Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Authenticity verification on social data outsourcing

Haowen Chen[a,*], *Qiang Qu*[a], *Yexiong Lin*[a], *Xia Chen*[a,b], *Keqin Li*[c]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, China
[b] School of Basic Education, Changsha Aeronautical Vocational and Technical College, Changsha, Hunan, China
[c] Department of Computer Science, State University of New York, New Paltz, New York 12561, USA

ARTICLE INFO

Article history:
Received 30 January 2020
Revised 30 August 2020
Accepted 5 October 2020
Available online 8 October 2020

Keywords:
Social data
Fake data
Fake detection
Authenticity verification
Social network

ABSTRACT

In the social data transaction model, online social networks sell their collected social data to a third-party data service provider, which commonly resells such social data to data users for further mining potential information. However, the data service provider may not be trustworthy and collude with others to return fake data to users. To prevent such malicious activities, data users should verify the *correctness* and *completeness* of social data purchased from the data service provider to make sure that no data would tamper and no qualifying results would be omitted. Accordingly, we first propose an authenticity verification scheme, called *FakeDetection*, for one-dimensional data query. To make our scheme becoming efficient, we further devise an enhanced probabilistic scheme *FakeDetection*$^+$, which takes partial vertices and neighbors with the identical profile value to generate auxiliary information. To evaluate the efficiency of our schemes, we utilize the real Twitter datasets with 1.6M twitters to perform the experiments. The *Twitter* with *FakeDetection*$^+$, takes 69K edges (47M in *FakeDetection*) into account to detect fake activities with a probability of more than 99%. For the computation overhead, the *FakeDetection*$^+$ scheme only consumes 27.9s (3.8% of that in *FakeDetection*) to generate auxiliary information for a social graph with 1.6M twitters and their social network.

1. Introduction

With the development of online social networks (OSNs), we are facing a revolutionary way of social interactions and communications. Notably, the *Twitter*, as a kind of OSN, is becoming pervasive because of its effectiveness. The *Twitter* possesses a significant amount of social data, which can be viewed as a social network with the particular graph structure composed of individuals (or organizations) and connections among these individuals. In this graph, each individual or organization possesses its profile, like gender, age, and political affiliation for personal, and band, features, and specialized services for organizations, etc.

Massive social media data have a critical potential economic valuation for many real-world applications. Puschmann and Burgess (2013) proposed a social data selling model, in which the Twitter first sells social data to the third-party data service provider (like *Gnip* and *DataSift*), which then resells such data to data users for commercial or academic data mining.

However, many black stores show that the data service provider may collude with advertising companies to *add/delete/modify* its social data to achieve business profits. For

* Corresponding author.
E-mail addresses: hwchen@hnu.edu.cn (H. Chen), quqiang@hnu.edu.cn (Q. Qu), yexiong_lin@hnu.edu.cn (Y. Lin), 19557092@qq.com (X. Chen), lik@newpaltz.edu (K. Li).
https://doi.org/10.1016/j.cose.2020.102077
0167-4048/

instance, Google was deemed to manipulating search suggestions to support Democratic presidential hopeful Hillary Clinton when she was stuck in "Email Controversy" (Richardson, 2016). Specifically, when someone types "Hillary Clinton cri", Google provides search suggestions, like "Hillary Clinton crime reform" and "Hillary Clinton crisis", while Yahoo or Bing suggests "Hillary Clinton crimes". Another example is that the famous review website Yelp forced businesses to manipulate reviews to achieve business profits (Richardson, 2013). Similarly, the data service provider in this paper also may tamper social data by launching adding/deleting/modifying attacks or returning fake query-results. These behaviors violate the interests of data users and have severe effects on our daily lives and our legal right seriously. To guarantee the integrity of query results, some methods have been proposed to verify the correctness and completeness of query results, like Merkle hash tree (Bertino et al., 2004; Li et al., 2007; Mouratidis et al., 2009a; Pang and Mouratidis, 2008; Papadopoulos et al., 2007; Yang et al., 2008; Zhang et al., 2012) and signature aggregation (Narasimha and Tsudik, 2006; Pang and Tan, 2008; Pang et al., 2009). However, such schemes did not take into consideration friendships among vertices in social networks. To verify the completeness of query graphs, Goodrich et al. (2003) proposed a scheme to verify whether two nodes are connected in the graph, and Yiu et al. (2010) proposed the landmark-based verification method (LDM) to verify whether the query result is the shortest path on the original graph. Nevertheless, these previous studies focus on the *completeness* of social networks while ignoring the profiles of vertices.

In this study, we are among the first to describe the definitions of correctness and completeness of social data. For the *correctness* aspect, social data, received by users, are all correct (refer to Definition 1). While for the *completeness* of social data, we define it as the following three aspects: *Vertex Completeness, Profiles Completeness*, and *Friendships Completeness* (refer to Definition 2). To verify social data, we first propose a deterministic basic scheme *FakeDetection* to implement the above *correctness* and *completeness* verification. However, the *Twitter* needs to take a large overhead to generate auxiliary information. To further make our scheme efficent, we propose a probabilistic scheme *FakeDetection+* to check whether the data service provider tampers the original social data before selling to data users.

Here, our contributions are listed as follows:

- Aiming at real applications, we consider the correctness and completeness verification of social data.
- To address the verifiable problem, we first propose a deterministic verifying method, *FakeDetection*, which can detect fake activities.
- To further reduce the computation overhead, we propose a probabilistic scheme *FakeDetection*$^{+}$, which takes nearly 3.8% computation overhead of the *FakeDetection* to achieve the detection probability of 99% for 1.6M twitters.
- To demonstrate the performance of our schemes, we conduct the experiment on real social data (*Twitter* data), and the results demonstrate our schemes are very efficient for the current data size.

The rest of this paper is organized as follows. We first describe related work in Section 2. We then introduce the system and adversary models in Section 3. Section 4 presents the problem definitions. Next, we present our basic scheme in Section 5 and our enhanced scheme in Section 6. In Section 7, we analyze the detection results. Section 8 analyzes the performance of our proposed methods. Sections 9 and 10 present our experimental results and conclude our study.

2. Related work

Our work is the most similar to data outsourcing (Hacigümüs et al., 2002), for which we can discuss state-of-the-art. The framework of data outsourcing, including data owner, service provider, and other users, was first introduced in Hacigümüs et al. (2002). A data owner outsources its data to a third-party service provider who is responsible for answering the data queries from either the data owner or other users. Generally, query result verification is the main security concern in data outsourcing (Ku et al., 2009).

To ensure the query integrity, the service provider returns a Verification Object ($\mathcal{VO}$) with the answer to each query, which permits other users to verify the correctness and completeness of the answer. Many techniques were proposed for signature and $\mathcal{VO}$ generations, such as those (Narasimha and Tsudik, 2006; Pang and Tan, 2008; Pang et al., 2009) based on signature chaining and those (Bertino et al., 2004; Li et al., 2007; Mouratidis et al., 2009a; Pang and Mouratidis, 2008; Papadopoulos et al., 2007; Yang et al., 2008) based on the Merkle hash tree (MHT) (Merkle, 1989) or its variants.

Signature chaining **:** The *data owner* first sorts the data tuples and generates the signature with its details and the tuples immediately to its predecessor and to its successor. For the proof of a query result, it contains the signature of every returned tuple. The chain generated by signing consecutive triples of tuples ensures the completeness of the result and the authenticity of each returned tuple. Mykletun et al. (2006) first proposed a signature-based method for range query authentication. Narasimha and Tsudik (2006) proposed an approach based on a signature chain (named *DSAC*), which provided the correctness and completeness for the more challenging case of dynamic databases. However, Pang et al. (2009) pointed out that the *DSAC* either costs huge correctness proofs due to requiring a pair of boundary values for each unmatched record or materializes the join result. In order to reduce the overhead of the proof construction, they proposed a novel signature caching scheme, called *SigCache*. Besides, Pang and Tan (2008) first introduced the range query scheme, and proposed efficient authentication schemes for single- and multi-attribute range aggregate queries. However, each element, under signature chain schemes, should be with a large overhead signature operation and it is intuitive. We focus on how can the Merkle hash trees effectively applied to solve the problem mentioned above, and the amount of the profile of the users, direct use of the Merkle hash tree is expensive and inefficient. Thus, we do not adopt the signature chaining schemes in this study.

MHT authentication**:** The original designation is that an MHT with an identity structure is embedded into the data

index. Such an index is typically a B^+-tree (Mouratidis et al., 2009b) for one-dimensional data; for multi-dimensional data, the *R*-tree (Guttman, 1984) and the *KD*-tree (Bentley, 1975) have both been considered. For the proof of a query result, it reconstructs the MHT root digest and tests whether it matches the owner's root signature. MHT techniques have been utilized to verify spatial (Mouratidis et al., 2009a; Mykletun et al., 2003; Niaz and Saake, 2015; Yang et al., 2008), continuous (Li et al., 2007; Papadopoulos et al., 2007), XML (Devanbu et al., 2003) and text search (Pang and Mouratidis, 2008) queries. Though MHT suffers from the limitations in coping with updates because every data modification must propagate from the leaf to the root of the index, which generates multiple I/Os if the index resides on disk, the update for the twitters' profiles is limited, so we do adopt the MHT in this study.

Table 1 – Notations and Explanations.

Notation	Explanation
SSP	Social Service Provider
DSP	Data Service Provider
DUs	Data Users
G	Graph
$e_{i,j}$	Edge
$v_{i,j}$	Vertex
$a_{i,j}$	Profiles values
E	The edge set
V	The vertex set
A	Profiles
VI	Auxiliary information

3. System and adversary models

3.1. System model

Here, we describe the overall system model for social data transactions, including three entities: *Social Service Provider, Data Service Provider* and *Data Users*.

- *Social Service Provider* (called *SSP*): It is well-known as the service provider to provide social data service, e.g., *Twitter, Facebook* and *LinkedIn*. On these platforms, social users fulfill their own profiles to let other users know them easily. Hence, the *SSPs* commonly contain a large amount of social data, e.g., *users' profiles* and *postings*. Here, we take the *Twitter* as an specific example of the *SSP*, and define it as social data selling entity to sell its data to the *Data Service Provider*. To simplify our discussion here, we only consider the static data. Dynamic data is left for the future work.
- *Data Service Provider* (termed *DSP*): In *social data* market, the *DSP* commonly exists as an data reseller to resell social data, including *Gnip* and *DataSift, etc.* In general, the *DSP* first buys the original social data ("*firehose*") from the *Twitter*, and then resell specific social data to users according to their demands.
- *Data Users* (called *DUs*): With the massive potential economic valuations of social data, *DUs*, like personnel, companies and academic institutes, wish to mine such potential information with the data mining methods. For such cases, *DUs* first buy social data from the *DSP*, and then analyze these data. For example, a localized consulting company, to investigate the following situations in "New York" area, such as "*How many citizens will go to Target to buy gift cards in Christmas Day?*" or "*What are the top-k popular restaurants in the New York area?*", sends the query $\tilde{Q}$(*location*="New York") to the *DSP*, and buys the corresponding social data.

3.2. Adversary model

Based on the system model, we further design our adversary model to meet the requests of efficient applications. As an original social data provider, the *Twitter* is assumed as a trusted entity. In other words, it does not maliciously *add/delete/modify* the original social data, and honestly runs our designed verification protocols. Notice that, the *Twitter* may preprocess social data before selling for guaranteeing the *twitters*' privacy, we state that such pre-processing, differing from the above malicious activities, does not affect the final results of data mining on the side of the *DUs*. Thus, we denote social data after such pre-processing as the original social data.

While the *DSP*, in our adversary model, is assumed as an untrusted entity. To pursue economic profits, the *DSP* may collude with *personnel* or *advertisement companies* to *add/delete/modify* social data, hence leading to change data analysis results. Obviously, such malicious activities directly affect the profits of *DUs*.

To prevent such malicious activities, the *DUs* should verify the *correctness* and *completeness* of the social data bought from the *DSP*. To achieve such goals, the *Twitter* generates auxiliary information and attaches this information with the original social data before selling to the *DSP*. Aiming at buying requests from the *DUs*, the *DSP* returns not only the social data satisfying their demands, but also the corresponding verification information. Then, the *DUs* verify the *correctness* and *completeness* of the purchased data according to the verification information.

4. Problem definitions

In this section, we formally depict the definition of our problem based on the system and adversary models. Before that, we first introduce the representations of social data on the *Twitter* platform. For convenience, all the notations used in this paper are listed in Table 1.

Social data in *Twitter* can be classified into two categories: the twitters' data (e.g., the twitters' profiles and the friendships among the twitters) and social information (such as tweets[1] with contextual, image or short video). Generally speaking, the twitters' data is considered as a social network, which can be represented as a graph *G*. In the social graph *G*, it also consists of two element sets as well as

[1] In this paper, we only focus on the twitters' data, and further address the tweets completeness in our future research. If not specific statement, social data means the twitters' data.

other graphs: the vertex set $V=\{v_1, \cdots, v_n\}$, and the edge set $E=\{e_{1,2}, \cdots, e_{i,j}, \cdots, e_{n-1,n}\}$ ($i \neq j$), where n denotes the number of vertices, and $e_{i,j}=1$ if there is a friendship between vertices v_i and v_j, otherwise 0. Here, we focus on mutual friendships among twitters for brevity, and then expand our schemes to the more efficient friendships (i.e., *follower/followee* relationship). Therefore, the social graph $G=(V,E)$ can be viewed as an undirected graph.

For each twitter in social data, we assume that he/she has the same number of attributes, from A_1 to A_w (w means the number of the profiles). More specifically, each vertex in the social graph G has specific profile values, namely, the vertex v_i has the profiles values $(a_{i,1}, \cdots, a_{i,w})$.

In our system model, the *DSP* returns specific social data to the *DUs* according to their demands. Based on the previous representations of social data, the returned social data is actually an specific *isomorphism* subgraph $G'=(V',E')$ of the social graph G. However, the *DSP*, as an untrusted entity, may tamper with the subgraph G', such as *adding/deleting/modifying* the twitters' profile values (for vertices) or the friendships among the twitters (for edges). To detect such malicious activities systematically, we formally define the *correctness* and *completeness* of the subgraph G' as follows.

Definition 1. For a value a_s of the profile A_k in a subgraph G', *if and only if* all vertices in the V' are with the identical profile value a_s of the profile A_k, the subgraph G' satisfies the *correctness*. Mathematically, $\forall v_{i'} \in V'$, s.t. $a_{i',k} = a_s$.

Definition 2. The *completeness* of the subgraph G' is defined in three following perspectives: 1) ***vertexcompleteness***: for a profile $A_k = a_s$, the subgraph G' should contain all vertices with such profile value a_s in the original graph G, namely, $|V_{G-G'} \cap V_{G'}|_{A_k=a_s} = \emptyset$; 2) ***profilecompleteness***: for the profile of all vertices in the subgraph G', its values should be the same as those in the original social graph G, i.e., $\forall v_{i'} \in V', \exists v_i \in V$, s.t. $a_{i',k} = a_{i,k}$, $k \in [1,w]$; 3) ***friendshipscompleteness***: the friendships among the vertices in the subgraph G' should be the same as those in the graph G, i.e., $\forall v_{i'}, v_{j'} \in V', \exists v_i, v_j \in E$, s.t. $e_{i',j'} = e_{i,j}$. *If and only if* all these perspectives happen on simultaneously, the subgraph G' can be considered as satisfying the *completeness*.

5. Basic scheme

To guarantee the *correctness* and *completeness* of the subgraph G', we first propose a naïve scheme *FakeDetection* for single profile query in this section.

5.1. Overview of basic scheme

For the social graph G, we assume that it consists of n vertices, corresponding to n twitters. The x-th vertex v_x is with the profile values $(a_{x,1}, \cdots, a_{x,w})$, and m_x^s neighbors (from $vn_{x,1}$ to vn_{x,m_x^s}). In our basic scheme, the *Twitter* first generates auxiliary information, and attaches this information with the original graph G to form a verification graph G_{veri} (refer to Section 5.2). To buy social data, the *DUs* send queries on-demand to the *DSP* and the *Twitter*. This procedure is shown in Fig. 1. After receiving these queries, the *DSP* and the *Twitter* search the social graph, and returns corresponding feedbacks to *DUs*, respectively (as described in Section 5.3). Finally, the *DUs* verify the *correctness* and *completeness* of the query results (as shown in Sect. 5.4). This procedure is shown in Fig. 2.

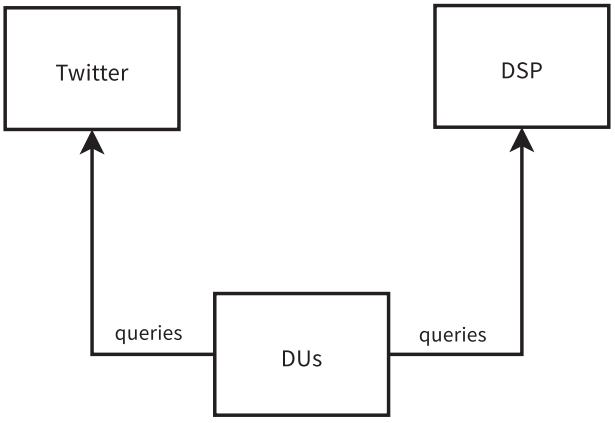


Fig. 1 – Sending queries.

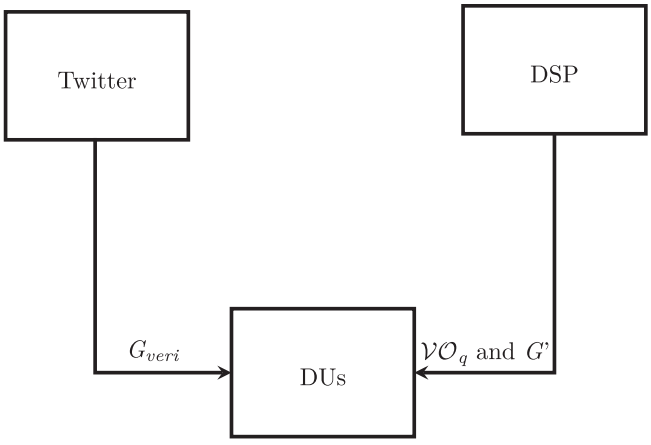


Fig. 2 – Getting the query results.

5.2. Generating a verifiable social graph

For an arbitrary social graph G, the *Twitter* leverages the Algorithm 1 to generate a verifiable graph G_{veri}. Before detailing the algorithm *Gen_Verinfo*, we first define the profile relationship between two vertices as follows.

Definition 3. For two neighbors v_1 and v_2, consisting of w profile values (i.e., $(a_{1,1}, \cdots, a_{1,w})$ and $(a_{2,1}, \cdots, a_{2,w})$), we define dv_k as the *difference value* between the profile values of v_1 and v_2 on the profile A_k, e.g., $dv_{1-2,k}$ and $dv_{2-1,k}$ are equivalent to $a_{1,k} - a_{2,k}$ and $a_{2,k} - a_{1,k}$, respectively. Apparently, the value of $dv_{1-2,k}$ is opposite to that of $dv_{2-1,k}$.

In algorithm *Gen_Verinfo*, the original graph G, and a private key sk for signatures are taken as the input. Firstly, the algorithm initializes an array set $\{\Psi_k\}$, $k \in [1, w]$. Each element in the array Ψ_k contains two parts: prefix and suffix. The initial array Ψ_k consists of two boundary elements, the prefix of the first element is the defined public minimum value ξ_k^{min} for

Algorithm 1: *Gen_Verinfo.*

Input: G, private key sk
Output: G_{veri}

1 Initialize an array set $\{\Psi_k\}$, $k \in [1, w]$;
2 **for** *Traverse each vertex* v_x **do**
3 Generate the auxiliary information $VI_{x,0} = \mathcal{S}_{sk}(h(a_{x,1}) \oplus \cdots \oplus h(a_{x,w}) \oplus r_x)$;
4 **for** *Traverse each profile* A_k **do**
5 Find the corresponding position ν via comparing the prefix values of the elements of the Ψ_k with the profile value $a_{x,k}$ with the alphabetical/numerical order;
6 **if** *The value* $a_{x,k}$ *is equivalent to the prefix value of* $\Psi_k(\nu)$ **then**
7 Update the suffix of the $\Psi_k(\nu)$ with the *Bit-XOR* value between the suffix and $h(ID_x)$;
8 **else**
9 Insert a new element at the position ν of the array Ψ, and setting the prefix as$a_{x,k}$, and the suffix as $h(ID_x)$;
10 $VI_{x,k}$=*Compute_VIA*$(v_x, vn_{x,y}, A_k)$, $y \in [1, m_x^s]$;
11 Sign suffixes of the array Φ_k with the private key sk of *Twitter*, and calculate all internal nodes of MHT and the root node signed by the key sk with the input of the prefixes of the Ψ_k;
12 Attach VI_x=$\{VI_{x,0}, \cdots, VI_{x,w}\}$ with the original vertex v_x to form verifiable vertex $v'_x = v_x||VI_x$;
13 Return the verifiable graph G_{veri};

the profile A_k, and that of the second element is the defined public maximum value ξ_k^{max}. For the suffixes of the first and second elements, we randomly set their values. For generating auxiliary information, the algorithm traverses all vertices in the graph G. For each vertex v_x, the *Twitter* generates the first auxiliary information $VI_{x,0}$ as shown in Line 3. Here, the algorithm calculates hash values for all profile values of the current vertex, and the *Bit-XOR* value for these hash values. To resist known plaintext attack, we introduce a disturbing part $r_x = h(ID_x)$. Finally, the *Twitter* utilizes its private key sk to sign the *Bit-XOR* value. After generating the auxiliary information $VI_{x,0}$, the algorithm traverses each profile value $a_{x,k}$ of the vertex v_x. Based on the comparing between the profile value $a_{x,k}$ and the prefixes in the array Ψ_k, the algorithm first finds the corresponding position ν. If the profile value $a_{x,k}$ is identical with the prefix of $\Psi_k(\nu)$, the suffix of the $\Psi_k(\nu)$ is updated with the *Bit-XOR* value between the suffix and $h(ID_x)$; Otherwise, the algorithm inserts a new element at the position ν of the array Ψ, and sets the prefix as $a_{x,k}$, and the suffix as $h(ID_x)$. Subsequently, the *Twitter* calls Algorithm 2 to generate the auxiliary information $VI_{x,k}$. Finally, the algorithm signs all suffixes of the array Ψ_k with the private key sk, and utilizes the prefixes of the array Ψ_k to generate all internal nodes of Merkle Hash Tree, and the root node signed with the key sk. Meanwhile, the algorithm attaches auxiliary information with the original vertex v_x to form the verifiable vertex v'_x, i.e., $v_x||VI_x$. Here, we denote (V_{veri}, E) as the verifiable graph G_{veri}. In the following paragraph, we detail the algorithm *Compute_VIA*.

Algorithm 2: *Compute_VIA.*

Input: v_x, $vn_{x,y}$, A_k
Output: $VI_{x,k}$

1 Initialize an array Φ;
2 **for** *Traverse each neighbor vertex* $vn_{x,y}$ **do**
3 Calculate the *difference value dv* between the vertex v_x and $vn_{x,y}$ for the profile A_k;
4 Find the corresponding position τ via comparing the prefix values of the elements of the Ψ_k with the value dv;
5 **if** *The value dv is equivalent to the prefix value of* $\Phi(\tau)$ **then**
6 Update the suffix with the *Bit-XOR* value between the previous suffix of the $\Phi(\tau)$ and $h(ID_{vn_{x,y}})$;
7 **else**
8 Insert a new element at the position τ of the array Φ, and setting the prefix asdv, and the suffix as $h(ID_{vn_{x,y}}) \oplus h(a_{x,k}) \oplus r_x$;
9 Sign suffixes of the array Φ with the private key sk of *Twitter*, and calculate the root with the prefixes of the array Φ as inputs of the MHT scheme, and signing its value;
10 Return all internal nodes of Merkle Hash Tree, and the array Φ denoted as $VI_{x,k}$;

In Algorithm 2, it first initializes an array Φ as well as the array Ψ_k containing the prefix and suffix for each element, and two boundary elements. Then, the algorithm traverses each neighbor vertex $vn_{x,y}$, and calculates the difference value dv between the vertex v_x and the neighbor vertex $vn_{x,y}$ for profile A_k. Note that if the profile value $a_{x,k}$ at the vertex v_x is larger than that at $vn_{x,y}$, dv is positive; If they are equal, dv is zero; Otherwise, negative. After finding the position ν, the algorithm finds the corresponding position τ by comparing the prefix values of the elements of the Ψ_k with the value dv. While the value dv is the same with the prefix value of $\Phi(\tau)$, the algorithm updates the suffix of the $\Phi(\tau)$ with the Bit-XOR value between the suffixes of the $\Phi(\tau)$ and $h(ID_{vn_{x,y}})$. Otherwise, the algorithm inserts a new element at position τ of the array Φ, and sets the prefix as dv, and the suffix as $h(ID_{vn_{x,y}}) \oplus h(a_{x,k}) \oplus r_x$. After traversing all neighbors, the *Twitter* signs the suffixes in the array Φ with its private key sk, and generates all internal nodes of Merkle Hash Tree, and the root node also signed with the key sk. Hence, the auxiliary information $VI_{x,k}$ includes two perspectives: all internal nodes of Merkle Hash Tree, and the array Φ.

5.3. *Query processing*

Social data contains the massive economic valuations, but the *DUs* commonly request a specific part of data instead of the whole social data. For instance, an investigation company investigates the living habits of the citizen located in "New York", it may only request social data with the profile value "New York" from the *DSP*.

Recall that each twitter has w profiles, from A_1 to A_w. Under this circumstance, the query can be divided into two cat-

egories: *single-profile* query and *multi-profiles* query. For the former query, the *DUs* only focus on one profile for each query (e.g., $\widetilde{Q}(q)$), while the *multi-profiles* query means the *DUs* can combine any profiles with *intersection* and *union* operations. In this study, we only focus on the *single-profile* query, and the *multi-profiles* query is beyond our discussion. For the *single-profile* query, it commonly consists of three basic query styles: *equal* query (e.g., q="New York"), *range* query (e.g., q="20 < *age* < 30"), and *subset* query (e.g., q={"Temp", "New York"}). Generally, *DUs* generate the query based on their demands, and send the query to the *DSP*.

For each query from the *DUs*, the *DSP* searches all vertices in the verifiable social graph G_{veri}, and returns the subgraph *G'* satisfying the query condition q, and the corresponding auxiliary information. During the query processing, the *DSP* first finds out all elements on the array Ψ_k, whose prefix satisfies the query condition q. To verify the *completeness* of the prefix, the *DSP* also needs to consider the boundary data. Based on these prefixes and the boundary data, the *DSP* extracts all internal nodes to generate the root of Merkle Hash Tree, and the corresponding suffix. Hence, the verification object $\mathcal{VO}_0^q$ consists of two parts: all internal nodes, and the corresponding suffixes. Subsequently, the *DSP* traverses all vertices in the social graph, and judges whether the value of each vertex satisfies the query condition q or not. If *false*, the *DSP* continues to search next vertex until all vertices have been traversed; Otherwise, the *DSP* first extracts the signature $VI_{x,0}$ as the verification object $\mathcal{VO}_{x,1}^q$, and then calculates a *difference value dv* between the profile value $a_{x,k}$ and the query condition q, i.e., dv=$a_{x,k}$-q[2] For the value dv, the *DSP* finds out all elements, whose prefixes satisfy the dv. As previously described, the boundary values also need to be considered, and all internal nodes are extracted to generate the root of Merkle Hash Tree, and the corresponding suffixes. Therefore, the verification object $\mathcal{VO}_{x,2}^q$ is defined as two parts: all internal nodes, and the corresponding suffixes. As a result, the final verification object $\mathcal{VO}_q$ can be defined as $\{\mathcal{VO}_0^q, \{\mathcal{VO}_{x,1}^q\}, \{\mathcal{VO}_{x,2}^q\}\}$, where x denote the IDs of the vertices whose profile value satisfy the query condition q. Finally, the verification object $\mathcal{VO}_q$ and the subgraph *G'* are returned to the *DUs*.

5.4. *Correctness and completeness verification*

As assumed in the adversary model, the *DSP* may return the fake query results with *adding, deleting* or *modifying* operations. To prevent such malicious activities and guarantee the rights of the *DUs*, the subgraph *G'* should satisfy the *correctness* and *completeness*, as described in Section 4. In this section, we mainly address the following two problems step by step:

- How to verify the *correctness* of the subgraph *G'* according to the auxiliary information?
- How to verify the *completeness* of the subgraph *G'* according to the auxiliary information?

Recall that the *correctness* of the subgraph means that all vertices in the subgraph contain a profile value identical with the query condition q. To implement such verification, for the subgraph *G'* with m vertices, the *DUs* needs to take the computation overhead of $O(m)$[3] to check whether all vertices in subgraph *G'* contain the profile value identical with the query condition q or not. If *False*, the *DUs* can consider the subgraph *G'* issued from the *DSP* as fake; If *True*, according to Definition 1, the subgraph *G'* can be considered as satisfying *correctness*.

For the *completeness* of the subgraph *G'*, Definition 2 depicts three perspectives, *vertex completeness, profile completeness* and *friendship completeness*. Here, the *DUs* separately verify these perspectives.

Firstly, to verify the *completeness* of the *vertices*, the *DUs* first re-construct the root node according to the internal nodes of Merkle Hash Tree in the verification object $\mathcal{VO}_0^q$. With the root value, the public key of the *Twitter*, and the root signature, the *DUs* can verify the *completeness* of the prefixes in the array Ψ. If the result of the verification is true, the *DUs* continue to verify the completeness of the suffixes in the array Ψ, and calculate hash values for the *ID* information of all vertices in the subgraph *G'*. With the *Bit-XOR* value of these hash values, and the public key of the *Twitter*, the *DUs* can verify the *vertex completeness* according to verify its value and the verification object $\mathcal{VO}_0^q$. If the signature verification can pass, the vertices of the subgraph *G'* can be viewed as *complete*; Otherwise, the subgraph *G'* includes *fake* vertices.

Secondly, for verifying the *profiles completeness*, the *DUs* calculate the hash values for all profile values and the ID information of each vertex in the subgraph *G'*, and utilize the *Bit-XOR* operation to these hash values. With the final *Bit-XOR* value, the public key of the *Twitter*, and the verification object $\mathcal{VO}_{x,1}^q$ as the inputs, the *DUs* can verify the *completeness* of the signature of $VI_{x,0}$. If the signature $VI_{x,0}$ of each vertex in the subgraph *G'* can be verified, the *DUs* consider the subgraph as the *profile completeness*.

Thirdly, to implement the verification of *friendships completeness*, the *DUs* also traverse all vertices in the subgraph *G'* with the following processes. For the vertex v_x, they first calculate the difference values, and utilize these values with the internal nodes of Merkle Hash Tree in the $\mathcal{VO}_{x,2}^q$ to re-construct the root node. With this root value, the public key of the *Twitter*, and the signature of the root, the *DUs* can verify the *correctness* and the *completeness* of the prefixes in the array Φ. If such verification can pass, the *DUs* generate the suffixes for various difference values with the same method in Algorithm 2. As the same with the verification for signature, the *DUs* take the generated suffixes, the public key of the *Twitter*, and the signed suffixes in the $\mathcal{VO}_{x,2}^q$ as the input, and verify the *completeness* of the suffixes. Likewise, if all suffixes of all vertices can pass, the subgraph *G'* satisfies the *friendships completeness*.

In summary, to guarantee the right of the *DUs*, the subgraph *G'* should be verified the *correctness* and three perspectives of the *completeness*.

5.5. *Security analysis*

To formally analyze the security of the basic scheme, we first describe the following three malicious activities: 1) Firstly, the

[2] Note that, when the query q is *equal, range* or *subset* query, the value dv is a *determine, range*, or *set* value, respectively.

[3] m denotes the number of vertices in subgraph *G*.

DSP may *add/delete* vertices of the subgraph. In this case, some fake vertices and corresponding edges may be inserted into the original subgraph; some true vertices and corresponding edges may be removed from the subgraph. 2) Secondly, the *DSP* may *modify* the profile values of vertices in the subgraph G'. The profile values of the *twitters* as one aspect of social data, may be changed, e.g., modifying *age, political affiliation* or *gender* of the *twitters*. 3) The *DSP* may *modify* the friendships among vertices of the subgraph. In this activity, some fake/true friendships among vertices may be *added/deleted* to/from the subgraph. Even though, both the first and third malicious activities affect edges in the original subgraph, we state that they are different since the change of edges in the first case relies on *adding* or *deleting* vertices, while the third case only changes the friendships among the vertices while keeping on the original vertices.

For the first malicious activity, the *DUs* only need to check whether the vertices in the subgraph are *correct* and *complete* for a specific query condition. Recall that the *Twitter* generates the signatures for vertices with various profile values, and stores them in the array Ψ. In the verification phase as shown in Section 5.4, the *DUs* can easily verify whether the vertices in the subgraph are *correct* and *complete* with the verification object $\mathcal{VO}_0^q$. For the second malicious activity, the *Twitter* in the algorithm *Gen_Verinfo* generates the auxiliary information of the profiles, i.e., $VI_{x,0}$. Obviously, any modifications on the profile values can be detected by the *DUs* with the auxiliary information $VI_{x,0}$. For the third malicious activity, the *Twitter* generates the auxiliary information for the friendships between the current vertex and its neighbors based on their difference values of the profile values. When receiving the query results, the *DUs* first verify the *completeness* of the difference values, and further verify the *completeness* of the suffixes. To summarize, the *FakeDetection* scheme is an effective method to guarantee the *correctness* and *completeness* of the subgraph G'.

According to the above security analysis, we conclude that when the auxiliary information is not compromised, all malicious activities for vertices or edges can be detected. Subsequently, we analyze the security of the auxiliary information, which consists of four operations: *hash, Bit-XOR, Merkle Hash Tree*, and *Signature*. For the *Bit-XOR* operation, the attackers can infer the two values with a very small probability according to the result of the *Bit-XOR*. In this study, we utilize 256-bit for hash function, so the attackers infer the two values with the probability $2/(2^{256}\text{-}1)$. From Algorithms 1 and 2, we can conclude that the security of the auxiliary information can be deduced to the security of *Merkle Hash Tree*, and *Signature* schemes, which are proved to be secure in the previous work (MHT and RSA (Rivest et al., 1978)). Since the *hash* functions are the *one-way* functions, and the attackers do not know the private key *sk* of the *Twitter*, they cannot compromise the security of the auxiliary information.

6. Enhanced scheme

6.1. Overview of enhanced scheme

For the profile A_k, we introduce some notations. For instance, the number of various profile values is m_k (i.e., from $a_{k,1}$ to a_{k,m_k})[4], and the number of twitters with profile value $a_{k,i}$ is $\psi_{k,i}$ ($i \in [1, m_k]$). In other words, the value of $\sum_{i=1}^{m_k} \psi_{k,i}$ is equal to n. For neighbors of the vertex with the profile value $a_{k,i}$, we assume that the number of neighbors is μ_j, the number of various profile values is δ_j ($j \in [1, \psi_{k,i}]$), and the number of neighbors with this profile value is $\chi_{j,z}$ ($z \in [1, \delta_j]$). Therefore, the value of $\sum_{z=1}^{\delta_j} \chi_{j,z}$ is equal to μ_j.

In the *FakeDetection* scheme, we mainly focus on the computation overhead of the signature and the memory overhead of Merkle Hash Tree. Intuitively, either the computation overhead or the memory overhead is very huge, because there are many signatures and a Merkle Hash Tree for each vertex. Next, we formally analyze the computation overhead and the memory overhead in the *FakeDetection* scheme. For computation overhead, the *Twitter* takes

$$N_{sign} = \sum_{k=1}^{w} m_k + 1 + \sum_{k=1}^{w} \sum_{i=1}^{m_k} \sum_{j=1}^{\psi_{k,i}} (\delta_j + 1)$$

signatures. While, for memory overhead, the *Twitter* takes

$$Mem_{sign} = \left(\sum_{k=1}^{w} m_k + 1 + \sum_{k=1}^{w} \sum_{i=1}^{m_k} \sum_{j=1}^{\psi_{k,i}} (\delta_j + 1) \right) \cdot sizeof(sign)$$

and

$$Mem_{MHT} = \left(\log_2 (\sum_{k=1}^{w} m_k) + \log_2 \left(\sum_{k=1}^{w} \sum_{i=1}^{m_k} \sum_{j=1}^{\psi_{k,i}} \delta_j \right) \right) \cdot sizeof(hash)$$

for the social graph *G*. Since the amount of the twitters are large and increase quickly, the computation overhead and memory overhead of the auxiliary information is impractical. Thus, we propose an enhanced scheme based on the following observations to make our scheme more efficient.

- *Observation 1*. To guarantee the *friendship completeness* between the current vertex and its neighbors, for each vertex a Merkle Hash Tree is built, and all suffixes in the array Φ are assigned.
- *Observation 2*. For the assumption of *single-profile* query, we can directly consider the vertices with the same profile value as a whole to guarantee the *friendships completeness*.

According to the observations 1 and 2, we propose an enhanced scheme *FakeDetection*$^+$, where the *Twitter* takes the vertices with the identical profile into account as a whole. In the following, we first depict the generation of auxiliary information and query processing (refer to Sections 6.2 and 6.3), and then discuss the query-result verification (as described in Section 6.4).

6.2. Generating verifiable social graph

In the *FakeDetection*$^+$ scheme, we also leverage the identical notations as in Section 5. Algorithms 3 and 4 show how to

[4] Notation that $a_{x,k}$ denote the profile value of the profile A_k of the vertex v_x, while $a_{k,i}$ means the *i*-th profile value for the profile A_k.

Algorithm 3: *Gen_Verinfo*$^+$.

Input: G, private key sk, the thresholds $t_{k,i}$ and $t_{j,z}$
Output: G_{veri}

Initialize a dictionary dic;
for *Traverse each vertex* v_x **do**
  Generate the auxiliary information $VI_{x,0}$ as described in Alg. 1;
  for *Traverse each profile* A_k, $k \in [1, w]$ **do**
    if *The* $a_{x,k}$ *is in dic* **then**
      Update the element $dic(a_{x,k})$ with the *Bit-XOR* value between the element and $h(ID_x)$;
    else
      Set the element $dic(a_{x,k})$ as $h(ID_x)$;
    $VI_{x,k}$=*Compute_VIA*$^+$$(v_x, vn_{x,y}, A_k, t_{k,i}, t_{j,z})$, $y \in [1, m_x^s]$;
  Sign all elements of the dictionary dic, and attach $VI_x=\{VI_{x,0}, \cdots, VI_{x,w}\}$ with the original vertex v_x to form verifiable vertex $v'_x = v_x||VI_x$;
Return the verifiable graph G_{veri};

Algorithm 4: *Compute_VIA*$^+$.

Input: v_x, $vn_{x,y}$, A_k, $t_{k,i}$, $t_{j,z}$
Output: $VI_{x,k}$

Initialize an empty array Φ;
for *Traverse each neighbor vertex* $vn_{x,y}$ **do**
  Calculate *difference value* dv between the vertex v_x and $vn_{x,y}$ on the profile A_k;
  Find the corresponding position τ based on the value dv;
  if *The number of traversed vertices with profile value* $a_{x,k}$ *is less than* $t_{k,i}$+1 *and the number of traversed neighbors for difference value* dv *is less than* $t_{j,z}$+1 **then**
    if *The element in position* τ *is identical with* dv **then**
      Update the suffix with the *Bit-XOR* value between the previous suffix of the $\Phi(\tau)$ and $h(ID_{vn_{x,y}})$;
    else
      Insert dv as the prefix of $\Phi(\tau)$, and $h(ID_{vn_{x,y}}) \oplus h(a_{x,k}) \oplus r_x$ as the suffix;
  else
    if *The element in position* τ *is not identical with* dv **then**
      Insert the prefix of $\Phi(\tau)$ as dv, and the suffix as an random value;
Sign suffixes of the array Φ with the private key sk of *Twitter*, and calculate the root with the prefixes of the array Φ as inputs of the MHT scheme, and signing its value;
Return signature and the suffixes of the array Φ denoted as $VI_{x,k}$;

generate the auxiliary information for the social graph with our probabilistic scheme.

Compared with the *FakeDetection* scheme, we take two additional parameters as the input, i.e., two thresholds $t_{k,i}$ and $t_{j,z}$, denoting $t_{k,i}$ vertices with the profile value $a_{k,i}$, and $t_{j,z}$ neighbors with the identical profile value. In Algorithm 3, apart from calling for algorithm *Compute_VIA*$^+$, the *Twitter* generates the auxiliary information as the almost identical method in Algorithm 1.

In *FakeDetection*$^+$ scheme, Algorithm 4 takes two other parameters $t_{k,i}$ and $t_{j,z}$ as inputs compared with Algorithm 2. Firstly, the algorithm initializes an empty array Φ to record the difference values between the current vertex and its neighbors. Secondly, the algorithm traverses neighbors following the descending-order permutated *ID* set. For a neighbor vertex $vn_{x,y}$, the *Twitter* calculates the difference value dv for the profile A_k, and finds the corresponding position τ of the array DV following with ascending-order. If the number of traversed vertices with profile value $a_{x,k}$ is less than $t_{k,i}$ + 1, and the number of traversed neighbors for difference value dv is less than $t_{j,z}$ + 1, the algorithm judges whether Φ contains the element dv. If true, the suffix of the current position is updated with the *Bit-XOR* value between the previous suffix of the position τ and $h(ID_{vn_{x,y}})$. Otherwise, the algorithm inserts the new prefix and suffix into the array Φ. If the condition in Line 5 of Algorithm 4 is not satisfied, the algorithm judges whether Φ does not contain the element dv. If true, the algorithm inserts dv and a random value as the prefix and suffix, respectively. In Section 7, we discuss how to determine the parameters $t_{k,i}$ and $t_{j,z}$ for various distributions.

6.3. Query

In *FakeDetection*$^+$ scheme, we also leverage the identical query generation as that in *FakeDetection* scheme. The *DUs* can generate three basic queries, i.e., *equal* query (e.g., q= "New York"), *range* query (e.g., q= "20 < *age* < 30"), and *subset* query (e.g., q={"Temp", "New York"}). Then, the *DUs* send these queries to the *DSP*.

The *DSP* also finds out all social information(i.e., vertices and edges) satisfying the query condition, and generates auxiliary information $\mathcal{VO}_q$ including three kinds of information: hash values to re-construct the root MHT for each vertex, the suffix array for each vertex, signature $dic(q)$.

6.4. Correctness and completeness verification

In this section, we describe the *correctness* and *completeness* verification for the query results. As in Section 5.4, the *DUs* just needs to test whether all vertices of the query results satisfy the query condition q. If this test can be passed, the query results are viewed as satisfying the *correctness*. Otherwise, the *DSP* returns fake query results.

Likewise, the *completeness* verification consists of three perspectives as described in Definition 2, i.e., *vertex completeness*, *profile completeness* and *friendship completeness*.

To verify the *vertex completeness*, the *DSP*, in either *FakeDetection* or *FakeDetection*$^+$ scheme, returns the signature $dic(q)$ about the *ID* information of all vertices satisfying the query

condition. Thus, the *DUs* only need to generate a auxiliary information for the *ID* information of all returned vertices with the identical rule of the *Twitter*, and verify the signature $dic(q)$ with the inputs of the public key of the *Twitter*, and the above auxiliary information. If the signature can be accepted, the vertices are *complete*; Otherwise, the *DSP* does not guarantee the *vertex completeness*.

To prevent the *DSP* from modifying the profile values of twitters, the *Twitter* generates the auxiliary information $VI_{*,0}$ for each vertex. Thus, to verify the *completeness* of *profiles*, the *DUs* only need to generate the auxiliary information with the identical method for all vertices, and compare the results. Obviously, if such comparisons can be passed, the *profile completeness* of the query results can be guaranteed.

Last but not least, the *DUs* verify the *friendship completeness*. As in Section 6.2, the *DUs* traverse all vertices. If the number of traversed vertices is less than $t_{k,i}$ and the number of traversed neighbors is less than $t_{j,z}$, the *DUs* construct the root of the MHT scheme according to the $\mathcal{VO}_{x,q}$, and generate the auxiliary information for the suffixes. Then, the *DUs* compare its signature and suffixes with the returned signature and suffixes. If they are identical, it can verify the *difference values* and *suffix completeness*. If the number of traversed vertices is larger than $t_{k,i}$ and the number of traversed vertices is larger than $t_{j,z}$, the *DUs* only need to construct the root of the MHT scheme, and compare its signature and the returned signature. If they are also identical, it can verify the *difference values completeness*. When the above two tests have both passed, the *DUs* can consider the query results are *friendships completeness* with a possibility p.

Intuitively, when $t_{k,i}$ and $t_{j,z}$ are closer to $\psi_{k,i}$ and $\chi_{j,z}$, the *DUs* detect the fake activities with a higher possibility p, but the overhead is also closer to that in our basic scheme. Thus, *how to choose* $t_{k,i}$ *and* $t_{j,z}$ directly affects the performance of our enhanced scheme, either the computation overhead or detection possibility p. Next, we discuss about the analysis of selecting $t_{k,i}$ and $t_{j,z}$ for various distributions, e.g., *random* distribution and *power-law* distribution.

7. Detection analysis

The auxiliary information in the basic and enhanced schemes chains the authentic nodes, attribute values, and edges with cryptographic methods. As long as the hash operation and signature operation used for constructing the MHTs are secure, the SDP cannot modify the authentic query result.

We assume the DSP knows the number of leaf nodes and the value of the root node in the MHT. After DSP modifies the data, to pass the validation, they must find a preimage of the root node with twice of bit-length of the hash value. And the adversary has to recursively search for preimage of each half until the preimage corresponds to the leaves. It needs $\lceil \log(n) \rceil$ times, where n is the number of leaf nodes in the tree. However, each hash value is an output of the cryptographic hash function, it is unpreimageable Merkle (1980, 1990). Therefore, the probability of the DSP forging a valid proof is negligible.

Next, we check the validity of this algorithm.

In our *FakeDetection*$^+$ scheme, the *Twitter* only takes partial vertices and neighbors to generate auxiliary information ($t_{k,i}$ vertices and $t_{j,z}$ neighbors). In this section, we mainly address the following problem:

- *What is the relationship* between the detection probability p and the parameters $t_{k,i}$ and $t_{j,z}$?
- *How to select* the parameters $t_{k,i}$ and $t_{j,z}$ for various subgraphs?

Based on the notations in Section 6.1, we first evaluate the relationships between the parameters $t_{k,i}$ and $t_{j,z}$ and the probability p for the single profile value query. For a specific query $\widetilde{Q}(q{=}a_{k,i})$, we assume that the *DSP* deletes/adds c_j edges for each vertex with the profile value $a_{k,i}$, and t_e is the number of various edges for which the *DUs* ask proof in a challenge. Besides, let X be a discrete random variable that is defined to be the number of edges chosen by the *DUs* that match the edges deleted/added by the *DSP*. Next we compute p_X^d or p_X^a, the probability that at least one of the edges deleted/added by the *DSP* should be detected. The p_X^d and p_X^a are presented as Eq. (1) and Eq. (2).

$$\begin{aligned} p_X^d &= P\{X \geq 1\} = 1 - p\{X = 0\} \\ &= 1 - \frac{\mu_s - c}{\mu_s} \cdot \frac{\mu_s - 1 - c}{\mu_s - 1} \cdot \cdots \cdot \frac{\mu_s - t_e + 1 - c}{\mu_s - t_e + 1} \end{aligned} \tag{1}$$

$$\begin{aligned} p_X^a &= P\{X \geq 1\} = 1 - p\{X = 0\} \\ &= 1 - \frac{\mu_s}{\mu_s + c} \cdot \frac{\mu_s - 1}{\mu_s - 1 + c} \cdot \cdots \cdot \frac{\mu_s - t_e + 1}{\mu_s + c - t_e + 1} \end{aligned} \tag{2}$$

where $\mu_s = \sum_{j=1}^{\psi_{k,i}} \mu_j$ and $c = \sum_{j=1}^{\psi_{k,i}} c_j$. Since $\frac{\mu_s - j - c}{\mu_s - j} \leq \frac{\mu_s - j - 1 - c}{\mu_s - j - 1}$, it follows that:

$$1 - \left(\frac{\mu_s - c}{\mu_s}\right)^{t_e} \leq p_X^d \leq 1 - \left(\frac{\mu_s - t_e + 1 - c}{\mu_s - t_e + 1}\right)^{t_e}$$

and

$$1 - \left(\frac{\mu_s}{\mu_s + c}\right)^{t_e} \leq p_X^a \leq 1 - \left(\frac{\mu_s - t_e + 1}{\mu_s - t_e + 1 + c}\right)^{t_e}.$$

Here, we utilize the minimal probability to denote the probability, i.e.,

$$p_X^d = 1 - \left(\frac{\mu_s - c}{\mu_s}\right)^{t_e} \tag{3}$$

and

$$p_X^a = 1 - \left(\frac{\mu_s}{\mu_s + c}\right)^{t_e} \tag{4}$$

separately.

To quantify these probabilities, we assume the *DSP* only deletes/adds 1% edges (i.e., $c{=}0.01{\cdot}\mu_s$). We can easily conclude that when t_e is set larger than 459 or 463, the probability p_X^d and p_X^a can achieve at least 99%, respectively. That is, for the fixed c, our scheme only needs to a fixed number of verification edges to achieve the detection probability of 99%. The variant t_e is

the result of $t_{k,i}$ and $t_{j,z}$. For further analysis, we then discuss how the *Twitter* chooses $t_{k,i}$ vertices and $t_{j,z}$ neighbors for specific subgraphs to generate auxiliary information for the fixed t_e. Obviously, when the parameters $t_{k,i}$ and $t_{j,z}$ are closing to $\psi_{k,i}$ and $\chi_{j,z}$, the *FakeDetection*$^+$ scheme is identical with the *FakeDetection* scheme.

Since no previous work point at which distributions the number of neighbors $\psi_{k,i}$ and the number of neighbors for various difference values $\chi_{j,z}$ belong to, we consider two common distributions, i.e., the *random* distribution and the *power-law* distribution. Next, we will discuss based on the following four cases: *random* and *random*[5], *random* and *power-law, power-law* and *random, power-law* and *power-law*

7.1. Random and random

In this section, we discuss the relationship between t_e and the parameters $t_{k,i}$ and $t_{j,z}$ based on the assumption that both parameters $\psi_{k,i}$ and $\chi_{j,z}$ are randomly selected from the range [1,*n*]. Here, we separately set $t_{k,i}$ and $t_{j,z}$ as $\alpha \cdot \psi_{k,i}$ and $\beta \cdot \chi_{j,z}$ ($j \in [1, \psi_{k,i}]$ and $z \in [1, \delta_j]$), where α and β are in the range (0,1]. Thus, t_e is presented as follows:

$$t_e = \sum_{j=1}^{t_{k,i}} \cdot \sum_{z=1}^{\delta_j} t_{j,z} = \sum_{j=1}^{\alpha\cdot\psi_{k,i}} \sum_{z=1}^{\delta_j} \beta \cdot \chi_{j,z} = \sum_{j=1}^{\alpha\cdot\psi_{k,i}} \beta \cdot \mu_j$$

and the mean value of t_e is displayed as follows:

$$\langle t_e \rangle = \frac{1}{4} \cdot \alpha \cdot \beta \cdot (n+1)^2$$

For a fixed value *n*, when the value of α and β are identical, t_e can reach the maximum value. In other words, the detection rate can reach the highest. Thus, mathematically, α and β can be set as:

$$\alpha, \beta = \sqrt{\frac{4 \cdot \langle t_e \rangle}{(n+1)^2}} \tag{5}$$

Obviously, the bigger the value of *n* is, the smaller the value of α and β is.

7.2. Random and power-law

In this section, we discuss the relationship between t_e and the parameters $t_{k,i}$ and $t_{j,z}$ based on the assumption that the parameters $\psi_{k,i}$ is randomly selected from the range [1,*n*], and the distribution of $\chi_{j,z}$ satisfies the power-law distribution. Here, the parameters, either $\psi_{k,i}$ or $\chi_{j,z}$, are discrete integers, but the authors in Clauset et al. (2009) claimed that discrete power-law behavior can be approximated with its continuous counterpart due to the mathematical convenience. Thus, we also utilize the formulas for continuous distributions, such as Eq. (6), instead of discrete distributions.

$$p(x) = \frac{\gamma - 1}{x_{min}} \cdot \left(\frac{x}{x_{min}}\right)^{-\gamma} \tag{6}$$

[5] The formal denote the distribution of $\psi_{k,i}$, while the latter denote that of $\chi_{j,z}$.

To make the computation brevity, we first compute the mean value as follows Newman (2005) $\langle x \rangle = \int_{x_{min}}^{\infty} p(x) \cdot x \cdot dx = |\frac{\gamma-1}{2-\gamma} \cdot x_{min}^{\gamma-1} \cdot x^{2-\gamma}|_{x_{min}}^{\infty}$. Here we utilize *n* approximately to denote ∞, and assume the parameter γ_2 to denote the power-law distribution of the parameter $\chi_{j,z}$, so $\langle x \rangle \approx |\frac{\gamma_2-1}{2-\gamma_2} \cdot x_{min}^{\gamma_2-1} \cdot x^{2-\gamma_2}|_{x_{min}}^{n}$. As it is well known, the *power-law* distribution contains a heavy tail. Intuitively, the values $t_{j,z}$ for the head and heavy-tailed should be different. To separate the head and heavy-tail, we assume the separating point as x_s. Thus, we design a method to select $t_{j,z}$ for the *power-law* distribution. Here, we utilize $\beta_1 \cdot \chi_{j,z}$ for the top part of the *power-law* part, while we randomly extract $\beta_2 \cdot (\delta_j - x_s - 1)$ difference values for the heavy-tailed part. Thus, the t_e is presented as follows:

$$t_e = \sum_{j=1}^{t_{k,i}} \cdot \sum_{z=1}^{\delta_j} t_{j,z} = \sum_{j=1}^{\alpha\cdot\psi_{k,i}} \left(\sum_{z=1}^{\delta_{x_s}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s}-1)} \chi_{j,z} \right)$$

Likewise, the mean value of t_e can be displayed as follows:

$$\langle t_e \rangle = \alpha \cdot \frac{n+1}{2} \cdot \left(\beta_1 \cdot \delta_{x_s} \cdot |\frac{\gamma_2 - 1}{2-\gamma_2} \cdot x_{min}^{\gamma_2-1} \cdot x^{2-\gamma_2}|_{x_{min}}^{n} + \beta_2 \cdot (\delta_j - \delta_{x_s} - 1) \cdot |\frac{\gamma_2 - 1}{2-\gamma_2} \cdot x_{min}^{\gamma_2-1} \cdot x^{2-\gamma_2}|_{x_{min}}^{n} \right) \tag{7}$$

Then, we discuss how to set the separating point x_s. The fraction of the total edges is

$$W(x_s) = \frac{\int_{x_s}^{n} x \cdot p(x)dx}{\int_{x_{min}}^{n} x \cdot p(x)dx} = \frac{n^{2-\gamma_2} - x_s^{2-\gamma_2}}{n^{2-\gamma_2} - x_{min}^{2-\gamma_2}}$$

In this study, we set $W(x_s)$ as 50% at least to compute the separating point x_s for a fixed *n*. Thus, we can get another determining condition, i.e.,

$$\frac{\beta_1 \cdot \delta_{x_s} \cdot |\frac{\gamma_2-1}{2-\gamma_2} \cdot x_{min}^{\gamma_2-1} \cdot x^{2-\gamma_2}|_{x_{min}}^{n}}{\beta_2 \cdot (\delta_j - \delta_{x_s} - 1) \cdot |\frac{\gamma_2-1}{2-\gamma_2} \cdot x_{min}^{\gamma_2-1} \cdot x^{2-\gamma_2}|_{x_{min}}^{n}} = 50\%. \tag{8}$$

From Section 7, we can set the mean value of t_e as 463, and calculate the parameters α, β_1 and β_2 based on Eqs (7) and (8).

7.3. Power-law and random

In this section, we assume that the parameter $\chi_{j,z}$ are randomly selected from the range [1,*n*], while the distribution of $\psi_{k,i}$ satisfies the power-law distribution with the parameters γ_1 and x_{min}. As such, we can obtain the identical conclusion by replacing γ_2 with γ_1 in Section 7.2. However, according to the following experiment, we set the fraction of the total edges for γ_1 as 80%.

7.4. Power-law and power-law

In this section, we assume that both the parameters $\psi_{k,i}$ and $\chi_{j,z}$ satisfy the power-law distribution with their own parameters γ_1 and γ_2. The separating points x_s^1 and x_s^2 for the distributions of $\psi_{k,i}$ and $\chi_{j,z}$. Here, we present t_e as follows:

$$t_e = \sum_{j=1}^{\psi_{x_s^1}} \alpha_1 \cdot \left(\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{j=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \sum_{z=1}^{\delta_j} \chi_{j,z}$$

In this study, we first compute the separating points x_s^1 and x_s^2, and accordingly set the fractions $W(x_s)$ as 80% and 50% for the distributions of $\psi_{k,i}$ and $\chi_{j,z}$, respectively. Furthermore, we can get the values $\alpha_1, \alpha_2, \beta_1$ and β_1 according to Eq (8).

8. Performance analysis

In this section, we analyze the performance for our designed *FakeDetection* and *FakeDetection*$^+$ scheme.

8.1. Generating verification information

In Section 6.1, we list the computation overhead in the *Twitter* side to generate auxiliary information for the social graph, e.g., hash computations, signatures, *HMAC* operations and *Bit-XOR* operations.

Due to space constraints, we only consider the performance of the case of *power-law* and *power-law* in the *FakeDetection*$^+$ scheme. For each profile value, the *Twitter* takes $\sum_{j=1}^{\psi_{k,i}} (2\cdot\delta_j\text{-}1)$ hash computations and one signature. While, for *HMAC* and *Bit-XOR* operations, it takes

$$\alpha_1 \cdot \sum_{i=1}^{\psi_{x_s^1}} \left(\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{i=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \chi_{j,z}$$

and

$$\alpha_1 \cdot \sum_{i=1}^{\psi_{x_s^1}} \left((\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{i=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \chi_{j,z}$$

, respectively. Thus, the computation overhead of *HMAC* and *Bit-XOR* operations can be computed as follows

$$N_{HMAC}^{+} = \sum_{k=1}^{w}\sum_{i=1}^{m_k} \left(\alpha_1 \cdot \sum_{i=1}^{\psi_{x_s^1}} \left(\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{i=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \chi_{j,z} \right) + (w+1)\cdot n$$

and

$$N_{\text{Bit-XOR}}^{+} = \sum_{k=1}^{w}\sum_{i=1}^{m_k} \left(\alpha_1 \cdot \sum_{i=1}^{\psi_{x_s^1}} \left(\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{i=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \chi_{j,z} \right) + (w+1)\cdot n$$

respectively. Obviously, the overhead for hash computations and signatures in the *FakeDetection*$^+$ scheme are equivalent to that in the *FakeDetection* scheme, while the overhead for *HMAC* and *Bit-XOR* operations, compared with the *FakeDetection* are decreased. Here, we further discuss the number of edges utilized to generate auxiliary information. In *FakeDetection* scheme, the number of edges utilized to generate auxiliary information for the profile value $a_{k,i}$ is $\sum_{j=1}^{\psi_{k,i}} \sum_{z=1}^{\delta_j} \chi_{j,z}$. While in the *FakeDetection*$^+$ scheme, the number of edges is

$$\alpha_1 \cdot \sum_{i=1}^{\psi_{x_s^1}} \left(\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{i=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \chi_{j,z}$$

In Table 4, we list the concrete number of edges for various samples.

8.2. Query

For the performance of the query, there is no difference between the *FakeDetection* scheme and *FakeDetection*$^+$ scheme. For a specific query $\widetilde{Q}(A_k = a_{k,i})$, the *DSP* takes $O(n)$ to traverse all vertices in the social graph to find out vertices satisfying the query condition. For each vertex satisfying the query condition, the *DSP* takes $O(\delta_j)$ to search difference values satisfying the query condition, and $O(\log(\delta_j))$ to construct the auxiliary information.

8.3. Verification

In this section, we analyze the performance of verifications. For a query $\widetilde{Q}(A_k = a_{k,i})$, the *DUs* take $O(\psi_{k,i})$ to traverse all returned vertices. For verifying the prefix of the auxiliary information of each vertex, the *DUs* take $\log(\delta_j)$ hash to construct the root of the MHT, and one signature to sign the root. Moreover, the *DUs* take $\chi_{j,z}$ *HMAC* and $\chi_{j,z}$-1 *Bit-XOR* operations to verify the suffixes of the auxiliary information. Thus, the total overhead of verifying one query contains $\sum_{j=1}^{\psi_{k,i}} \log(\delta_j)$ hashes, $\psi_{k,i}$ signatures,

$$\alpha_1 \cdot \sum_{i=1}^{\psi_{x_s^1}} \left(\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{i=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \chi_{j,z} + w + 1$$

HMAC operations and

$$\alpha_1 \cdot \sum_{i=1}^{\psi_{x_s^1}} \left(\sum_{z=1}^{\delta_{x_s^2}} \beta_1 \cdot \chi_{j,z} + \sum_{z=\delta_{x_s^2}+1}^{\beta_2\cdot(\delta_j-\delta_{x_s^2}-1)} \chi_{j,z} \right) + \sum_{i=\psi_{x_s^1}+1}^{\alpha_2\cdot(\psi_{k,i}-\psi_{x_s^1}-1)} \chi_{j,z} + w\text{Bit} - \text{XOR}$$

operations.

9. Experimental results

In this section, we thoroughly evaluate the efficiency of *FakeDetection* and *FakeDetection*$^+$ schemes. Firstly, we introduce some implementation details, followed by the datasets used in

our evaluations. Secondly, we explore the distributions for the parameters $\psi_{k,i}$, μ_j and $\chi_{j,z}$. Finally, we evaluate and compare the performance of *FakeDetection* and *FakeDetection*$^+$ schemes under our datasets.

9.1. Experiment setup

Our experiments are carried out on a commodity PC, with 3.4GHz Intel-i7 3770 CPU, 16GB Memory, a 7200RPM hard disk, and OS Windows 10. Besides, we utilize Python 2.7 to implement the code with 1500+ lines for exploring the distributions and the performance between the *FakeDetection* and *FakeDetection*$^+$ scheme.

9.2. Datasets

We utilize the real Twitter datasets, which have been used in the previous research for inferring home locations (Li et al., 2012). This data set randomly selected 100,000 twitters as seeds to crawl in May 2011. For each user, they crawled his profile, followers, and friends. They obtained nearly 4M twitters' profiles and their social network. In this study, we extracted their registered ID and locations. Specifically, we extracted location with city-level labels with the form of "cityName, stateName" and "cityName, stateAbbreviation", where we considered all cities listed in the file of "List of Valid U.S. cities" downloaded from the White House[6] (More than 37,000 cities). Among them, we found 1,640,146 twitters and more than 51M edges. We used these twitters with their following relationships as our data set. We note that these twitters had at least one labeled friend or follower.

To evaluate the performance of our schemes for various number of twitters, we randomly sample three samples (*Sample*-100K, *Sample*-1M, *Sample*-Original) from the above datasets, i.e., 100K, 1M, 1.6M twitters and their social network. The size of sampled social data for *Sample*-100K, *Sample*-1M and *Sample*-Original are 90MB (2MB twitters' profile, 88MB social network), 904MB (20MB twitters' profile, 884MB social network), 1.48GB (32MB twitters' profile, 1.44GB social network).

9.3. Exploring the distribution of $\psi_{k,i}$

In this section, we first explore the distribution of the number of twitters with the identical *location* profile value. In Fig. 3a, 3b and 3c, we show that the probability density function (PDF) for the number of twitters with the identical *location* profile value in three samples are identical. Moreover, we present the log-log cumulative distribution function (CDF) of the parameter $\psi_{k,i}$ in Fig. 3d. We can conclude that the CDF curve in the *Sample*-100K is close to the left dot line with the slopes from -2 to -1 for the number of twitters smaller than 10^3; while the CDF curves in the *Sample*-1M and *Sample*-Original are all close to the right dot line with the slopes also from -2 to -1 for the number of twitters smaller than 10^4. According to the previous description, a power-law distribution with PDF

$$p(x) = \frac{\gamma_1 - 1}{x_{min}} \cdot \left(\frac{x}{x_{min}}\right)^{-\gamma_1}$$

has a CDF

$$\bar{F}(x) = \left(\frac{x}{x_{min}}\right)^{1-\gamma_1},$$

the number of twitters living in various cities follow a power-law distribution with parameter γ_1 between 2 and 3. Besides, we get the minimal value of $x_{min} = 15$.

Table 2 – Rate of γ located in the range [1,2].

Sample-100K	*Sample*-1M	*Sample*-Original
100%	98.16%	98.25%

9.4. Exploring the distribution of μ_j and $\chi_{j,z}$

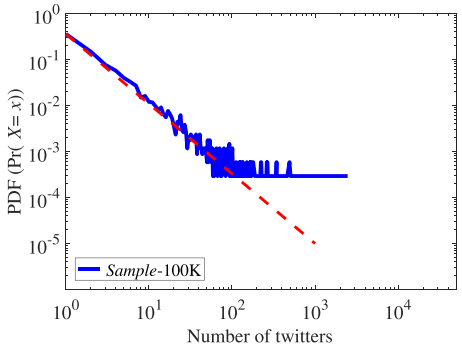
In this section, we first explore the distribution of the number of neighbors μ_j for a specific subgraph, and then we further explore the distribution of $\chi_{j,z}$. However, it is infeasible to explore all possible subgraphs. Thus, for our data set, we briefly assume that the subgraph contains all twitters with the identical *location* profile value, and their social network. For instance, the group of twitters with the *location* profile value "New York, NY" and their social network can be regarded as a subgraph. Based on such an assumption, we finally get 156, 760, 974 subgraphs for various samples.

For each subgraphs, we calculate the parameter γ for the number of neighbors for various subgraphs. As depicted in Fig. 3f, we conclude that the parameter μ_j in most of cities follows the power-law distribution with parameter γ located in the range [1.5,2.5], and with the minima x_{min}=5. Table 2 shows that the parameter γ for more than 98% subgraphs is located in the range [1.5,2.5]. Furthermore, the mean value of γ for all subgraphs is 1.7. Likewise, we show the values of γ_2 of $\chi_{j,k}$ in Fig. 3g. According to the figure, we conclude that more than 99% cases γ_2 locates in the range [1.4, 2.3]. Thus, the distribution of $\chi_{j,k}$ also follows *power-law* distribution with a mean value of γ_2=1.6 and a minimal value x_{min}=2.

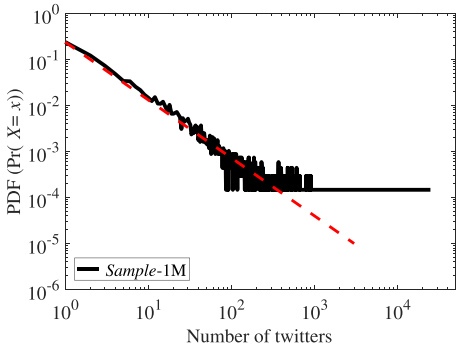
9.5. Generating verification information

In this section, we detail the performance for generating auxiliary information of our *FakeDetection* and *FakeDetection*$^+$ schemes. Next, we separately consider the computation and memory overhead for various samples. According to Sections 9.3 and 9.4, we conclude that the distributions of μ_j and $\chi_{j,z}$ in our data set both follow power-law distribution. Thus, we will first determine the separating points x_s for $\psi_{k,j}$ and μ_j. Here, we set the separating point x_s^{μ} and x_s^{χ} as the $(x_{min}+n)\cdot 0.8$ and $(x_{min}+n)\cdot 0.5$. Fig. 3e and 3g show the fraction of the total edges for μ_j and $\chi_{j,z}$. According to these figures, we can conclude that 90% of the total edges are contained by 20%

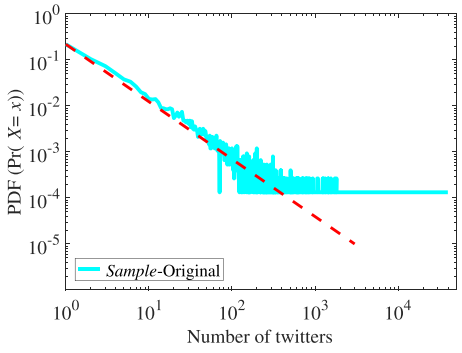
[6] https://www.whitehouse.gov/sites/default/files/omb/assets/procurement_fair/usps_city_state_list.xls.



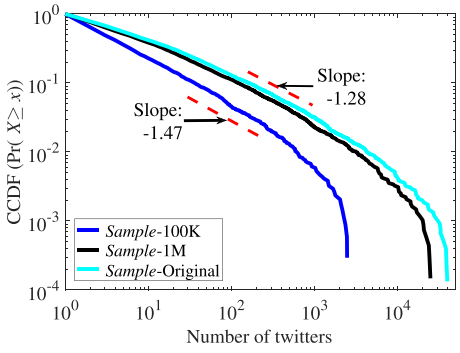
(a) PDF of $\psi_{k,i}$ in *Sample*-100K



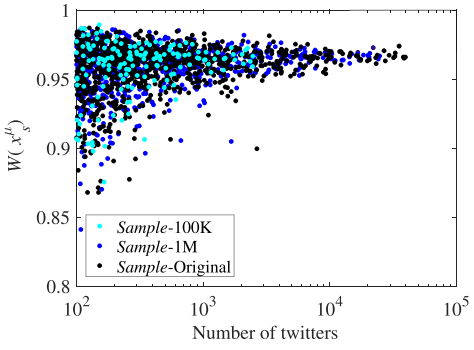
(b) PDF of $\psi_{k,i}$ in *Sample*-1M



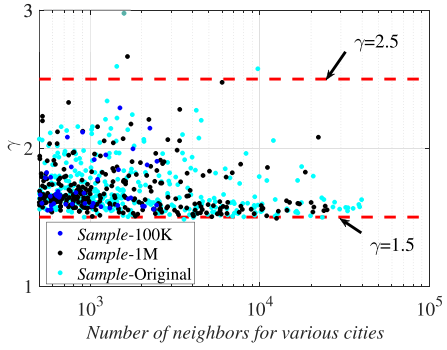
(c) PDF of $\psi_{k,i}$ in *Sample*-Original



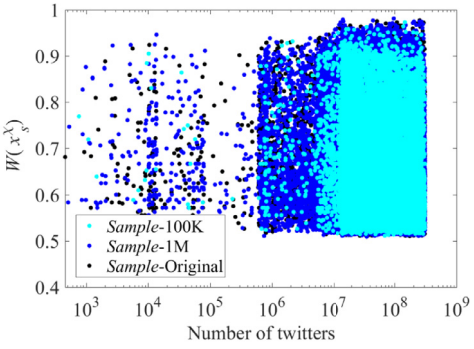
(d) CCDF of $\psi_{k,i}$



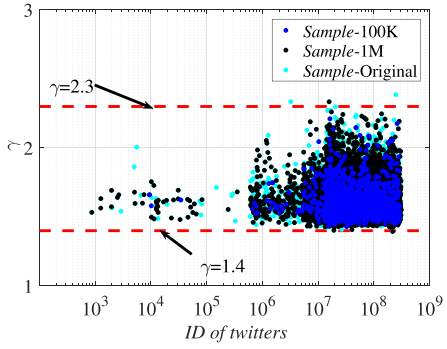
(e) CDF of $\psi_{k,i}$



(f) γ of μ_j



(g) CDF of $\psi_{k,i}$



(h) γ of $\chi_{j,z}$

Fig. 3 – Fig. 3 (a), 3(b) and 3(c) depict the PDF of $\psi_{k,i}$. Besides, the CCDF of the $\psi_{k,i}$ are shown in Fig. 3(d), respectively.

Table 3 – Performance Comparison between *FakeDetection* scheme and *FakeDetection*$^+$ scheme.

	FakeDetection		*FakeDetection*$^+$	
	HMAC	*Bit-XOR*	*HMAC*	*Bit-XOR*
Sample-100K	2.55M	2.48M	0.14M	0.079M
Sample-1M	31.18M	30.32M	1.76M	0.910M
Sample-Original	50.53M	49.13M	2.86M	1.46M

Table 4 – Number of Edges for Generating Verification Information.

	FakeDetection	*FakeDetection*$^+$
Sample-100K	2,413,257	10,755
Sample-1M	29,469,776	52,760
Sample-Original	47,738,327	69,103

Table 5 – Time Overhead.

	FakeDetection	*FakeDetection*$^+$
Sample-100K	37.27s	1.45s
Sample-1M	454.84s	17.37s
Sample-Original	740.35s	27.9s

twitters for μ_j, while only 50% of the total edges are contained by 50% twitters for $\chi_{j,z}$

Computation Overhead. To evaluate the computation overhead detail, we separately compare the following three aspects: number of *HMAC* and *Bit-XOR* operations, number of edges to generate auxiliary information, and computation time. Table 3 lists the number of *HMAC* and *Bit-XOR* operations. From the table, we can conclude that the number of *HMAC* operations in the *FakeDetection* scheme are 17 × at least and 18 × at most more than that in the *FakeDetection*$^+$ scheme for our samples, while the number of *Bit-XOR* operations in the *FakeDetection* scheme are 31 × at least and 33 × at most more than that in the *FakeDetection*$^+$ scheme. Furthermore, we list the number of edges for generating auxiliary information for various samples in Table 4. According to these experimental data, the number of edges in the *FakeDetection*$^+$ scheme only takes 0.44%, 0.17%, and 0.14% of those in the *FakeDetection* scheme for samples *Sample*-100K, *Sample*-1M, *Sample*-Original, respectively. In other words, the computation overhead with *FakeDetection* scheme is 224 × , 558 × and 690 × than those with *FakeDetection*$^+$ scheme. Finally, Table 5 lists the total computation time of generating auxiliary information for various samples. The computation time for generating auxiliary information in the *FakeDetection* scheme takes 26 × more than that in *FakeDetection*$^+$ scheme. Obviously, these times are less than that of number of edges, *HMAC* and *Bit-XOR* operations. This is the reason that the computation of generating MHT between *FakeDetection* and *FakeDetection*$^+$ are no difference.

Table 6 – Memory Overhead.

	FakeDetection	*FakeDetection*$^+$
Sample-100K	43.44MB	43.44MB
Sample-1M	424.44MB	424.44MB
Sample-Original	679.97MB	679.97MB

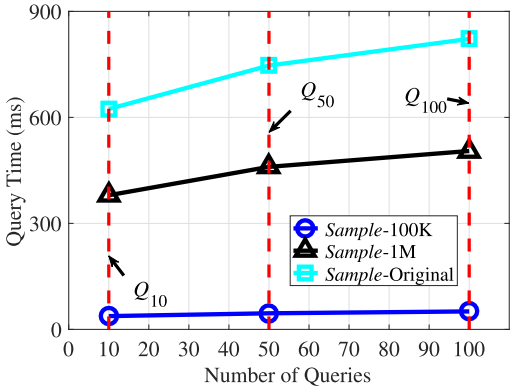
Memory Overhead. Another highlight point to evaluate the performance of our schemes is memory overhead. Here, we separately generate auxiliary information and list the memory overhead of the suffix in Table 6. From the table, we can conclude that the memory overhead of the suffix in *FakeDetection* scheme is identical with that in *FakeDetection*$^+$ scheme for various samples. For instance, the memory size for *Sample*-100K and *Sample*-Original with either *FakeDetection* scheme or *FakeDetection*$^+$ scheme are 43.44MB and 679.97MB. For memory overhead, it can be seen that our schemes are efficient for current applications.

9.6. *Query*

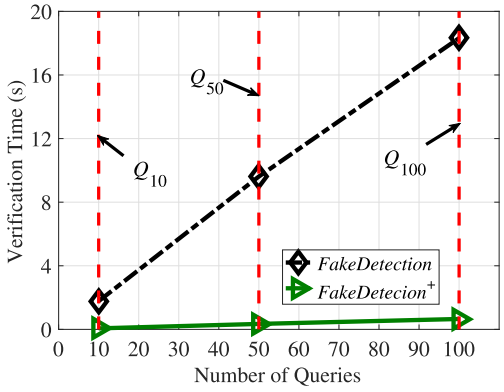
In this section, we evaluate the query efficiency. To evaluate the query efficiency for various query size, we generate various number query conditions ($\widetilde{Q}_{10}$, $\widetilde{Q}_{50}$ and $\widetilde{Q}_{100}$), where the query condition $\widetilde{Q}_{10}$ means that we randomly choose 10 cities as the query condition. Since the number of twitters with the identical profile value follows the *power-law* distribution, we process the query 100 times repeatedly and compute the average time as the query overhead. As shown in Fig. 4a, the average query time for the *Sample*-Original is 8ms per query, which is efficient to real applications.

9.7. *Verification*

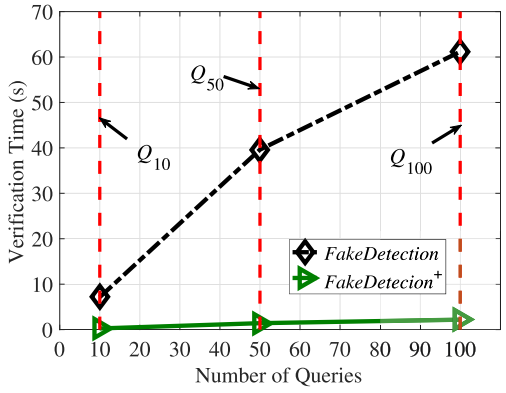
The verification performance on the side of data users is also a highlighting point to evaluate our schemes, so we evaluate the verification performance in this section. As such, we assume that *DUs* obtain the query results for the previous repeatedly query conditions $\widetilde{Q}_{10}$, $\widetilde{Q}_{50}$ and $\widetilde{Q}_{100}$. We calculate the average verification time for the query results, and depict the verification overhead in Fig. 4(b), 4(c) and 4(d). Obviously, the verification overhead with the *FakeDetection*$^+$ scheme is less than that with the *FakeDetection* scheme. Especially, for the data set *Sample*-Original, the average verification time with the *FakeDetection*$^+$ scheme for 100 queries takes only 2.9s and 3.5% of that with the *FakeDetection* scheme. Furthermore, to evaluate the detection probability, we account into the number of edges for generating auxiliary information, and utilize Eqs. (3) and (4) to calculate the detection probability p shown in Figs. 4(e). Obviously, all detection probabilities are over 99%, which satisfies the requirement of our schemes.



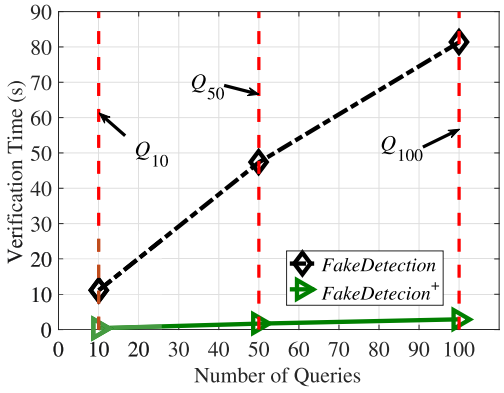
(a) Query Time



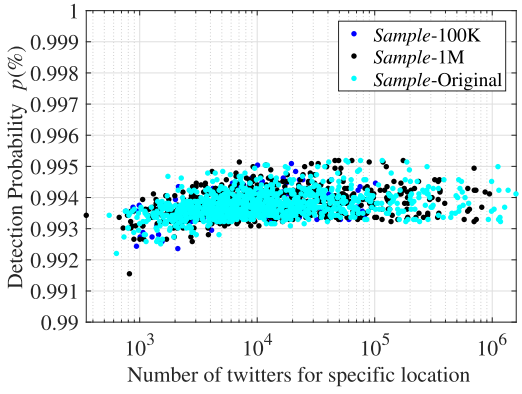
(b) Verification Time for *Sample*-100K



(c) Verification Time for *Sample*-1M



(d) Verification Time for *Sample*-Original



(e) Detection probability for deleting 1% edges

Fig. 4 – Fig. 4 (a) depicts the average query overhead. The verification overhead for various data set are listed in Fig. 4(b), 4(c) and 4(d)

10. Conclusions

In social data transaction model, the *DSP* may resell fake data to the *DUs*. To guarantee the correctness and completeness of social data, we proposed a deterministic scheme *FakeDetection* to detect any malicious activities for vertices, profile values, and friendships. However, it is unrealistic to consider all vertices and friendships due to a large volume of social data. Therefore, we proposed a probability scheme *FakeDetection*$^+$, in which partial vertices and neighbors with identical profile values are taken into account to reduce the computation overhead. Under the real social data with 1.6M twitters and their social network, we conduct our experiments and demonstrate that the enhanced scheme *FakeDetection*$^+$ takes only 27.9s (3.8% computation overhead of *FakeDetection*) to generate auxiliary information. Besides, the *DSP* only takes 8ms to process a query averagely.

But in fact, there are many users sign up every day, add friends and delete friend. The data are changing dramatically, so we can not consider they are static. For future work, we will continue authenticity verification on dynamic social data.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Haowen Chen: Conceptualization, Methodology, Writing - original draft. **Qiang Qu:** Software, Data curation. **Yexiong Lin:** Investigation, Writing - original draft. **Xia Chen:** Visualization. **Keqin Li:** Supervision, Writing - review & editing.

Acknowlgedgments

This work was partially supported by Natural Science Foundation of Hunan Province of China through grant 2018JJ2055.

REFERENCES

Bentley JL. Multidimensional binary search trees used for associative searching. Commun ACM 1975;18(9):509–17.
Bertino E, Carminati B, Ferrari E, Thuraisingham B, Gupta A. Selective and authentic third-party distribution of XML documents. IEEE Trans Knowl Data Eng 2004;16(10):1263–78.
Clauset A, Shalizi CR, Newman E. Power-law distributions in empirical data. SIAM Rev. 2009;51(4):661–703.
Devanbu P, Gertz M, Martel C, Stubblebine S. Authentic data publication over the internet. J. Comput. Secur. 2003;11(3):291–314.
Goodrich T, Tamassia R, Triandopoulos N, Cohen R. In: CT-RSA'03. Authenticated data structures for graph and geometric searching; 2003. San Francisco, CA
Guttman A. R-trees: A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on Management of data; 1984. p. 47–57.
Hacigümüs H, Iyer B, Mehrotra S. In: ICDE'02. Providing database as a service; 2002. San Jose, CA
Ku S, Hu L, Shahabi C, Wang H. Query integrity assurance of location-based services accessing outsourced spatial databases. Advances in Spatial and Temporal Databases 2009:80–97.
Li F, Yi K, Hadjieleftheriou M, Kollios G. In: VLDB'07. Proof-infused streams: enabling authentication of sliding window queries on streams; 2007. Vienna, Austria
Li R, Wang S, Deng H, Wang R, Chang C. In: KDD'12. Towards social user profiling: unified and discriminative influence model for inferring home locations; 2012. Beijing, China
Merkle C. In: CRYPTO'89. A certified digital signature; 1989. Santa Barbara, CA
Merkle RC. Protocols for public key cryptosystems. In: 1980 IEEE Symposium on Security and Privacy. IEEE; 1980. p. 122.
Merkle, R. C., 1990. A certified digital signature. in" advances in cryptology- crypto'89 proceedings.
Mouratidis K, Sacharidis D, Pang H. Partially materialized digest scheme: an efficient verification method for outsourced databases. The VLDB Journal 2009;18(1):363–81.
Mouratidis K, Sacharidis D, Pang H. Partially materialized digest scheme: an efficient verification method for outsourced databases. The VLDB Journal 2009;18(1):363–81.
Mykletun E, Narasimha M, Tsudik G. Providing authentication and integrity in outsourced databases using merkle hash trees. UCI-SCONCE Technical Report 2003.
Mykletun E, Narasimha M, Tsudik G. Authentication and integrity in outsourced databases. ACM Transactions on Storage (TOS) 2006;2(2):107–38.
Narasimha M, Tsudik G. In: DASFAA'06. Authentication of outsourced databases using signature aggregation and chaining; 2006. Singapore
Newman ME. Power laws, pareto distributions and Zipf's law. Contemp Phys 2005;46(5):323–51.
Niaz MS, Saake G. Merkle hash tree based techniques for data integrity of outsourced data.. In: GvD; 2015. p. 66–71.
Pang H, Mouratidis K. Authenticating the query results of text search engines. Proceedings of the VLDB Endowment 2008;1(1):126–37.
Pang H, Tan L. Verifying completeness of relational query answers from online servers. ACM Trans. Inf. Syst. Secur. 2008;11(2) 5:1–5:50.
Pang H, Zhang L, Mouratidis K. Scalable verification for outsourced dynamic databases. Proceedings of the VLDB Endowment 2009;2(1):802–13.
Papadopoulos S, Yang Y, Papadias D. In: VLDB'07. CADS: Continuous authentication on data streams; 2007. Vienna, Austria
Puschmann C, Burgess J. The politics of twitter data. Weller K, Bruns A, Burgess J, Mahrt M and Puschmann C (eds) Twitter and Society 2013:43–54.
Richardson, V., 2016. Google accused of manipulating searches, burying negative stories about hillary clinton, [EB/OL] https://www.washingtontimes.com/news/2016/jun/9/google-accused-burying-negative-hillary-clinton-st/ Accessed June 9.
Richardson, V., 2013. Yelp's newest weapon against fake reviews: Lawsuits. [EB/OL] https://www.bloomberg.com/news/articles/2013-09-09/yelps-newest-weapon-against-fake-reviews-lawsuits Accessed Sep 9.
Rivest L, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 1978;21(2):120–6.
Yang Y, Papadopoulos S, Papadias D, Kollios G. In: ICDE'08. Authentication of outsourced databases using signature aggregation and chaining; 2008. Cancum, Mexico
Yiu L, Lin M, Mouratidis K. In: ICDE'10. Efficient verification of shortest path search via authenticated hints; 2010. Long Beach, CA
Zhang R, Zhang Y, Zhang C. In: INFOCOM'12. Secure top-k query processing via untrusted location-based service providers; 2012. Orlando, FL

Haowen Chen received B.S. degree in Computer Science in 2002 from Hunan University, China, and M.S. in Computer science in 2007 from Hunan University, China. He received his Ph.D. degree from the College of Computer Science and Electronic Engineering, Hunan University, China. He is currently an associate professor at Hunan University, China. His research interests include security and privacy issues in social network.

Qiang Qu received the B.E. degree in Software Engineering from Qinghai Normal University, China, in 2019. Currently, he is pursuing the ME degree in the College of Computer Science and Electronic Engineering at Hunan University, China. His research focus on social network, bioinformatics and information security.

Yexiong Lin is pursuing a B.E. Degree in the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests Information Assurance, Computer Vision, GAN, Bioinformatics, and Graph Neural Network.

Xia Chen received B.S. degree in Computer Science in 2004 from Changsha University of Science and Technology, China, and M.S. in Computer Software in 2010 from Hunan University, China. She is currently a Ph.D. candidate at Hunan University, China. Her research interests include security and privacy issues in cloud computing.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a Distinguished Professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has published over 680 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.