. If the verification fails **return** reject. Else proceed to next step.
5: Parse $\mathsf{PKproof}$ as (b, c, r) and run the verification steps as in Figure 3.1 with $h = acc(I)$, $g' = W_{\mathsf{answer}, \mathcal{X}_{i_1}}$. If the verification fails **return** reject. Else proceed to next step.
6: Verify if $e(W_{(I,\mathcal{X}_j)}, \mathsf{acc}(I)) = e(\alpha_j, g)$ for $j \in [i_1, i_2]$. If verification fails for at least one α_j, **return** reject. Else proceed to next step.
7: Check if $\prod_{j\in[i_1,i_2]} e(F_j, W_{(I,\mathcal{X}_j)}) = e(g, g)$. If verification fails **return** reject. Else
8: **return** accept

$O(\rho \log \rho)$ by Lemma 1. By Lemma 1, Step 3 has complexity $O((\mathcal{X}_{i_1} - \rho)\log(\mathcal{X}_{i_1} - \rho))$. By the same lemma, Step 5 has complexity $O(|I| \log |I|)$. By the analysis for set intersection, we know that Steps 7-11 has complexity $O(N \log^2 N \log\log N)$. From Figure 3.1, we see, computing $\mathsf{PKproof}$ takes time $O(1)$ in Step 12. Finally, Step 13 has complexity $O(m^\varepsilon \log m)$ by Lemma 4. Therefore, the overall complexity is $O(\rho \log \rho + (\mathcal{X}_{i_1} - \rho)\log(\mathcal{X}_{i_1} - \rho) + |I| \log |I| + N \log^2 N \log\log N + m^\varepsilon \log m) = O(N \log^2 N \log\log N + m^\varepsilon \log m)$.

Verification: For the verification algorithm, let us look at each of the verification steps individually. Step 2 takes time $O(\rho)$. By Lemma 2. Step 3 has complexity $O(1)$ by Lemma 4. In Step 4 computing $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ has complexity $O(\rho)$. Step 6, 5 and 7 has complexity $O(1)$ each. Therefore, the overall complexity of verification is $O(\rho)$.

Proof size: The proof size is $O(\rho)$.

3.7.7 Efficiency Comparison with the Scheme of [131]

We compare the asymptotic complexity of the algorithms of our ZKASC scheme with that of [131] in Figure 3.4. Recall that our ZKASC scheme satisfies both authenticity and privacy (in particular, the strong notion of zero-knowledge property) for set algebra queries, while [131] provides only authenticity and trivially reveals information about the set collection $\mathbb{S} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ beyond query answers. We show that our setup, query and verify algorithms have the exact same asymptotic performance as those of [131]. The update algorithms are more expensive compared to that of [131]. At a high level, the extra cost is due to zero-knowledge property we achieve in return. We require all the proofs to be ephemeral, i.e., proofs should not hold good between updates. Hence, every set for which the client has received a proof between the updates has to be refreshed. Finally, we note that the construction of [36] offers faster union queries (roughly by a multiplicative $k/\log N$ factor) but its security relies on non-falsifiable type assumptions.

3.7.8 Security Proofs

In this section, we will prove that our ZKASC construction satisfies the security properties of soundness (Definition 11) and zero-knowledge (Definition 12). Completeness (Definition 10) directly follows from the constructions as described with the respective algorithms. We first prove a short lemma that we will use in our soundness proof.

Lemma 5. *For any adversarially chosen proof* Π_i*, if algorithm* $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, i, \Pi_i, \alpha_i)$ *accepts, then* $\alpha_i = \mathsf{acc}(\mathcal{X}_i)$ *with probability* $\Omega(1-\nu(\lambda))$

Proof. The lemma is identical to Lemma 5 in [131]. The only difference is that now each set is represented using a randomized accumulator instead of a general accumulator. We will show a reduction from Lemma 5 in [131] to this lemma. For the sake of contradiction, let us assume that there exists an adversary Adv that outputs Π_i such that $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, i, \Pi_i, \alpha_i) = \mathsf{accept}$ and $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i)$. Then there exists an adversary $\mathcal{A}$ that does the following:

- Initially $\mathcal{A}$ sends the public key vk to Adv.
- Adv comes up with a set collection $\mathbb{S}_0 = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$.
- $\mathcal{A}$ picks random elements $r_1, \ldots, r_m \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathbb{S}_0' = \{\mathcal{X}_1 \cup r_1, \ldots, \mathcal{X}_m \cup r_m\}$., where $r_i \notin \mathcal{X}_i$. Otherwise $\mathcal{A}$ picks a fresh randomness for that set.
- $\mathcal{A}$ forwards all the authentication units to Adv. Note that, from Adv's perspective, the blinding factor used for $\mathcal{X}_i$ is $s + r_i$.
- $\mathcal{A}$ forwards all the update requests to its challenger and forwards the challenger's responses to Adv. Note that, if Adv requests to insert element r_i is $\mathcal{X}_i$, $\mathcal{A}$ aborts. But this happens only with probability $\frac{1}{|\mathbb{Z}_p^*|}$ which is negligible in the security parameter λ.
- Finally $\mathcal{A}$ outputs Adv's forgery Π_i and inherits Adv's success probability.

Therefore, $\mathcal{A}$ breaks Lemma 5 in [131]. □ □

Proof of Soundness: In this section we prove that the ZKASC construction satisfies the soundness property. Formally, we prove Lemma 6.

Lemma 6. *Under the q-SBDH assumption, the ZKASC scheme is sound as per Definition 11.*

Proof. Let q be some polynomial in the security parameter of the scheme such that q is an upper bound on the maximum size the adversarial set collection can grow to. Since the adversary has to come with a set collection that is $poly(\lambda)$, and is allowed to ask a polynomial (in λ) number of queries, this upper bound exists. We will give a reduction to q-SBDH assumption. Let $\mathcal{A}$ be the reduction which receives a q-SBDH tuple $(g, g^s, \ldots, g^{s^q})$ as input. The reduction does the following:

Recall the soundness game where the adversary Adv sees the vk, then comes up with a set collection $\mathbb{S}_0 = \mathcal{X}_1, \ldots, \mathcal{X}_k$. and then sends it to $\mathcal{A}$. $\mathcal{A}$ runs all the steps of Algorithm 2 where it computes the proof units using $(g, g^s, \ldots, g^{s^q})$ instead of the secret key s and sends $(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0)$. Note that all the steps can be executed using the tuple $(g, g^s, \ldots, g^{s^q})$. For an update query, $\mathcal{A}$ runs the steps of Algorithm 3 and uses $(g, g^s, \ldots, g^{s^q})$ instead of s as before.

If Adv outputs a succesful forgery, it has to forge proof for at least one of the following queries: subset, set intersection, set union or set difference. Let us define the following events:

Event A_1: The adversary outputs a forged subset proof.

Event A_2: The adversary outputs a forged set intersection proof.

Event A_3: The adversary outputs a forged set union proof.

Event A_4: The adversary outputs a forged set difference proof.

The probability that Adv outputs a successful forgery is $Pr[A_1 \cup A_2 \cup A_3 \cup A_4] \leq Pr[A_1] + Pr[A_2] + Pr[A_3] + Pr[A_4]$. We will individually bound the probability of success of each of the events individually.

Lemma 7. *Under the q-SBDH assumption, there exists a negligible function $\nu(\lambda)$ such that $Pr[A_1] \leq \nu(\lambda)$*

Proof. If Adv outputs a succesful forgery, it has to forge on at least one of the following steps: Step 2, 3, and 5 or 8 in Algorithm subsetQ. Let us denote the corresponding events as E_i for $i \in [1,4]$ respectively and bound the probability of each event. By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

By Lemma 5 $Pr[E_2] \leq \nu_2(\lambda)$ for some negligible function ν_2.

Next we analyze the probability of forging a membership proof, i.e., $Pr[E_3]$.

- The equation $e(g^{\mathsf{Ch}_\Delta(s)}, W_{(\Delta,\mathcal{X}_i)}) = e(\mathsf{acc}(\mathcal{X}_i), g)$ verified to be true but $\Delta \nsubseteq \mathcal{X}_i$. This implies there exists at least one element $y \in \Delta$ such that $y \notin \mathcal{X}_i$.
- Therefore there exists polynomial $P[z]$and a scalar c such that $\mathsf{Ch}_{\mathcal{X}_i}[z] = (y+z)P[z] + c$.
- $\mathcal{A}$ computes polynomial $P[z]$ and scalar c using polynomial long division.
- $\mathcal{A}$ outputs $[y, (e(W_{(\Delta,\mathcal{X}_i)}, g^{\mathsf{Ch}_{\{\Delta-y\}}(s)})^{r_i^{-1}}) e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\Delta,\mathcal{X}_i)}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_3] \leq \nu_3(\lambda)$ for some negligible function ν_3.

Therefore, the probability of successfully forging a membership proof is $Pr[E_1 \wedge E_2 \wedge E_3] \leq Pr[E_1] + Pr[E_2] + Pr[E_3] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'.

Now let us look at the probability of forging a non-membership proof, i.e., $Pr[E_4]$.

- E_4 is the event when $e(F_1, g^{\mathsf{Ch}_\Delta(s)}) e(F_2, \mathsf{acc}(\mathcal{X}_i)) = e(g,g)$ but $\Delta \subseteq \mathcal{X}_i$. This implies for all $y \in \Delta$, $y \in \mathcal{X}_i$.
- Therefore for any $y \in \Delta$ there exists a polynomials $P[z]$ such that $\mathsf{Ch}_{\mathcal{X}_i}[z] = (y+z)P[z]$.
- Compute polynomial $P[z]$ using polynomial long division.
- Compute $e(F_1, g^{\mathsf{Ch}_{\Delta-y}(s)}) e(F_2, g^{r_i P(s)})$ which equals $e(g,g)^{\frac{1}{y+s}}$
- Output $[y, e(F_1, g^{\mathsf{Ch}_{\Delta-y}(s)}) e(F_2, g^{r_i P(s)})]$

Therefore using F_j's, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_4] \leq \nu_4(\lambda)$ for some negligible function ν_4.

Therefore, the probability of successfully forging a membership proof is $Pr[E_1 \wedge E_2 \wedge E_3] \leq Pr[E_1] + Pr[E_2] + Pr[E_4] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_4(\lambda) \leq \nu''(\lambda)$ for some negligible function ν''. □ □

Lemma 8. *Under the q-SBDH assumption, there exists a negligible function $\nu(\lambda)$ such that $Pr[A_2] \leq \nu(\lambda)$*

Proof. If Adv outputs a succesful forgery, it has to forge on at least one of the following steps: Step 2, 3, 4 and 5 in Algorithm insertQ. Let us denote the corresponding events as E_i for $i \in [1,4]$ respectively and bound the probability of each event.

By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

Next we bound $Pr[E_2]$. By Lemma 5, the probability that for some set $\mathcal{X}_i$, $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i) \leq \nu^i(\lambda)$ for some negligible function ν^i. Let $\nu(\lambda)$ be the maximum of the negligible functions $\nu^i(\lambda)$ for $i \in [1,k]$. Note that all the ν^i functions are independent of each other by By Lemma 4. Therefore, by union bound, $Pr[E_2] \leq k\nu(\lambda) = \nu_2(\lambda)$ for some negligible function ν_2 since $k = poly(\lambda)$. Therefore, we have, $Pr[E_2] \leq \nu_2(\lambda)$.

Now let us bound $Pr[E_3]$.

- At least for one j, it must be the case that $e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, W_{(\mathsf{answer},\mathcal{X}_j)}) = e(\mathsf{acc}(\mathsf{st}_j), g)$ but $\mathsf{answer} \nsubseteq \mathcal{X}_j$.
- This in turn implies, there exists at least one element $y \in \mathsf{answer}$ such that $y \notin \mathcal{X}_j$.
- This implies $(y+z)$ does not divide $\mathsf{Ch}_{\mathcal{X}_j}[z]$. Therefore there exists a polynomial $q[z]$ and a scalar c such that $\mathsf{Ch}_{\mathcal{X}_j}[z] = (y+z)q[z]+c$.
- Use polynomial long division to compute $q[z]$ and c.
- Output $[y, (e(W_{(\mathsf{answer},\mathcal{X}_j)}, g^{\mathsf{Ch}_{\{\mathsf{answer}-y\}}(s)})^{r_j^{-1}} e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\mathsf{answer},\mathcal{X}_j)}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_3] \leq \nu_3(\lambda)$ for some negligible function ν_3.

Now we look at $Pr[E_4]$.

- E_4 is the event when $\prod_{j\in[i_1,i_k]} e(F_j, W_{(\mathsf{answer},\mathcal{X}_j)}) = e(g,g)$ but the intersection $(\mathcal{X}_{i_1} - \mathsf{answer}) \cap (\mathcal{X}_{i_2} - \mathsf{answer}) \cap \ldots \cap (\mathcal{X}_{i_k} - \mathsf{answer}) \neq \phi$, i.e., there exists at least one element y that belongs to all the sets $(\mathcal{X}_j - \mathsf{answer})$ for $j \in [i_1, i_k]$.
- Therefore, there exists polynomials $\tilde{P}_j[z]$ such that $\mathsf{Ch}_{\mathcal{X}_j - \mathsf{answer}}[z] = (y+z)\tilde{P}_j[z]$ for $j \in [i_1, i_k]$.
- Compute polynomial $\tilde{P}_j[z]$ using polynomial long division.
- Compute $\prod_{j\in[i_1,i_k]} e(F_j, g^{r_j \tilde{P}_j(s)})$ which equals $e(g,g)^{\frac{1}{y+s}}$
- Output $[y, \prod_{j\in[i_1,i_k]} e(F_j, g^{r_j\tilde{P}_j(s)})]$

Therefore using F_j's, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_4] \leq \nu_4(\lambda)$ for some negligible function ν_4.

Therefore, we have $Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4] \leq Pr[E_1] + Pr[E_2] + Pr[E_3] + Pr[E_4] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'. □. □

Lemma 9. *Under the q-SBDH assumption, there exists a negligible function* $\nu(\lambda)$ *such that* $Pr[A_3] \leq \nu(\lambda)$

Proof. If Adv outputs a succesful forgery, it has to forge on at least one of the following steps: Step 3, 4, 5, 6 and 8 in Algorithm unionQ. Let us denote the corresponding events as E_i for $i \in [1,5]$ respectively and bound the probability of each event.

By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

Now let us look at $Pr[E_2]$. By Lemma 5, we have that the probability that for some set $\mathcal{X}_i$, $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i) \leq \nu^i(\lambda)$ for some negligible function ν^i. Let $\nu(\lambda)$ be the maximum of the negligible functions $\nu^i(\lambda)$ for $i \in [1,k]$. Note that all the ν^i functions are independent of each other by By Lemma 4. Therefore, by union bound, $Pr[E_2] \leq k\nu(\lambda) = \nu_2(\lambda)$ for some negligible function ν_2 since $k = poly(\lambda)$. Therefore, we have, $Pr[E_2] \leq \nu_2(\lambda)$.

Now let us bound $Pr[E_3]$.

- At least for one j, it must be the case that $e(\alpha_j, W_{(\mathcal{X}_j,\mathsf{answer})}) = e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, g)$ but $\mathcal{X}_j \nsubseteq \mathsf{answer}$.

- This in turn implies, there exists at least one element $y \in \mathcal{X}_j$ such that $y \notin \mathsf{answer}$.
- This implies $(y+z)$ does not divide $\mathsf{Ch}_{\mathsf{answer}}[z]$. Therefore there exists a polynomial $q[z]$ and a scalar c such that $\mathsf{Ch}_{\mathsf{answer}}[z] = (y+z)q[z]+c$.
- Use polynomial long division to compute $q[z]$ and c.
- Output $[y, (e(W_{(\mathcal{X}_j,\mathsf{answer})}, g^{\mathsf{Ch}_{\{\mathcal{X}_j - y\}}(s)})^{r_j} e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\mathcal{X}_j,\mathsf{answer})}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_3] \leq \nu_3(\lambda)$ for some negligible function ν_3.

Now we look at $Pr[E_4]$.

Forgery for some node v in the tree with children v_1, v_2 implies $e(a(v), g) = e(a(v_1), a(v_2))$ but $a(v_1)$ is not the correct accumulation value at v_1 or $a(v_2)$ is not the correct accumulation value at v_2 (or both). Since the verification happens bottom up, and by definition, $a(v) = g^{r_1 r_2 \mathsf{Ch}_M(s)}$ therefore, it must be the case that for some leaf node v_l, $a(v_l)$ was not the correct accumulation for the corresponding set (say, $\mathcal{X}_i$), i.e., $a(v_l) \neq \mathsf{acc}(\mathcal{X}_i)$. But by Lemma 4, this probability is negligibly small. Therefore, following the same argument as for E_2, we have $Pr[E_4] \leq \nu_4(\lambda)$ for some negligible function ν_4.

Now let us estimate the probability of $Pr[E_5]$. We will show that if $e(W_{(\mathsf{answer},\tilde{U})}, g^{\mathsf{Ch}_{\mathsf{answer}}(s)}) = e(\mathsf{acc}(\tilde{U}), g)$, but $\mathsf{answer} \nsubseteq \tilde{U}$, then, $\mathcal{A}$ can break the q-SBDH assumption with non-negligible probability.

- Since $\mathsf{answer} \nsubseteq \tilde{U}$ it must be the case that there exists at least one element $y \in \mathsf{answer}$ such that $y \notin \tilde{U}$.
- This implies $(y+z)$ does not divide $\mathsf{Ch}_{\tilde{U}}[z]$. Therefore there exists a polynomial $q[z]$ and a scalar c such that $\mathsf{Ch}_{\tilde{U}}[z] = (y+z)q[z]+c$.
- Use polynomial long division to compute $q[z]$ and c.
- Output $[y, (e(W_{(\mathsf{answer},\tilde{U})}, g^{\mathsf{Ch}_{\{\mathsf{answer}-y\}}(s)})^{(\prod_{j\in[i_1,i_k]} r_j^{-1})} e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\mathsf{answer},\tilde{U})}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_5] \leq \nu_5(\lambda)$ for some negligible function ν_5.

Therefore, we have $Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4 \wedge E_5] \leq Pr[E_1] + Pr[E_2] + Pr[E_3] + Pr[E_4] + Pr[E_5] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) + \nu_5(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'. □. □

Lemma 10. *Under the q-SBDH assumption, there exists a negligible function* $\nu(\lambda)$ *such that* $Pr[A_4] \leq \nu(\lambda)$

Proof. If Adv outputs a successful forgery, it has to forge on at least one of the following steps: Step 2, 3, 4, 5, 6 and 7 in Algorithm $\mathsf{differenceQ}$. Let us denote the corresponding events as E_i for $i \in [1,6]$ respectively and bound the probability of each event.

By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

Now let us look at $Pr[E_2]$. By Lemma 5, we have that the probability that for some set $\mathcal{X}_i$, $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i) \leq \nu^i(\lambda)$ for some negligible function ν^i. Let $\nu(\lambda)$ be the maximum of the negligible functions $\nu^i(\lambda)$ for $i \in [1,k]$. Note that all the ν^i functions are independent of each other by By Lemma 4. Therefore, by union bound, $Pr[E_2] \leq k\nu(\lambda) = \nu_2(\lambda)$ for some negligible function ν_2 since $k = poly(\lambda)$. Therefore, we have, $Pr[E_2] \leq \nu_2(\lambda)$.

By the soundness of subset query, there exists negligible functions ν_3, ν_4 such that $Pr[E_3] \leq \nu_3(\lambda)$ and $Pr[E_4] \leq \nu_4(\lambda)$.

Therefore, by the soundness of the zero knowledge protocol PK there exists negligible functions ν_4 such that $Pr[E_4] \leq \nu_4(\lambda)$.

In Step 4, the soundness of $W_{\mathsf{answer},\mathcal{X}_{i_1}}$ has been verified and in Step 5, it has been verified that $\mathsf{acc}(I)$ is $W_{\mathsf{answer},\mathcal{X}_{i_1}}$ raised to some exponent that the prover has knowledge of. By the knowledge soundness of PK, there exists an efficient extractor that can extract the exponent. Let us call this exponent x and let $\gamma = x/r_{i_2}$. $\mathcal{A}$ can us the extractor and compute $\mathsf{acc}(I)^{x^{-1}{r_{i_1}}^{-1}}$. $\mathsf{acc}(I)^{x^{-1}{r_{i_1}}^{-1}}$ can be though of as $g^{\mathsf{Ch}_{\mathcal{X}_{i_1}-\mathsf{answer}}(s)}$. Now $\mathcal{A}$ can compute polynomial $Q[z]$ and scalar c such that $\mathsf{Ch}_{\mathcal{X}_j} = (y+z)Q[z]+c$ and output $[y,(e(W_{I,\mathcal{X}_j},g^{\mathsf{Ch}_{\{\mathcal{X}_{i_1}-\mathsf{answer}-y\}}(s)})^{r_{j'}\gamma}e(g,g^{-Q(s)}))^{c^{-1}}] = [y,e(g,g)^{\frac{1}{y+s}}]$ (where j' is $(i_1,i_2)\setminus j$). This breaks the q-SBDH assumption. Hence we conclude that $Pr[E_5] \leq \nu_5(\lambda)$ for some negligible function ν_5.

Finally, we $Pr[E_6]$. Using γ as above, $\mathcal{A}$ can use forged F_j, $j \in [i_1,i_2]$ as in the proof of set intersection to come up with $[y',e(g,g)^{\frac{1}{y'+s}}]$ which breaks the q-SBDH assumption. Therefore using F_j's, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability as Adv. Hence it must be the case that $Pr[E_6] \leq \nu_6(\lambda)$ for some negligible function ν_6.

Therefore, we have $Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4 \wedge E_5 \wedge E_6] \leq Pr[E_1]+Pr[E_2]+Pr[E_3]+Pr[E_4]+Pr[E_5]+Pr[E_6] \leq \nu_1(\lambda)+\nu_2(\lambda)+\nu_3(\lambda)+\nu_4(\lambda)+\nu_5(\lambda)+\nu_6(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'. □. □

Therefore, by Lemma 7, Lemma 8, Lemma 9 and Lemma 10 we have $Pr[A_1 \cup A_2 \cup A_3 \cup A_4] \leq \nu(\lambda)$ for some negligible function ν. □

□

Proof of Zero-Knowledge: We prove that the ZKASC construction satisfies the zero-knowledge property. Formally, we prove Lemma 11.

Lemma 11. *The ZKASC scheme satisfies zero-knowledge property as per Definition 12.*

Proof. We write a simulator Sim that acts as follows:

- It first runs $\mathsf{GenParams}(1^\lambda)$ to receive bilinear parameters $pub = (p,\mathbb{G},\mathbb{G}_T,e,g)$
- Then it picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and sends $\mathsf{vk} := g^s, pub$ to Adv and saves s as its secret key in its state information st_S.
- Given vk, Adv comes up with a set collection $\mathbb{S}_0$ of its choice. Sim is given $|\mathbb{S}_0| = m$ which is the public information of the scheme.
- Sim picks m random numbers $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ for $i \in [1,m]$ and computes $\mathsf{acc}(\mathcal{X}_i) \xleftarrow{\$} g^{r_i}$ and saves all the r_i's in its state information.
- Then Sim invokes $\mathsf{AT.ATSetup}(\mathsf{sk},(\mathsf{acc}(\mathcal{X}_1),\ldots,\mathsf{acc}(\mathcal{X}_m)))$. Let $\mathsf{AT.ATSetup}(\mathsf{sk},(\mathsf{acc}(\mathcal{X}_1),\ldots,\mathsf{acc}(\mathcal{X}_m))) = (\mathsf{auth}(\mathbb{S}_0),\mathsf{digest}_0)$.
- Sim saves $\mathsf{auth}(\mathbb{S}_0),\mathsf{digest}_0$ in st_S and sends digest_0 to Adv.

After this, when Adv adaptively asks for op. Sim maintains a list of previously asked queries and uses the proof units used by the Sim. If a query is repeated by Adv between two updates, then Sim uses the same proof units. Otherwise Sim does the following:

If $\mathsf{op} = \mathsf{update}$**:** Sim makes oracle call to $D()$ to check the validity of update. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, for all indices $i \in [1,m]$, Sim looks up its state information to see if any query touched index i since the last Update. Let $[i_1,\ldots,i_k]$ be the set of indices. Let $\mathbb{S}_t$ is the most recent set collection

- For all the indices $j \in [i_1,\ldots,i_k]$, Sim picks fresh random elements $r'_j \xleftarrow{\$} \mathbb{Z}_p^*$.
- It updates sets $\mathsf{acc}(\mathcal{X}_j) \leftarrow g^{r_j r'_j}$ and updates r_j with $r_j r'_j$ in its state information.
- Then it calls $\mathsf{AT.ATUpdate}(\mathsf{sk}, j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$. Let $\mathsf{AT.ATUpdate}(\mathsf{sk}, j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t) = (\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}_t)$.

Return digest_{t+1} to Adv.

If $\mathsf{op} = \mathsf{subsetQ}$ **for** (Δ, i) Sim makes oracle call to $D()$ to check the validity of subsetQ. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, it makes oracle call to $\mathbb{S}_t$ (where $\mathbb{S}_t$ is the most recent set collection) and gets answer. Let answer $= 1$. Then Sim computes $W_{\Delta,\mathsf{st}_i} \leftarrow g^{\frac{r_i}{\mathsf{Ch}_\Delta(s)}}$ Else if answer $= 0$, Sim does the following:

- let $\Delta = \{x_1,\ldots,x_k\}$. Compute coefficients $[a_0,\ldots,a_k]$ of the polynomial Ch_δ.
- Now let $x = GCD(r_i, \mathsf{Ch}_\Delta[z])$.
- Using Extended Euclidian algorithm, compute q_1, q_2 such that $q_1(\mathsf{Ch}_\Delta[z]/x) + q_2(r_i/x) = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$.
- Set $q'_1[z] := \frac{q_1[z]+\gamma r_i}{x}$ and $q'_2[z] := \frac{q_2[z]-\gamma \mathsf{Ch}_\Delta[z]}{x}$.
- Compute $F_1 \leftarrow g^{q'_1(s)}, F_2 \leftarrow g^{q'_2(s)}$.
- Set $W_{\Delta,\mathcal{X}_i} := (F_1, F_2)$

Set $\mathsf{proof} = ([a_0,\ldots,a_k], W_{\Delta,\mathcal{X}_i}, \Pi_i, \alpha_i)$ Return $(\mathsf{answer}, \mathsf{proof})$ to Adv.

If $\mathsf{op} = \mathsf{IntersectionQ}$ **for indices** $i_1,\ldots,i_k$**:** Sim makes oracle call to $D()$ to check the validity of IntersectionQ. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, it makes oracle call to $\mathbb{S}_t$ (where $\mathbb{S}_t$ is the most recent set collection) and gets answer. Let answer $= \{x_1,\ldots,x_\delta\}$. Now Sim does the following:

- Compute coefficients $[a_\delta,\ldots,a_0]$ of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
- For each index $j \in [i_1, i_k]$ compute $W_{(\mathsf{answer},\mathcal{X}_j)} \leftarrow g^{\frac{r_j}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$.
- Now let $x = GCD(r_j)$ for $j \in [i_1, i_k]$ and let us denote as $\tilde{r_j} = r_j/x$.
- Note that $\tilde{r_j}$'s are co-prime. Using Extended Euclidian algorithm, compute q_j's such that $\sum_{j\in[i_1,i_k]} q_j \tilde{r_j} = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q'_j[z] := (q_j + (\gamma \frac{r_{j+1}}{\mathsf{Ch}_{\mathsf{answer}}[z]})) \frac{\mathsf{Ch}_{\mathsf{answer}}[z]}{x}$.
- Compute $F_j \leftarrow g^{q'_j(s)}$.
- Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t)) = (\Pi_j, \alpha_j)$.
- Set Set $\mathsf{proof} = ([a_\delta,\ldots,a_0], \{F_j, W_{(\mathsf{answer},\mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j\in[i_1,i_k]})$.

Return $(\mathsf{answer}, \mathsf{proof})$ to Adv.

If $\mathsf{op} = \mathsf{unionQ}$ **for indices** $i_1,\ldots,i_k$**:** Sim makes oracle call to $D()$ to check the validity of unionQ. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, it makes oracle call to $\mathbb{S}_t$ (where $\mathbb{S}_t$ is the most recent set collection)

and gets answer. Let answer $= \{x_1, \ldots, x_\delta\}$. Now Sim does the following:

- Compute coefficients $[a_\delta, \ldots, a_0]$ of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
- For each index $j \in [i_1, i_k]$ compute $W_{(\mathcal{X}_j, \mathsf{answer})} \leftarrow g^{\frac{\mathsf{Ch}_{\mathsf{answer}}(s)}{r_j}}$.
- Build a binary tree on k leaves (representing the k indices in the query) as follows: For a leaf node, let $a(v) = \mathsf{acc}(\mathcal{X}_j) = g^{r_j}$. For each internal node, $a(v) = g^{r_1 r_2}$ where r_1 and r_2 are the corresponding blinding factors for v_1 and v_2 respectively.
- Compute $W_{(\mathsf{answer}, \tilde{U})} \leftarrow g^{\frac{(\prod_{j \in [i_1, i_k]} r_j)}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$
- Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t)) = (\Pi_j, \alpha_j)$.
- $\mathsf{proof} := ([a_\delta, \ldots, a_0], \{a(v)\}_{v \in V(\mathsf{UT})}, W_{(\mathsf{answer}, \tilde{U})}, \{W_{(\mathcal{X}_j, \mathsf{answer})}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$

Sim returns $(\mathsf{answer}, \mathsf{proof})$ to Adv.

If $\mathsf{op} = \mathsf{differenceQ}$ **for indices** i_1, i_2**:** Sim makes oracle call to $D()$ to check the validity of differenceQ. If $D()$ returns 0, Let answer $= \{x_1, \ldots, x_\delta\}$. Now Sim does the following:

- Compute coefficients $[a_\delta, \ldots, a_0]$ of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
- Compute $W_{\mathsf{answer}, \mathcal{X}_{i_1}} \leftarrow g^{\frac{r_{i_1}}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$.
- Pick $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $\mathsf{acc}(I) := g^{\frac{r_{i_1} r_{i_2} \gamma}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$.
- Compute $W_{I, \mathcal{X}_{i_1}} \leftarrow g^{\frac{1}{r_{i_2} \gamma}}$
- Compute $W_{I, \mathcal{X}_{i_2}} \leftarrow g^{\frac{1}{r_{i_1} \gamma}}$
- Now let $x = GCD(r_{i_1}, r_{i_2})$ and let us denote as $\tilde{r_j} = r_j / x$ for $j \in [i_1, i_2]$.
- Note that $\tilde{r_j}$'s are co-prime. Using Extended Euclidian algorithm, compute q_j's such that $\sum_{j \in [i_1, i_2]} q_j \tilde{r_j} = 1$.
- Pick a random $\beta \xleftarrow{\$} \mathbb{Z}_p^*$.
- Set $q'_{i_1}[z] := (q_{i_1}[z] + \beta r_{i_2}) \frac{r_{i_1} r_{i_2} \gamma}{x}$.
- Set $q'_{i_2}[z] := (q_{i_2}[z] - \beta r_{i_1}) \frac{r_{i_1} r_{i_2} \gamma}{x}$.
- Compute $F_j \leftarrow g^{q'_j(s)}$.
- Compute PKproof by invoking PK with $h = acc(I)$, $g' = W_{\mathsf{answer}, \mathcal{X}_{i_1}}$ and the exponent as $r_{i_2} \gamma$.
- Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_2]$. Let $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t)) = (\Pi_j, \alpha_j)$.
- Set $\mathsf{proof} := ([a_\delta, \ldots, a_0], W_{(\mathsf{answer}, \mathcal{X}_{i_1})}, \mathsf{acc}(I), \{F_j, W_{(I, \mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1, i_2]}, \mathsf{PKproof})$.

Return $(\mathsf{answer}, \mathsf{proof})$ to Adv

It is easy to see that all the verification steps are satisfies. Observe that random values r are chosen independently after each update (and initial setup) in both cases and all the units of proof and digest have a one random blinding factor in the exponent. Hence they are distributed identically to random elements. This follows from a simple argument. Let $x, y, z \in \mathbb{Z}_p^*$ where x is a fixed element and $z = x + y$ or $z = xy$. Then z is identically distributed to y in $\mathbb{Z}_p^*$ in both cases. In other words, if y is picked with probability γ, then so is z. The same argument holds for elements in G and G_T.

Thus simulator Sim produces a view that is identically distributed to that produced by the challenger during $\mathbf{Real}_{\mathsf{Adv},k}$ and the simulation is perfect. □

□

We summarize the efficiency and the security properties of the ZKASC scheme in Theorem 3.

Theorem 3. *The* $\mathsf{ZKASC} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Update}, \mathsf{UpdateServer}, \mathsf{Query}, \mathsf{Verify})$ *scheme satisfies the properties of completeness (Definition 10), soundness (Definition 11), and zero-knowledge (Definition 12). Let* $\mathbb{S} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ *be the set original set collection. Define* $M = \sum_{i \in m} |\mathcal{X}_i|$, $n_j = |\mathcal{X}_j|$, *and* $N = \sum_{j \in [i_1, i_k]} n_j$. *Let* k *be the number of group elements in the query input (for the subset query, it is the cardinality of a queried subset, and for the rest of the queries it is the number of set indices). Let* ρ *be the size of a query answer,* L *be the number of sets touched by the queries between updates* u_{t-1} *and* u_t, *and* $0 < \varepsilon < 1$ *be a constant chosen at the time of setup. We have:*

- Gen *has access complexity* $O(1)$*;*
- Setup *has complexity* $O(M+m)$*;*
- Update *and* $\mathsf{UpdateServer}$ *have complexity* $O(L)$*;*
- Query *and* Verify *have the following access complexity:*
  - *For is-subset, the access complexity is* $O(N \log^2 N \log\log N + m^{\varepsilon} \log m)$. *The proof size is* $O(k)$ *and the verification has complexity* $O(k)$.
  - *For set intersection, the access complexity is* $O(N \log^2 N \log\log N + km^{\varepsilon} \log m)$. *The proof size is* $O(\rho + k)$ *and the verification has complexity* $O(\rho + k)$.
  - *For set union, the access complexity is* $O(k\rho \log \rho + N \log N \log k + km^{\varepsilon} \log m)$. *The proof size is* $O(\rho + k)$ *and the verification has complexity* $O(\rho + k)$.
  - *For set difference, the access complexity is* $O(N \log^2 N \log\log N + m^{\varepsilon} \log m)$. *The proof size is* $O(\rho)$ *and the verification has complexity* $O(\rho)$.

3.8 Conclusion

In this work, we have introduced zero-knowledge as a privacy notion for cryptographic accumulators. Zero-knowledge is a very strong security property that requires that witnesses and accumulation values leak nothing about the accumulated set at any given point in the protocol execution, even after insertions and deletions. We have shown that zero-knowledge accumulators are located between zero-knowledge sets and the recently introduced notion of primary-secondary-resolver membership proof systems, as the they can be constructed (in a black-box manner) from the former and they can be used to construct (in a black-box manner) the latter.

We have then provided the first construction of a zero-knowledge accumulator that achieves computational soundness and perfect zero-knowledge. Using this construction as a building block, we have designed a zero-knowledge authenticated set collection scheme that handles set-related queries that go beyond set (non-)membership. In particular, our scheme supports set unions, intersections, and differences, thus offering a complete set algebra.

There are plenty of future research directions in the area that, we believe, can use this work as a stepping stone. For example, our construction is secure under a parametrized (i.e., q-type) assumption. It would be interesting to develop an alternative construction from a constant-size assumption (such as RSA). Another interesting research direction would be to come up with an alternative protocol for the set-difference operation, since the one we present here utilizes a Σ-protocol and, in order to make it non-interactive zero-knowledge, without compromising on efficiency, we must rely on the Fiat-Shamir heuristic. Finally, the relations between primitives presented here complement the related results of [123], but are far from being complete; we believe there exist plenty of other directions to be explored.

Notation: The notation $q[z]$ denotes polynomial q over undefined variable z and $q(s)$ is the evaluation of the polynomial at point s. All arithmetic operations are performed $\bmod p$. N is a variable maintained by the owner.

Key Generation $(sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda)$

Run $\mathsf{GenParams}(1^k)$ to receive bilinear parameters $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g)$. Choose $s \xleftarrow{\$} \mathbb{Z}_p^*$. Return $sk = s$ and $vk = (g^s, pub)$.

Setup $(\mathsf{acc}, ek, \mathsf{aux}) \leftarrow \mathsf{Setup}(sk, \mathcal{X})$

Choose $r \xleftarrow{\$} \mathbb{Z}_p^*$. Set value $N = |\mathcal{X}|$. Return $\mathsf{acc} = g^{r \cdot \mathsf{Ch}_{\mathcal{X}}(s)}$, $ek = (g, g^s, g^{s^2}, \ldots, g^{s^N})$ and $\mathsf{aux} = (r, N)$.

Witness Generation $(b, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{acc}, \mathcal{X}, x, ek, \mathsf{aux})$

If $x \in \mathcal{X}$ compute $\mathsf{proof} = (\mathsf{acc})^{\frac{1}{s+x}} = g^{r \cdot \mathsf{Ch}_{\mathcal{X}\setminus\{x\}}(s)}$ and return $(1, \mathsf{proof})$.

Else, proceed as follows:

- Using the Extended Euclidean algorithm, compute polynomials $q_1[z], q_2[z]$ such that $q_1[z]\mathsf{Ch}_{\mathcal{X}}[z] + q_2[z]\mathsf{Ch}_{\{x\}}[z] = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q_1'[z] = q_1[z] + \gamma \cdot \mathsf{Ch}_{\{x\}}[z]$ and $q_2'[z] = q_2[z] - \gamma \cdot \mathsf{Ch}_{\mathcal{X}}[z]$.
- Set $W_1 := g^{q_1'(s) r^{-1}}, W_2 = g^{q_2'(s)}$ and $\mathsf{proof} := (W_1, W_2)$. Return $(0, \mathsf{proof})$.

Verification $(\mathsf{ACCEPT}/\mathsf{REJECT}) \leftarrow \mathsf{Verify}(\mathsf{acc}, x, b, \mathsf{proof}, vk)$

If $b = 1$ return ACCEPT if $e(\mathsf{acc}, g) = e(\mathsf{proof}, g^x \cdot g^s)$, REJECT otherwise. If $b = 0$ do the following:

- Parse proof as (W_1, W_2).
- Return ACCEPT if $e(W_1, \mathsf{acc}) e(W_2, g^x \cdot g^s) = e(g, g)$, REJECT otherwise.

Update $(\mathsf{acc}', ek', \mathsf{aux}') \leftarrow \mathsf{Update}(\mathsf{acc}, \mathcal{X}, x, sk, \mathsf{aux}, \mathsf{upd})$

Parse aux as (r, N). If $(\mathsf{upd} = 1 \wedge x \in \mathcal{X})$ or $(\mathsf{upd} = 0 \wedge x \notin \mathcal{X})$ output $\perp$ and halt. Choose $r' \xleftarrow{\$} \mathbb{Z}_p^*$. If $\mathsf{upd} = 1$:

- Compute $\mathsf{acc}' = \mathsf{acc}^{(s+x)r'}$.
- If $|\mathcal{X}| + 1 > N$, set $N = |\mathcal{X}| + 1$ and compute $\mathsf{ek}' = g^{s^N}$.

Else, compute $\mathsf{acc}' = \mathsf{acc}^{\frac{r'}{s+x}}$ and $ek' = \emptyset$. In both cases, set $\mathsf{aux}' := (r \cdot r', N)$ and return $(\mathsf{acc}', ek', \mathsf{aux}')$.

Witness Update $(\mathsf{upd}, \mathsf{proof}') \leftarrow \mathsf{Upd}(\mathsf{acc}, \mathsf{acc}', x, \mathsf{proof}, y, ek', \mathsf{aux}, \mathsf{aux}', \mathsf{upd})$

Parse $\mathsf{aux}, \mathsf{aux}'$ to get r, r'.

- If proof is a membership witness:
  If $\mathsf{upd} = 1$ output $(1, \mathsf{proof}' = (\mathsf{acc} \cdot \mathsf{proof}^{x-y})^{r'})$. Else, output $(0, \mathsf{proof}' = (\mathsf{acc}'^{-1} \cdot \mathsf{proof})^{\frac{r'}{(x-y)}})$.
- If proof is a non-membership witness:
  Let $\mathcal{X}'$ be the set produced after the execution of Update for element x (i.e., the currently accumulated set). Run
  $\mathsf{Query}(\mathsf{acc}', \mathcal{X}', y, ek', \mathsf{aux}')$ and return its output.

Figure 3.3: *Zero-knowledge Dynamic Universal Accumulator Construction*

		[131] \ This paper
Setup		$M+m$
Update	Owner	$1 \setminus L$
	Server	$1 \setminus L$
Subset	Query	$N\log^2 N \log\log N + m^{\varepsilon}\log m$
	Verify/Proof size	k
Instersection	Query	$N\log^2 N \log\log N + km^{\varepsilon}\log m$
	Verify/Proof size	$\rho + k$
Union	Query	$kN\log N + km^{\varepsilon}\log m$
	Verify/Proof size	$\rho + k$
Difference	Query	$N\log^2 N \log\log N + m^{\varepsilon}\log m$
	Verify/Proof size	ρ

Figure 3.4: *This table compares the access complexity of each operation with that of [131]. When only one value appears in the last column, it applies to both constructions. We note that the access complexity of Union Query was originally mistakenly reported as $O(N\log N)$ in [131].*
Notation: $m = |\mathbb{S}|$, $M = \sum_{i\in m}|\mathcal{X}_i|$, $n_j = |\mathcal{X}_j|$, $N = \sum_{j\in[i_1,i_k]} n_j$, k is the number of group elements in the query input (for the subset query it is the cardinality of a queried subset Δ and for the rest of the queries it is the number of set indices), ρ denotes the size of a query answer, and L is the number of sets touched by queries between updates u_{t-1} and u_t, and $0 < \varepsilon < 1$ is a constant chosen during setup. The reported complexities are asymptotic.

Chapter 4

Efficient and Verifiable Graph Encryption Schemes for Shortest Path Queries

4.1 Introduction

Structured encryption (STE) schemes encrypt data structures in such a way that they can be privately queried. An STE scheme is secure if it reveals no partial information about the structure and queries beyond a well-defined leakage profile. Graph encryption schemes are a special case of STE that encrypt graph data structures. Graph encryption schemes have been proposed for adjacency queries (given two vertices, return 1 if they are adjacent) [43], neighbor queries (given a vertex, return all its neighbors) [43] and approximate shortest distance queries [113].

In this work, we propose the first graph encryption scheme for single-pair shortest path (SPSP) queries; that is, given two vertices u and v in a graph $G = (V, E)$, return the shortest path between u and v.

Applications. Graph databases have received significant attention due to their ability to efficiently store and query large-scale graphs derived from social networks, collaborative networks, communication networks and geographic navigation systems. Examples from industry include Titan [9], Neo4j [6], OrientDB [7] and GraphDB [5]. Graph encryption schemes allow one to encrypt a graph database and query it privately. This is particularly useful in the context of cloud computing where the database is stored and managed by a third-party cloud provider (e.g., Amazon's graph database services [1]).

SPSP queries are one of the most basic query types in any graph-based application. For example, in road networks they return the shortest route between two locations; in social networks they return the shortest chain of contacts between two people; and in communication networks they return the shortest route between two networked devices.

4.1.1 Our Techniques and Contributions

The classical approach to answering SPSP queries is to use a single-source shortest path (SSSP) algorithm like Dijkstra's algorithm [59] for graphs with non-negative edge weights, or Bellman-Ford for graphs with negative edge weights. [1] Dijkstra and Bellman-Ford run in time $O(m+n\log n)$ and $O(n \cdot m)$, respectively, where $n = |V|$ and $m = |E|$.

This is too slow for applications that require sub-linear query time so in such scenarios, one must pre-process the graph in order to build a data structure that will answer SPSP queries efficiently. The best-known solutions for arbitrary graphs are based on computing the reachability matrix (i.e., the transitive closure) [49], the SP-matrix [49] and 2-hop labelings [46]. We summarize the performance of each approach in Table 4.1. These solutions provide different tradeoffs between setup time, space and query time. In particular, the setup time and query time of 2-hop labeling are worse than for the SP Matrix but the former produces more compact structures. For dense graphs, however, 2-hop labeling requires more space than the SP Matrix.

SP-matrix encryption. Our graph encryption schemes for SPSP queries are STE schemes for an SPSP data structure. In particular, we work with the SP-matrix due to its faster setup time and optimal query time. Furthermore, the SP-matrix is simple to implement. [2]

At a high level, the approach works as follows. The structure is an $n \times n$ matrix M that stores at location $M[i, j]$ the first vertex on the shortest path between vertices v_i and v_j. If no path exists, $\perp$ is stored. SPSP queries are answered using a recursive algorithm that works as follows. Given v_i and v_j, one looks up the first vertex on the shortest path $v_z := M[i, j]$ and recurses on the pair (v_z, v_j). The recursion ends when $\perp$ is returned.

Our goal is to design an encryption scheme that takes as input the SP-matrix of a graph and that handles SPSP queries over the encrypted matrix.

Recursion. One of the main challenges in designing such a scheme is handling recursive algorithms over encrypted structures. A-priori there are two ways we could handle recursion in STE. One could be to use an interactive query protocol where the client issues new tokens at every step of the recursion. Another approach could be to use chaining as introduced in [43]. The first approach would require the client to do work that is linear in the number of recursions, whereas the second approach would only work for a bounded number of recursions and would yield an encrypted structure that grows in the number of recursions.

We show how to encrypt SP-matrices and how to handle the recursion in SPSP queries non-interactively, for an unbounded number of recursive steps and with an encrypted structure that grows only in the size of the graph. In addition, our encrypted structure has the same asymptotic setup time, query time and space as a plaintext SP-matrix data structure. Our scheme makes black-box use of a dictionary encryption scheme which can be instantiated with a variety of constructions [43, 37].

Leakage. As any STE scheme, our construction leaks some information. The setup leakage includes the number of vertices n and the query leakage includes the search pattern (i.e., if and when a query is repeated),

[1] There is no known algorithm that solves the single-pair shortest path problem asymptotically faster than the single-source shortest path problem.

[2] We also note that other work on privacy-preserving graph analysis uses the SP-matrix including the two-party shortest path protocol for road networks of Wu et al. [158].

Structure	Setup time	Space	Query time
Transitive closure	$n^2 \log n + nm$	n^3	t
SP Matrix	$n^2 \log n + nm$	n^2	t
2-Hop Labeling [46]	$O(n^3\|TC\|)$ [91] $\tilde{O}(n\sqrt{m}\log n)$[3]		average[3]: $\tilde{O}(t\sqrt{m}\log n)$
			worst case[3]: $\tilde{O}(n\sqrt{m}\log n)$

Table 4.1: *$n = |V|$; $m = |E|$; t is the length of the shortest path between u and v; TC is edge transitive closure of G*

the length of the shortest path and what we refer to as the path intersection pattern (PIP). Given a sequence of SPSP queries $(q_1, \ldots, q_t)$, where $q_i = (u_i, v_i)$, the PIP of q_t reveals the the intersections of $p_t = SPSP(G, u_t, v_t)$ with previous shortest paths that have the same destination v_t. That is, the PIP reveals, $p_t \cap p_i$ i.e., the nodes in the intersection along with the rank of the nodes (on the corresponding paths) in the intersection, for all $1 \leq i \leq t-1$ such that $\{p_t \cap p_i\}_{v_i = v_t}$.

Experiments. To evaluate the efficiency of our graph encryption scheme we implemented it in Java. We used the RR2Lev dictionary encryption scheme provided in Clusion [11] as our building block. In our experiments, we used graph data sets obtained from [105, 74]. Setting up an encrypted graph database ran between 5.9 seconds (for a graph on 47 vertices) to 16.7 hours (for a graph on 10,876 vertices). As is standard in STE schemes, this expensive setup is a one-time cost. Searching the database is very fast in our scheme. Search time is proportional to the length of the shortest path and it is independent of the size or structure of the graph. The time to search for a length 1 path is always ∼0.8ms and the search time grows linearly with the length of the path, irrespective of the size of the graph.

4.2 Related work

STE was introduced by Chase and Kamara [43] as a generalization of searchable symmetric encryption (SSE) [147]. SSE schemes encrypt search structures such as inverted indexes or search trees [51, 43, 95, 38, 37, 94, 148, 79, 25, 26, 66]. In recent years, much of the work on SSE has focused on supporting more complex queries including boolean, range, substring, wildcard and phrase queries [38, 134, 93, 62].

Graph encryption was introduced in [43] as a special case of STE for graph structures. [43] showed how to design graph encryption schemes for adjacency neighbor and focused subgraph queries. Poh, Mohamed and Z' aba show how to encrypt conceptual graphs [137]. Meng, Kamara, Nissim and Kollios showed how to design schemes for approximate shortest distance queries [113]. Wang, Ren, Du, Li and Mohaisen design a scheme for exact shortest distance [155].

Privacy-preserving graph processing has been considered in other models. One example is the work of Wu et al. [158] which proposes a two-party protocol to compute shortest paths on road networks. While the protocol of [158] also uses the SP-matrix, their security goal is different from ours. In their protocol, the clients are using a cloud-based navigation system. The clients hold their location and destination while the

[3] All these complexities are assuming a conjecture by [46] which states that the largest size of a 2-hop cover for any m edge directed graph is $\tilde{O}(n\sqrt{m})$. The space complexity and worst case query time are otherwise $O(|H|\log m)$ where H is the size of the minimal 2-hop cover of the graph. The average query time becomes $O(\frac{|H|}{n}\log m)$.

cloud service provider has its proprietary routing information. The goal of their work is to protect privacy of both parties: the clients keep their location and destination private, and the service provider keeps its proprietary routing information private except for the routing information associated with the specific shortest path requested by the client. Moreover, their work applies to road networks exclusively. In contrast, our work is applicable to all graphs and is completely non-interactive. Other approaches for privacy-preserving graph analysis include structural anonymization [35], differential privacy techniques [61] and private information retrieval [72].

4.3 Preliminaries

In this section, we introduce the notations and the primitives that we will use. The security parameter is denoted λ. We will use the notation $\leftarrow_{\$}$ to denote that the input was picked uniformly at random from the input domain. We will use $|\cdot|$ to denote size of an object.

4.3.1 Abstract data types and data structures.

An *abstract data type* (ADT) is a collection of data objects along with a set of operations defined on those data objects. The operations associated with an abstract data type fall into two categories: immutable query operations and mutable update operations. Query operations return information about the data, but do not modify it whereas update operations modify the data. If an ADT supports only query operations it is static, otherwise it is dynamic. Examples of ADT includes sets, dictionaries, graphs etc. A *data structure* for type $\mathcal{T}$ is a concrete representation of type-$\mathcal{T}$ objects for a particular computational model and that supports the type's operations efficiently.

Graphs. A graph $\mathsf{G} = (V,E)$ consists of a set of vertices V and a set of edges E that connect vertices pairwise. Edges can be directed or undirected and may or may not have weights on them. Different kinds of query and update operations can be supported on graphs, but here we restrict ourselves to static graphs that handle SPSP queries. An SPSP query $\mathsf{SPath}(G,(u,v))$ takes a graph $G = (V,E)$ and two vertices $u,v \in V$ as input and returns a simple path $\mathsf{p}_{u,v}$, i.e, an ordered collection of nodes $(w_1,...,w_t)$ such that $(u,w_1),(w_1,w_2),\ldots,(w_t,v) \in E$. If no path exists from u to v in G, SPath returns $\perp$. A tree is a graph that is connected and has no cycle.

SP-matrix. As discussed in the Introduction, in this work we focus on the SP-matrix graph *structure* since it supports SPSP queries with optimal query time and has reasonable setup time and storage space. The SP-matrix (cf., [49]) is a classic graph data structure that has is used widely in the literature (e.g., see [143, 158]).

Given a graph $G = (V,E)$, an SP-matrix structure is constructed by executing an APSP algorithm between all pair of vertices (e.g., using [67]) and generating a $|V| \times |V|$ matrix M_G. In the following, we identify the rows and columns of M_G with vertices in V and refer to the item at location (v_i,v_j) to denote the item located at row v_i and column v_j. For each vertex pair (v_i,v_j) we store at location $\mathsf{M}_G[v_i,v_j]$ the first vertex on a shortest path from v_i to v_j and $\perp$ if no path exists. All the diagonal entries in M_G this matrix are set to $\perp$.

Given M_G, one can recover the shortest path between two vertices (v_i, v_j) as follows. Look up position (v_i, v_j) in M_G to recover either an item w. If $w \neq \perp$, then w the first vertex on the shortest path between v_i and v_j and repeat the procedure with position (w, v_j). The recursion stops when a $\perp$ is encountered. The query time is optimal, i.e., it is linear in the length of the shortest path.

Dictionaries. A dictionary is an ADT that consists of a collection of label/value pairs $\{(\mathsf{lab}_i, \mathsf{val}_i)\}_{i\in[1,n]}$ (where each label is unique) and which supports a put operation that inserts label/value pairs and a get operation that, given a label, returns the associated value. The dictionary ADT can be instantiated using various data structures including hash tables and balanced binary trees. In this work, we only make black-box use of dictionaries so we do not specify any particular instantiation and simply refer to a dictionary structure as DX. We denote by $\mathsf{val} := \mathsf{DX}[\mathsf{lab}]$ the get operation and by $\mathsf{DX}[\mathsf{lab}] := \mathsf{val}$ the put operation. If lab is not held in DX, $\mathsf{DX}[\mathsf{lab}]$ returns $\perp$. The size of a dictionary, $|\mathsf{DX}|$, is the number of label/value pairs it stores.

Symmetric encryption. A symmetric encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ scheme that has pseudorandom ciphertexts under chosen-plaintext attack or RCPA security. Informally, RCPA security states that an adversary cannot distinguish between an oracle returning encryptions of chosen messages from one returning a random string with length equal to a fresh ciphertext for the chosen message.

4.3.2 Structured Encryption

Structured encryption schemes encrypt data structures in such a way that they can be privately queried. In a client/server setting, STE can be used to securely outsource encrypted data while supporting sub-linear query operations. There are many variants of STE depending on whether the schemes support queries that are interactive, non-interactive, response-hiding or response-hiding. Interactive STE schemes have multi-round query operations whereas non-interactive schemes have single-round query operations where the client sends a single message (i.e., the token) to the server. Response-revealing schemes reveal to the server the response to a query whereas response-hiding schemes do not. Our main constructions are response-hiding so we recall the syntax and security definitions of this variant below.

Definition 1 (Interactive structured encryption)**.** *Let $\mathcal{T}$ be an ADT. An interactive response-hiding type-$\mathcal{T}$ structured encryption scheme $\Sigma^{\mathrm{RH}}_{\mathcal{T}} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Query}_{\boldsymbol{C},\boldsymbol{S}})$consists of two algorithms and one two-party protocols that that work as follows:*

$(\mathsf{key}, \mathsf{st}) \leftarrow \mathsf{Gen}(1^\lambda)$:

is a probabilistic algorithm that takes as input the security parameter λ and outputs a secret key key *and a state* st.

$(\mathsf{EDB}, \mathsf{st}) \leftarrow \mathsf{Encrypt}(\mathsf{key}, \mathsf{st}, \mathsf{DS})$:

is a probabilistic algorithm that takes as input a secret key key *and a type-$\mathcal{T}$ data structure* DS *and outputs an encrypted data structure* EDB *and updated state* st.

$((\mathsf{st}, m), \perp) \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}((\mathsf{key}, \mathsf{st}, q), (\mathsf{EDB}))$:

is a two-party protocol executed between a client and a server where the client inputs a secret key key*, a state* st *and a query q and the server inputs an encrypted data structure* EDB*. The client receives as output an updated state* st *and a response m while the server receives $\perp$.*

Note that in Definition 1 we depart slightly from the traditional STE syntax and define a key generation algorithm Gen *and an encryption algorithm* Encrypt *as opposed to a single* Setup *algorithm. We make this syntactic change because the setup procedures of our graph encryption constructions have a particular form that is easier to express with this syntax. More precisely, our constructions will require one to generate query tokens* before *producing the final encrypted structure. As such we must be able to generate the key before the encrypted structure is produced.*

The security definition for STE requires that the view of an adaptive adversary be efficiently simulatable given a well-specified leakage on the plaintext data structure and queries. In other words, the encrypted structure should reveal at most some given setup leakage $\mathcal{L}_{\mathsf{S}}$ and the encrypted structure and token should reveal at most some given query leakage $\mathcal{L}_{\mathsf{Q}}$.

Definition 2 (Adaptive security [43, 51])**.** *Let* $\Sigma_{\mathcal{T}} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Query}_{\boldsymbol{C},\boldsymbol{S}})$ *be a type*$-\mathcal{T}$ *structured encryption scheme and consider the following probabilistic experiments, where* $\mathcal{A}$ *is a stateful adversary,* Sim *is a stateful simulator and* $\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}$ *are stateful leakage algorithms,* $\lambda \geq 1$ *is a query capacity and* $z \in \{0,1\}^*$.

Real$_{\Sigma_{\mathcal{T}},\mathcal{A}}(1^\lambda)$**:** *Given* z *and* λ*, the adversary* $\mathcal{A}$ *chooses a type-*$\mathcal{T}$ *data structure* DS *and sends it to the challenger. The challenger generates* $(\mathsf{key}, \mathsf{st}) \leftarrow \mathsf{Gen}(1^\lambda)$*, computes* $(\mathsf{EDB}, \mathsf{st}) \leftarrow \mathsf{Encrypt}(\mathsf{key}, \mathsf{st}, \mathsf{DS})$ *and returns* EDB *to* $\mathcal{A}$*.* $\mathcal{A}$ *makes a polynomial number of adaptive queries* $(q_1, \ldots, q_p)$ *and for each* q_i *the challenger and* $\mathcal{A}$ *execute* $\mathsf{Query}_{\boldsymbol{C},\mathcal{A}}((\mathsf{key}, \mathsf{st}, q_i), \mathsf{EDB})$*. Finally* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

Ideal$_{\Sigma_{\mathcal{T}},\mathcal{A},\mathsf{Sim}}(1^\lambda)$**:** *Given* z *and* λ*, the adversary* $\mathcal{A}$ *chooses a type-*$\mathcal{T}$ *data structure* DS *and sends it to the challenger. Given* z *and* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ *from the challenger,* Sim *returns* EDB *to* $\mathcal{A}$*.* $\mathcal{A}$ *makes a polynomial number of adaptive queries* $(q_1, \ldots, q_p)$*. For each* q_i*, the challenger, the simulator and adversary execute* $\mathsf{Query}_{\mathsf{Sim},\mathcal{A}}((\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i)), \mathsf{EDB})$*. Finally* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

$\Sigma_{\mathcal{T}}$ *is adaptively* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$*-secure if there exists a ppt simulator* Sim *such that for all ppt adversaries* $\mathcal{A}$ *and for all* $\lambda \geq 1$*, for all* $z \in \{0,1\}^*$*, there exists a negligible function* $\nu(\cdot)$ *such that:*

$$|Pr[\textbf{\textit{Real}}_{\Sigma_{\mathcal{T}},\mathcal{A}}(1^\lambda) = 1] - Pr[\textbf{\textit{Ideal}}_{\Sigma_{\mathcal{T}},\mathcal{A},\mathsf{Sim}}(1^\lambda) = 1]| \leq \nu(\lambda)$$

Note that the security definition does not change based on whether a STE is RR or RH.

Remark. We note that even though the traditional Setup algorithm is split into key generation and encryption algorithms, Gen and Encrypt, respectively, Definition 2 is the same as the standard notion of security for STE. This holds simply because the challenger generates both the key and the encrypted structure EDB before returning the latter to the adversary. Interestingly, however, this suggests a possibly stronger notion of security for STE in which the adversary is allowed to receive the output of various functions on the key *before* the encrypted structure is produced. In particular, this allows the adversary to choose its data structure DS as a function of the information it learns about the key. Such a definition would be a leakage-resilient form of STE which would guarantee security even when the adversary could learn information about the key through various side-channel attacks.

We also recall the definition of the classical non-interactive STE [43]. The security definition can be easily derived from the interactive definition.

Definition 3. *[Non-interactive structured encryption] Let $\mathcal{T}$ be an ADT. A non-interactive response-revealing type-$\mathcal{T}$ structured encryption scheme $\Sigma_{\mathcal{T}}^{\mathrm{RR}} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Token}, \mathsf{Query})$ consists of four polynomial-time algorithms where* Gen *and* Encrypt *are as in Definition 1 and* Token *and* Query *work as follows:*
$\mathsf{tk} \leftarrow \mathsf{Token}(\mathsf{key}, q)$:
is a (possibly) probabilistic algorithm. On input the secret key key *and a query q it outputs a token* tk.
$m \leftarrow \mathsf{Query}(\mathsf{EDB}, \mathsf{tk})$:
is deterministic algorithm that takes as input an encrypted data structure EDB *and a token* tk. *It outputs a response m.*

4.4 Graph Encryption for SPSP Queries

We show how to construct a static adaptively-secure graph encryption scheme that supports SPSP queries. Our constructions has optimal query complexity and leaks the query pattern (i.e., if and when two queries are the same), the path intersection pattern (PIP) which, roughly speaking, reveals the intersections of the queried path with previously queried paths that end at the same vertex and the length of the path. The scheme makes black-box use of a static encrypted dictionary which can be instantiated from a variety of constructions (e.g., [51, 43, 37]).
Challenges. Our high-level goal is to encrypt an SP-matrix while supporting a SPSP queries. In particular, the construction should satisfy the following constraints:

- the encrypted graph should be roughly the same size of the SP-matrix, i.e., $O(n^2)$;
- the queries should be non-interactive, i.e., the client will send a token to the server and get back the entire encrypted path;
- the tokens should be small, i.e., independent of the size of the graph or the path;
- the query time should be optimal, i.e., $O(t)$ where t is the length of the path.

To achieve these properties, we have to encrypt the SP-matrix in such a way that we can support *recursive* operations on the encrypted structure. Specifically, each recursive step of the standard SPSP query algorithm should be executed by the server non-interactively and on the same encrypted structure. This clearly rules out using an interactive query protocol and standard chaining techniques where an encrypted structure is generated for each possible operation in a loop or recursion. Instead, we need a kind of recursive chaining where each step of the recursion queries the *same* structure.

Overview. Our construction is described in detail in Fig. 4.1. At a high-level, it works as follows. It makes use of a non-interactive response-revealing dictionary encryption scheme $\mathsf{DES} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Token}, \mathsf{Get})$[4]

[4] we use Get instead of Query, as per the convention of DES

and of a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$. The key generation algorithm simply generates a key K for SKE and a key K_{DX} for DES.

Given a graph $G = (V, E)$, the encryption algorithm constructs a dictionary-based representation of G's SP-matrix. Recall that the SP-matrix of a graph G is a $|V| \times |V|$ matrix with rows and columns indexed by the vertices $v \in V$ and such that cell $\mathsf{M}_G[v_i, v_j]$ holds the first vertex on the shortest path between v_i and v_j. Instead of representing G's SP-matrix as a two-dimensional array, we construct an SP-dictionary SPDX as follows. For every pair of vertices v_i, v_j such that $i \neq j$, if there is a path from v_i to v_j, we add the pair $((v_i, v_j), (w, v_j))$ to SPDX; that is, we set $\mathsf{SPDX}[(v_i, v_j)] := (w, v_j)$, where (v_i, w) is the first edge on the path. Note that, for vertex pairs that are are not connected and for self-pairs (i.e., of the form (v_i, v_i)) there is no entry in the dictionary. A recursive SPSP query on SPDX simply proceeds by looking up the next label (w, v_j) until no entry is found. The recursive step is possible since the current val acts as the next lab to query. At a first glance, it might seem wasteful to set $\mathsf{val} = (w, v_j)$ rather than to just $\mathsf{val} = w$ since the shortest path algorithm technically only needs to know the first hop w at each recursive step. We briefly mention here that the reason we add v_j is to improve security which we discuss (informally) below.

Given SPDX, we then generate a second dictionary SPDX' as follows. For every value (w, v_j) in SPDX, we generate a token $\mathsf{tk}_{(w,v_j)}$ using the Token algorithm from DES. For each $((v_i, v_j), (w, v_j))$ pair in SPDX, we then add a pair

$$\left((v_i, v_j), \left(\mathsf{tk}_{(w,v_j)}, \mathsf{ct}_{(w,v_j)}\right)\right)$$

to SPDX', where $\mathsf{ct}_{(w,v_j)} = \mathsf{SKE.Encrypt}(K, (w, v_j))$. Finally, we encrypt SPDX' using DES, resulting in $\mathsf{EDX} \leftarrow \mathsf{DES.Encrypt}(K_{\mathsf{DX}}, \mathsf{SPDX}')$ which is output as the encrypted graph EDB.

A token for an SPSP query (v_i, v_j) is simply a DES token for the pair (v_i, v_j); that is, $\mathsf{tk} \leftarrow \mathsf{DES.Token}(K_{\mathsf{DX}}, (v_i, v_j))$. Given an encrypted graph $\mathsf{EDB} = \mathsf{EDX}$ and a token tk, the server simply queries EDX with tk. This results in either $\perp$ or a pair consisting of a new token tk' and a encrypted vertex ct'. In the latter case, it queries EDX with tk' and so on. It stops when the query returns $\perp$ and sends all the ciphertexts to the client.

Now we can see the benefit of storing (w, v_j) as opposed to just w. Because all practical dictionary encryption schemes have deterministic tokens, if we were to store only w then all the tokens for w stored in EDX would be the same which, in turn, would allow the server to correlate the shortest paths across different queries. Suppose, for example that the shortest paths of two queries (v_1, v_2) and (v_1, v_3) intersect at vertex w. By storing tokens for (w, v_j) as opposed to just w, during the query operation the server will see tokens corresponding to (w, v_2) and (w, v_3) and, thus will not learn that the paths intersect at w. Note that the construction does leak information about the intersection of paths, but a lot less than what would be leaked without this approach. We describe the exact leakage of our scheme in Section 4.4.1.

Remark on circularity. One concern that might arise with our construction is that we use DES to encrypt SPDX' which stores tokens that were generated using the key for DES. Such a usage of the encrypted dictionary is supported by the security definition of STE and is different than the *leakage-resilient* setting discussed in Section 4.3. The main difference is that in the latter the choice of the structure can depend on (a function of) the secret key whereas in our case the choice of the structure cannot (since the tokens are generated after the input structure is chosen).

ComputeSPDX

Input:

Graph $G = (V, E)$. Let $V = \{v_1, \ldots v_n\}$

Computation:

- Initialize a dictionary SPDX.
  - Compute all pair shortest paths (APSP) for all pair of vertices in V. Let us denote this collection of paths as $\mathbb{P}$. Note that $\mathbb{P}$ has exactly one shortest path between each pair of vertices.
  - For each $(v_i, v_j) \in V$, $i \neq j$ do the following:
    - Find the path $\mathsf{p}_{v_i,v_j} \in \mathbb{P}$. If $\mathsf{p}_{v_i,v_j} \neq \perp$, let (v_i, w) be the first edge on the path.
  - set $\mathsf{SPDX}[(v_i, v_j)] := (w, v_j)$

Output:

Output SPDX.

Efficiency.

We recall that standard dictionary encryption schemes (e.g., [37]) have encryption algorithms that are $O(|\mathsf{DX}|)$ and token generation and get algorithms that are $O(1)$.

Using any such construction, our scheme's encryption algorithm $\mathsf{Encrypt}$ is $O(n^3)$ due to the cost of computing the APSP [5] in the construction of SPDX. The token generation algorithm Token is $O(1)$ and the query and reveal algorithms, Search and Reveal are $O(t)$ where t is the length of the shortest path (which is optimal). The size of the encrypted graph is $O(n^2)$.

4.4.1 Security

In this section, we prove that our construction is adaptively-secure with respect to a well-defined leakage profile. It is not clear what this leakage implies in practice but we do not know of any leakage attacks against the leakage profiles of our constructions. The design of such attacks, however, would help improve our understanding of the practical implications of this leakage.

Setup leakage. The setup leakage $\mathcal{L}_\mathsf{S}$ of our construction is the number of vertex pairs in G that are connected by a path. Formally, we define this is as

$$\mathcal{L}_\mathsf{S}(G) = |\{(u, v) \in V \times V : \mathsf{SPath}(u, v) \neq \perp\}|.$$

Query leakage. The query leakage $\mathcal{L}_\mathsf{Q}$ of our construction is

$$\mathcal{L}_\mathsf{Q}(G, (u, v)) = (\mathsf{QP}, \mathsf{PIP}, t)$$

[5] For graphs with non-negative edge weights, APSP can be computed in $O(n^2 \log n + nm)$.

GES Implementation

$\underline{\mathsf{key} \leftarrow \mathsf{Gen}(1^\lambda)}$

1. $K_{\mathsf{DX}} \leftarrow \mathsf{DES.Gen}(1^\lambda)$.
2. $K \leftarrow \mathsf{SKE.Gen}(1^\lambda)$.
3. Set $\mathsf{key} = (K, K_{\mathsf{DX}})$ and output key

$\underline{\mathsf{EDB} \leftarrow \mathsf{Encrypt}(\mathsf{key}, G)}$

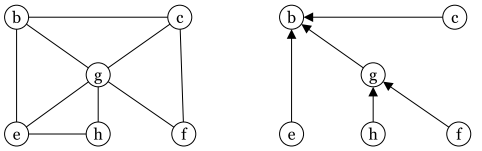
1. $\mathsf{SPDX} \leftarrow \mathit{ComputeSPDX}(G)$.
2. Initialize a dictionary SPDX'.
3. For each $(\mathsf{lab}, \mathsf{val})$ in SPDX, do the following:
   (a) $\mathsf{tk}_{\mathsf{val}} \leftarrow \mathsf{DES.Token}(K_{\mathsf{DX}}, \mathsf{val})$.
   (b) set $\mathsf{SPDX}'[\mathsf{lab}] := \big(\mathsf{tk}_{\mathsf{val}}, \mathsf{ct}\big)$, where $\mathsf{ct} = \mathsf{SKE.Encrypt}(K, \mathsf{val})$.
4. $\mathsf{EDX} \leftarrow \mathsf{DES.Encrypt}(K_{\mathsf{DX}}, \mathsf{SPDX}_s)$.
5. Output $\mathsf{EDB} = \mathsf{EDX}$.

$\underline{(m, \perp) \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}((\mathsf{key}, q), \mathsf{EDB})}$

This protocol is non-interactive in this instantiation and consists of the following steps.

1. $\mathbf{C}$: $\underline{\mathsf{tk} \leftarrow \mathsf{Token}(\mathsf{key}, q)}$
   (a) Parse q as (u, v)
   (b) Parse key as (K, K_{DX}).
   (c) Set $\mathsf{lab} = (u, v)$.
   (d) $\mathsf{tk} \leftarrow \mathsf{DES.Token}(K_{\mathsf{DX}}, \mathsf{lab})$.
   (e) Return tk.
2. $\mathbf{S}$: $\underline{\mathsf{resp} \leftarrow \mathsf{Search}(\mathsf{EDB}, \mathsf{tk})}$
   (a) parse EDB as EDX.
   (b) Initialize a variable status $:= \varepsilon$ and a variable resp $:= \varepsilon$.
   (c) Set variable curr $:=$ tk.
   (d) Until status $=$ "*SearchEnd*" do the following:
      i. $a \leftarrow \mathsf{DES.Get}(\mathsf{EDX}, \mathsf{curr})$.
      ii. If $a = \perp$, set status $:=$ "*SearchEnd*".
      iii. Else parse a as (τ, c).
      iv. Set resp $:=$ resp $\circ\, c$ and curr $:= \tau$.
      v. Go back to Step 2d.
   (e) Output resp.
3. $\mathbf{C}$: $m \leftarrow \mathsf{Reveal}(\mathsf{key}, \mathsf{resp})$
   (a) Parse st as (K, K_{DX}).
   (b) Set variable $m := \varepsilon$.
   (c) Parse resp as $c_1 \circ c_2 \circ \ldots \circ c_k$.
   (d) For $i = 1, \ldots, k$, do the following:
      i. Compute $m_i \leftarrow \mathsf{Decrypt}_K(c_i)$
      ii. Set $m := m \circ m_i$.
   (e) Output m.

Figure 4.1: *A graph encryption scheme for SPSP queries.*



(a) *Example graph* **(b)** *All paths to vertex b*

Figure 4.2: *Illustrative example*

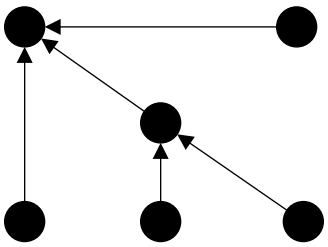


Figure 4.3: *Leakage profile. All the incoming shortest paths to vertex b form a tree. The server learns the structure of this tree in the construction of* GES *(Section 4.4), if it has seen tokens for* all the paths to *b.*

where QP is the query pattern, PIP is the path intersection pattern and t is the length of the shortest path between u and v. The query pattern QP reveals if and when a query has been made in the past. Formally, we define it as

$$\mathsf{QP}(G,(u,v)) = M$$

where M is a binary matrix with a 1 at location (i,j) if the ith query was the same as the jth query. The path intersection pattern PIP reveals the following. Let $p_{u,v}$ be the shortest path between vertices u and v. For all previously queried vertex pairs (u_i,v_i), if $v_i = v$, then the PIP reveals the ranked intersection $p_{u_i,v_i} \cap_r p_{u,v}$. We define ranked intersection as the nodes on the intersection of two paths, along with the rank of the nodes on the corresponding paths. More precisely,

$$\mathsf{PIP}(G,(u,v)) = \left(p_{u_i,v_i} \bigcap\nolimits_r p_{u,v} \right)_{(u_i,v_i):v_i=v}$$

More intuitively, for any vertex b, all the incoming shortest paths to vertex b form a tree. In the worst-case (i.e., if the server has tokens for all paths to b) the server learns the structure of this tree. We illustrate this on a concrete example in Figure 4.2. In this case, the leakage structure that the server learns is Figure 4.3a.

Overview. To prove the security of our construction we will need to describe a simulator $\mathsf{Sim}_{\mathsf{GES}}$ that, given the setup and query leakages, can simulate the encrypted graph and appropriate tokens. Because we make use of an adaptively-secure dictionary encryption scheme, our simulator can make use of a simulator $\mathsf{Sim}_{\mathsf{DES}}$ for the underlying encrypted dictionary.

The simulation of the encrypted graph is relatively straightforward and will consist of using $\mathsf{Sim}_{\mathsf{DES}}$ to simulate an encrypted dictionary EDX. Note that for this to work, the only requirement is for $\mathsf{Sim}_{\mathsf{GES}}$ to provide the appropriate setup leakage to $\mathsf{Sim}_{\mathsf{DES}}$. The more challenging part of the proof is the simulation of

the tokens. For some query, let t be the length of its shortest path. Recall that for this query the simulator will receive as input the query pattern QP, the path intersection pattern PIP and t and that from this information it has to generate a token tk for the query. The main challenge is that, tk has to be generated in such a way that if the adversary executes the Search algorithm on the simulated encrypted graph and the simulated token, the number of recursive steps has to stop exactly after t steps. Because the simulator does not know t when it simulates the encrypted graph it cannot design the simulated encrypted graph to stop answering queries after t steps.

Security of DES. As discussed above, our simulator $\mathsf{Sim}_{\mathsf{GES}}$ relies on a simulator for DES which is guaranteed to exist by its adaptive security. More precisely, if we instantiate DES with any standard dictionary encryption scheme, then its leakage profile will be as follows. The setup leakage is

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{des}}(\mathsf{DX}) = |\mathsf{DX}|$$

and the query leakage is

$$\mathcal{L}_{\mathsf{Q}}^{\mathsf{des}}(\mathsf{DX}, \mathsf{lab}) = \mathsf{QP}^{\mathsf{des}}$$

where $\mathsf{QP}^{\mathsf{des}}$ is the query pattern which reveals if and when the label lab was queried in the past. More formally we define it as

$$\mathsf{QP}^{\mathsf{des}}(\mathsf{DX}, \mathsf{lab}) = M$$

where M is a binary matrix with a 1 at location (i, j) if the ith label queried was the same as the jth label queried. Note that because DES is response-revealing we do not consider the response it outputs when queried as leakage. We summarize the security of our scheme in Theorem 2.

Theorem 2. *Our graph encryption scheme* GES *is* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$*-secure according to Definition 2. Let n be the number of vertices and m be the number of edges of the graph. We have that* $\mathsf{Encrypt}$ *runs is time* $O(n^3)$ *and generates an encrypted graph of size* $O(n^2)$*;* Token *runs is time* $O(1)$ *and produces a token of size* $O(1)$*; for a path of length t,* Search *and* Reveal *run each in time* $O(t)$ *and the size of the encrypted path is* $O(t)$*.*

Proof. We give a stateful simulator $\mathsf{Sim}_{\mathsf{GES}}$ that simulates the view of the adversary in the Ideal experiment of Definition 2. The focus of the simulation is simulating the leakage profile of a DES, given the leakage of GES.

$\underline{\mathsf{Encrypt}}$**:** On receiving output N from leakage function $\mathcal{L}_{\mathsf{S}}(\mathsf{G})$, $\mathsf{Sim}_{\mathsf{GES}}$ invokes $\mathsf{Sim}_{\mathsf{DES}}$ with input N. $\mathsf{Sim}_{\mathsf{DES}}$ outputs EDX which $\mathsf{Sim}_{\mathsf{GES}}$ forwards to $\mathcal{A}$ as EDB.

$\underline{\mathsf{Token}}$**:** $\mathsf{Sim}_{\mathsf{GES}}$ initializes an empty list L, to keep track of the queries generated by the adversary. Note that, $\mathsf{Sim}_{\mathsf{GES}}$ will never repeat a query for $\mathsf{Sim}_{\mathsf{DES}}$, so the query pattern $\mathsf{QP}^{\mathsf{des}}$ for $\mathsf{Sim}_{\mathsf{DES}}$ for each query that it receives from $\mathsf{Sim}_{\mathsf{GES}}$ is $\emptyset$. $\mathsf{Sim}_{\mathsf{GES}}$ can receive two kinds of leakage depending on the query q. We show $\mathsf{Sim}_{\mathsf{GES}}$ simulates the input to $\mathsf{Sim}_{\mathsf{DES}}$ in each case.

The adversary asks queries $q_1, \ldots, q_{\mathsf{poly}(\lambda)}$. Let $\lambda_1 = \mathsf{poly}(\lambda)$ and $\lambda_2 = \mathsf{poly}(\lambda)$ be the range sizes for $\mathsf{DES.Token}$ and $\mathsf{SKE.Encrypt}$ respectively.

$\underline{\mathsf{Sim}_{\mathsf{GES}} \text{ receives } k \text{ on query } q_j}$: This case reflects that the simulator has not seen this query token before and it only received the length of the path k as leakage. $\mathsf{Sim}_{\mathsf{GES}}$ does the following:

1. Initilize empty variables $\rho_1, \rho_2, \mathsf{val}, \mathsf{p}$.
2. Pick $c \leftarrow_\$ \{0,1\}^{\lambda_1}, d \leftarrow_\$ \{0,1\}^{\lambda_2}$.
3. Set $\rho_1 := c, \rho_2 := d$.
4. For $i = k, \ldots, 1$, do the following:
   (a) Set $\mathsf{val} = (\rho_1, \rho_2)$.
   (b) Invoke $\mathsf{Sim}_{\mathsf{DES}}$ with val. Let $\mathsf{Sim}_{\mathsf{DES}}$ return tk.
   (c) Set $\mathsf{p} = \mathsf{p} \circ (\mathsf{tk}, \mathsf{val})$.
   (d) Set $\rho_1 := \mathsf{tk}, \rho_2 \leftarrow_\$ \{0,1\}^{\lambda_2}$ and go back to step 4.
5. Let $\mathsf{p} = (\mathsf{tk}_k, \mathsf{val}_k) \circ (\mathsf{tk}_{k-1}, val_{k-1} \ldots \circ (\mathsf{tk}_1, \mathsf{val}_1)$
6. Set $\mathsf{p}' := (\mathsf{tk}_1, \mathsf{val}_1) \circ (\mathsf{tk}_2, \mathsf{val}_2) \circ (\mathsf{tk}_k, \mathsf{val}_k)$.
7. Insert (j, p') to list L and output tk_1.

$\mathsf{Sim}_{\mathsf{GES}}$ receives a list of tuples: This case reflects that the simulator has seen the query token before. Let q_j be the current query, p_j be the corresponding shortest path in the graph, j' be the highest index in L such that the $\mathsf{PIP}(G,(u,v)) \geq 1$. Let $|p_j| = k_1, |p'_j| = k_2$. For each $(k_1, j' < j, k_2)$, depending on whether $k_1 <, >, = k_2$, $\mathsf{Sim}_{\mathsf{GES}}$ does the following:

$k_1 < k_2$: In this case, $\mathsf{Sim}_{\mathsf{GES}}$ retrieves $(j', \mathsf{p}_{j'})$ from L. It retrieves the last k_1 tokens on $\mathsf{p}_{j'}$. Let the k_1^{th} token value pair be $(\mathsf{tk}_1, \mathsf{val}_1)$, k_2^{th} pair be $(\mathsf{tk}_2, \mathsf{val}_2)$ and so on and the last pair be $(\mathsf{tk}_{k-1}, \mathsf{val}_{k-1})$. Then, $\mathsf{Sim}_{\mathsf{GES}}$ sets $\mathsf{p}_j := (\mathsf{tk}_1, \mathsf{val}_1) \circ (\mathsf{tk}_2, \mathsf{val}_2) \ldots \circ (\mathsf{tk}_{k-1}, \mathsf{val}_{k-1})$, inserts (j, p_j) to L and outputs tk_1.

$k_1 = k_2$: In this case, $\mathsf{Sim}_{\mathsf{GES}}$ retrieves $(j', \mathsf{p}_{j'})$ from L and sets $\mathsf{p}_j := \mathsf{p}_{j'}$. Let tk_1 be the first token on p_j. $\mathsf{Sim}_{\mathsf{GES}}$ inserts (j, p_j) to L and outputs tk_1.

$k_1 > k_2$: $\mathsf{Sim}_{\mathsf{GES}}$ does the following:

1. Initialize empty variables $\rho_1, \rho_2, \mathsf{val}, \mathsf{p}$.
2. Set variable $\mathsf{length} = k_2 - k_1$.
3. Retrieve $(j', \mathsf{p}_{j'})$ from L.
4. Let $(\mathsf{tk}_{k_2}, \mathsf{val}_{k-2})$ be the $k_2{}^{th}$ token value pair from the end on $\mathsf{p}_{j'}$.
5. Set $\rho_1 := \mathsf{tk}_{k_2}, \rho_2 := \mathsf{val}_{k_2}$.
6. For $i = \mathsf{length}, \ldots, 1$, do the following:
   (a) Set $\mathsf{val} = (\rho_1, \rho_2)$.
   (b) Invoke $\mathsf{Sim}_{\mathsf{DES}}$ with val. Let $\mathsf{Sim}_{\mathsf{DES}}$ return tk.
   (c) Set $\mathsf{p} = \mathsf{p} \circ (\mathsf{tk}, \mathsf{val})$.
   (d) Set $\rho_1 := \mathsf{tk}, \rho_2 \leftarrow_\$ \{0,1\}^{\lambda_2}$ and go back to step 6.
7. Let $\mathsf{p} = (\mathsf{tk}_{\mathsf{length}}, \mathsf{val}_{\mathsf{length}}) \ldots \circ (\mathsf{tk}_1, \mathsf{val}_1)$
8. Set $\mathsf{p}' := (\mathsf{tk}_1, \mathsf{val}_1) \circ (\mathsf{tk}_2, \mathsf{val}_2) \circ (\mathsf{tk}_{\mathsf{length}}, \mathsf{val}_{\mathsf{length}}) \circ \mathsf{p}''$ where p'' contains the last $k_2 - 1$ token value pairs from $\mathsf{p}_{j'}$ (in that order).

9. Insert (j, p') to list L and output tk_1.

This concludes the description of the simulator. Next we give the game hops for the hybrid argument.

Hyb 0: This the identical to the real challenger.

Hyb 1: Same as the previous Hyb, except the following: every ciphertext c is replaced with $\mathsf{Encrypt}_K(0)$. We will actually need N hybrids here, to change one ciphertext at a time. We combine them into one Hyb for succinctness.

An adversary that can distinguish between these two, will break the RCPA security of the underlying encryption scheme with non-negligible advantage.

Hyb 2: Same as the previous Hyb except the following: instead of invoking the real algorithm $\mathsf{DES.Gen}, \mathsf{DES.Setup}, \mathsf{DES.Token}$, this game invokes the simulator $\mathsf{Sim}_{\mathsf{DES}}$ and simulates its leakage functions $\mathcal{L}_{\mathsf{S}}^{\mathsf{des}}, \mathcal{L}_{\mathsf{Q}}^{\mathsf{des}}$.

An adversary that can distinguish between these two, will break the adaptive security of the underlying DES with non-negligible advantage.

Note that the distribution Hyb 2 identical to that of $\mathsf{Sim}_{\mathsf{GES}}$. This concludes the proof of security. □

4.5 Achieving security in the malicious model

Achieving security against malicious servers has been considered in the setting of SSE [148, 104, 26], but has not been addressed in the general setting of STE. The main challenge in supporting malicious servers is to detect when a server is returning incorrect answers to the client. This is particularly challenging in the case of an empty answer. This was precisely the case that the verifiable SSE in [148] could not handle (and was later addressed in [26]).

However, in our GES construction, we can support verifiability (even when the path is empty) without adding any significant overhead to the server or the client. We extend our GES construction to achieve this as follows:

1. While generating SPDX, for each unreachable and self pair of vertices (u, v) add entry $(\mathsf{lab} = (u, v), \mathsf{val} = \perp)$. Note that this step does not add any additional overhead asymptotically and it does not affect the stopping condition of the server. The server only has to do one more step in the recursion.

2. Client processes this SPDX ad before and generates the corresponding EDB. After that, the client stores *message authentication code* (MAC) for each $(\mathsf{lab}, \mathsf{val})$ pair of EDB. To distinguish it from plain dictionary entries, let us denote these as $(\mathsf{lab}^e, \mathsf{val}^e)$

3. The server returns the $(\mathsf{lab}^e, \mathsf{val}^e)$ pairs in the search result resp along with its MAC.

4. The client verifies the correctness of the returned resp by computing $\mathsf{lab}^e \in \mathsf{resp}$ from the corresponding tk and then verifying MAC on each of the returned $(\mathsf{lab}^e, \mathsf{val}^e)$ pairs.

Table 4.2: *Data Sets*

Name	n	m	Source	Extracted from
p2p-Gnutella30	36,682	88,328	[105]	–
p2p-Gnutella25	22,687	54,705	[105]	–
p2p-Gnutella04	10,876	39,994	[105]	–
p2p-Gnutella08	6,301	20,777	[105]	–
email-Eu-core	986	24929	[105]	–
Facebook1	424	23397	[105]	facebook-combined [105]
InternetRouting	78	1174	[74]	ir [74]
Facebook2	162	11018	[105]	facebook-combined [105]
CA-GrQc	47	947	[105]	CA-GrQc [105]

4.6 Experiments

In this section we discuss the experimental evaluation of our graph encryption schemes. We have evaluated our protocol GES on real world networks obtained from [105, 74]. This section is organized as follows. First we describe the system specifications, cryptographic primitives and the datasets we used for our implementation. Then we describe the performance of each of the algorithms separately. Finally we simulate the communication latency for our $\mathsf{GES}_{\mathsf{RL}}$ protocol.

4.6.1 System specifications, Primitives and Data Sets

We implemented our scheme in Java and ran our experiments on Amazon's EC2 cloud server with Microsoft Windows Server 2012 R2 Base OS the following system specifications: 208 ECUs, 64 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 488 GiB memory, EBS only.

We build our scheme on the RR2Lev implementation of DES scheme from [37] provided in Clusion [11]. In the DES implementation, three cryptographic primitives are used. 1) Pseudo Random Function (PRF): CMAC-AES with 16B output. 2) A symmetric Encryption Scheme: AES-CTR with 16B key. 3) Collision Resistant Hash Function: SHA-256.

4.6.2 Graph Dataset

Our data set consists of 9 graph data sets obtained from [105, 74]. In Table 4.2 we summarize the number of vertices (n) and number of edges (m) of each graph, source of the graph and if further processing was done. Four of our experimental data sets were extracted from larger graphs using the library [84] that provides an implementation of the dense subgraph extraction algorithm of Charikar *et al.* [41]. For the rest of this section, we mark the name of the graph with an asterisk if the results obtained for that graph was extrapolated from the results obtained from the experiments.

4.6.3 Setup

In this section, we measure the cost of computing the encrypted the graph database EDB. The costs reported are averaged over 2 runs. We do not report any variance since it is very low.

Table 4.3: *Storage overhead over the graph. Let n = number of vertices, m = number of edges of the graph,* EDB *be encrypted graph and* SPDX *be the plaintext shortest path dictionary. Density of the graph = m/n^2, overhead over graph (O-G) = (size of* EDB *- size of* SPDX*)/size of* SPDX *overhead over* SPDX *(O-SPDX)= (size of* EDB *- size of* SPDX*)/size of* SPDX

Name	n	Density	O-G	O-SPDX
CA-GrQc	47	0.43	19	3.8
InternetRouting	78	0.20	64.7	4
Facebook2	162	0.42	20.5	4
Facebook1	424	0.13	72	4.2
email-Eu-core	986	0.02	395.6	5.4
p2p-Gnutella08	6,301	0.0005	5856.1	4.2
p2p-Gnutella04	10,876	0.0003	10021.6	4
p2p-Gnutella25*	22,687	0.0001	25282.6*	4.1*
p2p-Gnutella30*	36,682	0.00006	38593.4*	4.1*

Storage overhead: The only significant cost that our scheme incurs, is in terms of storage overhead at the server. Note that the client only needs to store the PRF key and the encryption key, i.e, only 32B in total. The space overhead of our scheme is proportional to the number of reachable pair of nodes in the graph. On a graph of n nodes and m edges, if the graph is fully connected, i.e., if there exists a path between every pair of nodes in the graph, our scheme will store an encrypted dictionary of size $O(n^2)$. For dense graphs, the graph size if $\Omega(n^2)$, hence the storage overhead is asymptotically constant. However, for a sparse graph that is fully connected, (an undirected tree, for example), the storage overhead in $O(n)$.

We experimentally verify this claim. In Table 4.3 we show how the density correlates with the EDB storage overhead. In the overhead calculation, we measure the sizes of the graphs (stored in the SNAP format [105]), the shortest path dictionaries (SPDX) and the encrypted graphs (EDB) as space required to store on disk. We calculate the overhead as (size of EDB - size of Graph)/size of Graph. The density, is calculated as m/n^2. As expected, we see high density results in low overhead whereas low density increases the overhead. The size of the SPDX of a graph is proportional to the total number of reachable pair of nodes in the graph, as in the size of EDB. As we expect, we see that this storage overhead is very low, namely, 4.15 on average. This overhead is computed as (size of EDB - size of SPDX)/size of SPDX.

Setup time and space: Next, we compute the size of shortest path dictionary SPDX, the size of encrypted graph database EDB and the time to compute EDB, given SPDX. We report the cost in Table 4.4. Note that, computing SPDX, given a graph G is a classic parallelizable graph algorithm and have been widely used and optimized [136, 135, 159, 144]. This cost is negligibly small compared to the cost of computing EDB, so we do not report that cost here. We extrapolate the costs for p2p-Gnutella25 and p2p-Gnutella30 from our experimental results, (we mark those with an asterix to denote that those are extrapolated numbers).

4.6.4 Search

Search token generation: The size of a search token is constant, i.e, it is always 32B. The time to generate this token is negligibly small ($\sim 100\mu s$). This token generation is carried out at the client. Next we evaluate

Table 4.4: *Setup costs. n is the number of vertices on the graph,* SPDX *is the shortest path dictionary and* EDB *is the encrypted graph database.*

Name	n	SPDX size	EDB size	EDB setup time
CA-GrQc	47	43KB	208 KB	5.9s
InternetRouting	78	76.1 KB	555KB	10.12s
Facebook2	162	478KB	2.38MB	35.2s
Facebook1	424	3.31 MB	17.1 MB	3.2m
email-Eu-core	986	11.7MB	74.9 MB	14.2 m
p2p-Gnutella08	6,301	237MB	1.23 GB	3.5 h
p2p-Gnutella04	10,876	885 MB	4.43GB	16.7h
p2p-Gnutella25*	22,687	3.17MB*	16.38GB*	41.8h*
p2p-Gnutella30*	36,682	7.56GB*	38.84GB*	93.5h*

the the following: (1) time to search at the server (2) size of the returned answer (i.e., encrypted path) (3) time to decrypt the answer using Reveal (at the client).

We generate the search queries as follows. For each graph, we generate 1000 random source destination pairs to search. For each pair searched, we repeat the experiment 100 times and report the average cost. Finally, we aggregate all the searches that returned answers of same length and report the average time and size of the answer.

We report the answer size (in Bytes) in Figure 4.5 and the search and decryption time (in millisec) in Figure 4.4. As expected, we see that the size of the answer, time to generate the answer at the server and the time to decrypt the answer using Reveal, all are proportional to the length of the returned path and that alone. In other words, none of the graph parameters, like density or size of a graph affect these numbers. Our experimental evaluation supports the asymptotic costs reported in Section 4.4. We describe the color encoding used for the plots in Figure 4.6.

4.7 Conclusion

In this work, we developed a scheme and built an initial prototype for efficiently answering shortest path queries on an encrypted graphs. Our scheme is the first structured encryption scheme that supports a recursive algorithm, where the number of recursion steps is not needed at the setup time (unlike the chaining technique from [Chase and Kamara, 2010]). It will be interesting to see if our technique can be extended to develop other practical encrypted structures for recursive algorithms.

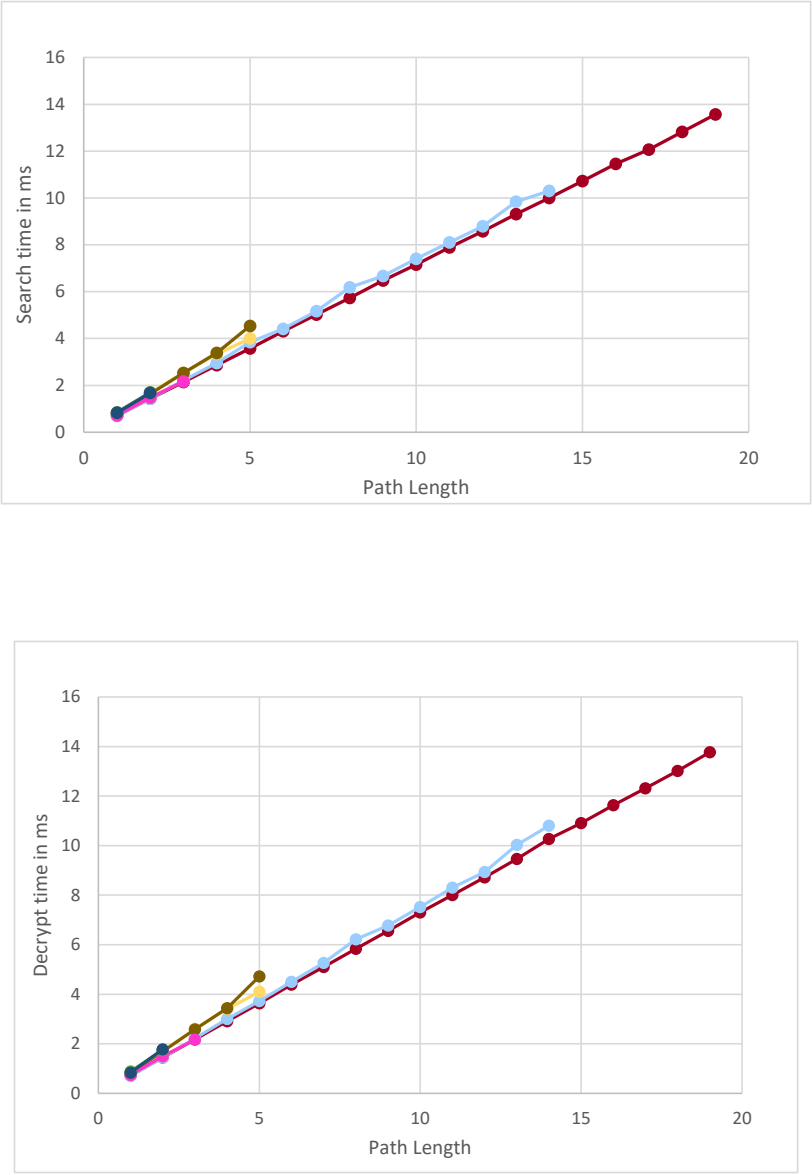


Figure 4.4: *Search and Decryption time in ms*

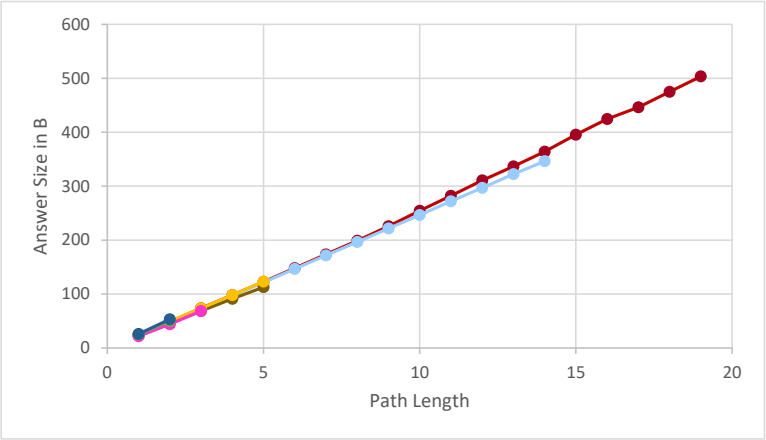


Figure 4.5: *Answer size in Bytes*

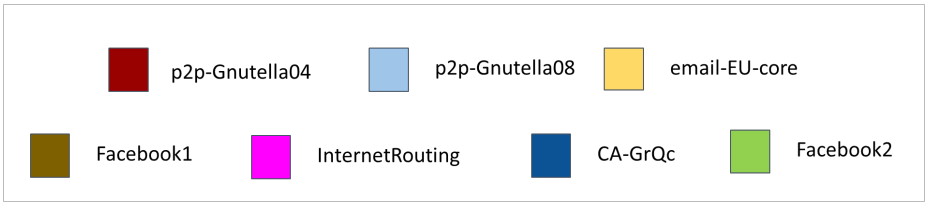


Figure 4.6: *Color encoding used in the plots*

Chapter 5

Efficient and Verifiable Reachability Queries on General Graphs in Zero-Knowledge

5.1 Introduction

Graph databases are designed to efficiently store and query large-scale graphs derived from social networks,collaborative networks, communication networks and geographic navigation systems. Due to its vast applications, graph databases have received significant attention both from the industry and the academic community. Titan [9], Neo4j [6], OrientDB [7] and GraphDB [5] are some of the commercial graph databases. In graph databases, reachability is a fundamental query that asks if there is a path between two vertices u, v in a graph. Due to its wide applications, reachability queries have been actively studied for a long time [56, 97, 98].

In the context of cloud services, there has been some attempt to look at verifiable graph queries [162, 83], but not much work has been done in answering recahability queries on both *static and dynamic graphs* while maintaining privacy. More specifically, we are looking at a three party setting where a (trusted) data owner outsources a large graph to a server, who will then answer reachability queries about the graph made by various clients. The goal is to ensure verifiability of the returned results, thus protecting clients against malicious behavior by the server, or server compromise. Moreover, we require the proofs returned by the server should be privacy-preserving, i.e., they should not leak any information beyond what can be inferred from the query answers. The privacy property is crucial in the context of cloud security. Consider the case of cloud VM isolation, for example. While cloud services have significantly reduced the cost of computations for organizations, many are still hesitant to adopt them due to security and privacy concerns. Recent works such as the Spectre [100] and Meltdown [110] show how side-channel attacks may compromise co-located cloud VMs. In such a situation, accountability becomes even more important for cloud service providers. There are

well known ways to prevent attacks due to colocation on the cloud, in particular, *security groups*[21, 156], if implemented correctly, can help prevent information flow to VMs with high security requirement. Note that security groups can be represented as directed graphs where reachability of node u from node v represents a possibility for information flow from the entity represented by u to the entity represented by v. This motivates the scenario where an organization can obtain proof of sufficient network isolation from different groups using zero-knowledge authenticated graph reachability queries.

5.1.1 Related Work

There is a large body of wok that focuses on general-purpose verifiable computation. But here, we discuss the work that is most closely related to us. Namely, we disuss the work of verifible queries on outsourced graph databases. In ALITHEIA [162] the authors build schemes where a server can verifiably compute shortest-path, longest-path, and maximum-flow queries over general graphs. Their technique builds on top of a generic SNARK based technique and their verification cost depends on the size of the graph. Moreover, they do not provide any privacy. In [160], the authors provide a verifiable protocol for shortest-path queries. Their protocols have worst-case proof size linear in the number of the edges of the graph and they do not provide any privacy guarantee. In [83], the authors build protocols for various connectivity queries on outsourced graphs, namely, connectivity, bi-connectivity and tri-connectivity. But they do not provide any privacy either. In contrast to these works, our construction is less general (we handle reachability queries only), but more efficient (verification cost is independent of the size of the graph) and comes with strong privacy guarantee, namely, zero-knowledge privacy. There has been a line of work on providing zero-knowledge privacy for queries on outsourced datastructures [153, 150, 152, 76, 123], but none of these handled graph reachability queries.

5.1.2 Our Contribution

In this work, we design schemes for answering graph reachability queries in zero knowledge on both static and dynamic graphs. In case of dynamic graphs, we additionally require that the updates on the graphs do not leak any information about the graph beyond what can be inferred from queries. This property, in particular, implies that the proofs generated by the server should be ephemeral, i.e., the same proof cannot be used before and after an update. The design of our static scheme uses a set theoretic characterization of reachability queries due to [47]. We extend techniques from [152] to implement the zero-knowledge version of the set theoretic characterization of reachability on static graphs. The scheme for dynamic graphs uses traditional cryptographic commitments and Sigma protocols for proving simple statements of equality (and disjunction).

The rest of the chapter is organized as follows. In Section 5.2, we recall some cryptographic primitives, hardness assumptions and the Zero-Knowledge Authenticated Data Structure (ZKADS) model that we will use. In Section 5.3, we develop the scheme for proving (non-)empty set intersection in zero knowledge. Using this primitive, we design the static reachability scheme in Section 5.4. Finally, we give the dynamic reachability scheme in Section 5.5.

5.2 Preliminaries

In this section we introduce notation and cryptographic tools that we will be using for the rest of the paper.

We denote with λ the security parameter and with $\nu(\lambda)$ a negligible function. A function $f(\lambda)$ is negligible if for each polynomial function $poly(\lambda)$ and all large enough values of λ, $f(\lambda) < 1/(poly(\lambda))$. We say that an event can occur with negligible probability if its occurrence probability can be upper bounded by a negligible function. Respectively, an event takes place with overwhelming probability if its complement takes place with negligible probability. The symbol $\xleftarrow{\$} \mathbb{X}$ denotes uniform sampling from domain $\mathbb{X}$. We denote the fact that a party Adv (instantiated as Turing machine) is probabilistic and runs in polynomial-time by writing PPT Adv.

Bilinear pairings. Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p, generated by g. Let also $\mathbb{G}_T$ be a cyclic multiplicative group with the same order p and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear pairing with the following properties: (1) Bilinearity: $e(P^a, Q^b) = e(P,Q)^{ab}$ for all $P, Q \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$; (2) Non-degeneracy: $e(g,g) \neq 1_{\mathbb{G}_T}$; (3) Computability: There is an efficient algorithm to compute $e(P,Q)$ for all $P, Q \in \mathbb{G}$. We denote with $pub := (p, \mathbb{G}, \mathbb{G}_T, e, g)$ the bilinear pairings parameters, output by a randomized polynomial-time algorithm $\mathsf{GenParams}$ on input 1^λ. For clarity of presentation, we assume for the rest of the paper a symmetric (Type 1) pairing e. We note though that both our constructions can be securely implemented in the (more efficient) asymmetric pairing case, with straight-forward modifications (see [44] for a general discussion on pairings). Our security proofs make use of the q-Strong Bilinear Diffie-Hellman (q-SBDH) assumption over groups with bilinear pairings introduced in [22].

Assumption 2 (q-Strong Bilinear Diffie-Hellman)**.** *For any PPT adversary* Adv *and for q being a parameter of size polynomial in* λ*, there exists negligible function* $\nu(\lambda)$ *such that the following holds:*

$$\Pr[pub \leftarrow \mathsf{GenParams}(1^\lambda); s \leftarrow_R \mathbb{Z}_p^*; (z, \gamma) \in \mathbb{Z}_p^* \times \mathbb{G}_T \leftarrow \mathsf{Adv}(pub, (g^s, ..., g^{s^q})) : \gamma = e(g,g)^{1/(z+s)})] \leq \nu(\lambda)]$$

Complexity model. For ease of notation, we measure the asymptotic performance of our schemes by counting numbers of operations and group elements, ignoring a, poly-logarithmic in λ, factor (e.g., an operation in $\mathbb{G}$ takes one unit time).

Non Interactive Zero Knowledge proof of Knowledge Protocols (NIZKPoK). A non-interactive zero-knowledge proof of knowledge protocol (NIZKPoK) for any NP language membership L proof system operates in the public random string model, and consists of polynomial-time algorithms P and V that work as follows: The algorithm P takes the common reference string σ and values (x,w) such that $x \in L$ and w is a witness to this. P outputs a proof π. The algorithm V takes (σ, x, π) as input and outputs accept or reject. The security properties of a NIZKPoK are the following:

Competeness: For all $x \in L$, for all witnesses w for x, for all values of random string σ and for all outputs π of $P(\sigma, x, w)$, $V(\sigma, x, \pi) = \mathsf{accept}$.

Soundness: For all adversarial prover algorithms P^*, for a randomly chosen σ, the probability that P^* can produce (x, π) such that $x \notin L$ but $V(\sigma, x, \pi) = \mathsf{accept}$ is negligible.

Knowledge Soundness with error δ*:* A zero-knowledge proof of knowledge protocol for any NP language membership L has a stronger form of soundness that says that if a cheating prover P^* convinces the

verifier that $x \in L$ with noticeable probability (i.e., more than the soundness error), then not only this means that $x \in L$ but it actually means that P^* "knows" a witness in the sense that it could obtain a witness by running some algorithm. To put more formally, for every possibly cheating prover P^*, and every x, if P^* produces π such that $Pr[V(\sigma,x,\pi) = \mathsf{accept}] > \delta + \rho$, ($\delta$ is the soundness error) then there's a algorithm E (called a knowledge extractor) with running time polynomial in $1/\rho$ and the running time of P^*, that on input x outputs a witness w for x with probability at least $1/2$.

Zero-Knowledge: There exist algorithms ZKSim-Setup and ZKSim-Prove, such that the following holds. ZKSim-Setup takes the security parameter as input and outputs (σ, s). For all x, ZKSim-Prove takes (σ, s, x) as input and outputs simulated proof π^S. Even for a sequence of adaptively and adversarially picked $(x_i, \ldots, x_m)$ (where m is polynomial in the security parameter), if $x_i \in L$ for $i \in [1,m]$, then the simulated proofs ${\pi_1}^S, \ldots {\pi_m}^S$ are distributed indistinguishably from proofs $\pi_1, \ldots, \pi_m$ that are computed by running $P(\sigma, x_i, w_i)$ where w_i is some witness that $x_i \in L$.

By $\mathsf{PoK}\{(w) : \textit{statement}(w)\}$ we denote a generic non-interactive zero-knowledge proof protocol of knowledge of a witness w such that the $\textit{statement}(w)$ is true. Sometimes we need witnesses to be online-extractable, which we make explicit by denoting with $\mathsf{PoK}\{(\underline{w_1}, w_2) : \textit{statement}(w_1, w_2)\}$ the proof of witnesses w_1 and w_2, where w_1 can be extracted. We use PoKs for notational convenience, in our scheme all proofs could also be done interactively.

Characteristic polynomial. A set $\mathcal{X} = \{x_1, \ldots, x_n\}$ with elements $x_i \in \mathbb{Z}_p$ can be represented by a polynomial following an idea introduced in [68]. The polynomial $\mathsf{Ch}_\mathcal{X}(z) = \prod_{i=1}^{n}(x_i + z)$ from $\mathbb{Z}_p[z]$, where z is a formal variable, is called the *characteristic polynomial* of $\mathcal{X}$. In what follows, we will denote this polynomial simply by $\mathsf{Ch}_\mathcal{X}$ and its evaluation at a point y as $\mathsf{Ch}_\mathcal{X}(y)$. Characteristic polynomials enjoy a number of homomorphic properties w.r.t. set operations.

5.2.1 Unforgeable Signature Scheme

A *digital signature scheme* specifies the following.

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SIG.GenKey}(1^\lambda)$ A probabilistic key-generation algorithm takes as input the security parameter and produces a pair of (private) signature key sk and (public) verification key vk.

$\sigma \leftarrow \mathsf{SIG.Sign}_{\mathsf{sk}}(m)$ A a (possibly probabilistic) signature algorithm takes as input a secret key and a message m and produces a signature σ.

$1/0 \leftarrow \mathsf{SIG.Verify}_{\mathsf{vk}}(\sigma, m)$ A (deterministic) verification algorithm that takes as input vk, a signature σ and a message m and produces a boolean.

The security definition we use in this paper is the standard existential unforgeability, i.e., it should be infeasible for an adversary to generate a valid signature on a message where the signature was not produced by the legitimate signer.

5.2.2 Aggregate Signature Scheme

A bilinear aggregate signature scheme is a 5 tuple of algorithm *Key Generation, Signing, Verification, Aggregation*, and *Aggregate Verification*. We discuss the construction for the case of a single user signing *n distinct*

messages $M_1, M_2, \ldots, M_n$ here. The description of the generic case of n different users can be found at [24]. The following notation is used in the scheme:

- G, G_1 are multiplicative cyclic groups of prime order p
- g is a generator of G
- e is computable bilinear nondegenerate map $e : G \times G \to G_1$
- $H : \{0,1\}^* \to G$ is a full domain hash function viewed as a random oracle that can be instantiated with a cryptographic hash function.

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{AGG.GenKey}(1^\lambda)$**:** $\mathsf{sk} \leftarrow \mathbb{Z}_p^*$ and $\mathsf{vk} \leftarrow g^{\mathsf{sk}}$.

$(\sigma_i) \leftarrow \mathsf{AGG.Sign}_{\mathsf{sk}}(M_i)$**:** Sign the hash of each *distinct message* $M_i \in \{0,1\}^*$ via $\sigma_i \leftarrow H(M_i)^{\mathsf{sk}}$.

$(1/0) \leftarrow \mathsf{AGG.Verify}_{\mathsf{vk}}(M, \sigma)$**:** Accept if $e(\sigma, g) = e(H(M), \mathsf{vk})$ holds, reject otherwise.

$(\sigma) \leftarrow \mathsf{AGG.AggSign}(M_1, \ldots, M_n, \sigma_1, \ldots, \sigma_n)$**:** This is a public algorithm which does not need the user's secret key to aggregate the individual signatures. Let σ_i be the signature on a distinct message $M_i \in \{0,1\}^*$ according to the Signing algorithm ($i = 1, \ldots, n$). The aggregate signature σ for any subset of k signatures, where $k \leq n$, is produced via $\sigma \leftarrow \prod_{i=1}^{k} \sigma_i$.

$(1/0) \leftarrow \mathsf{AGG.AggVerify}_{\mathsf{vk}}(M_1, \ldots, M_k, \sigma)$**:** Given the aggregate signature σ, k original messages $M_1, M_2, \ldots, M_k$ and the verification key vk:

1. ensure that all messages M_i are distinct, and reject otherwise.
2. accept if $e(\sigma, g) = e(\prod_{i=1}^{k} H(M_i), \mathsf{vk})$.

Bilinear Aggregate Signature Security. The formal model of security is called the aggregate chosen-key security model. The security of aggregate signature schemes is expressed via the following game where an adversary is challenged to forge an aggregate signature:

Setup: The adversary $\mathcal{A}$ is provided with a public key vk of the aggregate signature scheme.
Query: $\mathcal{A}$ adaptively requests signatures on messages of his choice.
Response: Finally, $\mathcal{A}$ outputs k distinct messages $M_1, M_2, \ldots, M_k$ and an aggregate signature σ.

$\mathcal{A}$ wins if the aggregate signature σ is a valid aggregate signature on messages $M_1, M_2, \ldots, M_k$ under vk, and σ is nontrivial, i.e., $\mathcal{A}$ did not request a signature on $M_1, M_2, \ldots, M_k$ under vk. A formal definition and a corresponding security proof of the scheme can be found in [24].

5.2.3 Commitment Scheme

A cryptographic commitment scheme consists of the following algorithms.

$\mathsf{par} \leftarrow \mathsf{COM.GenKey}(1^\lambda)$: The probabilistic setup algorithm outputs global parameters par for the commitment scheme. We will denote the domain of values to be committed by $D(\mathsf{par})$

$\mathsf{com} \leftarrow \mathsf{COM.Com}_{\mathsf{par}}(m; r)$: Using the global parameters par, the commit algorithm produces commitment com to message $m \in D(\mathsf{par})$ using randomness r (also called the opening).

$1/0 \leftarrow \mathsf{COM.Verify}_{\mathsf{par}}(\mathsf{com}, m, r)$: This algorithm either accepts or rejects the decommitment of com to message m using the opening r.

A commitment scheme has two important security properties: *hiding and binding*. Informally, hiding ensures that a commitment reveals no information about the message committed to and binding ensures that a commitment can be opened to exactly one value.

5.2.4 Zero-Knowledge Authenticated Data Structure

The notion of Zero-Knowledge Authenticated DataStructures (ZKADS) was introduced in [150]. We recall the model here.

An Abstract Data Type (ADT) is a data structure (DS) $\mathcal{D}$ with two types of operations defined on it: immutable operations $Q()$ and mutable operations $U()$. $Q(\mathcal{D},\delta)$ takes as input a query δ on the elements of $\mathcal{D}$ and returns the answer and it does not alter $\mathcal{D}$. $U(\mathcal{D},u)$ takes as input an update request u (e.g., insert or delete), changes $\mathcal{D}$ accordingly, and outputs the modified data structure, $\mathcal{D}'$.

We present a three party model where a trusted owner generates an instantiation of an ADT, denoted as $(\mathcal{D},Q,U)$, and outsources it to an untrusted server along with some auxiliary information. The owner also publicly releases a short digest of $\mathcal{D}$. The curious (potentially malicious) client(s) issues queries on the elements of $\mathcal{D}$ and gets answers and proofs from the server, where the proofs are zero-knowledge, i.e., they reveal nothing beyond the query answer. The client can use the proofs and the digest to verify query answers. Additionally, the owner can insert, delete or update elements in $\mathcal{D}$ and update the public digest and the auxiliary information that the server holds. In this model the owner is required to keep a copy of $\mathcal{D}$ to perform updates. We also require the updates to be zero-knowledge, i.e., an updated digest should be indistinguishable from a new digest generated for the unchanged $\mathcal{D}$.

5.2.5 Model

ZKADS is a tuple of six probabilistic polynomial time algorithms $(\mathsf{KeyGen},\mathsf{Setup},\mathsf{UpdateOwner},\mathsf{UpdateServer},\mathsf{Query},\mathsf{Verify})$. We describe how these algorithms are used between the three parties and give their API.

The owner uses KeyGen to generate the necessary keys. He then runs Setup to prepare $\mathcal{D}_0$ for outsourcing it to the server and to compute digests for the client and the server. The owner can update his data structure and make corresponding changes to digests using UpdateOwner. Since the data structure and the digest of the server need to be updated on the server as well, the owner generates an update string that is enough for the server to make the update itself using UpdateServer. The client can query the data structure by sending queries to the server. For a query δ, the server runs Query and generates ans. Using its digest, it also prepares a proof of the ans. The client then uses Verify to verify the query answer against proof and the digest she has received from the owner after the last update.

$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ where 1^λ is the security parameter. KeyGen outputs a secret key (for the owner) and the corresponding public key pk.

$(\mathsf{state}_O,\mathsf{digest}_C^0,\mathsf{digest}_S^0) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{pk},\mathcal{D}_0)$ where $\mathcal{D}_0$ is the initial data structure. Setup outputs the internal state information for the owner state_O, digests digest_C^0 and digest_S^0 for the client and the server, respectively.

$(\mathsf{ans}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S^t, \mathcal{D}_t, \delta)$ where δ is a query on elements of $\mathcal{D}_t$, ans is the query answer, and proof is the proof of the answer.

$b \leftarrow \mathsf{Verify}(\mathsf{pk}, \mathsf{digest}_C^t, \delta, \mathsf{ans}, \mathsf{proof})$ with input arguments are defined above. The output bit b is accept if $\mathsf{ans} = Q(\mathcal{D}_t, \delta)$, and reject, otherwise.

$(\mathsf{state}_O, \mathsf{digest}_C^{t+1}, \mathsf{Upd}_{t+1}, \mathcal{D}_{t+1}, u_t) \leftarrow \mathsf{UpdateOwner}(\mathsf{sk}, \mathsf{state}_O, \mathsf{digest}_C^t, \mathsf{digest}_S^t, \mathcal{D}_t, u_t, \mathsf{SID}_t)$ where u_t is an update operation to be performed on $\mathcal{D}_t$. SID_t is a session information and is set to the output of a function f on the queries invoked since the last update (Setup is counted as the 0^{th} update). UpdateOwner returns the updated internal state information state_O, the updated public/client digest digest_C^{t+1}, update string Upd_{t+1} that is used to update digest_S^t and the updated $\mathcal{D}_{t+1} := U(\mathcal{D}_t, u_t)$.

$(\mathsf{digest}_S^{t+1}, \mathcal{D}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{digest}_S^t, \mathsf{Upd}_{t+1}, \mathcal{D}_t, u_t)$ where Upd_{t+1} is used to update digest_S^t to digest_S^{t+1} and u_t is used to update $\mathcal{D}_t$ to $\mathcal{D}_{t+1}$.

Our model also supports the execution of a batch of updates as a single operation, which may be used to optimize overall performance. We note that SID and f are introduced for efficiency reasons only. Intuitively, function f can be instantiated in a way that helps reduce the owner's work for maintaining zero-knowledge property of each update. We leave f to be defined by a particular instantiation (e.g., f may the cardinality of its input). Once defined, f remains fixed for the instantiation. Since the function is public, anybody, who has access to the list of (authentic) queries performed since the last update, can compute it.

A ZKADS has three security properties: completeness, soundness and zero-knowledge.

Completeness dictates that if all three parties are honest, then for an instantiation of any ADT, the client will always accept an answer to her query from the server. Here, honest behavior implies that whenever the owner updates the data structure and its public digest, the server updates $\mathcal{D}$ and its digest accordingly and replies client's queries faithfully w.r.t. the latest data structure and digest.

Definition 13 (Completeness)**.** *For an ADT $(\mathcal{D}_0, Q, U)$, any sequence of updates $u_0, u_1, \ldots, u_L$ on the data structure $\mathcal{D}_0$, and for all queries δ on $\mathcal{D}_L$:*

$$
\begin{aligned}
\Pr[&(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda); (\mathsf{state}_O, \mathsf{digest}_C^0, \mathsf{digest}_S^0) \leftarrow \mathsf{Setup}(\mathsf{sk}, \mathsf{pk}, \mathcal{D}_0); \\
&\big\{(\mathsf{state}_O, \mathsf{digest}_C^{t+1}, \mathsf{Upd}_{t+1}, \mathcal{D}_{t+1}, u_t) \leftarrow \\
&\qquad \mathsf{UpdateOwner}(\mathsf{sk}, \mathsf{state}_O, \mathsf{digest}_C^t, \mathsf{digest}_S^t, \mathcal{D}_t, u_t, \mathsf{SID}_t); \\
&(\mathsf{digest}_S^{t+1}, \mathcal{D}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{digest}_S^t, \mathsf{Upd}_{t+1}, \mathcal{D}_t, u_t); \big\}_{0 \le t \le L} \\
&(\mathsf{ans}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S^L, \mathcal{D}_L, \delta) : \\
&\qquad \mathsf{Verify}(\mathsf{pk}, \mathsf{digest}_C^L, \delta, \mathsf{ans}, \mathsf{proof}) = \mathsf{accept} \wedge \mathsf{ans} = Q(\mathcal{D}_L, \delta)] = 1.
\end{aligned}
$$

Soundness protects the client against a malicious server. This property ensures that if the server forges the answer to a client's query, then the client will accept the answer with at most negligible probability. The definition considers adversarial server that picks the data structure and adaptively requests updates. After seeing all the replies from the owner, it can pick any point of time (w.r.t. updates) to create a forgery.

Since, given the server digest, the server can compute answers to queries herself, it is superfluous to give Adv explicit access to Query algorithm.

Definition 14 (Soundness)**.** *For all PPT adversaries* Adv *and for all possible valid queries* δ *on the data structure* $\mathcal{D}_j$ *of an ADT, there exists a negligible function* $\nu(.)$ *such that, the probability of winning the following game is negligible:* ***Setup:*** Adv *receives* pk *where* $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. *Given* pk, Adv *picks an ADT of its choice,* $(\mathcal{D}_0, Q, U)$ *and receives the server digest* digest_S^0 *for* $\mathcal{D}_0$*, where* $(\mathsf{state}_O, \mathsf{digest}_C^0, \mathsf{digest}_S^0) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{pk},\mathcal{D}_0)$.

Query: Adv *adaptively chooses a series of updates* $u_1, u_2, \ldots, u_L$ *and corresponding* SIDs*, where* $L = \mathsf{poly}(\mathsf{k})$. *For every update request* Adv *receives an update string. Let* $\mathcal{D}_{i+1}$ *denote the state of the data structure after the* (i)*th update and* Upd_{i+1} *be the corresponding update string received by the adversary, i.e.,* $(\mathsf{state}_O, \mathsf{digest}_C^{i+1}, \mathsf{Upd}_{i+1}, \mathcal{D}_{i+1}, u_i) \leftarrow \mathsf{UpdateOwner}(\mathsf{sk}, \mathsf{state}_O, \mathsf{digest}_C^i, \mathsf{digest}_S^i, \mathcal{D}_i, u_i, \mathsf{SID}_i)$.

Response: *Finally,* Adv *outputs* $(\mathcal{D}_j, \delta, \mathsf{ans}, \mathsf{proof})$, $0 \le j \le L$*, and wins the game if* $\mathsf{ans} \neq Q(\mathcal{D}_j, \delta)$ *and* $\mathsf{Verify}(\mathsf{pk}, \mathsf{digest}_C^j, \delta, \mathsf{ans}, \mathsf{proof}) = \mathsf{accept}$.

Zero-knowledge captures privacy guarantees about the data structure against a curious (malicious) client. Recall that the client receives a proof for every query answer. Periodically she also receives an updated digest, due to the owner making changes to $\mathcal{D}$. Informally, (1) the proofs should reveal nothing beyond the query answer, and (2) an updated digest should reveal nothing about update operations performed on $\mathcal{D}$. This security property guarantees that the client does not learn which elements were updated, unless she queries for an updated element (deleted or replaced), before and after the update.

Definition 15 (Zero-Knowledge)**.** *Let* $\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}$ *and* $\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}$ *be defined as follows where, wlog the adversary is asks only for valid data and update queries.*[1]

A ZKADS $\mathcal{E}$ *is zero-knowledge if there exists a PPT algorithm* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *s.t. for all malicious stateful adversaries* $\mathsf{Adv} = (\mathsf{Adv}_1, \mathsf{Adv}_2)$ *there exists a negligible function* $\nu(.)$ *s.t.*

$$|\Pr[\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}(1^\lambda) = 1]| \le \nu(k).$$

We note that SID argument to UpdateOwner need not be used explicitly in the definition: Adv implicitly controls the input of f by choosing queries on $\mathcal{D}$ (recall that $\mathsf{SID} = f(\ldots)$), while the challenger and the simulator know all the queries and can compute f themselves.

5.3 Set intersection in Zero-Knowledge

In this section, we will give a protocol for proving the intersection of two sets is empty/non-empty in zero-knowledge. We denote our scheme for proving set intersection result in zero knowledge as ZKSI. In particular, we consider a static collection $\mathbb{S}$ of m sets $\mathcal{X}_1, \ldots, \mathcal{X}_m$ that is remotely stored on an untrusted server. We then develop a mechanism to answer intersection queries on this sets such that the answers to the queries can be verified publicly and in zero-knowledge. That is, the proofs of the queries should reveal nothing beyond

[1] This is not a limiting constraint, as we can easily force this behavior by checking if a query/update is valid in the Real game.

$\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda)$	$\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$
The challenger, $\mathcal{C}$, runs $\mathsf{KeyGen}(1^\lambda)$ to generate sk,pk, sends pk to Adv_1.	Sim_1 generates a public key, pk, sends it to Adv_1 and keeps a state, state_S.
Given pk, Adv_1 picks an ADT $(\mathcal{D}_0, Q, U)$ of its choice.	
Given $\mathcal{D}_0$, $\mathcal{C}$ runs $\mathsf{Setup}(\mathsf{sk},\mathsf{pk},\mathcal{D}_0)$ and sends digest_C^0 to Adv_1.	Sim_1 generates digest_C^0, sends it to Adv_1, updates state_S. (It is not given $\mathcal{D}_0$.)
With access to Adv_1's state, Adv_2 adaptively queries $\{q_1, q_2, \ldots, q_M\}$, $M = \mathsf{poly}(\mathsf{k})$: (Let $\mathcal{D}_{t-1}$ denote the state of the data structure at the time of q_i.)	
On data query q_i:	
$\mathcal{C}$ runs Query algorithm for the query on $\mathcal{D}_{t-1}$ and the corresponding digest as its parameters. $\mathcal{C}$ returns ans and proof to Adv_2.	Given the answer to the query, $Q(\mathcal{D}_{t-1}, q_i)$, and state_S, Sim_2 generates ans and proof, sends them to Adv_2 and updates its state.
On update query q_i:	
$\mathcal{C}$ runs UpdateOwner algorithm on q_i and returns the public digest digest_C^t.	Given state_S, Sim_2 returns updated digest digest_C^t and updates its state. (It is not given the update query q_i.)
Adv_2 outputs a bit b.	

the query answer (which is a boolean indicating true if intersection is non-empty and false otherwise). The *sets collection* data structure $\mathbb{S}$, consists of m sets, denoted with $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, each containing elements from a universe $\mathbb{X}$. A set does not contain duplicate elements, however an element can appear in more than one set. Proving that the intersection is empty is almost identical to the batch proof of non-membership in [150]. This protocol for proving non-empty intersection is our contribution and may be of independent interest. At a very high level, the proof proceed as follows. When the intersection of $\mathcal{X}_1, \mathcal{X}_2$ is empty, the server proves that the polynomials $\mathsf{Ch}_{\mathcal{X}_1}[z]$ and $\mathsf{Ch}_{\mathcal{X}_2}[z]$ are co-prime to each other. When the intersection in non-empty, the server finds some $x \in \mathcal{X}_1 \cap \mathcal{X}_2$, forms a zero-knowledge accumulator acc_x for the singleton $\{x\}$, proves that the accumulator is well formed and then proves that $x \in \mathcal{X}_1$ and $x \in \mathcal{X}_2$ using the corresponding zero knowledge accumulators. We note that, we give the construction for queries that involve intersection of two sets only for simplicity, but our scheme trivially extends to answering intersection queries of multiple sets.

5.3.1 Construction

Notation: The notation $q[z]$ denotes polynomial q over undefined variable z and $q(s)$ is the evaluation of the polynomial at point s. All arithmetic operations are performed mod p. N is a variable maintained by the owner. Let $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, be a collection of sets, each containing elements from a universe $\mathbb{X}^2$

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{GenKey}(1^\lambda)$**:** Run $\mathsf{GenParams}(1^\lambda)$ to receive bilinear parameters $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g)$. Choose $s \xleftarrow{\$} \mathbb{Z}_p^*$. Return $\mathsf{sk} = s$ and $\mathsf{vk} = (g^s, pub)$.

$(\mathsf{digest}_C, \mathsf{digest}_S) \leftarrow \mathsf{Setup}(\mathsf{sk}, \mathsf{vk}, \mathbb{S},)$**:** For each $\mathcal{X}_i \in \mathbb{S}$, do the following:

- Choose $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. Set value $n_i = |\mathcal{X}_i|$.
- Compute $\mathsf{acc}_i \leftarrow g^{r_i \cdot \mathsf{Ch}_{\mathcal{X}_i}(\mathsf{sk})}$, $ek_i = (g, g^{\mathsf{sk}}, g^{\mathsf{sk}^2}, \ldots, g^{\mathsf{sk}^{n_i}})$ and $\mathsf{aux}_i = (r_i, n_i)$.
- Set $\mathsf{ek} := (g, g^{\mathsf{sk}}, \ldots, g^{\mathsf{sk}^N})$ where $N = \max_{i \in [1,m]} n_i$
- Set $\mathsf{aux} := ((r_i \mid \mathcal{X}_i \in \mathbb{S}), N)$.
- Return $\mathsf{digest}_C = \mathsf{vk}, \mathsf{digest}_S = (\{\mathsf{acc}_i\}_{i \in [1,m]}, \mathsf{ek}, \mathsf{vk})$

$(\mathsf{ans}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S, \mathbb{S}, \delta)$**:** Given a query (i, j) where $i, j \in [1, m]$, the server computes the answer and proof as follows[3].

$\mathcal{X}_i \cap \mathcal{X}_j \neq \phi$**:** The server sets $\mathsf{ans} := \mathsf{true}$ and computes the proof as follows

- Find any element $v \in \mathcal{X}_i \cap \mathcal{X}_j$
- Pick a random $r \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $\mathsf{acc}_x \leftarrow g^{r(\mathsf{sk}+x)}$
- Compute the proof of well formdness of acc_x by computing $\Pi = \mathsf{PoK}\{(\underline{r}, \underline{x}) \mid \mathsf{acc}_x = g^{r(\mathsf{sk}+x)}\}$
- Compute witness $w_1 \leftarrow (\mathsf{acc}_i)^{\frac{1}{r(\mathsf{sk}+x)}}, w_2 \leftarrow (\mathsf{acc}_j)^{\frac{1}{r(\mathsf{sk}+x)}}$
- Return $(\mathsf{ans}, \mathsf{proof} = (\mathsf{acc}_x, \mathsf{acc}_i, \mathsf{acc}_j, \Pi, w_1, w_2))$

$\mathcal{X}_i \cap \mathcal{X}_j = \phi$**:** The server sets $\mathsf{ans} := \mathsf{false}$ and computes the proof as follows

- Using the Extended Euclidean algorithm, compute polynomials $q_1[z], q_2[z]$ such that $q_1[z]\mathsf{Ch}_{\mathcal{X}_i}[z] + q_2[z]\mathsf{Ch}_{\mathcal{X}_j}[z] = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q_1'[z] = r_i^{-1} q_1[z] + \gamma \cdot \mathsf{Ch}_{\mathcal{X}_j}[z]$ and $q_2'[z] = r_j^{-1} q_2[z] - \gamma \cdot \mathsf{Ch}_{\mathcal{X}_i}[z]$.
- Set $w_1 := g^{q_1'(\mathsf{sk})}, w_2 = g^{q_2'(\mathsf{sk})}$
- Return $(\mathsf{ans}, \mathsf{proof} := (\mathsf{acc}_i, \mathsf{acc}_j, w_1, w_2))$.

$(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{Verify}(\mathsf{digest}_C, \delta, \mathsf{ans}, \mathsf{proof})$**:** For a query (i, j), if $\mathsf{ans} = \mathsf{true}$, do the following.

- Parse proof as $(\mathsf{acc}_x, \mathsf{acc}_i, \mathsf{acc}_j, \Pi, w_1, w_2)$
- Verify Π. If the verification passes, proceed. Else output reject
- Check if $e(\mathsf{acc}_x, w_1) = e(\mathsf{acc}_i, g)$ and $e(\mathsf{acc}_x, w_2) = e(\mathsf{acc}_j, g)$. If both check passes, return accept. Else return reject.

If $\mathsf{ans} = \mathsf{false}$, do the following:

- Parse proof as $(\mathsf{acc}_i, \mathsf{acc}_j, w_1, w_2)$
- Return true if $e(w_1, \mathsf{acc}_i) e(w_2, \mathsf{acc}_j) = e(g, g)$, false otherwise.

Figure 5.1: *Instantiation of* ZKSI

5.3.2 Security Proof

Completeness is easy to see. Here we prove that our construction is both sound and zero-knowledge as per Definition 14 and Definition 15 respectively.

Prover, P has $(g^{\mathsf{sk}}, r, x, \mathsf{acc} = g^{r(\mathsf{sk}+x)})$ and Verifier V has $(\mathsf{acc}, g^{\mathsf{sk}})$. The protocol proceeds as follows. Here $\mathcal{H}$ is a cryptographic hash function.

- P randomly generates $R, \overline{r}, \overline{x}, \overline{R} \xleftarrow{\$} \mathbb{Z}_p^*$.
- Then P computes $\mathsf{acc} \leftarrow g^{r(\mathsf{sk}+x)}$, $\mathsf{acc}' \leftarrow g^{(r\mathsf{sk}+R)}$ and $\frac{\mathsf{acc}}{\mathsf{acc}'} = g^{\alpha}$, where $\alpha = rx - R$.
- P also computes $\overline{\mathsf{acc}} \leftarrow g^{\overline{r}(\mathsf{sk}+\overline{x})}$, $\overline{\mathsf{acc}'} \leftarrow g^{(\overline{r}\mathsf{sk}+\overline{R})}$, and $\frac{\overline{\mathsf{acc}}}{\overline{\mathsf{acc}'}} = g^{\overline{\alpha}}$, where $\overline{\alpha} = \overline{r}\overline{x} - \overline{R}$.
- Further, P computes $c \leftarrow \mathcal{H}(\mathsf{acc}, \mathsf{acc}', g^{\alpha}, \overline{\mathsf{acc}}, \overline{\mathsf{acc}'}, g^{\overline{\alpha}})$. Then P computes $\tilde{r} \leftarrow \overline{r} + cr$, $\tilde{x} \leftarrow \overline{r}\overline{x} + rxc$, $\tilde{R} = \overline{R} + cR$ and $\tilde{\alpha} = \overline{\alpha} + c\alpha$
- Finally, P sets $\Pi := (\mathsf{acc}, \mathsf{acc}', \overline{\mathsf{acc}}, \overline{\mathsf{acc}}, g^{\alpha}, g^{\overline{\alpha}}, \tilde{r}, \tilde{x}, \tilde{R}, \tilde{r}, \tilde{\alpha})$ and sends Π to V.

The verification proceeds as follows:

- V parses Π as $(\mathsf{acc}, \mathsf{acc}', \overline{\mathsf{acc}}, \overline{\mathsf{acc}}, g^{\alpha}, g^{\overline{\alpha}}, \tilde{r}, \tilde{x}, \tilde{R}, \tilde{r}, \tilde{\alpha})$.
- V computes $c \leftarrow \mathcal{H}(\mathsf{acc}, \mathsf{acc}', g^{\alpha}, \overline{\mathsf{acc}}, \overline{\mathsf{acc}'}, g^{\overline{\alpha}})$.
- V verifies if (1) $(\overline{\mathsf{acc}})\mathsf{acc}^c \stackrel{?}{=} (g^{\mathsf{sk}})^{\tilde{r}} g^{\tilde{x}}$ (2) $(\overline{\mathsf{acc}'})\mathsf{acc}'^c \stackrel{?}{=} (g^{\mathsf{sk}})^{\tilde{r}} g^{\tilde{R}}$ (3) $\frac{\overline{\mathsf{acc}}}{\overline{\mathsf{acc}'}} \times \left(\frac{\mathsf{acc}}{\mathsf{acc}'}\right)^c \stackrel{?}{=} g^{\tilde{\alpha}}$ If any of the verification fails, **return** reject. Else **return** accept.

Figure 5.2: $\mathsf{PoK}\{(\underline{r}, \underline{x}) \mid \mathsf{acc} = g^{r(\mathsf{sk}+x)}\}$

Lemma 12. *Under the q-SBDH assumption and the soudness of Proof of Knowledge system the* ZKSI *scheme is sound as per Definition 14.*

Proof. Let q be some polynomial in the security parameter of the scheme such that q is an upper bound on the maximum size the adversarial set be. Since the adversary has to come with a set collection that is $poly(\lambda)$, and is allowed to ask a polynomial (in λ) number of queries, this upper bound exists. We will give a reduction to q-SBDH assumption. Let $\mathcal{A}$ be the reduction which receives a q-SBDH tuple $(g, g^s, \ldots, g^{s^q})$ as input. The reduction does the following:

Recall the soundness game where the adversary Adv sees the vk, then comes up with a set collection $\mathbb{S}_0 = \mathcal{X}_1, \ldots, \mathcal{X}_k$. and then sends it to $\mathcal{A}$. $\mathcal{A}$ runs the query algorithm where it computes the proof units using $(g, g^s, \ldots, g^{s^q})$ instead of the secret key s and sends it to Adv. Note that all the steps can be executed using the tuple $(g, g^s, \ldots, g^{s^q})$.

Now let A_1 be the event that Adv outputs a forged a non-empty-intersection proof and A_2 be the event that it outputs a forged proof of empty intersection. The probability that Adv outputs a successful forgery is $Pr[A_1 \cup A_2] \leq Pr[A_1] + Pr[A_2]$. We will individually bound the probability of success of each of the events individually.

Forging a proof of non-empty intersection: This implies, that for some sets $\mathcal{X}_i, \mathcal{X}_j \in \mathbb{S}$, Adv provided an accepting proof that $\mathcal{X}_i \cap \mathcal{X}_j \neq \phi$, when, actually $\mathcal{X}_i \cap \mathcal{X}_j = \phi$. By knowledge soundness of the underlying Proof of Knowledge system, there exists an efficient extractor that can extract r, x from the PoK. Now, since $\mathcal{X}_i \cap \mathcal{X}_j = \phi$, it must be the case that $x \notin \mathcal{X}_j$. Since $x \notin \mathcal{X}_j$, it follows that $(x+z) \not| \mathsf{Ch}_{\mathcal{X}_j}[z]$ which guarantees the existence of $q[z], c$. Also observe that c is a scalar (zero-degree polynomial) since it is the remainder of the polynomial division and it must have degree less than that of $(x+z)$. Since verify accepts we can write:

$$e(\mathsf{acc}_x, w_2) = e(g, w_2)^{r(s+x)} = e(\mathsf{acc}_j, g) = e(g^{r_j \cdot \mathsf{Ch}_{x_j}(s)}, g) = e(g, g)^{r_j((x+s)q(s)+c)}$$

From this, it follows that $[e(w_2,g)^{rr_j^{-1}} e(g,g)^{-q(s)}]^{c^{-1}} = e(g,g)^{1/(x+s)}$. Therefore, $\mathcal{A}$ can output $e(g,g)^{1/(x+s)}$, thereby breaking the q-SBDH assumption. Hence, there exists a negligible function ν_1, such that $Pr[A_1] \leq \nu_1(\lambda)$.

Forging a proof of empty intersection: This implies, that for some sets $\mathcal{X}_i, \mathcal{X}_j \in \mathbb{S}$, Adv provided an accepting proof that $\mathcal{X}_i \cap \mathcal{X}_j = \phi$, when, actually $\mathcal{X}_i \cap \mathcal{X}_j \neq \phi$. Therefore, there exists at least one element y that belongs to all the sets $\mathcal{X}_i \cap \mathcal{X}_j$. Therefore, there exists polynomials $\tilde{P}_t[z]$ such that $\mathsf{Ch}_{\mathcal{X}_t}[z] = (y+z)\tilde{P}_t[z]$ for $t \in \{i,j\}$. Compute polynomial $\tilde{P}_t[z]$ using polynomial long division. Compute $\prod_{t\in\{i,j\}} e(w_t, g^{r_t \tilde{P}_t(s)})$ which equals $e(g,g)^{\frac{1}{y+s}}$ Therefore using w_t's, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_2] \leq \nu_2(\lambda)$ for some negligible function ν_2. Thus we have $Pr[E_1 \cup E_2] \leq \nu_1(\lambda) + \nu_2(\lambda)$. This concludes the soundness proof. □

Lemma 13. *The* ZKSI *scheme satisfies zero-knowledge property as per Definition 15.*

Proof. We write a simulator Sim that acts as follows:

- It first runs $\mathsf{GenParams}(1^\lambda)$ to receive bilinear parameters $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g)$
- Then it picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and sends $\mathsf{vk} := g^s, pub$ to Adv and saves s as its secret key.
- It runs ZKSim-Setup and generates parameter σ.
- Given vk, Adv comes up with a set collection $\mathbb{S}_0$ of its choice. Sim is given $|\mathbb{S}_0| = m$ which is the public information of the scheme.
- Sim picks m random numbers $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ for $i \in [1,m]$ and computes $\mathsf{acc}_i \xleftarrow{\$} g^{r_i}$ and saves all the r_i's in its state information.
- Sim sends $\mathsf{digest}_C = (g, g^s)$ to Adv.

After this, when Adv adaptively asks for queries (no updates since this is a static structure). Sim maintains a list of previously asked queries and uses the proof units used by the Sim. If a query is repeated by Adv then Sim uses the same proof units.

Intersection is non-empty: In this case, Sim generates picks a fresh random element $r \xleftarrow{\$} \mathbb{Z}_p^*$. It then computes $\mathsf{acc}_x \leftarrow g^r, w_1 \leftarrow g^{r_i/r}, w_2 \leftarrow g^{r_j/r}$. It then generates Π using ZKSim-Prove and sends $(\mathsf{ans} = \mathsf{true}, \mathsf{proof} = (\mathsf{acc}_x, \mathsf{acc}_i, \mathsf{acc}_j, \Pi, w_1, w_2))$ back to Adv.

Intersection is empty: In this case, Sim does the following:

- Let $x = GCD(r_i, r_j)$ and let us denote as $\tilde{r}_t = r_t/x, t \in \{i,j\}$.
- Note that $\tilde{r}_t$'s are co-prime. Using Extended Euclidian algorithm, compute q_t's such that $\sum_{t\in\{i,j\}} q_t \tilde{r}_t = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q_i'[z] := (q_i + (\gamma r_j))\frac{1}{x}, q_j'[z] := (q_j - (\gamma r_i))\frac{1}{x}$.
- Set $w_1 \leftarrow g^{q_i'(s)}, w_2 \leftarrow g^{q_j'(s)}$
- Return $(\mathsf{ans} = \mathsf{false}, (\mathsf{acc}_i, \mathsf{acc}_j, w_1, w_j))$

This concludes the description of Sim. □

We summarize the efficiency and the security properties of the ZKSI scheme in Theorem 4.

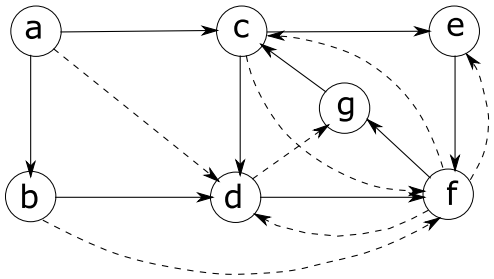
Theorem 4. *The* $\mathsf{ZKSI} = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Verify})$ *scheme satisfies the properties of completeness (Definition 13), soundness (Definition 14), and zero-knowledge (Definition 15). Let* $\mathbb{S} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ *be the set original set collection. Define* $n_j = |\mathcal{X}_j|$ *and* $N = \max_{j\in[1,m]} n_j$.

- KeyGen *has access complexity* $O(1)$*;*
- Setup *has complexity* $O(Nm)$*;*
- Query *has access complexity is* $O(N \log^2 N \log\log N)$*. The proof size is* $O(1)$ *and the verification* Verify *has complexity* $O(1)$*.*

5.4 Zero Knowledge Reachability on Static Graphs (sZKReach)

In this section, we will give a construction of answering reachability queries on outsourced directed graphs (which subsumes undirected graphs) in zero knowledge. The basic intuition for the construction is the following. Using the labeling algorithm from [47], we construct a 2-hop cover of a graph. This labeling algorithm, attaches to each vertex u of the graph two labels $L_{\text{in}}(u)$ and $L_{\text{out}}(u)$ such that, for any two vertices u and v, the four labels $L_{\text{in}}(u)$, $L_{\text{out}}(u)$, $L_{\text{in}}(v)$ and $L_{\text{out}}(v)$ would contain enough information to answer the reachability query (u to v or v to u). More formally we use the following.

Theorem 5. *[47] Let* $G = (V,E)$ *be a directed graph. A 2-hop reachability labeling of* G *assigns to each vertex* $v \in V$ *a label* $L(v) = (L_{in}(v), L_{out}(v))$*, such that* $L_{in}(v), L_{out}(v) \subseteq V$ *and there is a path from every* $x \in L_{in}(v)$ *to* v *and from* v *to every* $x \in L_{out}(v)$*. Furthermore, for any two vertices* $u, v \in V$ *, we should have* `reach(u, v)` *= 1 iff* $L_{out}(u) \cap L_{in}(v) \neq \phi$



v	$L_{in}(v)$	$L_{out}(v)$
a	**a**	**a**,b,c,d
b	**b**	**b**,f,d
c	f,**c**	**c**,d,f
d	f,c,**d**	g,f,**d**
e	c,f,**e**	f,**e**
f	d,c,**f**	g,c,e,**f**
g	d,f,**g**	c,**g**

Figure 5.3: *The graph and the corresponding labels* $L_{in}(v), L_{out}(v)$ *are shown here. Note that each label include the vertex label itself. The dotted lines represent hops that are not edges of the graph. Now, for a query* (c,g)*, the answer is* 1 *since* $L_{out}(c) \cap L_{in}(g) \neq \phi$*. But for a query* (c,a) *, the answer is* 0 *since* $L_{out}(c) \cap L_{in}(a) = \phi$*.*

For an example (taken from [47]), see Figure 5.3. We use this set theoretic characterization of reachability queries and prove reachability between two vertices u, v by proving whether the intersection of $L_{\text{out}}(u) \cap L_{\text{in}}(v)$ is empty or not. Now we can directly invoke our ZKSI scheme to answer reachability queries. The first characterization of reachability using 2-hop cover was proposed in [47]. The authors showed that computing exact 2-hop cover is NP-hard, but they gave a $\log |V|$ factor approximation algorithm where V is the set of vertices of the graph. There have been several follow up works on improving efficiency of this construction [161, 16].

Time and Space Complexity: Here we analyze the asymptotic complexity of our sZKReach scheme.

Notation: Let $G=(V,E)$ be a graph whose vertex set is denoted by V and edge set is denoted by E. For simplicity, we assume that the queries are on publicly known ID's of the vertices present in the graph. If we allow queries on non-existent nodes as well, we will first need to prove that the queried ID is indeed present/absent in the graph. This can be easily proven using ZK-Accumulator [152] or ZK-PSR systems [122].

$(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{GenKey}(1^\lambda)$**:** Pick a collision resistant cryptographic hash function $\mathcal{H}$. Run the key generation algorithm for any unforgeable signature scheme to compute $(\mathsf{sk}_{\mathsf{Sig}},\mathsf{vk}_{\mathsf{Sig}}) \leftarrow \mathsf{SIG.GenKey}(1^\lambda)$. Run $(\mathsf{sk}',\mathsf{vk}') \leftarrow \mathsf{ZKSI.GenKey}(1^\lambda)$. Set $\mathsf{sk} := (\mathsf{sk}',\mathsf{sk}_{\mathsf{Sig}}), \mathsf{vk} := (\mathsf{vk}',\mathsf{vk}_{\mathsf{Sig}},\mathcal{H})$.

$(\mathsf{digest}_C,\mathsf{digest}_S) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{vk},G=(V,E))$**:** First compute the 2-hop label cover for the graph G. Let $L_{\mathtt{out}}(v), L_{\mathtt{in}}(v)$ be the out and in label sets for vertex v.

- Assign a unique ID for each of the label sets as $\mathsf{ID}_{L_{\mathtt{out}}(v)} \leftarrow \mathcal{H}(\mathsf{ID}(v),\mathtt{out}), \mathsf{ID}_{L_{\mathtt{in}}(v))} \leftarrow \mathcal{H}(\mathsf{ID}(v),\mathtt{in})$. For notational convenience, we will denote $\mathsf{ID}_{L_t(v)}$, $t \in \{\mathtt{in},\mathtt{out}\}$ as (t,v)
- Define $\mathbb{S} = \{L_{\mathtt{in}}(v), L_{\mathtt{out}}(v)\}_{v\in V}$.
- Run $(\mathsf{digest}'_C,\mathsf{digest}'_S) \leftarrow \mathsf{ZKSI.Setup}(\mathsf{sk}',\mathsf{vk}',\mathbb{S})$.
- Parse digest'_S as $(\{\mathsf{acc}_{t,v}\}_{t\in\{\mathtt{in},\mathtt{out}\},v\in V},\mathsf{ek}',\mathsf{vk}')$.
- Generate $\sigma_{t,v} \leftarrow \mathsf{SIG.Sign}_{\mathsf{sk}_{\mathsf{Sig}}}(\mathsf{ID}(v),\mathsf{ID}(L_t(v)),\mathsf{acc}_{t,v})$ for $t \in \{\mathtt{in},\mathtt{out}\}, v \in V$.
- Set $\mathsf{digest}_C := (\mathsf{vk})$, $\mathsf{digest}_S = (\mathbb{S},\mathsf{vk},\mathsf{digest}'_S,\{\sigma_{t,v}\}_{t\in\{\mathtt{in},\mathtt{out}\},v\in V})$

$(\mathsf{ans},\mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S,G,\delta)$**:** On a query $\delta = (\mathsf{ID}(u),\mathsf{ID}(v))$, do the following:

- Parse digest_S as $(\mathbb{S},\mathsf{vk},\mathsf{digest}'_S,\{\sigma_{t,v}\}_{t\in\{\mathtt{in},\mathtt{out}\},v\in V})$
- Call $(\mathsf{ans}',\mathsf{proof}') \leftarrow \mathsf{ZKSI.Query}(\mathsf{digest}'_S,\mathbb{S},((\mathtt{out},u),(\mathtt{in},(v))))$
- Set $\mathsf{ans} := \mathsf{ans}'$ and $\mathsf{proof} = [\sigma_{\mathtt{out},v},(\mathsf{ID}(v),\mathsf{ID}(L_{\mathtt{out}}(v)),\mathsf{acc}_{\mathtt{out},v}),\sigma_{\mathtt{in},u},(\mathsf{ID}(u),\mathsf{ID}(L_{\mathtt{in}}(u)),\mathsf{acc}_{\mathtt{in},u}),\mathsf{proof}']$

$(\mathsf{acceptreject}) \leftarrow \mathsf{Verify}(\mathsf{digest}_C,\delta,\mathsf{ans},\mathsf{proof})$**:**

- Let $\delta = (\mathsf{ID}(u),\mathsf{ID}(v))$ and $\mathsf{digest}_C = (\mathsf{vk}',\mathsf{vk}_{\mathsf{Sig}},\mathcal{H})$
- Parse proof as $[\sigma_{\mathtt{out},v},(\mathsf{ID}(v),\mathsf{ID}(L_{\mathtt{out}}(v)),\mathsf{acc}_{\mathtt{out},v}),\sigma_{\mathtt{in},u},(\mathsf{ID}(u),\mathsf{ID}(L_{\mathtt{in}}(u)),\mathsf{acc}_{\mathtt{in},u}),\mathsf{proof}']$
- Verify $\mathsf{SIG.Verify}_{\mathsf{vk}_{\mathsf{Sig}}}(\sigma_{\mathtt{out},v},(\mathsf{ID}(v),\mathsf{ID}(L_{\mathtt{out}}(v)),\mathsf{acc}_{\mathtt{out},v}))$ and $\mathsf{SIG.Verify}_{\mathsf{vk}_{\mathsf{Sig}}}(\sigma_{\mathtt{in},u},(\mathsf{ID}(u),\mathsf{ID}(L_{\mathtt{in}}(u)),\mathsf{acc}_{\mathtt{in},u}))$. If both verification pass, proceed to next step. Else return reject.
- Run $\mathsf{ZKSI.Verify}(\mathsf{vk}',\delta,\mathsf{ans},\mathsf{proof}')$. Return the response.

Figure 5.4: *Zero Knowledge Reachability on Static Graphs (*sZKReach*)*

Setup The algorithm in [48], computes an approximate 2-hop cover whose size is larger than the smallest possible cover by an $O(\log n)$ factor. The largest size of a 2-hop cover for any m edge directed graph is conjectured [48] to be $\tilde{O}(n\sqrt{m})$. This conjecture is acknowledged in [91].

The time taken to genertate the label set for a graph on n vertices and m edges by the algorithm in [48] is $O(n^3\cdot|TC|)$ (see, [91]) where TC is the transitive closure of the graph. The authentication information is generated as $\mathsf{ZKSI.Setup}$ runs in time $\tilde{O}(n^2\sqrt{m})$. Therefore, the overall time is $O(n^3\cdot|TC|)$.

Query To query two vertices (u,v) in the worst case, the server has to scan the label sets $L_{\mathtt{out}}(u)$ and $L_{\mathtt{in}}(v)$ to find if they intersect. Let $t = |L_{\mathtt{out}}(u)| + |L_{\mathtt{in}}(v)|$. Then, $\mathsf{ZKSI.Query}$ runs in time $O(t\log^2 t\log\log t)$. Assuming the conjecture that largest size of a 2-hop cover for any m edge directed graph is $\tilde{O}(n\sqrt{m})$, the average cost per query is $\tilde{O}(\sqrt{m})$. The server's storage cost is $\tilde{O}(n\sqrt{m})$.

Verify The proof size and the verification time is $O(1)$.

Security: Completeness is easy to see. The proof of soundness directly follows from the soundness of ZKSI and the unforgeability of the signature scheme SIG. Here we prove that our construction satisfies zero-knowledge property by constructing a simulator Sim.

Sim simulates the proofs for the queries and also the random oracle $\mathcal{H}$ (the hash function that we model as a RO). For each query to the RO, Sim maintains a table. If any query is repeated Sim, looks up its table to response. Otherwise, it generates a random string and inserts the query point and the string in its table.

Sim picks $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{vk}_{\mathsf{Sig}}) \leftarrow \mathsf{SIG.GenKey}(1^{\lambda})$ and runs $\mathsf{Sim}_{\mathsf{ZKSI}}$ to get vk'. It then sends $\mathsf{digest}_C = (\mathsf{vk}', \mathsf{vk}_{\mathsf{Sig}}, \mathcal{H})$ to the adversary.

On every subsequent (new) query $(\mathsf{ID}(u), \mathsf{ID}(v))$, Sim first uses $\mathcal{H}$ to generate the random IDs for $L_{\mathtt{in}}(v), L_{\mathtt{out}}$ (if they have not been generated already) and then invokes $\mathsf{Sim}_{\mathsf{ZKSI}}$ to simulate proof'. Sim then generates $\mathsf{Sig}_{\mathtt{out},u}, \mathsf{Sig}_{\mathtt{in},v}$ as per the SIG.Sign algorithm to complete the simulation of the entire proof proof. This concludes the description of Sim.

We summarize the performance of our static graph reachability scheme in Theorem 6

Theorem 6. *Our zero knowledge static graph reachability scheme,* sZKReach*, satisfies completeness (Definition 13), soundness (Definition 14) and zero-knowledge (Definition 15). The asymptotic performance of the scheme is as follows. For a graph with n vertices and m edges, let m^* denote the transitive closure of the graph. We have that setup runs in time and space $O(n^3 \cdot m^*)$. Also, the proof size and verification time are each $O(1)$. Finally, assuming the conjecture in [47], the storage space at the server is $\tilde{O}(n\sqrt{m})$, and the average cost per query is $\tilde{O}(\sqrt{m})$.*

5.5 Zero-Knowledge Reachability Queries on General Dynamic Graphs(dZKR)

In this section, we will use the algorithm to maintain a dynamic transitive closure of a directed graph from [139]. This algorithm has many steps and build on some previous constructions by [69, 88, 89]. We will first describe the subroutines that we will use to build the final algorithm.

5.5.1 Algorithm to maintain a Directed Acyclic Graph under deletion [88, 89]

Algorithm Ital

Notation For a tree T, and a vertex w, $T(w)$ denotes the pointer to the node w in T if it exists, and is $NULL$ otherwise. For any pointer p to a node in T, $p.parent$ denotes is the pointer to its parent in T.

Setup: Given an input graph $G_0 = (V_0, E_0)$, do the following:

- Initialize two empty matrices of pointers PARENT and HOOK, each of dimension $|V_0| \times |V_0|$.
- For each $u \in V_0$
  - Build a spanning tree T_u of u
  - For each $v \in V_0$, set PARENT$[u, v] \leftarrow T_u(v)$

- Initialize $IN[u]$ to be a linked list of incoming edges to u.
- For each $v \in V_0$, set $\text{HOOK}[v,u] = IN[u].head$

• Output $\text{state}_0 = (IN_0 = IN, \text{HOOK}_0 = \text{HOOK}, \text{PARENT}_0 = \text{PARENT}, \{T_u\}_{u \in G_0})$

Deletion: Let (i,j) be the edge to be deleted. Let the latest graph be denoted at $G_{\text{curr}} = (V_{\text{curr}}, E_{\text{curr}})$ and the current state be $\text{state}_{\text{curr}} = (IN_{\text{curr}}, \text{HOOK}_{\text{curr}}, \text{PARENT}_{\text{curr}}, \{T_u\}_{u \in G_{\text{curr}}})$

For each $u \in V_{\text{curr}}$ if $\text{PARENT}[u,j] \neq NULL$ and $\text{PARENT}[u,j].parent = i$

- If there exists a vertex $v \in \text{HOOK}[u,j], v \neq i$, such that $\text{PARENT}[u,v] \neq NULL$, then $\text{PARENT}[u,j] = v$.
- Else
  - Delete (i,j) from T_u
  - For Each edge (a,b) in the sub-tree $T_u(j)$, resursively delete it.
  - Set $\text{PARENT}[u,j] \leftarrow NULL$

Query: On query (u,v) do the following:

- If $\text{PARENT}[u,v]_{\text{curr}} \neq NULL$ return true.
- Else, return false.

5.5.2 Algorithm to maintain a General Graph under deletion [69]

Algorithm Frig

Notation

- Let $SCC(G)$ denote all the strongly connected components of a graph G.
- Let $\mathcal{C} = \{C \mid C \in SCC(G)\}$. Define skeleton graph of G to be the adjacency matrix M_G of order $|\mathcal{C}| \times |\mathcal{C}|$ such that for any $C_1, C_2 \in SCC(G)$, $M_G[C_1, C_2] = 1$ if and only if there is an edge (i,j) common to $C_1.OUT$ and $C_2.IN$.
- For any graph G and a node $u \in G$ define $DFSTree(G,u)$ to be a DFSTree of G with root u.
- For any directed graph G, let G^R denote the graph obtained by reversing the direction of every edge in G.

Setup: Let the input directed graph be $G_0 = (V_0, E_0)$. Set up SCC a $|V_0|$ dimensional array of pointers to a structure representing strongly connected componenets of G_0 defined as a follows. For every strongly connected component $C \in G_0$, denote by C a structure with following elements:

- $C.G$: the induced subgraph of the edges and vertices forming C
- C.sparseCert : a pair of DFS trees of $C.G$. This pair is computed by first building a DFS tree $T_{C.G}$ rooted at any vertex $r \in C.G$ and then reversing the direction of edges in $C.G$ and building a second DFS tree $T^R_{C.G}$ rooted at r and then restoring the original direction of edges in $T^R_{C.G}$.
- $C.IN$: a linked list of edges in coming to C initialized to ϕ

- $C.OUT$: a linked list of edges out going from C initialized to ϕ
- A function $C.G.\mathsf{Delete}(i,j)$ removing the edge (i,j) from the graph $C.G$

Do the following initialization :

- Initialize two empty matrices of pointers PARENT and HOOK, each of dimension $|V_0| \times |C|$.
- For every $v \in G_0$, set $SCC[v] = C$ where C is the strongly connected component containing v in G_0.
- Setup M'_{G_0}, the adjacency matrix of the skeleton graph of strongly connected components. That is, for any two C_1, C_2 in the strongly connected components of G_0, $M'_{G_0}[C_1][C_2] = 1$ if and only if threre is directed path from C_1 to C_2 in G_0.
- For each connected compoenent C of G_0 :
  - Set up $C.G$, the underlying graph of C.
  - Set up $C.\text{sparseCert} = (T_C, T_C^R)$, a pair of DFS trees where $T_C = DFSTree(C.G,u)$ and $T_C^R = DFSTree(C.G^R,u)$ where u is an arbitrary vertex in C and $C.G^R$ is the reverse of the graph $C.G$.
  - Set up $C.IN$ (respectively, $C.OUT$) the linked list of edges that are incoming (respectively, outgoing) to C.
  - For each $v \in G$: Set up $\text{HOOK}[v,C] \leftarrow C.IN$.
  - Set up T_C directed reachability tree of C in M'_G.
  - For each $v \in G_0$: Set up $\text{PARENT}[v,C]$ to be the incoming edge of C which connects $SCC[v]$ to C. That is, $\text{PARENT}[v,C] \leftarrow T_{SCC[v]}(C)$.
  - Output $\mathsf{state}_0 = (SCC_0, M'_{G_0}, \{C.\text{sparseCert}, C.IN, C.OUT, C.G\}_{C \in SCC(G_0)}, \text{HOOK}_0, \text{PARENT}_0)$

Deletion: Let (i,j) be the edge to be deleted. Let $C_i = SCC_{\mathsf{curr}}[i]$ and $C_j = SCC_{\mathsf{curr}}[j]$. Let the latest graph be denoted at $G_{\mathsf{curr}} = (V_{\mathsf{curr}}, E_{\mathsf{curr}})$ and the current state be

$\mathsf{state}_{\mathsf{curr}} = (SCC_{\mathsf{curr}}, M'_{G_{\mathsf{curr}}}, \{C.\text{sparseCert}, C.IN, C.OUT, C.G\}_{C \in SCC(G_{\mathsf{curr}})}, \text{HOOK}_{\mathsf{curr}}, \text{PARENT}_{\mathsf{curr}})$
Also let $,\text{HOOK} = \text{HOOK}_{\mathsf{curr}}$ and $,\text{PARENT} = \text{PARENT}_{\mathsf{curr}}$

If $C_i \neq C_j$, $\mathsf{Ital.Delete}(\mathsf{state}_{\mathsf{curr}}, (i,j))$.
Else if $(i,j) \in C_i.\text{sparseCert}$

- Let $C = C_i$. Set $C.G_{\mathsf{curr}}[(i,j)].\mathsf{state} = \text{deleted}$, $G_{\mathsf{curr}}[(i,j)].\mathsf{state} = \text{deleted}$.
- Compute connected components of $C.G_{\mathsf{curr}}$, say $C_1, \ldots, C_\ell$
- If C is not broken (i.e., $\ell = 1$), then recompute $C.\text{sparseCert}$.
- Else, For each $p \in [\ell]$
  - Compute $C_p.G$, $C_p.OUT$ and $C_p.\text{sparseCert}$
  - Update $\text{PARENT}[v,C_p]$ for every $v \in V[G_{\mathsf{curr}}]$.
  - Update $SCC_{\mathsf{curr}}[v]$ for every $v \in C_p$
  - Update $C_p.IN$

– Update $\text{HOOK}[C_p, v]$ to first edge common to both $\text{HOOK}[v, C]$ and $C_p.IN$, for every $v \in V[G_{\mathsf{curr}}]$.

• Update $M'_{G_{\mathsf{curr}}}$.

Else, let $C = C_i$, call $C.G_{\mathsf{curr}}.\mathsf{Delete}(i, j)$, and set $G_{\mathsf{curr}}[(i, j)].\mathsf{state} = \mathsf{deleted}$.

Query: On query (u, v) do the following:

• If $\text{PARENT}[u, SCC_{\mathsf{curr}}[v]] \neq NULL$ return true.

• Else, return false.

Now we are ready to describe the our construction of dZKR. Our construction extends the algorithm by [139]. The main intuition behind the construction is the following. The algorithm maintains two matrices, (atmost) $n \times n$ matrices (where n is the number of vertices in the underlying graph), PARENT and PATH. We are giving a simplified version of the algorithm here, to give the intuition. Each matrix has an entry for each u, v pair which is either empty (denoted as $\perp$) or not. The invariant that the dynamic algorithm maintains, is the following: there is a path from vertex u to v if either $\text{PARENT}[u, v] \neq \perp$ or $\text{PATH}[u, v] \neq \perp$. Therefore, there is no path from u to v if and only if $\text{PARENT}[u, v] = \perp$ and $\text{PATH}[u, v] = \perp$. We will use this characterization to generate proof for reachability queries. We will use cryptographic commitments and proof-of-knowledge protocol to prove the statements above in zero knowledge. To ensure that the server uses the correct commitments and to protect against replay attack (i.e., to prevent the server for producing stale answers), we use aggregate signature on the commitment schemes.

Zero Knowledge Reachability on Dynamic Graphs (dZKR)

Notation: Let $G = (V, E)$ be a graph whose vertex set is denoted by V and edge set is denoted by E. For simplicity, we assume that the queries are on publicly known ID's of the vertices present in the graph. If we allow queries on non-existent nodes as well, we will first need to prove that the queried ID is indeed present/absent in the graph. This can be easily proven using ZK-Accumulator [152] or ZK-PSR systems [122].

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{GenKey}(1^\lambda)$**:** Pick a collision resistant cryptographic hash function H. Run the key generation algorithm for any unforgeable signature scheme to compute $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{vk}_{\mathsf{Sig}}) \leftarrow \mathsf{AGG.GenKey}(1^\lambda)$. Run $(\mathsf{par}) \leftarrow \mathsf{COM.GenKey}(1^\lambda)$. Set $\mathsf{sk} := \mathsf{sk}_{\mathsf{Sig}}, \mathsf{vk} := (\mathsf{vk}_{\mathsf{Sig}}, H, \mathsf{par})$.

$(\mathsf{state}_0, \mathsf{digest}_C, \mathsf{digest}_S) \leftarrow \mathsf{Setup}(\mathsf{sk}, \mathsf{vk}, G = (V, E))$**:** Given an input graph $G_0 = (V_0, E_0)$,

• $\mathsf{DD}_0 \leftarrow \mathsf{Frig.Setup}(G_0)$, where $\mathsf{DD}_0 = (SCC_0, M'_{G_0}, \{C.\text{sparseCert}, C.IN, C.OUT, C.G\}_{C \in SCC(G_0)}, \text{HOOK}_0, \text{PARENT}_0)$.

• For each $C \in SCC(G_0)$, assign a unique id, $\mathsf{ID}_C \leftarrow H(C)$.

• For each $v \in V_0$, compute $\mathsf{com}_1^v \leftarrow \mathsf{COM.Com}(v; r_1^v), \mathsf{com}_2^v \leftarrow \mathsf{COM.Com}(\mathsf{ID}_C; r_2^v)$ where $SCC_0[v]$ points to C. Then compute $\sigma_{v,C} \leftarrow \mathsf{AGG.Sign}_{\mathsf{sk}_{\mathsf{Sig}}}(\mathsf{com}_1^v, \mathsf{com}_2^v)$.

• Let $\mathsf{com}_V = \left\{\mathsf{com}_1^v, r_1^v, \mathsf{com}_2^v, r_2^v, \sigma_{v,C}\right\}_{v \in V_0}$

• For each entry (i, j) in PARENT_0, compute $\mathsf{com}_i \leftarrow \mathsf{COM.Com}(i; r_i), \mathsf{com}_j \leftarrow$

$\mathsf{COM.Com}(j;r_j), \mathsf{com}_{val} \leftarrow \mathsf{COM.Com}(val;r_{val})$ where $val = 0$ if $\text{PARENT}[i,j] = NULL$, $val = 1$, otherwise.

- Compute $\sigma_{i,j,parent} \leftarrow \mathsf{AGG.Sign}_{\mathsf{sk_{Sig}}}(\mathsf{com}_i, \mathsf{com}_j, \mathsf{com}_{val})$.
- Let $\mathsf{com}_{parent} = \left\{\mathsf{com}_i, r_i, \mathsf{com}_j, r_j, \mathsf{com}_{val}, r_{val}, \sigma_{i,j,parent}\right\}_{(i,j)\in \text{PARENT}_0}$
- Set $S_0 \leftarrow \phi$ and fix parameter $\mathbb{T}$.
- Set up a matrix PATH of dimension $|V_0| \times |V_0|$ defined over $\mathbb{Z}_{\mathbb{T}}$ initialized to all zeroes. Let $\text{PATH}_0 = \text{PATH}$.
- For each entry (i,j) in PATH_0, compute $\mathsf{com}_i \leftarrow \mathsf{COM.Com}(i;r_i), \mathsf{com}_j \leftarrow \mathsf{COM.Com}(jr_j), \mathsf{com}_{val} \leftarrow \mathsf{COM.Com}(val;r_{val})$ where $val = 0/1$
- Compute $\sigma_{i,j,path} \leftarrow \mathsf{AGG.Sign}_{\mathsf{sk_{Sig}}}(\mathsf{com}_i, \mathsf{com}_j, \mathsf{com}_{val})$.
- Let $\mathsf{com}_{path} = \left\{\mathsf{com}_i, r_i, \mathsf{com}_j, r_j, \mathsf{com}_{val}, r_{val}, \sigma_{i,j,path}\right\}_{(i,j)\in \text{PATH}_0}$
- Aggregate all the signatures generated using $\mathsf{AGG.AggSign}$. Let σ_{agg} denote the aggregate signature.
- Set $\mathsf{state}_0 := (\mathsf{DD}_0, S_0, G_0, \text{PATH}_0, \mathbb{T}, \mathsf{ID}_{C\forall C\in SCC(G_0)}, \sigma_{\mathsf{agg}}), \mathsf{digest}_S := (\mathsf{com}_V, \mathsf{com}_{parent}, \mathsf{com}_{path}, \sigma_{\mathsf{agg}})$ and $\mathsf{digest}_C = (\mathsf{vk}, \sigma_{\mathsf{agg}})$.

$\underline{(\mathsf{ans}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{digest}_S^t, G_t, \delta)}$**:** On a query $\delta = (u,v)$, do the following:

- Let $SCC_t[v]$ points to C. Look up $\sigma_{v,C}$ and the corresponding commitments $\mathsf{com}_1, \mathsf{com}_2$ from com_V
- Look up $\text{PARENT}[u, SCC_{\mathsf{curr}}[v]]$ and the corresponding commitments and signature from com_{parent}. Let us denote these as $(\mathsf{com}_3, \mathsf{com}_4, \mathsf{com}_5, \sigma_{u,\mathsf{ID}_{SCC_{\mathsf{curr}}[v]},parent})$.
- Look up $\text{PATH}[u][v]$ and the corresponding commitments and signature from com_{path}. Let us denote these as $(\mathsf{com}_6, \mathsf{com}_7, \mathsf{com}_8, \sigma_{u,v,path})$.
- Let r_i be the openings of com_i. Recall that
  - $\mathsf{com}_1, \mathsf{com}_7$ are commitments to v
  - $\mathsf{com}_2, \mathsf{com}_4$ are commitments to $\mathsf{ID}_{SCC_{\mathsf{curr}}[v]}$
  - $\mathsf{com}_3, \mathsf{com}_6$ are commitments to u.
  - $\mathsf{com}_5, \mathsf{com}_8$ are commitments to 0 or some $x \in \mathbb{Z}_p^*$ depending on the corresponding entries in $\text{PARENT}_t, \text{PATH}_t$ matrices.
  - Compute $\sigma_{\mathsf{ans}} \leftarrow \sigma_{v,C} \times \sigma_{u,\mathsf{ID}_{SCC_{\mathsf{curr}}[v]}} \times \sigma_{u,v,path}$
  - Let M denote the the set of the rest of the messages whose signture was not aggregated into σ_{ans}. Compute $H_{\mathsf{compl}} \leftarrow \prod_{m\in M} H(m)$ (the computation time can be done in to $2\log s$ time where s is the total number of signatures in the system using the technique from [78]).
- Using NIZK, generate proofs for equality as follows.

$$\pi_1 := \mathsf{PoK}\{(\underline{x, r_2, r_4} : \mathsf{com}_2 = \mathsf{COM.Com}(x, r_2) \wedge \mathsf{com}_4 = \mathsf{COM.Com}(x, r_4)\}$$
$$\pi_2 := \mathsf{PoK}\{(\underline{y, r_1, r_7} : \mathsf{com}_1 = \mathsf{COM.Com}(y, r_1) \wedge \mathsf{com}_7 = \mathsf{COM.Com}(y, r_7)\}$$
$$\pi_3 := \mathsf{PoK}\{(\underline{z, r_3, r_6} : \mathsf{com}_3 = \mathsf{COM.Com}(z, r_3) \wedge \mathsf{com}_6 = \mathsf{COM.Com}(z, r_6)\}$$

- If $\mathsf{Frig.Query}(\mathsf{state}_{\mathsf{curr}}, (u,v)) = \mathsf{true}$ or $\text{PATH}[u][v] \neq 0$, set $\mathsf{ans} := \mathsf{true}$ and compute the following:

1. Using NIZK, generate proofs that either of $\mathsf{com}_5, \mathsf{com}_8$ opens to 1.

   $\pi_4 := \mathsf{PoK}\{(\underline{w_1, w_2, r_5, r_8} : \mathsf{com}_5 = \mathsf{COM.Com}(w_1, r_5) \wedge \mathsf{com}_8 = \mathsf{COM.Com}(w_8, r_8) \wedge (w_1 = 1 \vee w_2 = 1)\}$

2. Set $\mathsf{proof} := (\mathsf{com}_1, r_1, \mathsf{com}_2, \mathsf{com}_3, r_3, \mathsf{com}_4, \mathsf{com}_5, \mathsf{com}_6, \mathsf{com}_7, \mathsf{com}_8, \pi_1, \pi_2, \pi_3, \pi_4, \sigma_{\mathsf{ans}}, H_{\mathsf{compl}})$

- If $\mathsf{Frig.Query}(\mathsf{state}_{\mathsf{curr}}, (u,v)) = \mathsf{false}$ and $\mathrm{PATH}[u][v] = 0$, set $\mathsf{ans} := \mathit{false}$ and set $\mathsf{proof} := (\mathsf{com}_1, r_1, \mathsf{com}_2, \mathsf{com}_3, r_3, \mathsf{com}_4, \mathsf{com}_5, r_5, \mathsf{com}_6, \mathsf{com}_7, \mathsf{com}_8, r_8, \pi_1, \pi_2, \pi_3, \sigma_{\mathsf{ans}}, H_{\mathsf{compl}})$

$\underline{(\mathsf{accept/reject}) \leftarrow \mathsf{Verify}(\mathsf{digest}_C^t, \delta, \mathsf{ans}, \mathsf{proof})}$**:** On query $\delta = (u,v)$, if $\mathsf{ans} = \mathit{true}$, parse proof as $(\mathsf{com}_1, r_1, \mathsf{com}_2, \mathsf{com}_3, r_3, \mathsf{com}_4, \mathsf{com}_5, \mathsf{com}_6, \mathsf{com}_7, \mathsf{com}_8, \pi_1, \pi_2, \pi_3, \pi_4, \sigma_{\mathsf{ans}}, H_{\mathsf{compl}})$. Else parse proof as $(\mathsf{com}_1, r_1, \mathsf{com}_2, \mathsf{com}_3, r_3, \mathsf{com}_4, \mathsf{com}_5, r_5, \mathsf{com}_6, \mathsf{com}_7, \mathsf{com}_8, r_8, \pi_1, \pi_2, \pi_3, \sigma_{\mathsf{ans}}, H_{\mathsf{compl}})$. Now do the following verification and accept if all of them pass. Reject, otherwise.

- Verify $\mathsf{COM.Verify}(\mathsf{par}, \mathsf{com}_1, v, r_1), \mathsf{COM.Verify}(\mathsf{par}, \mathsf{com}_3, u, r_3)$
- Let $M_1 = (\mathsf{com}_1, \mathsf{com}_2), M_2 = (\mathsf{com}_3, \mathsf{com}_4, \mathsf{com}_5), M_3 = (\mathsf{com}_6, \mathsf{com}_7, \mathsf{com}_8)$
- Verify $e(\sigma_{\mathsf{ans}}, g) = e(\prod_{i=1}^{3} H(M_i), \mathsf{vk})$.
- Verify $e(\sigma_{\mathsf{agg}}, g) = e(\sigma_{\mathsf{ans}}, g) \times e(H_{\mathsf{compl}}, \mathsf{vk}_{\mathsf{Sig}})$
- Verify π_1, π_2, π_3
- If $\mathsf{ans} = \mathsf{true}$, verify π_4. Else verify $\mathsf{COM.Verify}(\mathsf{par}, \mathsf{com}_5, 0, r_5), \mathsf{COM.Verify}(\mathsf{par}, \mathsf{com}_8, 0, r_8)$.

$\underline{(\mathsf{state}_{t+1}, \mathsf{digest}_C^{t+1}, \mathsf{Upd}_{t+1}, G_{t+1}, u_t) \leftarrow \mathsf{UpdateOwner}(\mathsf{sk}, \mathsf{state}_t, \mathsf{digest}_C^t, \mathsf{digest}_S^t, G_t, u_t, \mathsf{SID}_t)}$ Let the latest graph be denoted at $G_{\mathsf{curr}} = (V_{\mathsf{curr}}, E_{\mathsf{curr}})$. Let the current state be $\mathsf{state}_t = (\mathsf{DD}_{\mathsf{curr}}, S_{\mathsf{curr}}, G_{\mathsf{curr}}, \mathrm{PATH}_{\mathsf{curr}}, \mathbb{T}, \mathsf{ID}_{C \forall C \in SCC(G_{\mathsf{curr}})}), \mathsf{digest}_S^t := (\mathsf{com}_V, \mathsf{com}_{parent}, \mathsf{com}_{path})$. Let $\mathsf{SID}_t = \{(u,v)\}$ such that u, v has been queried since the last update. There are two types of update u_t: edge insertions and edge deletions.

Insertion: Let E_v be the edges centered around v that are being inserted.

- Set $E_{\mathsf{curr}} \leftarrow E_{\mathsf{curr}} \cup E_v$, $S_{\mathsf{curr}} \leftarrow S_{\mathsf{curr}} \cup v$
- If $|S_{\mathsf{curr}}| > \mathbb{T}$ call setup on G_{curr}.
- Else construct shortest path in-reachability tree $In(v)$ and out-reachability tree $Out(v)$. For every $x \in In(v)$ and for every $y \in Out(v)$, Update $\mathrm{PATH}[x][y] = \mathrm{PATH}[x][y] + 1$. Regenerate the commitments corresponding to entry (x,y) and regenerate the signatures. Let Δ_{insert} denote the fresh commitments and signatures.

Deletion: Let E' be the set of edges to be deleted.

- $E_{\mathsf{curr}} \leftarrow E_{\mathsf{curr}} \setminus E'$
- Delete it from deletion structure, $\mathsf{Frig.Delete}(\mathsf{state}_t, E')$.
- For each $w \in S_{\mathsf{curr}}$:
  - Set $In_{old}(w) = In(w)$, $Out_{old}(w) = Out(w)$
  - Rebuild trees $In(w), Out(w)$
  - For every $x \in In(w)$ and for every $y \in Out(w)$, Update $\mathrm{PATH}[x][y] = \mathrm{PATH}[x][y] + 1$
  - For every $x \in In_{old}(w)$ and for every $y \in Out_{old}(w)$, such that either $x \notin In(v)$ or $y \notin Out(v)$ or both, Update $\mathrm{PATH}[x][y] = \mathrm{PATH}[x][y] - 1$
- Regenerate the commitments corresponding to entries (x,y) in $\mathrm{PARENT}, \mathrm{PATH}$ and entries in $SCC[.]$ that changed. Regenerate the signatures. Let Δ_{delete} denote the fresh commitments and signatures.

- Finally, regenerate commitments and signatures for all tuples that contain vertices in SID_t. Let Δ_{SID} denote the fresh commitments and signatures. Update σ_{agg} by replacing the stale signatures with the fresh signatures (updating each signature takes 2 exponentiations).
- Update state_{t+1}, set $\mathsf{Upd}_{t+1} := (\Delta_{insert}, \Delta_{delete}, \Delta_{\mathsf{SID}})$ and $\mathsf{digest}_C^{t+1} := (\mathsf{vk}, \sigma_{\mathsf{agg}})$.

$(\mathsf{digest}_S^{t+1}, G^{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{digest}_S^t, \mathsf{Upd}_{t+1}, G_t, u_t)$ The server updates the graph using the update information u_t. It takes the update string $\mathsf{Upd}_{t+1} = (\Delta_{insert}, \Delta_{delete}, \Delta_{\mathsf{SID}})$ and update $\mathsf{com}_{parent}, \mathsf{com}_{path}, \mathsf{com}_V$ using commitments and signatures in Upd_{t+1}.

Time and Space Complexity: Here we analyze the amortized cost of updates and query. It is easy to see that the proof size and the verification time is $O(1)$

- **Setup**: The time taken by the original algorithm by Zwick et al. [139] for setup is $O(n_0^2 + m_0 n_0)$. We additionally set up the integer path matrix, which is initialized to all zeroes as there are no intermediate vertices. Hence the additional time needed for set up is just $O(n_0^2)$. Hence the total time complexity of setup is $O(n_0^2 + m_0 n_0)$.
  The space used by Zwick et al. [139] is $O(n_0^2)$. The modified algorithm uses an extra $O(n_0^2)$ space for storing the PATH matrix.
- **Deletion**: We define $\mathsf{SID} = \sum_{t \in \mathbb{T}, t \text{ is time of an update}} \mathsf{SID}_t$. Recall that, $\mathsf{SID}_t = \{(u,v)\}$ such that u, v has been queried since the last update. Hence SID thus defined is the total number of queries in a phase.
  The deletion in the dynamic deletion data structure takes $O(nm)$ cost in total for any number of deletions. Since in a phase there are at least $\mathbb{T}$ update operations, each update operation (insert/delete) is charged $O(\frac{nm}{\mathbb{T}})$ in an amortized fashion, towards maintaining the decremental delete structure of [69].
  Each delete of a set of edges connected to a center vertex of a phase, Zwick et al.'s algorithm rebuilds the reachability trees for each central vertex, incurring a cost of $O(m)$ for each central vertex. Since there are at most $\mathbb{T}$ central vertices, this takes at most $O(m\mathbb{T})$ time [139]. Updating the PATH matrix takes another $O(n^2)$ time for each central vertex. This is because, for each central vertex w we go through all $x \in IN(w)$ and all $y \in Out(w)$, and there are at most n vertices in either trees. Thus for all the central vertices it takes $O(n^2\mathbb{T})$. Since there are SID queries, if we charge each query cost c towards accounting for this cost, then the cost of rebuilding PATH matrix for each delete is $n^2\mathbb{T} - (\mathsf{SID} \cdot c)$.
  Since a delete is an update operation, one needs to recompute the signatures for all the queries in SID_t, but this can be accounted for by adding an $O(1)$ cost to each query itself, as $\mathsf{SID} = \sum_t \mathsf{SID}_t$.
  Hence amortized cost of each delete is $O(nm/\mathbb{T} + m\mathbb{T} + n^2\mathbb{T} - (\mathsf{SID} \cdot c))$.
- **Insertion**: For each central vertex that is inserted, the reachability trees are constructed by Zwick et al.'s algorithm. This takes $O(m)$ for a given vertex (i.e., a given insert operation) [139]. In our algorithm, the PATH matrix has to be updated, and this takes $O(n^2)$ time. This is because, if w is the new vertex to be inserted, we go through all $x \in IN(w)$ and all $y \in Out(w)$ to update PATH matrix, and there are at most n vertices in either trees.
  Since a insert is an update operation, one needs to recompute the signatures for all the queries in SID_t, but this can be accounted for by adding an $O(1)$ cost to each query itself, as $\mathsf{SID} = \sum_t \mathsf{SID}_t$.
  Thus, the total amortized cost of an insert is $O(m + n^2)$.

- **Query**: Each query is $O(\log n)$ time as it is two matrix look ups and one H_{compl} computation. We add another $O(1)$ cost to each query for recomputing SID_t. Also, we are adding a cost c to each query to account for each delete in the amortized analysis. Since there are at most $\mathbb{T}$ deletes in a phase, this adds $c \cdot \mathbb{T}$ cost to each query. Hence the amortized cost of a query is $O(\log n) + (c \cdot \mathbb{T})$.

If we set the parameters $\mathbb{T}$ and c to $\sqrt{n}$ and 0 respectively, to get the following amortized costs :

- **Setup**: Time : $O(n_0^2 + n_0 m_0)$. Space : $O(n_0^2 + n_0 m_0)$.
- **Deletion**: Amortized time per delete : $O(\sqrt{n}m + n^{2.5})$.
- **Insertion**: Amortized time per insert : $O(m + n^2)$
- **Query**: Amortized time per query : $O(\log n)$.

Security: Completeness is easy to see. The proof of soundness directly follows from the binding property of COM, the unforgeability of the signature scheme AGG and the knowledge-soundness of the underlying proof system PoK. Here we prove that our construction satisfies zero-knowledge property by constructing a simulator Sim. Sim proframs the Random Oracle H. It runs the key generation algorithm for the aggregate signature scheme to compute $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{vk}_{\mathsf{Sig}}) \leftarrow \mathsf{AGG.GenKey}(1^\lambda)$. and $(\mathsf{par}) \leftarrow \mathsf{COM.GenKey}(1^\lambda)$. Set $\mathsf{sk} := \mathsf{sk}_{\mathsf{Sig}}, \mathsf{vk} := (\mathsf{vk}_{\mathsf{Sig}}, H, \mathsf{par})$. Sim picks a random $r \leftarrow \mathbb{Z}_p^*$, sets $\sigma_{\mathsf{agg}} = g^r$ and sends $\mathsf{digest}_C^0 = (\mathsf{vk}, \sigma_{\mathsf{agg}})$ to the adversary $\mathcal{B}$.

For every query Sim receives $F(q)$ as input informing it whether q is an update or a (reachability) query. In order to simulate consistent answers to the queries, it maintains a table of its previous answers (in the order of their arrival).

For an update query, Sim picks fresh randomness $r' \leftarrow \mathbb{Z}_p^*$, sets $\sigma_{\mathsf{agg}} = g^{rr'}$ and send $(\mathsf{vk}, \sigma_{\mathsf{agg}})$ to $\mathcal{B}$.

For a reachability query (u, v), Sim receives the answer ans from the oracle. It finds the most recent query that included (u, v). If there was no update (on (u, v) or other elements) after that, then Sim uses the same randomness as it used last time. Otherwise, Sim picks fresh randomness and simulates all the corresponding commitments (that are returned in the real proof), generates signature on them using its signature key sk and simulates the PoK's. Sim simulates H_{compl} as σ_{agg}/C where $C = \prod_{\mathsf{com} \in \mathsf{proof}} H(\mathsf{com})$ This concludes the description of the Sim. Sim produces a sequence of answers and proofs that are identically distributed to the one produced by the real challenger (i.e., the owner and the server) by the hiding property of the commitment scheme and the zero-knowledge of the underlying proof system. This concludes the proof.

We summarize the performance of our dZKR scheme in the Theorem 7.

Theorem 7. *Our zero knowledge dynamic graph reachability scheme,* dZKR*, satisfies completeness (Definition 13), soundness (Definition 14) and zero-knowledge (Definition 15). On a graph G with n_0 initial vertices, m_0 initial edges, n current vertices, and m current edges (after updates), the asymptotic performance of our scheme is as follows. The setup time and the owners's storage (to remember its state) are each $O(n_0^2 + n_0 m_0)$. The server's storage is $O(n^2)$. The amortized insertion time is $O(m + n^2)$, the amortized deletion time is $O(\sqrt{n}m + n^{2.5})$, and the amortized query time is $O(\log n)$. The proof size and verification time are each $O(1)$.*

5.6 Conclusion

In this work, we designed lightweight cryptographic primitive to answer reachability queries on outsourced general graphs (both static and dynamic). To design the primitive for static graphs, we extended the zero-knowledge set algebraic construction from [152] and this can be of independent interest. One interesting direction will be to extend our framework to support more complex graph theoretic queries keeping the efficiency of our scheme. Prototype implementation of our schemes is left as future work.

... ...

Bibliography

[1] Amazon neptune. https://aws.amazon.com/neptune.

[2] Boston email data set. https://data.cityofboston.gov/Transportation/Boston-Taxi-Data/ypqb-henq. Accessed on 22 Feb 2016.

[3] Enron email data set. https://www.cs.cmu.edu/~./enron/. Accessed on 22 Feb 2016.

[4] GeoHash library. Accessed on 22 Feb 2016.

[5] Graphdb. http://graphdb.ontotext.com.

[6] Neo4j. https://neo4j.com.

[7] Orientdb. http://orientdb.com.

[8] Synthetic datasets for range queries used in this paper.

[9] Titan. http://titan.thinkaurelius.com.

[10] Efficient identification and signatures for smart cards. In *CRYPTO*, 1989.

[11] Clusion. https://github.com/encryptedsystems/Clusion, 2016.

[12] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. In *Theory of Cryptography - Proc. TCC*, 2012.

[13] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*, 2000.

[14] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In *Advances in Cryptology – Proc. ASIACRYPT*, 2012.

[15] Man Ho Au, Patrick P. Tsang, Willy Susilo, and Yi Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *CT-RSA*, 2009.

[16] Maxim Babenko, Andrew V. Goldberg, Anupam Gupta, and Viswanath Nagarajan. Algorithms for hub label optimization. *ACM Trans. Algorithms*, 13(1):16:1–16:17, November 2016.

[17] Niko Baric and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, 1997.

[18] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.

[19] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *EUROCRYPT*, 1994.

[20] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In *ASIACCS*, 2012.

[21] Sören Bleikertz, Matthias Schunter, Christian W Probst, Dimitrios Pendarakis, and Konrad Eriksson. Security audits of multi-tier virtual infrastructures in public infrastructure clouds. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pages 93–102. ACM, 2010.

[22] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, 2004.

[23] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Proc. Annual Int. Conf. on Theory and Applications of Cryptographic Techniques*, EUROCRYPT, pages 440–456, Berlin, Heidelberg, 2005. Springer-Verlag.

[24] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, 2003.

[25] Raphael Bost. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1143–1154. ACM, 2016.

[26] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. Verifiable dynamic symmetric searchable encryption: Optimality and forward security. *IACR Cryptology ePrint Archive*, 2016:62, 2016.

[27] Ernest F. Brickell, David Chaum, Ivan B. Damgaard, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *Advances in Cryptology CRYPTO*, volume 293 of *LNCS*, pages 156–166. Springer Berlin Heidelberg, 1988.

[28] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In *Proc. Applied Cryptography and Network Security ACNS*, 2010.

[29] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In *CCS*, 2000.

[30] Philippe Camacho and Alejandro Hevia. On the impossibility of batch update for cryptographic accumulators. In *LATINCRYPT*. 2010.

[31] Philippe Camacho, Alejandro Hevia, Marcos Kiwi, and Roberto Opazo. Strong accumulators from collision-resistant hashing. In *Information Security*. 2008.

[32] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT*, 2008.

[33] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography*, 2009.

[34] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002.

[35] Alina Campan, Yasmeen Alufaisan, and Traian Marius Truta. Preserving communities in anonymized social networks. *Trans. Data Privacy*, 8(1):55–87, December 2015.

[36] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable set operations over outsourced databases. In *PKC*, 2014.

[37] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[38] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 353–373. Springer, 2013.

[39] Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In *Advances in Cryptology – Proc. EUROCRYPT*, 2008.

[40] Ee-Chien Chang, Chee Liang Lim, and Jia Xu. Short redactable signatures using random trees. In *Topics in Cryptology — Proc. Cryptographer's Track at the RSA Conference CT-RSA*, 2009.

[41] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In Klaus Jansen and Samir Khuller, editors, *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1913 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2000.

[42] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In *Advances in Cryptology – Proc. EUROCRYPT*, 2005.

[43] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[44] Sanjit Chatterjee and Alfred Menezes. On cryptographic protocols employing asymmetric pairings - the role of ψ revisited. *Discrete Applied Mathematics*, 2011.

[45] Fei Chen and Alex X. Liu. Privacy- and integrity-preserving range queries in sensor networks. *IEEE/ACM Trans. Netw.*, 20(6):1774–1787, December 2012.

[46] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 937–946, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[47] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. In *SODA*, 2002.

[48] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. In *SODA*, 2002.

[49] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[50] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *FC*, 2010.

[51] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 79–88. ACM, 2006.

[52] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, 2009.

[53] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008.

[54] Hermann de Meer, Manuel Liedel, Henrich C. Pöhls, and Joachim Posegga. Indistinguishability of one-way accumulators. In *Technical Report MIP-1210, Faculty of Computer Science and Mathematics (FIM), University of Passau*, 2012.

[55] Hermann de Meer, Henrich C. Pöhls, Joachim Posegga, and Kai Samelin. Redactable signature schemes for trees with signer-controlled non-leaf-redactions. In *E-Business and Telecommunications*, 2014.

[56] C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure: breaking through the o(n2) barrier. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 381–389, 2000.

[57] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, 2015.

[58] Premkumar T. Devanbu, Michael Gertz, Charles U. Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In *Conference on Data and Applications Security and Privacy DBSec*, pages 101–112, 2000.

[59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[60] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM CCS*, 2013.

[61] Cynthia Dwork. Differential privacy. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 338–340. Springer, 2011.

[62] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, volume 9327 of *Lecture Notes in Computer Science*, pages 123–145. Springer, 2015.

[63] Prastudy Fauzi, Helger Lipmaa, and Bingsheng Zhang. Efficient non-interactive zero knowledge arguments for set operations. In *FC*, 2014.

[64] Nelly Fazio and Antonio Nicolosi. Cryptographic accumulators: Definitions, constructions and applications. In *Technical Report. Courant Institute of Mathematical Sciences, New York University*, 2002.

[65] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1987.

[66] Ben A. Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M. Bellovin. Malicious-client security in blind seer: A scalable private DBMS. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 395–410. IEEE Computer Society, 2015.

[67] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[68] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[69] Daniele Frigioni, Tobias Miller, Umberto Nanni, and Christos Zaroliagis. An experimental study of dynamic algorithms for transitive closure. *J. Exp. Algorithmics*, 6, December 2001.

[70] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, June 1998.

[71] Juan A. Garay, Philip MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In *EUROCRYPT*, 2003.

[72] William I. Gasarch. A survey on private information retrieval (column: Computational complexity). *Bulletin of the EATCS*, 82:72–107, 2004.

[73] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In *Advances in Cryptology – Proc. ASIACRYPT*, volume 2501, pages 548–566. 2002.

[74] GephiDatasets. Internet dataset, 2006. a symmetrized snapshot of the structure of the Internet at the level of autonomous systems, reconstructed from BGP tables, `https://gephi.org/datasets/internet_routers-22july06.gml.zip`.

[75] Esha Ghosh, Olga Ohrimenko, Dimitrios Papadopoulos, Roberto Tamassia, and Nikos Triandopoulos. Zero-knowledge accumulators and set operations. *IACR Cryptology ePrint Archive*, 2015:404, 2015.

[76] Esha Ghosh, Olga Ohrimenko, and Roberto Tamassia. Efficient verifiable range and closest point queries in zero-knowledge. *Privacy Enhancing Technologies Symposium (PETs)*, 2016(4).

[77] Esha Ghosh, Olga Ohrimenko, and Roberto Tamassia. Authenticated range \& closest point queries in zero-knowledge. *IACR Cryptology ePrint Archive*, 2015:1183, 2015.

[78] Esha Ghosh, Olga Ohrimenko, and Roberto Tamassia. Zero-knowledge authenticated order queries and order statistics on a list. In *ACNS*, 2015.

[79] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[80] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably preventing DNSSEC zone enumeration. In *The Network and Distributed System Security Symposium NDSS*, 2015.

[81] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *ACM Symposium on Theory of Computing (STOC '85)*, pages 291–304. ACM, 1985.

[82] Michael T. Goodrich, Duy Nguyen, Olga Ohrimenko, Charalampos Papamanthou, Roberto Tamassia, Nikos Triandopoulos, and Cristina Videira Lopes. Efficient verification of web-content searching through authenticated web crawlers. *PVLDB*, 5(10), 2012.

[83] Michael T. Goodrich, Roberto Tamassia, and Nikos Triandopoulos. Efficient authenticated data structures for graph connectivity and geometric search problems. *CoRR*, abs/0908.4116, 2009.

[84] Jonathon S. Hare, Sina Samangooei, and David P. Dupplaw. Openimaj and imageterrier: Java libraries and tools for scalable multimedia analysis and indexing of images. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 691–694, New York, NY, USA, 2011. ACM.

[85] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *J. Cryptology*, 25(3), 2012.

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

[87] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[88] G. F. Italiano. Amortized efficiency of a path retrieval data structure. *Theor. Comput. Sci.*, 48(2-3):273–281, December 1986.

[89] Giuseppe F. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Inf. Process. Lett.*, 28(1):5–11, May 1988.

[90] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, 2009.

[91] Ruoming Jin, Ning Ruan, Yang Xiang, and Haixun Wang. Path-tree: An efficient reachability indexing scheme for large directed graphs. *ACM Trans. Database Syst.*, 36(1):7:1–7:44, March 2011.

[92] Seny Kamara. Restructuring the NSA metadata program. In *Financial Cryptography and Data Security – FC Workshops*, pages 235–247, 2014.

[93] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 94–124, 2017.

[94] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, volume 7859 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2013.

[95] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and*

Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012, pages 965–976. ACM, 2012.

[96] Florian Kerschbaum. Oblivious outsourcing of garbled circuit generation. In Roger L. Wainwright, Juan Manuel Corchado, Alessio Bechini, and Jiman Hong, editors, *Proc. ACM Symp. on Applied Computing*, pages 2134–2140. ACM, 2015.

[97] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 81–, Washington, DC, USA, 1999. IEEE Computer Society.

[98] Valerie King and Mikkel Thorup. A space saving trick for directed dynamic transitive closure and shortest path algorithms. In Jie Wang, editor, *Computing and Combinatorics*, pages 268–277, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[99] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, 2005.

[100] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*, 2018.

[101] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: faster verifiable set computations. In *USENIX*, 2014.

[102] Ashish Kundu, Mikhail J. Atallah, and Elisa Bertino. Leakage-free redactable signatures. In *.ACM conference on Data and application security and privacy CODASPY*, 2012.

[103] Ashish Kundu and Elisa Bertino. Structural signatures for tree data structures. *Proceedings of Very Large Data Bases PVLDB*, 1(1), 2008.

[104] Kaoru Kurosawa, Keisuke Sasaki, Kiyohiko Ohta, and Kazuki Yoneyama. Uc-secure dynamic searchable symmetric encryption scheme. In Kazuto Ogawa and Katsunari Yoshioka, editors, *Advances in Information and Computer Security - 11th International Workshop on Security, IWSEC 2016, Tokyo, Japan, September 12-14, 2016, Proceedings*, volume 9836 of *Lecture Notes in Computer Science*, pages 73–90. Springer, 2016.

[105] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[106] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 121–132, 2006.

[107] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *ACNS*, 2007.

[108] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *Theory of Cryptography - Proc. TCC*, 2010.

[109] Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In *ACNS*, 2012.

[110] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[111] Moses Liskov. Updatable zero-knowledge databases. In *ASIACRYPT*, 2005.

[112] Philip MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In *EUROCRYPT*. 2004.

[113] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. GRECS: graph encryption for approximate shortest distance queries. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 504–517. ACM, 2015.

[114] R. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[115] Ralph C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[116] Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *Symposium on Foundations of Computer Science FOCS, Proceedings*, pages 80–91, 2003.

[117] Roberto Tamassia Michael T. Goodrich and Andrew Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. *DARPA Information Survivability Conference and Exposition II*, 2001.

[118] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE Symposium on Security and Privacy*, 2013.

[119] Ruggero Morselli, Samrat Bhattacharjee, Jonathan Katz, and Peter J. Keleher. Trust-preserving set operations. In *IEEE INFOCOM*, 2004.

[120] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. *Electronic Colloquium on Computational Complexity ECCC*, (043), 2002.

[121] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications*, 18(4), 2000.

[122] Moni Naor and Asaf Ziv. Primary-secondary-resolver membership proof systems. Cryptology ePrint Archive, Report 2014/905, 2014.

[123] Moni Naor and Asaf Ziv. Primary-secondary-resolver membership proof systems. In *Theory of Cryptography - Proc. TCC*, volume 9015, pages 199–228. 2015.

[124] Lan Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, 2005.

[125] Kaisa Nyberg. Commutativity in cryptography. In *1st International Trier Conference in Functional Analysis*, 1996.

[126] Kaisa Nyberg. Fast accumulated hashing. In *Fast Software Encryption*, 1996.

[127] Rafail Ostrovsky, Charles Rackoff, and Adam Smith. Efficient consistency proofs for generalized queries on a committed database. In *Automata, Languages and Programming: International Colloquium, ICALP, Proceedings*, pages 1041–1053, 2004.

[128] Dimitrios Papadopoulos, Stavros Papadopoulos, and Nikos Triandopoulos. Taking authenticated range queries to arbitrary dimensions. In *Proc.ACM Conf. on Computer and Communications Security*, CCS, pages 819–830, New York, NY, USA, 2014. ACM.

[129] Dimitrios Papadopoulos, Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Practical authenticated pattern matching with optimal proof size. *Proceedings of Very Large Data Bases PVLDB*, 8(7):750–761, 2015.

[130] Charalampos Papamanthou, Elaine Shi, Roberto Tamassia, and Ke Yi. Streaming authenticated data structures. In *Advances in Cryptology - Proc. EUROCRYPT*, pages 353–370, 2013.

[131] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *Advances in Cryptology – Proc. CRYPTO*, pages 91–110, 2011.

[132] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables based on cryptographic accumulators. *Algorithmica*, 2015.

[133] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables based on cryptographic accumulators. *Algorithmica*, 74(2):664–712, 2016.

[134] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 359–374. IEEE Computer Society, 2014.

[135] Wei Peng, Xiaofeng Hu, Feng Zhao, and Jinshu Su. A fast algorithm to find all-pairs shortest paths in complex networks. *Procedia Computer Science*, 9:557 – 566, 2012.

[136] Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *J. Artif. Intell. Res. (JAIR)*, 43:353–388, 2012.

[137] G. S. Poh, M. S. Mohamad, and M. R. Z'aba. Structured encryption for conceptual graphs. In *Advances in Information and Computer Security*, pages 105–122, 2012.

[138] F.P. Preparata, D.V. Sarwate, and ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB. *Computational Complexity of Fourier Transforms Over Finite Fields*. DTIC, 1976.

[139] L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 679–688, 2002.

[140] Kai Samelin, Henrich C. Poehls, Arne Bilzhause, Joachim Posegga, and Hermann De Meer. Redactable signatures for independent removal of structure and content. In *ISPEC*, 2012.

[141] Kai Samelin, Henrich Christopher Pöhls, Arne Bilzhause, Joachim Posegga, and Hermann de Meer. On structural signatures for tree data structures. In *Proc. Applied Cryptography and Network Security ACNS*, 2012.

[142] Tomas Sander. Efficient accumulators without trapdoor. In *ICICS*, 1999.

[143] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. Efficient query processing on spatial networks. In *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems*, GIS '05, pages 200–209, New York, NY, USA, 2005. ACM.

[144] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. Path oracles for spatial networks. *Proc. VLDB Endow.*, 2(1):1210–1221, August 2009.

[145] Bo Sheng and Qun Li. Verifiable privacy-preserving range query in two-tiered sensor networks. In *Proc. IEEE Int. Conf. on Computer Communications INFOCOM*, pages 46–50, 2008.

[146] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proc. IEEE Symp. on Security and Privacy*, pages 350–364, Washington, DC, USA, 2007. IEEE Computer Society.

[147] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[148] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[149] Roberto Tamassia. Authenticated data structures. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003*, volume 2832 of *Lecture Notes in Computer Science*, pages 2–5. Springer, 2003.

[150] **Esha Ghosh**, Michael T. Goodrich, Olga Ohrimenko, and Roberto Tamassia. Verifiable zero-knowledge order queries and updates for fully dynamic lists and trees. In *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016,*

Proceedings, volume 9841 of *Lecture Notes in Computer Science*, pages 216–236. Springer, 2016. Full version at https://eprint.iacr.org/2015/283.

[151] **Esha Ghosh**, Michael T. Goodrich, Olga Ohrimenko, and Roberto Tamassia. Verifiable zero-knowledge order queries and updates for fully dynamic lists and trees. In *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings*, volume 9841 of *Lecture Notes in Computer Science*, pages 216–236. Springer, 2016. Full version at https://eprint.iacr.org/2015/283.

[152] **Esha Ghosh**, Olga Ohrimenko, Dimitrios Papadopoulos, Roberto Tamassia, and Nikos Triandopoulos. Zero-knowledge accumulators and set algebra. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *22nd International Conference on the Theory and Application of Cryptology and Information Security, ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 67–100, 2016.

[153] **Esha Ghosh**, Olga Ohrimenko, and Roberto Tamassia. Zero-knowledge authenticated order queries and order statistics on a list. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015*, pages 149–171.

[154] **Esha Ghosh**, Olga Ohrimenko, and Roberto Tamassia. Authentiated range & closest point queries in zero-knowledge. *Proceedings on Privacy Enhancing Technologies Symposium PoPETs*, 2016(4):373–388, 2016.

[155] Qian Wang, Kui Ren, Minxin Du, Qi Cheng Li, and Aziz Mohaisen. Secgdb: Graph encryption for exact shortest distance queries with efficient updates. 2017.

[156] Yushun Wang. *TenantGuard: Scalable Runtime Verification of Cloud-Wide VM-Level Network Isolation*. PhD thesis, Concordia University, 2017.

[157] Zhiwei Wang. Improvement on Ahn et al.'s RSA P-homomorphic signature scheme. In *Security and Privacy for Emerging Areas in Communications Networks, SecureComm*, 2012.

[158] David J. Wu, Joe Zimmerman, Jérémy Planul, and John C. Mitchell. Privacy-preserving shortest path computation. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[159] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *Proc. VLDB Endow.*, 5(5):406–417, January 2012.

[160] M. L. Yiu, Y. Lin, and K. Mouratidis. Efficient verification of shortest path search via authenticated hints. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 237–248, March 2010.

[161] D. Yung and S. K. Chang. Fast reachability query computation on big attributed graphs. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3370–3380, Dec 2016.

[162] Yupeng Zhang, Charalampos Papamanthou, and Jonathan Katz. Alitheia: Towards practical verifiable graph processing. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 856–867, New York, NY, USA, 2014. ACM.

[163] Qingji Zheng and Shouhuai Xu. Verifiable delegated set intersection operations on outsourced encrypted data. *IACR Cryptology ePrint Archive*, 2014.