Verifiable Query Processing Over Outsourced Social Graph

Xin Yao, *Member, IEEE*, Rui Zhang, *Member, IEEE*, Dingquan Huang, *Student Member, IEEE*, and Yanchao Zhang, *Fellow, IEEE*

***Abstract*— Social data outsourcing is an emerging paradigm for effective and efficient access to the social data. In such a system, a third-party Social Data Provider (SDP) purchases social network datasets from Online Social Network (OSN) operators and then resells them to data consumers who can be any individuals or entities desiring social data through query interfaces. The SDP cannot be fully trusted and may return forged or incomplete query results to data consumers for various reasons, e.g., in favor of the businesses willing to pay. In this paper, we initiate the study on verifiable query processing over outsourced social graph whereby a data consumer can verify both the integrity and completeness of any query result returned by an untrusted SDP. We propose three schemes for single-attribute queries and another scheme for multi-attribute queries over outsourced social data. The four schemes all require the OSN provider to generate some cryptographic auxiliary information, based on which the SDP can construct a verification object to allow the data consumer to verify the integrity and completeness of the query result. They, however, differ in how the auxiliary information is generated and how the verification object is constructed and verified. Detailed analysis and extensive experiments using a real Twitter dataset confirm the efficacy and efficiency of the proposed schemes.**

***Index Terms*— Verifiable query processing, outsourced social graph, security.**

I. Introduction

ONLINE social networks (OSNs) are pervasive. As three exemplary popular OSNs, Twitter, Facebook, and Sina Weibo have 330 million, 2.37 billion, and 465 million monthly active users as of the first quarter of 2019, respectively. OSN users produce information at an unprecedented rate and scale. For example, there are about 3 million posts per minute on Facebook and 500 million tweets on Twitter per day. Besides facilitating social interactions, OSNs are increasingly used in massive information campaigns, public relations, political campaigns, pandemic and crisis situations, marketing, and many other public/private contexts. For instance, over 76% of businesses used social media to achieve their marketing objectives in 2014; business retailers have seen 133% increases in their revenues from social media marketing; about 71% of the consumers respond to the feedbacks and recommendations of social users regarding a particular product [2].

The traditional way to access the social network data is via the public APIs provided by each OSN itself, but the data obtained in this way are often incomplete, biased, and even incorrect. For example, Twitter provides the Filter API and Sample API to access real-time tweets as well as the Search API to retrieve historical tweets. These public APIs, however, often have very limited functionalities. For example, the Filter API and Sample API both return at most 1% random samples of all the data satisfying the query condition [3]; the data returned by the Filter API have been found strongly biased [3]; and the crawling process of the Sample API can be easily manipulated by attackers wishing to promote their social content [4]. In addition, the Search API searches against a sampling of recent tweets published in the past 7 days. There are also rate limits imposed on these public APIs. For instance, no more than 180 calls per user and 450 calls per application for the Search API can be made every 15 minutes. While the Twitter Firehose API yields 100% of all public tweets, it incurs a prohibitive monetary cost and very high server requirements to host and process the real-time tweets. The public APIs of other OSNs such as Facebook have similar constraints as well.

Social data outsourcing is an emerging paradigm for effective and efficient access to social data. A system built on this paradigm consists of a third-party Social Data Provider (SDP), many OSN operators, and numerous data consumers. The SDP purchases complete social dataset from OSN operators and offers paid data services to data consumers who can be any individuals or entities requiring the complete social data satisfying some criteria. Some popular SDPs include DataSift, Gnip, NTT Data, Brandwatch, Twazzup, CrowdEye, etc. For example, DataSift has 20 data sources until now such as Facebook, Instagram, Youtube, Tumblr, and Google+.

How much trust could a data consumer have in the data offered by these SDPs? There have been too many stories

Manuscript received January 30, 2021; revised April 29, 2021 and May 13, 2021; accepted May 26, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor N. Zhang. Date of publication June 18, 2021; date of current version October 15, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61902433, in part by the Hunan Provincial Natural Science Foundation of China under Grant 2019JJ50802, and in part by the U.S. National Science Foundation under Grant CNS-1933069, Grant CNS-1824355, Grant CNS-1651954 (CAREER), Grant CNS-1718078, and Grant CNS-1933047. The preliminary version of this article titled "Verifiable Social Data Outsourcing" was published in the proceedings of the IEEE INFOCOM, 2017 [1]. *(Corresponding authors: Yanchao Zhang; Xin Yao.)*

Xin Yao and Dingquan Huang are with the School of Computer Science and Engineering, Central South University, Changsha 410012, China (e-mail: xinyao@csu.edu.cn; fixright@csu.edu.cn).

Rui Zhang is with the Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716 USA (e-mail: ruizhang@udel.edu).

Yanchao Zhang is with the School of Electrical, Computer and Energy Engineering, Ira A. Fulton Schools of Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: yczhang@asu.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3085574, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3085574

revealing the dark side of the web industry. For instance, Yelp has "always had a complicated relationship with small businesses" according to BusinessWeek [5], and there have been wide allegations that Yelp has manipulated the online reviews based on the participation in its advertising programs [6]. As another famous example, the "death of Wei Zexi" has resulted in the official investigation of Baidu—the biggest search engine in China—which has been notoriously providing highly skewed search results due to paid placement that are unaware by the Internet users. In our context, a dishonest SDP can return incorrect query results to data consumers by adding fake data and deleting/modifying true data in favor of the businesses who would like to pay. An honest SDP may also be hacked to return incorrect query results.

In this paper, we coin the concept of *verifiable social data outsourcing* whereby a data consumer can verify the correctness of any query result returned by an untrusted SDP before making any critical decision. As the first work along this line, this paper tackles the following specific problem and leaves the investigations of other variations to future work. We view any specific OSN as an undirected social graph, in which each node corresponds to a unique OSN user and has a set of attributes (e.g., location, age, gender, and education level). An edge exists between two nodes if the two users are mutual friends, e.g., they are following each other on Twitter. We first study queries over a single attribute, which can be an equality query (e.g., age = "30"), a range query (e.g., age = "[20, 30]"), or a subset query (e.g., age = "20, 25, 30"). We then investigate queries over multiple attributes, which ask for nodes with attributes satisfying multiple criteria, e.g., age = "[20, 30]" `and` (or `or`) location = "Los Angeles". The query result is *authentic* if all the returned user profiles are present in the OSN operator's dataset and have not been tampered with and is *complete* if it contains all the nodes satisfying the query condition as well as all the associated edges among them.

We propose three schemes for verifiable single-attribute queries over outsourced social graph. The basic scheme requires each OSN provider to generate some cryptographic auxiliary information for its dataset, based on which the SDP can construct a verification object that allows the data consumer to verify the query-result integrity and completeness. The enhanced scheme significantly reduces the computation, storage, and communication overhead of the basic solution by generating the auxiliary information through grouping nodes with identical attribute values. The advanced scheme explores a memory-efficient Bloom filter to further reduce the overhead. Moreover, we propose a verifiable multi-attribute query scheme based on Cuckoo filter and verifiable set operations.

We thoroughly evaluate the four schemes on a real Twitter dataset with 1.6 million nodes and 50 million edges. Our experiments show that our basic, enhanced, advanced, and multi-attribute verification schemes can generate the auxiliary information with the size 38.85%, 3.23%, 0.14%, and 1.17% of the original social network data in 2954.26 s, 201.22 s, 208.42 s, and 916.63 s, respectively. In addition, a data consumer can verify a query result in 631.8 ms, 25.3 ms, 29.4 ms, and 114.33 ms under the proposed basic, enhanced, advanced, and multi-attribute verification schemes, respectively. Our experiment results confirm the efficacy and efficiency of the proposed schemes.

The rest of this paper is outlined as follows. Section II introduces the problem formulation. Section III presents three single-attribute query schemes. Section IV introduces a novel multi-attribute query scheme. Section V analyzes the performance of the four proposed schemes. Section VI evaluates the four proposed schemes on a real Twitter dataset. Section VII discusses related work. Section VIII concludes this paper.

II. Problem Formulation

In this section, we introduce the system, query, and adversarial models along with our design goals.

A. System Model

We consider a social data outsourcing system consisting of an SDP, many OSN operators, and many data consumers. The SDP acquires the complete social data from each OSN operator and profits by answering data queries from many data consumers. For convenience only, our subsequent illustrations focus on a single OSN operator, but similar operations should be independently performed on the data of each OSN operator.

We model the social data of the OSN operator as an undirected graph $G = (V, E)$, where V is a set of n nodes each of which corresponds to a unique OSN user, and E is the set of edges. Every edge $e_{i,j} \in E$ represents the social relationship, e.g., friendship in Facebook-like OSNs, between two nodes i and j. We will use nodes and users interchangeably hereafter. While this paper focuses on directed social graph, our schemes can be extended to support directed social graphs such as Twitter with minimal modification.

Each node has a profile consisting of w attributes specific to the OSN operator, such as location, age, education level, and hobbies. The profile of node i is denoted by $p_i = (ID_i, b_{i,1}, b_{i,2}, \ldots, b_{i,w})$, where ID_i is a unique user ID and $b_{i,j}$ is the j-th attribute value for all $1 \leq j \leq w$. An attribute value can be specified by the user him/herself or inferred by the OSN operator based on the user's posts and online interactions [7], [8]. Some attributes such as age have a numeric value by nature, while others such as location and hobbies may have non-numeric values. For the latter case, we assume that the OSN operator converts any non-numeric attribute value into a numeric one. For example, a location can be converted into a sequence of digits like a zipcode. Such conversions are done by the OSN operator in the background and totally unaware to the user. For simplicity, we assume that each attribute value $b_{i,j}$ in the social dataset represents a unique numeric value (possibly after conversion) in the range of the particular attribute. Each node is also affiliated with the data content s/he has ever generated (e.g., original posts, replies, and comments).

B. Query Model

We consider multiple types of queries over the social graph. Every query asks for a subgraph $G' = (V', E')$, where $V' \subseteq V$ is the subset of nodes whose attribute values satisfy the query

condition and $E' \subseteq E$ is the set of the edges among the nodes in V'. The SDP needs to return all the profiles $\{p_i | i \in V'\}$ as well as the edge set E'.

More specifically, we first consider queries over a single attribute, which can be an equality query (e.g., age = "30"), a range query (e.g., age = "[20, 30]"), or a subset query (e.g., age $\in \{20, 25, 30\}$). We then consider general multi-attribute queries, which can be viewed as multiple single-attribute queries combined by either logical operators AND (e.g., age = "[20, 30]" $\bigwedge$ location = "Los Angeles") or logical operators OR (e.g., age = "25" $\bigvee$ location = "New York"). If any queried attribute (e.g., location) has a non-numeric value, the same conversion rules used by the OSN operator are applied to generate the corresponding numeric value. The data consumer submits a query to the SDP, which specifies the query condition. An appropriate payment may also need to be submitted simultaneously or later.

C. Adversary Model

We assume that the OSN operator is trusted to follow all system operations. In contrast, the SDP cannot be trusted and may return incorrect query results by adding, deleting, or modifying user profiles or edges among them for various motives, e.g., in favor of the businesses which are willing to pay. We also assume that the communications between the OSN operator and SDP and those between the SDP and the data consumer are secured using traditional mechanisms such as TLS. Therefore, an incorrect query result can only be attributed to the misbehavior of the SDP. Also note that the OSN operator is highly motivated to help identify malicious SDPs, as data consumers who make critical decisions based on manipulated query results may eventually blame the OSN operator.

D. Design Goals

Under the above adversary model, we aim to enable the data consumer to verify the correctness of any query result returned by the SDP. A query result is considered correct if it satisfies the following two requirements.

- *Query Result Integrity:* Any returned user profile should appear in the OSN operator's original dataset and have not been tampered with.
- *Query Result Completeness:* The query result should contain all the user profiles that satisfy the query condition as well as all the edges among them.

III. Verifiable Single-Attribute Query

In this section, we illustrate how to achieve verifiable single-attribute query over outsourced social graph by presenting three schemes. The three schemes all require the OSN provider to generate some cryptographic auxiliary information, based on which the SDP can construct a verification object for the data consumer to verify the query-result correctness. They differ in how the auxiliary information is generated and how the verification object is constructed and verified. Table I summarizes the key notation used in the paper.

TABLE I
Summary of Notation

Symbol	Definition
p_i	Profile of node i
ID_i	ID of node i
$b_{i,j}$	j-th attribute value of node i
$e_{i,j}$	Edge between nodes i and j
m_k	# of unique attribute k's values
$v_{k,j}$	The j-th unique attribute k's value
T	A Merkle hash tree
$\mathcal{A}(\cdot)$	Verification object
$[\delta_{\min}, \delta_{\max}]$	Query range
BF	A Bloom filter
α	The length of the Bloom filter
β	# of elements in the Bloom filter
τ	# of hash functions of the Bloom filter
CF	A partial-key cuckoo filter
fp	A fingerprint
ϵ	The false positive probability

A. A Basic Scheme

The basic scheme enables verifiable single-attribute query by having the OSN operator chain all the nodes with common attribute value as well as adjacent attribute values using cryptographic primitives to force the SDP to return all the nodes and edges that satisfy the query condition. In what follows, we detail the its three phases: data preprocessing at the OSN operator, query processing at the SDP, and query result verification at the data consumer.

1) Data Preprocessing: Assume that the OSN operator has a dataset, it processes the dataset as follows.

First, the OSN operator signs the every profile to guarantee its integrity. Specifically, for each node $i \in [1, n]$, the OSN operator first computes $h_i = H(p_i)$, where $H(\cdot)$ denotes a good cryptographic hash function, and then signs h_i using its private key to obtain the signature s_i.

Second, the OSN operator groups nodes based on their attribute values and chains adjacent attribute values using cryptographic techniques. For each attribute $k \in [1, w]$, the OSN operator sorts the n attribute values $\{b_{i,k} | 1 \le i \le n\}$ into a list $v_{k,1}, \ldots, v_{k,m_k}$ in ascending order with no repetition, where m_k is the number of unique attribute values and $v_{k,j} < v_{k,j+1}$ for all $1 \le j \le m_k - 1$. Let $v_{k,0} = v_{k,\min}$ and $v_{k,m_k+1} = v_{k,\max}$ be two publicly known values that are smaller and larger than the minimum and maximum attribute k values, respectively. Also denote by $I_{k,j} = \{i | b_{i,k} = v_{k,j}, 1 \le i \le n\}$ the set of nodes whose attribute k equals $v_{k,j}$ for all $j \in [1, m_k]$. For each unique attribute value $v_{k,j}$, $1 \le j \le m_k$, the OSN operator groups all the nodes with attribute k value $v_{k,j}$ by computing $\psi_{k,j}$ with two fields as

$$\begin{aligned} \psi_{k,j}.\mathsf{val} &= v_{k,j-1} || v_{k,j} || v_{k,j+1}, \\ \psi_{k,j}.\mathsf{hash} &= H(||_{i \in I_{k,j}} ID_i) \,, \end{aligned} \tag{1}$$

where ID_i is the ID of node i for all $1 \le i \le n$. The OSN operator then builds a Merkle Hash Tree (MHT) T_k^{ψ} [9] over $\psi_{k,1}, \ldots, \psi_{k,m_k}$. Specifically, a MHT is a binary tree where every leaf node is the hash of one unique element, and every internal node is the hash of the concatenation of its two

children nodes.[1] Finally, the OSN operator uses its private key to sign the root of MHT T_k^{ψ}.

Third, the OSN operator binds the nodes with their neighbors according to their attribute values to enable completeness verifications of the edges (i.e., social links) in the query result. Let N_i be the set of neighbors of each node i and $N_{i,k,j} = \{z|b_{z,k} = v_{k,j}, z \in N_i\}$ the subset of node i's neighbors whose attribute k values equal $v_{k,j}$ for all $1 \leq j \leq m_k$. It follows that $N_i = \bigcup_{j=1}^{m_k} N_{i,k,j}$ for all $1 \leq k \leq w$. For each $1 \leq i \leq n$, $1 \leq k \leq w$, and $1 \leq j \leq m_k$, the OSN operator computes

$$\phi_{i,k,j} = \langle v_{k,j}, H(||_{i \in N_{i,k,j}} ID_i) \rangle \ . \tag{2}$$

The OSN operator then constructs one MHT $T_{i,k}^{\phi}$ from $\{\phi_{i,k,1}, \ldots, \phi_{i,k,m_k}\}$ for each $1 \leq i \leq n$ and $1 \leq k \leq w$.

Finally, the OSN operator sends to the SDP its dataset and all the auxiliary information, which includes $\{s_i|1 \leq i \leq n\}$, $\{T_k^{\psi}|1 \leq k \leq w\}$, $\{T_{i,k}^{\phi}|1 \leq i \leq n, 1 \leq k \leq w\}$, and its signature over every MHT root.

2) Query Processing: We now illustrate how the SDP processes a query. Since any subset query can be decomposed into multiple equality queries, each of which can be considered as a special range query over a single queried attribute value, our subsequent discussion focuses on a range query $[\delta_{\min}, \delta_{\max}]$ over a particular attribute $k \in [1, w]$, where $\delta_{\min}$ and $\delta_{\max}$ are the minimum and maximum attribute values of interest, respectively.

After receiving a query $\langle k, \delta_{\min}, \delta_{\max} \rangle$ from the consumer, the SDP searches the social graph G and constructs a subgraph $G' = (V', E')$, where V' consists of all the nodes whose attribute k value falls into the range $[\delta_{\min}, \delta_{\max}]$ and $E' = \{e_{i,j}|i, j \in V', e_{i,j} \in E\}$ consists of all the edges among nodes in V'. The query result consists of $\{p_i|i \in V'\}$, $\{(ID_i, ID_j)|e_{i,j} \in E'\}$, and a verification object constructed according to $\{T_k^{\psi}|1 \leq k \leq w\}$ and $\{T_{i,k}^{\phi}|1 \leq i \leq n, 1 \leq k \leq w\}$ as follows.

First, the SDP returns the OSN operator's signatures $\{s_i|i \in V'\}$ to prove the integrity of the returned profiles.

Second, the SDP returns the information to prove that all the attribute values within the query range has been returned. Given the list of attribute values $v_{k,0}, \ldots, v_{k,m_k+1}$, the SDP finds x and y such that $v_{k,x-1} < \delta_{\min} \leq v_{k,x} \leq v_{k,y} \leq \delta_{\max} < v_{k,y+1}$. It follows that attribute values $v_{k,x}, \ldots, v_{k,y}$ fall into the query range $[\delta_{\min}, \delta_{\max}]$. For each element $\psi_{k,j}, x \leq j \leq y$, the SDP finds the subset of internal nodes needed to compute the root of MHT T_k^{ψ}, denoted by $\mathcal{A}(T_k^{\psi}|\psi_{k,j})$. The SDP returns $\bigcup_{j=x}^{y} \mathcal{A}(T_k^{\psi}|\psi_{k,j})$ and the OSN operator's signature on the root of T_k^{ψ}.

Third, the SDP returns the information to prove that all the edges among nodes in V' have been returned. Specifically, the SDP returns $\{\phi_{i,k,j}|i \in V', x \leq j \leq y\}$, the subset of internal nodes needed for computing the root of $T_{i,k}^{\phi}$ for all $i \in V'$, i.e., $\bigcup_{i \in V'} \mathcal{A}(T_{i,k}^{\phi}|\phi_{i,k,j})$, and the OSN operator's signature on the roots of $\{T_{i,k}^{\phi}|i \in V'\}$.

[1]Note that if the number of leaf nodes is not power of two, some dummy leaf nodes need be introduced.

3) Query-Result Verification: On receiving the query result, the data consumer verifies its integrity and completeness as follows.

First, data consumer verifies the integrity of the query result. For every returned profile $p_i, i \in V'$, the data consumer verifies the OSN operator's signature s_i. Moreover, for every $\psi_{k,j}$ received, the data consumer recomputes the root of the MHT T_k^{ψ} using $\mathcal{A}(T_k^{\psi}|\psi_{k,j})$. Similarly, for every $\phi_{i,k,j}$ received, the data consumer recomputes the root of the MHT $T_{i,k}^{\phi}$ using $\mathcal{A}(T_{i,k}^{\phi}|\phi_{i,k,j})$. The data consumer then verifies the OSN operator's signature over the root of the corresponding MHT. If all the verifications succeed, the query result is considered authentic.

Second, the data consumer verifies that for every unique k-th attribute values in $[\delta_{\min}, \delta_{\max}]$, all the profiles have been returned. Without loss of generality, assume that the SDP has returned $\psi_{k,x'}, \ldots, \psi_{k,y'}$, where $\psi_{k,j}.\mathsf{val} = v_{k,j}^{-}||v_{k,j}||v_{k,j}^{+}$ for all $x' \leq j \leq y'$. The data consumer checks if the following two conditions hold.

- *Condition 1:* $v_{k,x'}^{-} < \delta_{\min}$ and $v_{k,y'}^{+} > \delta_{\max}$, i.e., $v_{k,x'}$ and $v_{k,y'}$ are the smallest and largest attribute values that falls into the range, respectively.
- *Condition 2:* $v_{k,j} = v_{k,j+1}^{-}$ for all $x' \leq j \leq y' - 1$, i.e., $\psi_{k,x'}, \ldots, \psi_{k,y'}$ contains all the attribute values between $v_{k,x'}$ and $v_{k,y'}$.

Third, the data consumer verifies that all the nodes of which the attribute k value fall in $[\delta_{\min}, \delta_{\max}]$ have been returned. Specifically, for each returned $\psi_{k,j}, x \leq j \leq y$, the data consumer finds the set of returned profiles $I'_{k,j} = \{p_i|b_{i,k} = v_{k,j}, i \in V'\}$ and verifies if $\psi_{k,j}.\mathsf{hash} = H(||_{i \in I'_{k,j}} ID_i)$. If so, the data consumer is ascertain of $I'_{k,j} = I_{k,j}$, i.e., all the nodes whose attribute k values equal $v_{k,j}$ have been returned.

Finally, the data consumer verifies that all the edges among nodes in V' have been returned. Specifically, for every $\phi_{i,k,j}$ received, the data consumer recovers ID_i from the corresponding profile p_i and identifies the set of its neighbors N'_i according to received $\{(ID_i, ID_j)|i, j \in V'\}$. For each $j, x' \leq j \leq y'$, the data consumer finds $N'_{i,k,j} = \{z|b_{z,k} = v_{k,j}, z \in N'_i\}$ and verifies if $\phi_{i,k,j} = \langle v_{k,j}, H(||_{i \in N'_{i,k,j}} ID_i) \rangle$. If so, the data consumer is ascertain of $N'_{i,k,j} = N_{i,k,j}$, i.e., all node i's neighbors whose attribute k values equal $v_{k,j}$ have been returned.

The query result is considered authentic and complete if all the above verifications succeed.

We refer readers to Appendix A of the supplement file for an example of the basic scheme.

B. An Enhanced Scheme

The basic scheme can effectively detect any incorrect query results but at the cost of significant computation overhead. Besides signing n user profiles, it also requires the OSN operator to construct one MHT for every attribute and every node, leading to nw MHTs in total. Since n is very large in practice, the computational overhead for signature generation can be significant. Motivated by the observation that many nodes share the same attribute value, we now introduce an

enhanced scheme that builds one MHT for every unique attribute value instead of each node.

1) Data Preprocessing: The OSN operator performs the following steps in sequel to generate the auxiliary information for its dataset.

First, the OSN operator groups nodes based on their attribute values. Specifically, for each node $i \in [1, n]$, the OSN operator computes $h_i = H(p_i)$. Recall that $v_{k,1}, \ldots, v_{k,m_k}$ is the list of unique attribute values, where $v_{k,j} < v_{k,j+1}$ for all $j \in [1, m_k - 1]$ and $I_{k,j} = \{i | b_{i,k} = v_{k,j} | 1 \le i \le n\}$ is the set of nodes whose attribute k equals $v_{k,j}$ for all $j \in [1, m_k]$. For each attribute $k, 1 \le k \le w$ and each unique attribute value $v_{k,j}, 1 \le j \le m_k$, the OSN operator computes $\psi_{k,j}$ consisting of two fields as

$$\begin{aligned} \psi_{k,j}.\mathsf{val} &= v_{k,j-1} || v_{k,j} || v_{k,j+1}, \\ \psi_{k,j}.\mathsf{hash} &= H(||_{i \in I_{k,j}} h_i), \end{aligned} \tag{3}$$

where $v_{k,0} = v_{k,\min}$ and $v_{k,m_k+1} = v_{k,\max}$. For each attribute $k \in [1, w]$, the OSN operator builds an MHT T_k^{ψ} over $\psi_{k,1}, \ldots, \psi_{k,m_k}$ and signs its root.

Second, the OSN operator binds every node with its neighbors according to their attribute values. Let N_i be the set of neighbors of node i for all $1 \le i \le n$ and $N_{i,k,j} = \{z | b_{z,k} = v_{k,j}, z \in N_i\}$ the set of neighbors whose attribute k values equal $v_{k,j}$ for all $1 \le k \le w$ and $1 \le j \le m_k$. It follows that $N_i = \bigcup_{j=1}^{m_k} N_{i,k,j}$ for all $1 \le k \le w$. For every node $i \in [1, n]$, every attribute $k \in [1, w]$, and every unique attribute value $v_{k,j}$, the OSN operator computes

$$\theta_{i,k,j} = \langle ID_i, v_{k,j}, \{ID_z | z \in N_{i,k,j}\} \rangle \ .$$

In other words, each $\theta_{i,k,j}$ consists of node i's ID, its attribute k value $v_{k,j}$, and the IDs of its neighbors whose attribute k's values equal $v_{k,j}$. For every attribute k and every pair of attribute values $v_{k,j}$ and $v_{k,z}$, $1 \le j, z \le m_k$, the OSN operator further computes

$$\phi_{k,j,z} = H(||_{i \in I_{k,j}} \theta_{i,k,z}) \ . \tag{4}$$

In other words, each element $\phi_{k,j,z}$ binds the subset of nodes with attribute k value $v_{k,j}$ and their neighbors with attribute k value $v_{k,z}$.

Finally, the OSN operator builds one MHT $T_{k,j}^{\phi}$ over $\phi_{k,j,1}, \ldots, \phi_{k,j,m_k}$ for all $1 \le k \le w$ and $1 \le j \le m_k$ and signs the root of each MHT tree.

As in the basic scheme, the auxiliary information includes the internal nodes of all the MHT trees and all the signatures.

2) Query Processing: Upon receiving a range query $\langle k, \delta_{\min}, \delta_{\max} \rangle$ from the data consumer, the SDP searches the social graph G and constructs a subgraph $G' = (V', E')$, where V' consists of all the nodes whose attribute k values fall into the range $[\delta_{\min}, \delta_{\max}]$ and $E' = \{e_{i,j} | i, j \in V', e_{i,j} \in E\}$ consists of all the edges among all the nodes in V'. The query result consists of $\{p_i | i \in V'\}$, $\{(ID_i, ID_j) | e_{i,j} \in E'\}$, and a verification object constructed according to $\{T_k^{\psi} | 1 \le k \le w\}$ and $\{T_{k,j}^{\phi} | 1 \le k \le w, 1 \le j \le m_k\}$ as follows.

First, the SDP finds x and y such that $v_{k,x-1} < \delta_{\min} \le v_{k,x} \le v_{k,y} \le \delta_{\max} < v_{k,y+1}$. It follows that attribute values $v_{k,x}, \ldots, v_{k,y}$ fall into the query range $[\delta_{\min}, \delta_{\max}]$.

Second, the SDP returns elements $\psi_{k,x}, \ldots, \psi_{k,y}$ as well as the subset of internal nodes needed to recompute the root of the MHT T_k^{ψ} from each of the returned elements, denoted by $\mathcal{A}(T_k^{\psi} | \psi_{k,j})$.

Third, the SDP returns every element $\phi_{k,j,z}$ for all $x \le j, z \le y$, the SDP also finds the subset of internal nodes needed to reconstruct the root of the MHTs $T_{k,j}^{\phi}$, denoted by $\mathcal{A}(T_{k,j}^{\phi} | \phi_{k,j,z})$.

Finally, the SDP returns the query result and the verification objects consisting of $\bigcup_{j=x}^{y} \mathcal{A}(T_k^{\psi} | \psi_{k,j})$, $\{\bigcup_{z=x}^{y} \mathcal{A}(T_{k,j}^{\phi} | \phi_{k,j,z})\}_{j=x}^{y}$, and the OSN operator's signatures on the roots of $T_k^{\psi} | \psi_{k,j}$ and $\{T_{k,j}^{\phi} | \phi_{k,j,z}\}_{j=x}^{y}$.

3) Query-Result Verification: The data consumer verifies the integrity and completeness of the query result as follows.

First, data consumer verifies the integrity of the query result. For every returned profile $p_i, i \in V'$, the data consumer first computes $h_i = H(p_i)$. It then reconstructs the set of nodes whose attribute values equal $v_{k,j}$ as $I'_{k,j} = \{i | b_{i,k} = v_{k,j}, i \in V'\}$ and verifies if $\psi_{k,j}.\mathsf{hash} = H(||_{i \in I'_{k,j}} h_i)$. If so, the data consumer recomputes the root of the MHT T_k^{ψ} using $\mathcal{A}(T_k^{\psi} | \psi_{k,j})$ for every $\psi_{k,j}$ received. Next, for every $\phi_{i,k,j}$ received, the data consumer recomputes the root of the MHT $T_{i,k}^{\phi}$ using $\mathcal{A}(T_{i,k}^{\phi} | \phi_{i,k,j})$. The data consumer then verifies the OSN operator's signatures over the root of each corresponding MHT. If all the verifications succeed, the query result is considered authentic.

Second, the data consumer verifies that for every unique k-th attribute value in $[\delta_{\min}, \delta_{\max}]$, the corresponding profiles have been returned. Assume that the SDP has returned $\psi_{k,x'}, \ldots, \psi_{k,y'}$. The data consumer verifies the two conditions in Section III-A.3 as in the basic scheme.

Finally, the data consumer verifies that all the edges among nodes in V' have been returned. Specifically, for every profile p_i received, the data consumer finds the set of its neighbors N'_i based on received edge $\{(ID_i, ID_j) | e_{i,j} \in E'\}$. For each $j, x' \le j \le y'$, the data consumer finds $N'_{i,k,j} = \{z | b_{z,k} = v_{k,j}, z \in N'_i\}$ and computes $\theta'_{i,k,j}$ by $\langle ID_i, v_{k,j}, \{ID_z | z \in N'_{i,k,j}\} \rangle$. For every pair of attribute values $v_{k,j}$ and $v_{k,z}$ where $x' \le j, z \le y'$, the data consumer verifies if $\phi_{k,j,z} = H(||_{i \in I'_{k,j}} \theta'_{i,k,z})$. If so, the data consumer is ascertain of $\{N'_{i,k,z}\}_{i \in I'_{k,j}} = \{N_{i,k,z}\}_{i \in I_{k,j}}$, i.e., all the edges among the subset of nodes with attribute k's value $v_{k,j}$ with their neighbors with attribute k's value $v_{k,z}$ have been returned.

The query result is considered authentic and complete only if it passes all the verifications above.

We refer readers to Appendix B of the supplement file for an example of the enhanced scheme.

C. An Advanced Scheme

We now introduce an advanced scheme that further reduces the communication overhead by using a Bloom filter to represent each set $\{\phi_{k,j,1}, \cdots, \phi_{k,j,m_k}\}$ for all $1 \le j \le m_k$.

A Bloom Filter [10] is a classical space-efficient probabilistic data structure that supports membership testing. Assume that we want to use an α-bit Bloom filter to represent a set $\{d_i\}_{i=1}^{\beta}$. Every bit of the Bloom filter is initialized to zero. Let $\{h_j(\cdot)\}_{j=1}^{\tau}$ be τ independent hash functions, each of which

maps an element to a bit position $\{0, 1, \ldots, \alpha - 1\}$. We add each element d_i into the Bloom filter by setting all the bit positions $\{h_j(d_i)\}_{j=1}^{\tau}$ to one. To check the membership of an arbitrary element e, we can verify whether all the bit positions $\{h_j(e)\}_{j=1}^{\tau}$ are one. If at least one bit position is zero, then e is certainly not in the set; otherwise, we deem element e in the set. As a probabilistic data structure, Bloom filter may have false positive, which happens when the element is not in the set, but all the bit positions corresponding to the element have been set to one by other elements. The false positive probability of a Bloom filter is jointly determined by α, β, and τ. It has been shown in [10] that the false-positive probability of a Bloom filter is $0.6185^{\alpha/\beta}$ when $\tau = \frac{\alpha}{\beta} \times \ln 2$.

In the advanced scheme, the OSN operator computes $\{\psi_{k,j} | 1 \leq k \leq w, 1 \leq j \leq m_k\}$ and $\{\phi_{k,j,z} | 1 \leq k \leq w, 1 \leq j, z \leq m_k\}$ as in the enhanced scheme. For each $1 \leq k \leq w$ and $1 \leq j \leq m_k$, the OSN operator generates a Bloom filter $\texttt{BF}_{k,j}$ to represent $\phi_{k,j,1}, \ldots, \phi_{k,j,m_k}$. Finally, the OSN operator signs $\{\texttt{BF}_{k,j} | 1 \leq k \leq w, 1 \leq j \leq m_k\}$.

The auxiliary information is now composed of all the internal nodes of the MHT T_k^{ψ}, the Bloom filters $\{\texttt{BF}_{k,j} | 1 \leq k \leq w, 1 \leq j \leq m_k\}$, and all the related signatures. In addition, the advanced scheme uses the almost identical processes for query processing and query-result verification to those in the basic and enhanced schemes. The only exception is that the verification object contains the signed Bloom filters $\texttt{BF}_{k,x}, \ldots, \texttt{BF}_{k,y}$ instead of individual elements $\phi_{k,j,z}$ for all $x \leq j, z \leq y$. The data consumer still computes $\{\phi_{k,j,z} | 1 \leq j, z \leq y\}$ from the query result and checks whether every element $\phi_{k,j,z}$ is in the corresponding Bloom filter. If any of the elements is not present in the corresponding Bloom filter, the query result is considered incomplete.

We refer readers to Appendix C of the supplement file for an example of the advanced scheme.

IV. Verifiable Multi-Attribute Query

In this section, we extend the enhanced scheme to the multi-attribute verification scheme whereby to support multi-attribute queries.

Assume that each OSN user has multiple attributes such as *age* and *followees number*. Since the multi-attribute query is an `and`/`or` combination of multiple single-attribute queries, the query-result of a multi-attribute query can be viewed as the intersection/union of multiple query-results accordingly, each satisfying the query condition over a single attribute.

In what follows, we first briefly introduce the partial-key cuckoo filter (CF) [11] that underpins the scheme and then describe how to realize verifiable set operations with CF. Finally, we detail auxiliary information generation, query processing, and query-results verification.

A. Partial-Key Cuckoo Filter

Partial-key cuckoo filter (CF) [11], [12] is another probabilistic data structure that supports membership checking. Different from the Bloom filter, it also supports deletion operation in addition to insertion and lookup, which is key to realize verifiable set operations. The key idea behind the CF is to store the fingerprints of a set of elements into a cuckoo hash dictionary. A cuckoo hash dictionary consists of multiple buckets, each of which is uniquely identified by the bucket index. Each bucket contains multiple entries, each of which is either empty or stores one fingerprint of an element in the set.

To insert an element a into a CF with f buckets, the insertion algorithm first calculates two candidate bucket indexes by two functions $h_1(a) = \mathsf{hash}(a) \mod f$ and $h_2(a) = h_1(a) \oplus \mathsf{hash}(\mathsf{fingerprint}(a)) \mod f$, where $\mathsf{hash}(\cdot)$ denotes a hash function that maps any element to an bucket index in $\{0, 1, \ldots, f-1\}$, and $\mathsf{fingerprint}(\cdot)$ denotes a fingerprint generation function that maps any element to a fingerprint of η bits. Both hash function $\mathsf{hash}(\cdot)$ and fingerprint function $\mathsf{fingerprint}(\cdot)$ can be realized by any non-cryptographic hash function suitable for general hash-based lookup such as MurmurHash, CityHash, and Jenkins hash. There are three possible cases.

- *Case 1:* if both candidate buckets have empty entries, then randomly choose one bucket and insert a to one of the empty entries.
- *Case 2:* if only one of the candidate buckets has empty entries, then a is inserted to an empty entry of that bucket.
- *Case 3:* if neither candidate buckets have any empty entry, then select one of the candidate buckets and kick out one of the existing fingerprints and re-insert it to its other candidate bucket. For example, suppose that we want to kick out a fingerprint $\mathsf{fingerprint}(e)$ out of bucket ℓ. The other candidate bucket of element e is given by $\ell \oplus \mathsf{hash}(\mathsf{fingerprint}(e))$ even if we cannot recover the original element e. Such insertion procedure runs until an empty bucket is found or until a maximum number of displacement is reached.

If no empty bucket is found, this CF is considered full.

To lookup an element a, we check the corresponding two candidate buckets and consider a present if either one has an entry storing $\mathsf{fingerprint}(a)$.

Lookup operation may incur false positive, which happens when element a is not in the CF but one of the candidate buckets contains an entry with a fingerprint same as $\mathsf{fingerprint}(a)$. We abuse the notations to let ξ denote the maximum number of entries per bucket, η denote the fingerprint size, and β denote the number of elements in an CF. We have the following theorem regarding the false positive probability of an CF.

Theorem 1: The false positive probability of an CF is given by

$$\epsilon = 1 - 2\Pr(W_1) + \Pr(W_1, W_2)\,, \tag{5}$$

where

$$\Pr(W_1) = \sum_{k=0}^{\xi} \binom{\beta}{k} \cdot \left(\frac{1}{f}\right)^k \cdot \left(1 - \frac{1}{f}\right)^{\beta - k} \cdot \left(1 - \frac{1}{2^{\eta}}\right)^k\,, \tag{6}$$

and

$$\Pr(W_1, W_2) = \sum_{k_1=0}^{\xi} \sum_{k_2=0}^{\xi} \binom{\beta}{k_1} \cdot \binom{\beta - k_1}{k_2} \cdot \left(\frac{1}{f}\right)^{k_1 + k_2}$$

$$\cdot \left(1-\frac{1}{f}\right)^{2\beta-2k_1-k_2} \cdot \left(1-\frac{1}{2^{\eta}}\right)^{k_1+k_2}. \quad (7)$$

We give the proof in Appendix D of the supplement file.

B. Verifiable Set Operation With Cuckoo Filter

Verifiable set operations [13] are a classical computation outsourcing problem in which a prover who has two sets $X = \{x_1, x_2, \dots\}$ and $Y = \{y_1, y_2, \dots\}$ sends the intersection or union set $E = \{e_1, e_2, \dots\}$ to a verifier who needs to verify the correctness of the intersection/union. Different from prior solutions [14]–[16] that incur high computation overhead, we propose a novel verifiable set-intersection/union scheme based on the Cuckoo filter.

We realize the verification of set-intersection/union with the Cuckoo filter. Specifically, we say two CFs $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ are *homologous* if and only if they have the same hash function, fingerprint generation function, and the number of buckets. Accordingly, any element a has identical fingerprints and candidate buckets across two homologous CFs.

Let us first take a look at how to realize verifiable set intersection. As in [17], we define the conditions for the correctness of set intersection as follows.

Definition 1: A set E is the intersection of two sets X and Y if and only if the following two conditions are met.

Subset condition: $(E \subseteq X) \wedge (E \subseteq Y)$;

Completeness condition: $(X \setminus E) \cap (Y \setminus E) = \emptyset$.

Suppose that the prover provides two homologous CFs $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ representing two sets X and Y, respectively, along with the intersection set $E = X \bigcap Y$. First, the verifier checks the subset condition by testing whether every element $e \in E$ appears in both $\mathtt{CF}_X$ and $\mathtt{CF}_Y$. Second, to verify the completeness condition, we delete each $e \in E$ from both $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ and then check whether resulting $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ have any element in common. Specifically, for each bucket $\ell \subseteq \mathtt{CF}_X$, we check each fingerprint fp in bucket ℓ to see if it appears in $\mathtt{CF}_Y$. While we cannot recover the original element a of fingerprint fp, we can calculate the other candidate bucket of element a in $\mathtt{CF}_X$ as

$$\ell' = \ell \oplus \mathsf{hash}(\mathsf{fp}) \mod f$$

Since $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ are homologous, element a would have the same candidate buckets ℓ or ℓ' in $\mathtt{CF}_Y$. We then check if either bucket ℓ or ℓ' of $\mathtt{CF}_Y$ contains the fingerprint fp. If not, we deem the fingerprint fp and its corresponding element a absent from $\mathtt{CF}_Y$. If none of the fingerprints in $\mathtt{CF}_X$ is found in $\mathtt{CF}_Y$, we consider the completeness condition is met. We summarize the procedure in Algorithm 1.

Algorithm 1 may incur false alarm, that is, it returns FALSE when $E = X \bigcap Y$. We have the following theorem about the false alarm probability of Algorithm 1.

Theorem 2: Let X and Y be two sets represented by CFs $\mathtt{CF}_X$ and $\mathtt{CF}_Y$, respectively, and $E = X \bigcap Y$. The false alarm probability of Algorithm 1 is upper bounded by

$$\varepsilon \leq 1 - (1-\epsilon_X)^{|Y \setminus E|}. \quad (8)$$

where ϵ_X is the false positive probability of $\mathtt{CF}_X$.

We give the proof in Appendix E of the supplement file.

Algorithm 1: Verifiable Set-Intersection Operation

Input: CF_X representing set X, CF_Y representing set Y, and set E

Output: Whether $E = X \bigcap Y$

```
foreach e ∈ E do
    if e ∉ CF_X or e ∉ CF_Y then
        return false;
    end
    Delete e from CF_X and CF_Y;
end
foreach bucket ℓ of CF_X do
    foreach fingerprint fp in bucket ℓ do
        ℓ' ← ℓ ⊕ hash(fp) mod f;
        foreach fp' ∈ bucket ℓ of CF_Y do
            if fp = fp' then
                return false;
            end
        end
        foreach fp' ∈ bucket ℓ' of CF_Y do
            if fp = fp' then
                return false;
            end
        end
    end
end
return true;
```

Now let us take a look at the verifiable set union operation. We define the conditions for the correctness of set union as follows.

Definition 2: A set E is the union of two sets X and Y if and only if the following two conditions are met.

Membership condition: For each $e \in E$, $e \in X$ or Y;

Superset condition: For each $e \in X$ or Y, $e \in E$.

Given a set E and two homologous CFs $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ representing two sets X and Y, respectively, the verifier first checks whether every element $e \in E$ is in both $\mathtt{CF}_X$ and $\mathtt{CF}_Y$. If so, the verifier deletes all elements in E from both $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ and then checks whether the remaining CFs $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ are both empty. If so, the membership and superset conditions are considered met. We summarize the procedure in Algorithm 2.

Different from Algorithm 1, Algorithm 2 does not incur any false alarm. In particular, given two sets X and Y and $E = X \bigcup Y$, the membership condition check will always pass because every element $e \in E$ must be present in both $\mathtt{CF}_X$ and $\mathtt{CF}_Y$. In addition, since every element in X or Y belongs to E, so both $\mathtt{CF}_X$ and $\mathtt{CF}_Y$ will become empty after deleting every element in E from them. Therefore, Algorithm 2 does not incur any false alarm.

In what follows, we detail the three phases of the scheme.

C. Data Preprocessing

The OSN operator preprocesses the dataset as follows. Recall that $v_{k,1}, \dots, v_{k,m_k}$ are the list of unique attribute k values, where $v_{k,j} < v_{k,j+1}$ for all $1 \leq j \leq m_k$, and that

Algorithm 2: Verifiable Set-Union Operation

Input: CF_X representing set X, CF_Y representing set Y, and set E
Output: Whether $E = X \bigcup Y$

```
foreach e ∈ E do
    flag ← flase;
    if e ∈ CF_X then
        flag ← true;
        Delete e from CF_X;
    end
    if e ∈ CF_Y then
        flag ← true;
        Delete e from CF_Y;
    end
    if flag = false; then
        return false;
    end
end
if CF_X ≠ empty or CF_Y ≠ empty then
    return false;
end
return true;
```

$I_{k,j} = \{i|b_{i,k} = v_{k,j}, 1 \leq i \leq n\}$ is the set of nodes whose attribute k values equal $v_{k,j}$ for all $1 \leq k \leq w$ and $1 \leq j \leq m_k$.

First, for each node $i \in [1, n]$, the OSN operator computes $h_i = H(p_i)$, where $p_i = (ID_i||b_{i,1}|| \ldots ||b_{i,w})$. Let N_i be the set of neighbors of each node i. The OSN operator constructs an CF $\mathtt{CF}_i$ from the set $\{H(ID_i||ID_j)|j \in N_i\}$ to encode its neighbor information. Further let $\theta_i = \langle h_i, \mathtt{CF}_i \rangle$ for all $1 \leq i \leq n$. The OSN operator constructs an MHT T^θ over elements $\theta_1, \ldots, \theta_n$ and signs the root.

Second, the OSN operator binds the IDs of nodes that share the same attribute k value. Specifically, for each attribute $k \in [1, w]$ and each unique attribute k value $v_{k,j}$, $1 \leq j \leq m_k$, the OSN operator computes $\psi_{k,j}$ that consists of two fields as

$$\begin{aligned} \psi_{k,j}.\mathsf{val} &= v_{k,j-1}||v_{k,j}||v_{k,j+1}, \\ \psi_{k,j}.\mathsf{cf} &= \mathtt{CF}_{k,j}, \end{aligned} \tag{9}$$

where $v_{k,0} = v_{k,\min}$, $v_{k,m_k+1} = v_{k,\max}$, and $\mathtt{CF}_{k,j}$ is an CF constructed from $\{h_i|i \in I_{k,j}\}$. Note that all CFs $\{\mathtt{CF}_{k,j}|1 \leq k \leq w, 1 \leq j \leq m_k\}$ are homologous. The OSN operator then builds an MHT T_k^ψ over $\psi_{k,1}, \ldots, \psi_{k,m_k}$ and signs the root of T_k^ψ for each $1 \leq k \leq w$.

D. Query Processing

Since any subset or equality query can be converted into a range query, our subsequent discussions focus on range query processing. Let $\mathsf{R}_1, \ldots, \mathsf{R}_w$ be the set of query ranges over attributes $1, \ldots, w$, respectively, where $\mathsf{R}_k = [\delta_{k,\min}, \delta_{k,\max}]$ for all $k \in [1, w]$. A multi-attribute range query has the form of

$$\mathsf{R}_1 \odot \mathsf{R}_2 \odot \cdots \odot \mathsf{R}_w,$$

where $\odot$ is either a logical operators AND operator $\wedge$ or a logical operators OR operator $\vee$. Without loss of generality, we take a range query $\mathsf{R}_1 \bigodot \mathsf{R}_2$ over attributes 1 and 2 as an example. Range queries over more than two attributes can be realized in a similar fashion.

After receiving the query, the SDP first finds the nodes whose attribute 1 and 2 values fall into the respective queried ranges. Let $V_1 = \{i|b_{i,1} \in \mathsf{R}_1, 1 \leq i \leq n\}$ and $V_2 = \{i|b_{i,2} \in \mathsf{R}_2, 1 \leq i \leq n\}$ be the two subsets of nodes whose attribute 1 and 2 values fall into the ranges R_1 and R_2, respectively. The query result is the subgraph $G' = (V', E')$, where $V' = V_1 \odot V_2$ and $E' = \{e_{i,j}|e_{i,j} \in E, i \in V', j \in V'\}$. More specifically, the query result consists of the set of profiles and CFs $\{\theta_i = \langle p_i, \mathtt{CF}_i \rangle | i \in V'\}$, the set of edges $\{(ID_i, ID_j)|e_{i,j} \in E'\}$, and an verification object constructed from T^θ and $\{T_k^\psi | k \in \{1, 2\}\}$ as follows.

First, the SDP returns elements $\{\theta_i | i \in V'\}$ as well as the subsets of internal nodes $\bigcup_{i \in V'} \mathcal{A}(T^\theta|\theta_i)$ of the MHT T^θ to ensure the integrity of the returned profiles and the completeness of the returned edges.

Second, the SDP finds x_1, x_2, y_1 and y_2 such that $v_{1,x_1-1} < \delta_{1,\min} \leq v_{1,x_1} \leq v_{1,y_1} \leq \delta_{1,\max} < v_{1,y_1+1}$ and $v_{2,x_2-1} < \delta_{2,\min} \leq v_{2,x_2} \leq v_{2,y_2} \leq \delta_{2,\max} < v_{2,y_2+1}$. It follows that attribute values $\{v_{1,x_1}, \ldots, v_{1,y_1}\}$ and $\{v_{2,x_2}, \ldots, v_{2,y_2}\}$ fall into the query ranges R_1 and R_2, respectively.

Third, the SDP returns elements $\psi_{1,x_1}, \ldots, \psi_{1,y_1}$ and $\psi_{2,x_2}, \ldots, \psi_{2,y_2}$ as well as the subsets of internal nodes $\bigcup_{j=x_1}^{y_1} \mathcal{A}(T_1^\psi|\psi_{1,j})$ of the MHT T_1^ψ and $\bigcup_{j=x_2}^{y_2} \mathcal{A}(T_2^\psi|\psi_{2,j})$ of the MHT T_2^ψ, which are needed for recomputing the roots of T_1^ψ and T_2^ψ from $\{\psi_{1,j}|x_1 \leq j \leq y_1\}$ and $\{\psi_{2,j}|x_2 \leq j \leq y_2\}$, respectively.

In summary, the query result consists of $\{\theta_i = \langle p_i, \mathtt{CF}_i \rangle | i \in V'\}$, $\{(ID_i, ID_j)|e_{i,j} \in E'\}$, $\bigcup_{i \in V'} \mathcal{A}(T^\theta|\theta_i)$, $\bigcup_{j=x_1}^{y_1} \mathcal{A}(T_1^\psi|\psi_{1,j})$, $\bigcup_{j=x_2}^{y_2} \mathcal{A}(T_2^\psi|\psi_{2,j})$, and the OSN operator's signatures on the roots of the MHTs T^θ, T_1^ψ and T_2^ψ.

E. Query-Result Verification

On receiving the query result, the data consumer verifies its integrity and completeness as follows.

First, the data consumer verifies the integrity of the query result through received MHTs and the OSN operator's signature. For each node $i \in V'$, the data consumer computes $h_i = H(p_i)$ and the root of MHT T^θ from $\theta_i = \langle h_i, \mathtt{CF}_i \rangle$ using $\mathcal{A}(T^\theta|\theta_i)$. Similarly, for each $\psi_{1,j}, x_1 \leq j \leq y_1$ and each $\psi_{2,j}, x_2 \leq j \leq y_2$, the data consumer recomputes the roots of T_1^ψ and T_2^ψ using $\mathcal{A}(T_1^\psi|\psi_{1,j})$ and $\mathcal{A}(T_2^\psi|\psi_{2,j})$, respectively. If the correct MHT root can be constructed from every received element, the data consumer further verifies the OSN operator's signatures on all the MHT roots. If all the verifications succeed, the data consumer considers the query result authentic.

Second, the data consumer verifies the completeness of the query result using the received CFs. Let V_1 and V_2 be the sets of nodes whose attribute 1 value falls into the range R_1 and R_2, respectively. The data consumer needs to verify whether $V' = V_1 \odot V_2$. The attribute 1 values that fall into the range R_1 are $v_{1,x_1}, \ldots, v_{1,y_1}$. Recall

that $I_{1,j}$ is the set of nodes with attribute 1 value being $v_{1,j}$ and that $\psi_{1,j}.\mathsf{cf} = \mathtt{CF}_{1,j}$ for all $x_1 \leq j \leq y_1$. It follows that $V_1 = \bigcup_{j=x_1}^{y_1} I_{1,j}$ and that $I_{1,x_1}, \ldots, I_{1,y_1}$ are represented by $\mathtt{CF}_{1,x_1}, \ldots, \mathtt{CF}_{1,y_1}$, respectively. Similarly, we have $V_2 = \bigcup_{j=x_2}^{y_2} I_{2,j}$ and $I_{2,x_1}, \ldots, I_{2,y_1}$ represented by $\mathtt{CF}_{2,x_2}, \ldots, \mathtt{CF}_{2,y_2}$, respectively. Further denote by $V_{l \bigwedge m} = \{i | b_{i,1} = v_{1,l}, b_{i,2} = v_{2,m}, i \in V'\}$ and $V_{l \bigvee m} = \{i | b_{i,1} = v_{1,l} \bigvee b_{i,2} = v_{2,m}, i \in V'\}$ for all $x_1 \leq l \leq y_1$ and $x_2 \leq m \leq y_2$.

There are two cases. Let us first consider the case of $\odot = \bigwedge$. It follows that $V' = \bigcup_{l=x_1}^{y_1} \bigcup_{m=x_2}^{y_2} V_{l \bigwedge m}$. To verify whether $V' = V_1 \bigcap V_2$, i.e., $\bigcup_{l=x_1}^{y_1} \bigcup_{m=x_2}^{y_2} V_{l \bigwedge m} = \bigcup_{l=x_1}^{y_1} I_{1,l} \odot \bigcup_{m=x_2}^{y_2} I_{2,m}$, it is equivalent to verify whether $V_{l \bigwedge m} = I_{1,l} \bigcap I_{2,m}$ with $I_{1,l}$ and $I_{2,m}$ being represented by $\mathtt{CF}_{1,l}$ and $\mathtt{CF}_{2,m}$, respectively, for all $x_1 \leq l \leq y_1$ and $x_2 \leq m \leq y_2$. In particular, given $V_{l \bigwedge m}, \mathtt{CF}_{1,l}$, and $\mathtt{CF}_{2,m}$, the data consumer can verify whether $V_{w \bigwedge z} = I_{1,l} \bigcup I_{2,m}$ using Algorithm 1. If every call to Algorithm 1 returns TRUE, the data consumer considers all the profiles that satisfy the query condition have been returned. In the second case where $\odot = \bigvee$, we have $V' = \bigcup_{l=x_1}^{y_1} \bigcup_{m=x_2}^{y_2} V_{w \bigvee z}$. The user needs to verify whether $V_{l \bigvee m} = I_{1,l} \bigcup I_{2,m}$ with $I_{1,l}$ and $I_{2,m}$ being represented by $\mathtt{CF}_{1,l}$ and $\mathtt{CF}_{2,m}$, respectively, for all $x_1 \leq l \leq y_1$ and $x_2 \leq m \leq y_2$. In particular, given $V_{l \bigvee m}, \mathtt{CF}_{1,l}$, and $\mathtt{CF}_{2,m}$, the data consumer can verify whether $V_{w \bigvee z} = I_{1,l} \bigcup I_{2,m}$ using Algorithm 2. If every call to Algorithm 2 returns TRUE, the data consumer considers all the profiles that satisfy the query condition have been returned.

Finally, the data consumer verifies whether all the edges among the nodes in V' have been returned. Specifically, given the received edges $\{(ID_i, ID_j) | e_{i,j} \in E'\}$, the data consumer first verifies whether $H(ID_i || ID_j)$ and $H(ID_j || ID_i)$ are in $\mathtt{CF}_i$ and $\mathtt{CF}_j$, respectively, for all $e_{i,j} \in E'$. In addition, for each $(ID_i, ID_j), e_{i,j} \notin E'$, the data consumer verifies whether $H(ID_i || ID_j)$ and $H(ID_j || ID_i)$ are not in $\mathtt{CF}_i$ and $\mathtt{CF}_j$, respectively. If all the verifications succeed, the data consumer considers that all the edges among the nodes in V' have been returned.

The query result is considered complete and correct if all the verifications above succeed.

V. Performance Analysis

In this section, we analyze the performance of the proposed schemes.

A. Security Analysis

1) Basic, Enhanced, and Advanced Schemes: The basic and enhanced schemes both allow a data consumer to detect any forged or incomplete query result. The reason is that the auxiliary information amounts to chaining profiles and edges grouped by their attribute values with cryptographic techniques. As long as the cryptographic hash function and digital signature scheme are secure, the SDP cannot return a forged and/or incomplete query result without being detected.

In contrast, the advanced scheme can detect any forged or incomplete query result with high probability. While the signed MHT can effectively prevent the SDP from inserting, deleting, or modifying profiles in the query result while escaping the detection, the Bloom filter $\mathtt{BF}_{k,j}$ used for encoding $\phi_{k,j,1}, \ldots, \phi_{k,j,m_k}$ may not be able to detect the deletion of some $\phi_{k,j,z}$ due to the false positive of Bloom filter. In particular, since $\phi_{k,j,z} = H(||_{i \in I_{k,j}} \theta_{i,k,z})$, we have $\phi_{k,j,z} = \phi_{k,z,j}$ for all $1 \leq z, j \leq m_k$. Therefore, each $\phi_{k,j,z}$ is inserted into two Bloom filters $\mathtt{BF}_{k,j}$ and $\mathtt{BF}_{k,z}$. Since the signed MHT can guarantee the integrity of all the profiles in the query result, it can also detect omission of any attribute values $b_{k,j}$ and $b_{k,z}$ as well as $\phi_{k,j,z}$ in the query result. Assume that the SDP deletes (or adds) one edge between node i with attribute k's value $b_{k,j}$ and node l with attribute k's value $b_{k,z}$ from the query result. In the query-result verification phase, the data consumer derives $\phi'_{k,j,z}$ and $\phi'_{k,z,j}$ and checks whether they are indeed in the corresponding Bloom filters. The insertion or deletion of an edge would result in $\phi'_{k,j,z} \neq \phi_{k,j,z}$ and $\phi'_{k,z,j} \neq \phi_{k,z,j}$. The SDP can escape the detection if and only if $\phi'_{k,j,z}$ is found in $\mathtt{BF}_{k,j}$ and $\phi'_{k,z,j}$ is found in $\mathtt{BF}_{k,z}$ at the same time. Since the false positive probability of a Bloom filter is $0.6185^{\alpha/\beta}$, the SDP can escape the detection with probability $(0.6185^{\alpha/\beta})^{2\chi}$ for adding (or deleting) $\chi \geq 1$ edges from the query result.

2) Verifiable Multi-Attribute Query Scheme: The verifiable multi-attribute query scheme can detect any modified or forged profile in a deterministic way. In particular, the OSN operator builds one MHT T^{θ} over $\theta_1, \ldots, \theta_n$, where each θ_i contains the hash of the profile p_i. The signed T^{θ} ensures that any modified or forged profile in the query result can be detected. Moreover, if the SDP inserts any authentic profile that does not satisfy the query condition, the user can easily detect it by checking the profile against the query condition. Therefore, the remaining options left for the SDP are (1) omitting one or more profiles (2) inserting or deleting one or more edges. We discuss these two attacks below by considering query $\mathsf{R}_1 \bigodot \mathsf{R}_2$ over attributes 1 and 2 as an example.

Let us first take a look at the case of $\bigodot = \bigwedge$, that is, the user asks for the set of nodes $V' = V_1 \bigcap V_2$, where V_1 and V_2 are the subsets of nodes with the attribute 1 value that falling in R_1 and the attribute 2 value falling in R_2, respectively. We have the following theorem regarding the detection probability with respect to the deletion of profiles.

Theorem 3: The deletion of any node from the query result can always be detected if $\bigodot = \bigwedge$.

We give the proof in Appendix F of the supplement file.

In addition, since query result verification relies on Algorithm 1 that may incur false alarm, we have the following theorem regarding the false alarm probability of the multi-attribute verification scheme.

Theorem 4: Suppose that the data consumer issues a query $\mathsf{R}_1 \bigcap \mathsf{R}_2$. *Assume that the attribute* 1 *values fall into* R_1 *are* $v_{1,x_1}, \ldots, v_{1,y_1}$ *and that the attribute* 2 *values fall into* R_2 *are* $v_{2,x_2}, \ldots, v_{2,y_2}$. *the false alarm probability of the multi-attribute verification scheme is upper bounded by*

$$P_{FA} \leq 1 - \prod_{l=x_1}^{y_1} \prod_{m=x_2}^{y_2} (1 - \epsilon_{1,l})^{|I_{m,2} \setminus I_{1,l}|}, \tag{10}$$

where $\epsilon_{1,l}$ *is the false positive probability of CF* $\mathtt{CF}_{1,l}$ *for all* $x_1 \leq l \leq y_1$.

We give the proof in Appendix G of the supplement file.

Now we consider the case of $\bigodot = \bigvee$, that is, the user asks for the set of nodes $V' = V_1 \bigcup V_2$. We have the following theorem regarding the detection probability of deletion of any single node from the query result.

Theorem 5: Suppose that the data consumer issues a query $\mathsf{R}_1 \bigcup \mathsf{R}_2$. *Assume that the attribute* 1 *values fall into* R_1 *are* $v_{1,x_1}, \ldots, v_{1,y_1}$ *and that the attribute* 2 *values fall into* R_2 *are* $v_{2,x_2}, \ldots, v_{2,y_2}$. *The deletion of any single node* s *with* $b_{s,1} = v_{1,x}$ *and* $b_{s,2} = v_{2,y}$ *from the query result can be detected with probability*

$$P_{det} \geq \min(\epsilon_{1,x} + (1-\epsilon_{1,x}) \prod_{m=x_2}^{y_2} \epsilon_{2,m}, \epsilon_{2,y} + (1-\epsilon_{1,l}) \prod_{l=x_1}^{y_2} \epsilon_{1,l}) , \quad (11)$$

where $\epsilon_{1,x}$ *and* $\epsilon_{2,y}$ *are the false positive probabilities of* $CF_{2,y}$ *and* $CF_{2,y}$, *respectively,* $\epsilon_{1,l}$ *is the false positive probability of* $CF_{1,l}$ *for all* $x_1 \leq l \leq y_1$, *and nd* $\epsilon_{2,m}$ *is the false positive probability of* $CF_{2,m}$ *for all* $x_2 \leq m \leq y_2$.

We give the proof in Appendix H of the supplement file.

In addition, since Algorithm 2 does not incur any false alarm, the multi-attribute verification scheme does not incur any false alarm when $\bigodot = \bigvee$.

We now analyze the detection probability over the attacks of inserting or deleting one or more edges. Since the data consumer will check each possible edge between any two nodes in the returned result, the cases of $\bigodot = \bigwedge$ and $\bigodot = \bigvee$ follow a similar analysis.

Theorem 6: The insertion of an edge between two nodes i *and* j *into the query result can be detected with probability*

$$P_{det} = 1 - \epsilon_i \epsilon_j, \quad (12)$$

where ϵ_i *and* ϵ_j *are the false positive probabilities of* CF_i *and* CF_j, *respectively.*

We give the proof in Appendix I of the supplement file.

Theorem 7: The deletion of any edge from the query result can always be detected.

We give the proof in Appendix J of the supplement file. We also refer readers to Appendix K of the supplement file for the analysis of the computation, storage, and communication overhead of the four proposed schemes.

VI. Experimental Results

In this section, we experimentally evaluate the four proposed schemes using real OSN datasets. We implement all four schemes using Python 2.7 with about 1800 lines of code. All the experiments are carried out on a commodity PC equipped with a 3.4 GHz Intel-i7 3770 CPU, 16 GB memory, a 7200 RPM hard disk, and Windows 10 OS. The false positive probability of the Bloom filter in the advanced scheme is set to 0.001 unless mentioned otherwise. We follow the prior work [11] to configure two candidate buckets for each item and four entries for each bucket of the partial-key cuckoo filter as such setting can achieve good space efficiency for the false positive probabilities suitable for most practical applications [18]. In addition, we adopt MurmurHash3 as both the hash function and the fingerprint generation function for partial-key cuckoo filter in our experiments.

A. Datasets

We use a real-world Twitter dataset that we collected in March 2016. Specifically, we randomly selected 100,000 Twitter users as seeds. For each seed user, we crawled his attribute, followers, and friends, and obtained nearly 4M users' attributes to construct the corresponding social graph. Our subsequent experiments focus on location and fans attributes. For verifiable single-attribute queries, we extracted the location with city-level labels in the form of "cityName,stateName" or "cityName,stateAbbreviation", where we considered all cities in the "List of Valid U.S. cities".[2] Among all the crawled users, we found 1.6M users with valid location and fans attributes and more than 50M edges among them. Besides, we further considered the fans number of these users as the second attribute for multi-attribute queries.

To evaluate the performance of our schemes for datasets with different sizes, we randomly sampled three groups of users (*S*-100K, *S*-1M, *S*-1.5M) from the above dataset, i.e., 100K, 1M, 1.5M users and their corresponding social graphs. The sizes of three datasets *S*-100K, *S*-1M and *S*-1.5M are 315.58 MB (10.95 MB users' attribute, 304.63 MB social network), 3.13 GB (109.57 MB users' attribute, 3.02 GB social network), and 4.71 GB (164.36 MB users' attribute, 4.54 GB social network), respectively.

B. Data Preprocessing

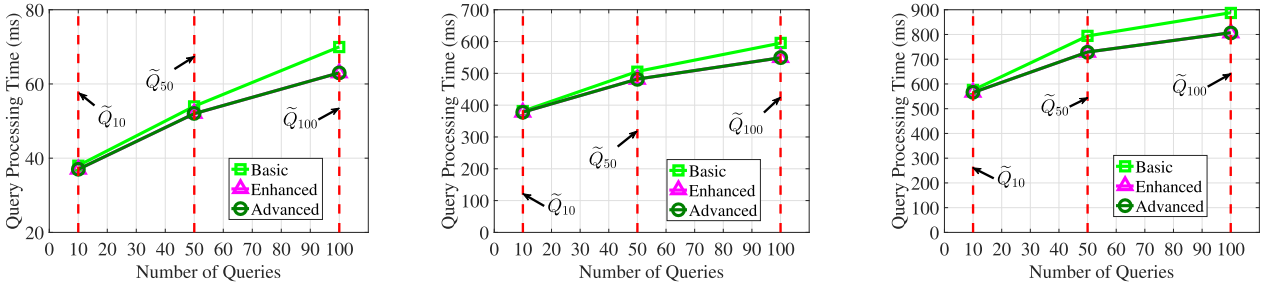
We first evaluate the computation and storage overhead incurred by data preprocessing in the basic, enhanced, advanced, and multi-attribute query schemes.

1) Computation Overhead: We compare the four schemes in terms of the numbers of hash and signature operations, and computation time. Table II shows the numbers of hash and signature operations for all four schemes and all three datasets. We can see from Table II that (1) the basic scheme incurs the highest number of hash operations, followed by the enhanced scheme and the advanced scheme; (2) the number of signature operations in the advanced scheme and enhanced schemes are significantly lower than that of the basic scheme; and (3) the number of signature operations in the multi-attribute query scheme is higher than that of the other three single-attribute query schemes. For example, the numbers of signature operations in the advanced and enhanced schemes are both only 0.13% of that in the basic scheme for *S*-1.5M. This is expected, as the numbers of signature operations in the three schemes are $O(n)$, $O(m_k)$ and $O(m_k)$ for each attribute, respectively, where m_k is the number of unique attribute k values. Besides, the number of signature operations in the multi-attribute query scheme is only 3 (i.e., $O(w)$) for *S*-1.5M. Finally, Table III shows the total computation time incurred by data preprocessing for three datasets. We can see that the advanced scheme and the

[2] https://www.whitehouse.gov/sites/default/files/omb/assets/procurement_fair/usps_city_state_list.xls

TABLE II
COMPUTATION OVERHEAD

	Basic		Enhanced		Advanced		Multi-attribute Verification	
	hash	*sign*	*hash*	*sign*	*hash*	*sign*	*hash*	*sign*
S-100K	4.92 M	192.28 K	3.71 M	1.2 K	3.38 M	1.2 K	1.01 M	3
S-1M	50.5 M	1.98 M	35.9 M	3.49 K	34.5 M	3.49 K	82.82 M	3
S-1.5M	76.0 M	2.98 M	53.74 M	4.07 K	52.06 M	4.07 K	186.44 M	3



(a) *S*-100K (b) *S*-1M (c) *S*-1.5M

Fig. 1. Comparison of query processing time.

TABLE III
COMPUTATION TIME

	Basic	Enhanced	Advanced	Multi-attribute Query
S-100K	190.48 s	15.23 s	15.67 s	4.63 s
S-1M	2105.36 s	135.41 s	140.01 s	452.97 s
S-1.5M	2954.26 s	201.22 s	208.42 s	916.63 s

TABLE IV
STORAGE OVERHEAD

	Basic	Enhanced	Advanced	Multi-attribute Query
S-100K	121.22 MB (38.41%)	31.16 MB (9.87%)	1.44 MB (0.45%)	121.13 KB (0.04%)
S-1M	1.22 GB (38.98%)	125.42 MB (3.91%)	5.49 MB (0.17%)	41.36 MB (1.29%)
S-1.5M	1.83 GB (38.85%)	155.8 MB (3.23%)	6.76 MB (0.14%)	56.35 MB (1.17%)

TABLE V
COMMUNICATION OVERHEAD OF MULTI-ATTRIBUTE QUERY

Query type	*S*-100K	*S*-1M	*S*-1.5M
$\bigwedge$	21.57 KB	156 KB	251.65 KB
$\bigvee$	96.43 KB	8.19 MB	23.86 MB

TABLE VI
VERIFICATION TIME OF MULTI-ATTRIBUTE QUERY

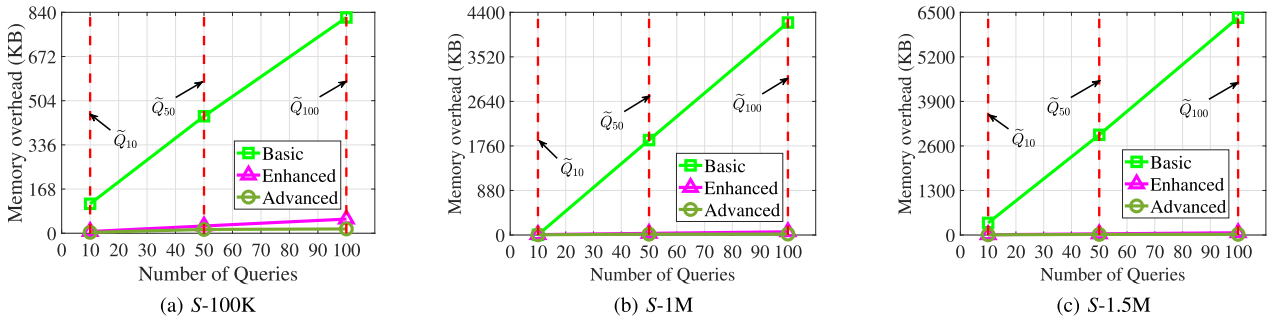
Query type	*S*-100K	*S*-1M	*S*-1.5M
$\bigwedge$	0.95ms	56.51ms	114.33ms
$\bigvee$	2.48s	716.72s	1,565.52s

enhanced scheme take the almost same amount of time for data preprocessing which are shorter than that of the basic scheme, and the multi-attribute query scheme takes longer time for data preprocessing than the enhanced advanced schemes. For example, the computation time in the basic scheme is 14.04 times of that in the advanced scheme for *S*-1.5M, and the computation time of the multi-attribute query scheme is more than the enhanced and advanced schemes.

2) Storage Overhead: Table V shows the storage overhead incurred by the auxiliary information, i.e., the total size of signatures, internal nodes of MHTs, Bloom filters, and Cuckoo filters, in the basic, enhanced, advanced, and multi-attribute query schemes in bits, respectively. We can see from Table V that the basic scheme incurs the highest storage overhead, followed by the enhanced scheme, the multi-attribute query scheme, and the advanced scheme. Consider *S*-1.5M as an example. The storage overhead of the basic, enhanced, advanced, and multi-attribute query scheme are 1.83 GB, 155.8 MB, 6.76 MB or 56.35 MB, respectively. This is of no surprise, as the storage overhead for signatures and hash values in the basic and enhanced schemes are $\langle O(n), O(m_k)\rangle$, $\langle O(m_k), O(nm_k)\rangle$, respectively, and the advanced scheme utilizes space-efficient data structure (Bloom filter) to reduce the storage overhead. Furthermore, the storage overhead for signatures in the multi-attribute query scheme is $O(w+\sum_{k=1}^{w} m_k)$. Table V also shows the ratio between the size of the auxiliary information and that of original user data. It is clear that the advanced scheme incurs very small additional storage overhead for auxiliary information, and the multi-attribute query scheme has less additional storage overhead than the enhanced scheme.

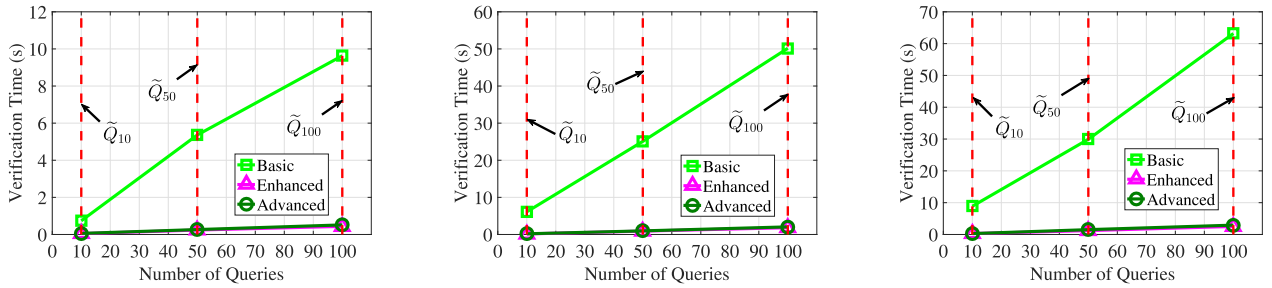
C. Query Processing

To evaluate the computation and communication overhead incurred by single-attribute query processing, we generate three types of queries: $\widetilde{Q}_{10}$, $\widetilde{Q}_{50}$, and $\widetilde{Q}_{100}$, where the query $\widetilde{Q}_x$ means randomly choosing x cities as the query condition for $x = 10, 50$ and 100. Figs. 1a, 1b and 1c show the query processing times of three schemes for the three types of queries for all three datasets, respectively, where each point represents the average of 100 runs, each with a random seed. We can see that the query processing time of the basic scheme is the longest and those of the enhanced and advanced schemes are similar. For *S*-1.5M, it takes only 807 ms to process 100 queries under the advanced scheme. In addition, we also evaluate the communication overhead incurred by transmitting verification objects as shown in Figs. 2a, 2b and 2c. We can see that the communication overhead incurred by verification objects under the advanced scheme is slightly lower than that of the enhanced scheme and significantly lower than that of the basic scheme. For *S*-1.5M, transmitting verification



(a) S-100K (b) S-1M (c) S-1.5M

Fig. 2. Comparison of communication overhead.



(a) S-100K (b) S-1M (c) S-1.5M

Fig. 3. Comparison of query-result verification time.

objects incurs only 19.43 KB of communication overhead for 100 queries under the advanced scheme.

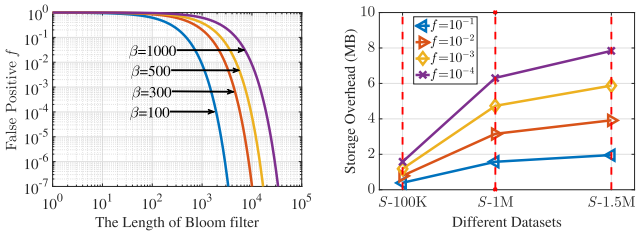
To evaluate the computation and storage overhead of the multi-attribute query scheme, we take two typical multi-attribute queries into account, i.e., the AND query and the OR query. We run the query processing 100 times and randomly select one attribute value for each attribute each time. The computation overhead of the query processing is tiny, e.g., 10^{-5} s, so we omit it here. In addition, Table V shows the communication overhead incurred by verification objects in multi-attribute query scheme. We can see that the communication overhead for AND queries is smaller than that for the OR queries. It is reasonable that the result of an OR query includes more nodes than an AND query, which results in more CFs being returned and thus higher communication overhead.

D. Query-Result Verification

Figs. 3a to 3c compare the computation time for single-attribute query-result verification under three proposed schemes for three datasets. We can see that the verification time in the advanced and enhanced schemes are almost the same, followed by the basic schemes. This is expected, as the computation complexities of signature verification in three schemes are $O(zm'_k)$, $O(m'_k)$, and $O(m'_k)$, respectively, where z and m'_k are the number of nodes and the number of unique attribute k values in the query result, respectively. Moreover, the average verification time in the advanced scheme is 29.4 ms for the dataset S-1.5M, which clearly shows its high efficiency. In addition, Table VI shows the average verification time of two queries. The verification overhead of the AND query is higher than that of the OR query and increases with the scale of the datasets.

E. Impact of Bloom Filter on the Advanced Scheme

Fig. 4a shows the false-positive probability of a Bloom filter varying with its length, where the number of elements



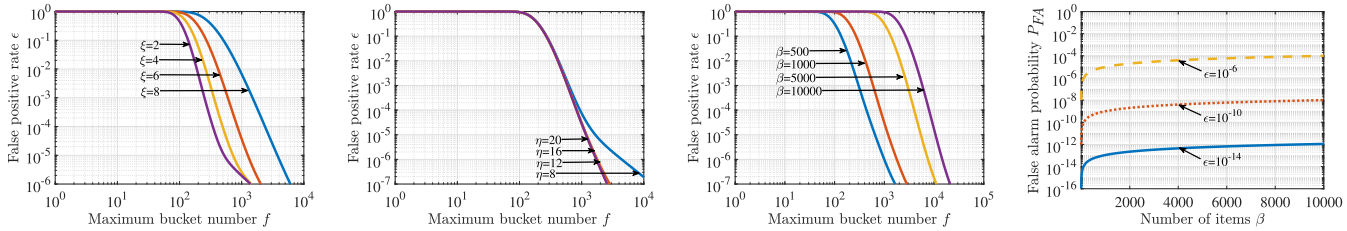
(a) Bloom filter size (b) Different f

Fig. 4. Impact of bloom filter size on false positive probability and storage overhead.

$\beta = 100, 300, 500$, and $1,000$. We can see that the lower the false positive probability we desire, the larger the Bloom filter needs to be. In addition, the more elements inserted into a Bloom filter, the higher the false positive probability, and vice versa. These results coincide with the property of Bloom filter. Fig. 4b shows the storage overhead incurred by all the Bloom filters for S-100K, S-1M and S-1.5M with false positive probability varying from 10^{-1} to 10^{-4}. We can see that if we reduce false positive probability from 10^{-3} to 10^{-4}, the storage overhead incurred by all the Bloom filters for S-1.5M increases by only 2 MB. Recall that Table V that the total storage overhead of the advanced scheme is 6.76 MB for S-1.5M when $f = 10^{-3}$, so the total storage overhead is 8.76 MB for S-1.5M when $f = 10^{-4}$, which is still significant lower than that of the basic and the enhanced schemes.

F. Impact of CF on the Multi-Attribute Query Scheme

Figs. 5a to 5c show the false-positive probability of a cuckoo filter varying with the number of non-empty entries per bucket ξ, the fingerprint size η, and the number of items β. We can see that the more entries per bucket, the larger the fingerprint size, the lower the false positive probability of the CF. In addition, the more items inserted into the memory-optimized cuckoo filter, the higher the false positive probability, and vice versa.



(a) Different ξ (b) Different η (c) Different β (d) Different ϵ

Fig. 5. Maximum number of non-empty entries per bucket ξ.

These results coincide with the properties of the cuckoo filter. Fig. 5d shows the false alarm probability of Algorithm 1 varying with the false positive probability. We can see that the more items in the cuckoo filter, the higher the false alarm probability, and vice versa.

VII. Related Work

Several efforts have been made to ensure the integrity of outsourced query over graph data. Goodrich *et al.* [19] proposed a query scheme to enable verification of the connectivity of two nodes in an outsourced graph. Yiu *et al.* [20] introduced a scheme for authenticating outsourced shortest path queries.Moreover, Fan *et al.* [21] proposed a technique for authenticating outsourced subgraph queries, which ask for all the graphs that contain a given subgraph. Since these works consider different types of queries we consider different queries, none of them can be applied to our problem.

Authenticating outsourced queries have been studied extensively in the past. Narasimha *et al.* [22] proposed an approach based on signature chain to verify the integrity of dynamic databases, which was subsequently improved by Pang *et al.* [23]. In [24], the authors proposed efficient authentication schemes for single- and multi-attribute range queries. Authenticating GET queries over outsourced multi-version key-value stores were studied in [25], [26]. In addition, Zhang *et al.* [27] presented several techniques to enable efficient verification of outsourced location-based top-k queries. Bajaj *et al.* [28] proposed ConcurDB to extend query authentication to multiple clients and presented a novel protocol to eliminate the inefficiency of authenticated data structures on updates. Other types of queries that have been studied include aggregation queries [29], kNN queries [30], [31], top-k spatial keyword queries [32], [33], Boolean queries [34], SQL queries [35], [36], skyline queries [37]–[40], and probabilistic queries over uncertain data [41]. None of these schemes target query over graph data, so they cannot be applied to our problem.

VIII. Conclusion

In this paper, we initiated the study of verifiable social data outsourcing and proposed four solutions to allow the data consumer to verify the social-graph correctness, social-graph completeness, and content authenticity of any query result returned by an untrusted SDP. The efficacy and efficiency of our solutions have been confirmed by extensive experiments based on real Twitter dataset.

References

[1] X. Yao, R. Zhang, Y. Zhang, and Y. Lin, "Verifiable social data outsourcing," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Atlanta, GA, USA, May 2017, pp. 1–9.

[2] B. Jason. (Mar. 2014). *The Impact of Social Media Marketing Trends on Digital Marketing*. [Online]. Available: http://www.socialmediatoday.com/content/impact-social-media-marketing-trends-digital-marketing

[3] F. Morstatter, J. Pfeffer, H. Liu, and K. Carley, "Is the sample good enough? Comparing data from Twitter's streaming API with Twitter's firehose," in *Proc. ICWSM*, Boston, MA, USA, Jul. 2013, pp. 400–408.

[4] F. Morstatter, H. Dani, J. Sampson, and H. Liu, "Can one tamper with the sample API?: Toward neutralizing bias from spam and bot content," in *Proc. 25th Int. Conf. Companion World Wide Web (WWW Companion)*, Montreal, QC, Canada, Apr. 2016, pp. 81–82.

[5] C. Patrick. (Jul. 2014). *Yelp's Newest Weapon Against Fake Reviews: Lawsuits*. [Online]. Available: http://www.bloomberg.com/news/articles/2013-09-09/yelps-newest-weapon-against-fake-reviews-lawsuits

[6] A. Chang. (Aug. 2013). *Tempers Flare at Yelp's Town Hall for Small Business Owners in L.A*. [Online]. Available: http://articles.latimes.com/2013/aug/21/business/la-fi-tn-yelp-town-hall-reviews-20130820

[7] J. Zhang, J. Sun, R. Zhang, and Y. Zhang, "Your actions tell where you are: Uncovering Twitter users in a metropolitan area," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Florence, Italy, Sep. 2015, pp. 424–432.

[8] J. Zhang, X. Hu, Y. Zhang, and H. Liu, "Your age is no secret: Inferring microbloggers' ages via content and interaction analysis," in *Proc. ICWSM*, Cologne, Germany, May 2016, pp. 476–485.

[9] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, "Authentic data publication over the Internet," *J. Comput. Secur.*, vol. 11, no. 3, pp. 291–314, Jul. 2003.

[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[11] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than Bloom," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, Sydney, NSW, Australia, Dec. 2014, pp. 75–88.

[12] R. Pagh and F. F. Rodler, "Cuckoo hashing," *J. Algorithms*, vol. 51, no. 2, pp. 122–144, May 2004.

[13] R. Canetti, O. Paneth, D. Papadopoulos, and N. Triandopoulos, "Verifiable set operations over outsourced databases," in *Proc. PKC*, Buenos Aires, Argentina, Mar. 2014, pp. 113–130.

[14] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. CRYPTO*, Santa Barbara, CA, USA, Aug. 2010, pp. 465–482.

[15] A. Kosba, D. Papadopoulos, C. Papamanthou, M. Sayed, E. Shi, and N. Triandopoulos, "TRUESET: Faster verifiable set computations," in *Proc. USENIX Security*, San Diego, CA, USA, Aug. 2014, pp. 765–780.

[16] C. Costello *et al.*, "Geppetto: Versatile verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, San Jose, CA, USA, May 2015, pp. 253–270.

[17] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Optimal verification of operations on dynamic sets," in *Proc. CRYPTO*, Santa Barbara, CA, USA, Aug. 2011, pp. 91–110.

[18] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, Jan. 2004.

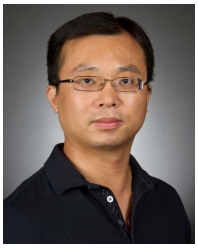
[19] T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated data structures for graph and geometric searching," in *Proc. CT-RSA*, San Francisco, CA, USA, Apr. 2003, pp. 295–313.

[20] M. L. Yiu, Y. Lin, and K. Mouratidis, "Efficient verification of shortest path search via authenticated hints," in *Proc. IEEE 26th Int. Conf. Data Eng. (ICDE)*, Long Beach, CA, USA, Mar. 2010, pp. 237–248.
[21] Z. Fan, Y. Peng, B. Choi, J. Xu, and S. S. Bhowmick, "Towards efficient authenticated subgraph query service in outsourced graph databases," *IEEE Trans. Services Comput.*, vol. 7, no. 4, pp. 696–713, Oct. 2014.
[22] M. Narasimha and G. Tsudik, "Authentication of outsourced databases using signature aggregation and chaining," in *Proc. DASFAA*, Singapore, Apr. 2006, pp. 420–436.
[23] H. Pang, J. Zhang, and K. Mouratidis, "Scalable verification for outsourced dynamic databases," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 802–813, Aug. 2009.
[24] H. Pang and L. Tan, "Verifying completeness of relational query answers from online servers," *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 2, pp. 5:1–5:50, May 2008.
[25] Y. Tang, T. Wang, L. Liu, X. Hu, and J. Jang, "Lightweight authentication of freshness in outsourced key-value stores," in *Proc. 30th Annu. Comput. Secur. Appl. Conf.*, New Orleans, LA, USA, Dec. 2014, pp. 176–185.
[26] Y. Hu, R. Zhang, and Y. Zhang, "KV-Fresh: Freshness authentication for outsourced multi-version key-value stores," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 1638–1647.
[27] R. Zhang, Y. Zhang, and C. Zhang, "Secure top-k query processing via untrusted location-based service providers," in *Proc. IEEE INFOCOM*, Orlando, FL, USA, Mar. 2012.
[28] S. Bajaj, A. Chakraborti, and R. Sion, "ConcurDB: Concurrent query authentication for outsourced databases," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1401–1412, Apr. 2021.
[29] C. Xu, Q. Chen, H. Hu, J. Xu, and X. Hei, "Authenticating aggregate queries over set-valued data with confidentiality," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 630–644, Apr. 2018.
[30] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi, "Spatial query integrity with Voronoi neighbors," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 863–876, Apr. 2013.
[31] N. Cui, X. Yang, B. Wang, J. Li, and G. Wang, "SVkNN: Efficient secure and verifiable k-nearest neighbor query on the cloud platform," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Dallas, TX, USA, Apr. 2020, pp. 253–264.
[32] D. Wu, B. Choi, J. Xu, and C. S. Jensen, "Authentication of moving top-k spatial keyword queries," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 4, pp. 922–935, Apr. 2015.
[33] Q. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, "Enabling verifiable and dynamic ranked search over outsourced data," *IEEE Trans. Services Comput.*, early access, Jun. 11, 2019, doi: 10.1109/TSC.2019.2922177.
[34] S. Jiang, X. Zhu, L. Guo, and J. Liu, "Publicly verifiable Boolean query over outsourced encrypted data," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 799–813, Jul. 2019.
[35] B. Zhang, B. Dong, and W. H. Wang, "Integrity authentication for SQL query evaluation on outsourced databases: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1601–1618, Apr. 2021.
[36] B. Zhang, B. Dong, and W. H. Wang, "CorrectMR: Authentication of distributed SQL execution on MapReduce," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 897–908, Mar. 2021.
[37] X. Lin, J. Xu, and H. Hu, "Authentication of location-based skyline queries," in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, New York, NY, USA, Oct. 2011, pp. 1583–1588.
[38] X. Lin, J. Xu, and J. Gu, "Continuous skyline queries with integrity assurance in outsourced spatial databases," in *Proc. WAIM*, Harbin, China, Aug. 2012, pp. 114–126.
[39] X. Lin, J. Xu, H. Hu, and W.-C. Lee, "Authenticating location-based skyline queries in arbitrary subspaces," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 6, pp. 1479–1493, Jun. 2014.
[40] W. Chen, M. Liu, R. Zhang, Y. Zhang, and S. Liu, "Secure outsourced skyline query processing via untrusted cloud service providers," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (IEEE INFOCOM)*, San Francisco, CA, USA, Apr. 2016, pp. 1–9.
[41] B. Zhang, B. Dong, H. Sun, and W. H. Wang, "AuthPDB: Authentication of probabilistic queries on outsourced uncertain data," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, New Orleans, LA, USA, Mar. 2020, pp. 121–132.

Xin Yao (Member, IEEE) received the B.S. degree in computer science from Xidian University in 2011, and the M.S. degree in software engineering and the Ph.D. degree in computer science and technology from Hunan University, in 2013 and 2018, respectively. From 2015 to 2017, he worked as a Visiting Scholar with Arizona State University. He is currently an Assistant Professor with Central South University. His research interests include security and privacy issues in social networks, the Internet of Things, cloud computing, and big data.

Rui Zhang (Member, IEEE) received the B.E. degree in communication engineering and the M.E. degree in communication and information system from the Huazhong University of Science and Technology, China, in 2001 and 2005, respectively, and the Ph.D. degree in electrical engineering from Arizona State University, in 2013. From 2013 to 2016, he was an Assistant Professor with the Department of Electrical Engineering, University of Hawaii. He is currently an Associate Professor with the Department of Computer and Information Sciences, University of Delaware. His primary research interests are network and distributed system security, wireless networking, and mobile computing. He received the U.S. NSF CAREER Award in 2016.

Dingquan Huang (Student Member, IEEE) received the B.E. degree in the Internet of Things engineering from Jinan University in 2019. He is currently pursuing the master's degree with the School of Computer Science and Engineering, Central South University, China. His research interests include security and privacy issues in the Internet of Things.



Yanchao Zhang (Fellow, IEEE) received the B.E. degree in computer science and technology from the Nanjing University of Posts and Telecommunications in 1999, the M.E. degree in computer science and technology from the Beijing University of Posts and Telecommunications in 2002, and the Ph.D. degree in electrical and computer engineering from the University of Florida in 2006. He is a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University. His primary research interests are security and privacy issues in computer and networked systems, with current focus areas in emerging wireless networks, mobile crowdsourcing, the Internet-of-Things, social networking and computing, wireless/mobile systems for disabled people, big data analytics, mobile/wearable devices, and wireless/mobile health. He received the U.S. NSF CAREER Award in 2009. He chaired the 2017 IEEE Conference on Communications and Network Security (CNS), the 2016 ARO Workshop on Trustworthy Human-Centric Social Networking, the 2015 NSF Workshop on Wireless Security, and the 2010 IEEE GLOBECOM Communication and Information System Security Symposium. He has been on the Editorial Board of IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE WIRELESS COMMUNICATIONS, IEEE CONTROL NETWORK SYSTEMS, and IEEE TRANSACTIONS VEHICULAR TECHNOLOGY.