Privacy Preserving Data Quality Assessment for High-Fidelity Data Sharing

Julien Freudiger, Shantanu Rane, Alejandro E. Brito and Ersin Uzun
PARC
Palo Alto, CA

ABSTRACT

In a data-driven economy that struggles to cope with the volume and diversity of information, data quality assessment has become a necessary precursor to data analytics. Real-world data often contains inconsistencies, conflicts and errors. Such dirty data increases processing costs and has a negative impact on analytics. Assessing the quality of a dataset is especially important when a party is considering acquisition of data held by an untrusted entity. In this scenario, it is necessary to consider privacy risks of the stakeholders.

This paper examines challenges in privacy-preserving data quality assessment. A two-party scenario is considered, consisting of a client that wishes to test data quality and a server that holds the dataset. Privacy-preserving protocols are presented for testing important data quality metrics: completeness, consistency, uniqueness, timeliness and validity. For semi-honest parties, the protocols ensure that the client does not discover any information about the data other than the value of the quality metric. The server does not discover the parameters of the client's query, the specific attributes being tested and the computed value of the data quality metric. The proposed protocols employ additively homomorphic encryption in conjunction with condensed data representations such as counting hash tables and histograms, serving as efficient alternatives to solutions based on private set intersection.

1. INTRODUCTION

Data is now the crux of the Internet economy. Businesses across many technological sectors are taking advantage of data commercialization opportunities and monetizing their data with other companies [1]. Popular approaches include data monetization with advertisers (e.g., social networks, credit services, cloud providers), and participation in collaborative data sharing initiatives (e.g., collaborative security, healthcare analytics). Real-world data typically contains inconsistencies, conflicts and errors. Enterprises commonly expect that about 1 to 5% of their data contains errors [2]. Such *dirty data* increases processing costs and has a negative impact on data commercialization [3]. Dirty data costs US businesses about 600 billion dollars annually because cleaning data accounts for 30 to 80% of the development time of most big data projects [4].

WISCS'14, November 3, 2014, Scottsdale, Arizona, USA.
Copyright 2014 ACM 978-1-4503-3151-7/14/11 ...$15.00.
http://dx.doi.org/10.1145/2663876.2663885.

Over time, a large body of work has focused on methods to estimate data quality according to various metrics [5–7] and to automatically clean the data [8–10]. These techniques involve running a series of algorithms that check for integrity constraints on the data, identify dependencies among data attributes and detect inconsistencies in the dataset.

In this paper, we identify a new challenge in data quality assessment that consists in verifying the quality of data owned by an untrusted party. Specifically, one organization might prefer to verify data quality in a privacy-preserving manner prior to acquiring it from another organization. As the data monetization model becomes increasingly prevalent, it is vital to develop the capability to assess data quality prior to data acquisition. Similarly, organizations that participate in collaborative data sharing environments would benefit from assessing the quality of a potential collaborator's data before initiating the data sharing process. In existing data quality assessment solutions, data quality is directly verified by data owners themselves, so these aspects of data collaboration have not received much attention.

The problem of assessing the quality of an untrusted party's data raises privacy risks since the data itself as well as the data quality parameters are sensitive. In particular, there is a need for techniques that protect the privacy of both the querying party and the data owner while they interactively assess the data quality of datasets.

For example, consider the following healthcare scenario. Access to healthcare data is a long and costly procedure that is strictly regulated by government regulations, such as HIPAA [11]. It would be beneficial for healthcare researchers to evaluate the quality of clinical trial data before engaging in the full data access procedure. However, healthcare data is sensitive and cannot be shared in plaintext with researchers. Similarly, data quality aspects — such as whether every element of a certain medical attribute is populated — are sensitive as they may reveal the research agenda.

Another example is that of a market research company that would like to assess whether an attribute related to a particular market demographic contains data that is both valid, and rich enough for useful analytics. It is desirable to have a method that allows such assessments while not only keeping the data private until the sharing officially takes place, but also keeping the demographic sector of interest confidential from the data owner.

A third example is cyber threat mitigation. Previous work has proposed to help organizations share security data with each other in order to improve the effectiveness of their cyber threat mitigation systems [12–19]. However, dirty security data reduces the effectiveness of collaboration because it makes data integration difficult, requires data cleaning, and increases the risk of false positives. Unfortunately, cyber threat data is sensitive [20–22], and so are data quality parameters such as testing of dependencies between attack

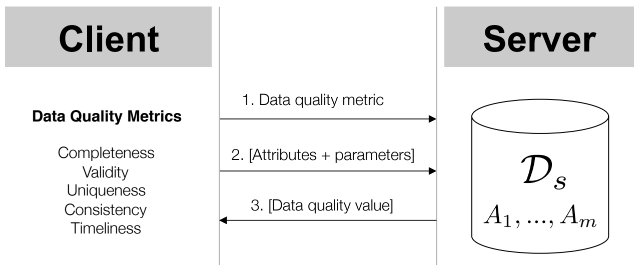


Figure 1: System architecture and basic protocol. After selecting the data quality metric, the client defines data quality constraints and associated parameters, interacts with the server via a secure protocol and obtains the data quality value. The secure protocol protects the privacy of $\mathcal{D}_s$, as well as that of the client's data quality constraint parameters, the queried data attributes, and the resulting data quality value.

vectors and port numbers. Therefore, obtaining data quality assurance in a privacy-preserving way prior to data sharing, would have tangible benefits for collaborative threat mitigation.

Contributions: This paper describes two novel contributions. First, we identify the class of problems in *privacy-preserving data quality assessment*. This is a subclass of problems in privacy-preserving data mining, in which the queries involve specific properties of the dataset. To the best of our knowledge, we are the first to explicitly consider the class of data quality assessment problems under privacy constraints and to propose a systematic analysis. Second, we provide a series of efficient protocols that evaluate data quality while keeping the data, the query parameters and resulting quality value private. Our solutions rely on two-party secure computations for oblivious computation of data quality metrics. Privacy-preserving data quality assessment ensures that poor quality data will not be acquired. Furthermore, since data quality is ascertained beforehand, our protocols have the potential to reduce the onerous overhead involved in cleaning the data and facilitate the emergence of high-fidelity data sharing platforms.

2. PRELIMINARIES

The main goal of this paper is to propose techniques where one party can verify various aspects of the quality of data held by another party in a privacy-preserving manner, i.e., the querying party should learn nothing besides the value of the data quality metric and the data owner should learn nothing besides the data quality metric under consideration.

2.1 System Model

We consider two entities, a client that checks for data quality and a server that owns the dataset under examination. The server holds a dataset $\mathcal{D}_s$ with a known schema including a list of m attributes $A_1, ..., A_m$. A tuple is a row in the dataset and n is the total number of tuples. The system architecture is illustrated in Fig. 1.

2.2 Threat Model

Following Kerckhoff's Principle [23], we assume that both the server and client know which data quality metric is being measured, which protocol is being executed, and the explicit constraints of each protocol. The proposed protocols do not reveal the server's data to the client (*data privacy*), and do not reveal the data quality constraint parameters, the data attributes under consideration, and the resulting values of the quality metric to the server (*query privacy*). All our protocols are two-party interactions. We regard the client and server as semi-honest (honest-but-curious, passive) adversaries that monitor all protocol communications and try to infer as much information as possible about data held by the other party. Being passive players, they follow the rules of the protocol and do not misrepresent their inputs at any stage of the protocol.

First Name	Last Name	Age	State	Zip
John	Steinbeck	32	CA	94043
Jimi	Hendrix	27	WA	01000
Isaac	Asimov	-15	NY	NULL

Table 1: Example of a dirty dataset $\mathcal{D}_s$. The second tuple is inconsistent because zip code does not match state. The third tuple is incomplete and invalid because it misses a zip code and age is negative.

2.3 Data Quality Constraints and Metrics

There are numerous ways of ascertaining the quality of information held in a dataset [5–7]. We consider two fundamental approaches to determining data quality, viz., by testing *integrity* constraints on individual data attributes, and *dependency* constraints across two or more attributes.

Testing integrity constraints involves defining a variety of constraints on dataset attributes with associated parameters and determining whether the data satisfies each constraint. Suppose that d is an element in dataset $\mathcal{D}_s$, θ is a possible singleton parameter, and Θ is a set of parameter values. Examples of integrity constraints include equality constraints (e.g., $d = \theta$), comparison constraints (e.g., $d > \theta$), subset constraints (e.g., $d \in \Theta$), interval constraints (e.g., $d \in [\theta_{min}, \theta_{max}]$), and conjunctions of these constraints (e.g., $d \in \Theta \wedge d > 0$). Note that, unlike integrity checks used for communication security (such as Message Authentication Codes and hashing techniques), integrity constraints test data semantics and do not detect changes to the data.

Testing dependency constraints involves defining a variety of relations between attributes and testing whether the data satisfies those dependencies. For example, "`zip code` $\leftrightarrow$ `state`" is a dependency constraint that verifies whether zip codes are consistent with the associated state.

The selection of constraints to be verified depends on the data quality metrics under consideration. There is a large body of work on determining proper data quality metrics [5, 6]. The selected data quality metric depends upon the type of data and the context or application in which the data is expected to be used. In this paper, we focus on metrics recommended by the US Department of Defense for cyber security data [24]: Completeness, Validity, Uniqueness, Consistency and Timeliness.

Completeness: This metric is defined as the percentage of elements that are properly populated. Specifically, a test for completeness asks whether each tuple has a value for each attribute. For example, in Table 1, the third tuple is missing a zip code and is thus incomplete. It is generally possible to test for completeness using equality and/or subset constraints. An example of a test for data completeness is an integrity constraint that checks for the presence and frequency of occurrence of values such as NULL, "", and other symbols that suggest unpopulated data attributes.

Validity: This metric is defined as the percentage of elements, whose attributes possess meaningful values. The validity of an attribute can be tested using a comparison constraint and/or an interval constraint, which is a composition of multiple comparison constraints. For example, in Table 1, the third tuple has a negative age and is thus clearly invalid. In this case, the integrity constraint could be a reasonable interval constraint on the age, such as [0,110].

Uniqueness: This metric is defined as the number of unique values taken by an attribute, or a combination of attributes in a dataset. This is an important test because it evaluates the variety and richness of the data; data without sufficient variety may not be worth acquiring or worth using for analytics.

Consistency: Consistency is defined as a measure of the degree to which the data attributes satisfy a set of constraints [24]. To explicitly exclude *validity* and *completeness* considerations from discussions on consistency, we narrow down the definition of consistency as follows: Consistency is a measure of the degree to which two or more data attributes satisfy a well-defined dependency constraint. One way to measure consistency for a certain dependency constraint is to report the percentage of tuples that satisfy that dependency constraint. As an example, in Table 1, the first tuple is consistent with respect to the "`zip code` $\leftrightarrow$ `state`" dependency because "94043" is a legitimate California zip code. However, the second tuple is inconsistent because "01000" is not a legitimate Washington zip code.

Timeliness: This metric is defined as the percentage of elements whose time attributes are within a specified time frame. For example, one may be interested in acquiring data only if the tuples have time stamps in the year 2014. Though timeliness is listed as a data quality metric, it is easy to see that it has the same abstract form as a validity metric. Thus, any protocol that can measure validity can also measure timeliness.

2.4 Cryptography Background

This section provides an overview of cryptographic tools used in subsequent sections to build protocols for privacy-preserving evaluation of the data quality metrics.

Additively Homomorphic Cryptosystems: These are public-key cryptosystems in which it is possible to perform mathematical operations on ciphertexts that result in addition operations on the underlying plaintext. There are several additively homomorphic cryptosystems in the literature. Our protocols can employ either the Paillier cryptosystem [25] or a generalization due to Damgård and Jurik [26]. For concreteness, we restrict the discussion below to the original Paillier cryptosystem only. Denoting by $E(\cdot)$ and $D(\cdot)$ respectively the encryption and decryption functions of the Paillier cryptosystem, we have the following two properties for any two integers d_1 and d_2 in the set of allowable messages:

$$D(E(d_1, r_1) \cdot E(d_2, r_2) \mod N^2) = d_1 + d_2 \mod N$$
$$D(E(d_1, r_1)^{d_2} \mod N^2) = d_1 \cdot d_2 \mod N$$

where $N = pq$ is a large modulus, p, q are two large primes and r_1, r_2 are random numbers in $\mathbb{Z}_N^*$, which is a subset containing those elements of $\mathbb{Z}_N = \{0, 1, ..., N-1\}$ that have a multiplicative inverse. As indicated above, the Paillier cryptosystem is probabilistic, i.e., a fresh random parameter r_i is used during every encryption operation. The cryptosystem is thus semantically secure. This property is desirable because it makes the cryptosystem secure against a Chosen Plaintext Attack (CPA). Note that the random parameters r_i are not required for decryption. For simplicity, we omit the random parameters in our development, with the implicit understanding that semantic security is ensured by the choice of a fresh random parameter whenever encryption is performed. Then, the above properties of the Paillier cryptosystem are depicted more concisely as:

$$E(d_1) \cdot E(d_2) = E(d_1 + d_2)$$
$$E(d_1)^{d_2} = E(d_1 \cdot d_2)$$

Private Set Intersection Cardinality (PSI-CA): [27–30] This is a cryptographic primitive involving two parties, a server with set $\mathcal{S}$, and a client with set $\mathcal{C}$ that execute a privacy-preserving protocol. At the end of the PSI-CA protocol, the client learns $|\mathcal{S} \cap \mathcal{C}|$, and the server only learns the client's set size. As PSI-CA only reveals the size of the intersection, but not the actual contents, it can be seen as a more conservative variant of Private Set Intersection (PSI) [27, 31, 32]. In what follows, we will briefly review how, using some preprocessing of the client's query and the server's data, PSI-CA can be adapted to privately evaluate certain data quality metrics.

3. PRIVACY-PRESERVING DATA QUALITY ASSESSMENT

We propose a number of protocols in which a client assesses the quality of a server's dataset according to the data quality metrics discussed earlier. As illustrated in Fig. 1, the client first chooses a quality metric and accordingly defines integrity constraints and dependencies. A secure protocol ensues between the client and the server. As a result of the secure protocol, the client only learns the value of the data quality metric.

3.1 PSI-based Protocols

Private Set Intersection (PSI) protocols enable two parties to compute the intersection set of their data while protecting data privacy for both parties. Data quality metrics discussed in Section 2 are aggregate measures in which the client is interested not in specific data items, but in the number of occurrences of specific data values. PSI-CA reveals to the client the cardinality of the set intersection and therefore appears better suited to data quality assessment than PSI. We illustrate the use of PSI-CA via a protocol in which the client assesses completeness of the server's data.

Setup: The client first determines the set of values to test, for example, $\mathcal{U} = \{\text{NULL}, \text{“”}\}$. The clients then tests the number of occurrences of each element $u \in \mathcal{U}$ in the server's dataset.

Protocol: Typically, the inputs to a PSI-CA protocol are sets, i.e., each element is unique. Therefore, a default execution of PSI-CA with the client possessing $\mathcal{U}$ and the server possessing $\mathcal{D}_s$ will simply yield information about whether a $u \in \mathcal{U}$ is present in $\mathcal{D}_s$ or not. For the protocol to output the number of occurrences of each $u \in \mathcal{U}$ in the server's dataset, the client must preprocess its input. For each value $u \in \mathcal{U}$, the client creates a set:

$$\mathcal{D}_{c,u} = \{(1, u), (2, u), ..., (n, u), (n+1, u), ..., (2n, u), ..., (n(m-1)+1, u), ..., (mn, u)\}$$

where, n is the number of dataset tuples. For each attribute A_i where $i \in 1, ..., m$, the server computes the following set:

$$\bar{\mathcal{D}}_s = \{(1, d_1), (2, d_2), ..., (n, d_n), (n+1, d_{n+1}), ..., (2n, d_{2n}), ..., (n(m-1)+1, d_{n(m-1)+1}), ..., (mn, d_{mn})\}$$

where $d_{(i-1)n+j}$ represents the value of the attribute A_i in the j^{th} tuple in $\mathcal{D}_s$. An execution of PSI-CA for each $u \in \mathcal{U}$ yields the number of occurrences of u in $\bar{\mathcal{D}}_s$. The client can then compute the completeness metric as follows:

$$\mathsf{Completeness} = 1 - \frac{\sum_u |\bar{\mathcal{D}}_s \cap \mathcal{D}_{c,u}|}{nm} \tag{1}$$

Correctness: The preprocessing step applied by the client and server ensures that the protocol correctly evaluates the number of occurrences of each $u \in \mathcal{U}$. A large value for Completeness means that the dataset is relatively complete and few elements in $\mathcal{D}_s$ match an element in the set $\mathcal{U}$. A small value of Completeness indicates a dirty dataset with many missing elements.

Privacy: The construction of the PSI-CA protocols ensures that the client and server only discover the cardinality of the intersection set of $\mathcal{D}_{c,u}$ and $\bar{\mathcal{D}}_s$.

Cost: The construction in [30] has complexity linear in the total number of elements under consideration, i.e., mn in this problem. Thus, the communication overhead is rather high. Even with state of the art computation performance for PSI protocols (about 500ms for sets containing 200 elements), the presented protocol would induce large overhead in the assessment of completeness. Similar or higher complexity would be incurred while testing for validity, uniqueness and consistency. Thus, even though PSI and PSI-CA can be adapted to data quality assessment problems, more practical solutions are necessary.

Recently, there have been several attempts to speed up PSI by relying on server-aided computations [33], garbled Bloom filters [34], or computational optimizations [35]. Inspired by those efforts, we propose efficient privacy-preserving protocols tailored to the data quality assessment problem in the next section.

3.2 Data Quality Assessment Protocols

Our approach is to transform datasets and queries into representations that are efficient for testing data quality metrics. Concretely, we propose to project the server's dataset $\mathcal{D}_s$ into a representative *vector* that is efficient in the sense that its length is typically small in comparison with the number of dataset tuples. Possible vector representations include binary vectors, hash maps, and histograms.

The vector representation makes it is possible to test for data quality constraints by selectively accessing vector values and aggregating them. For instance, it is possible to test for completeness by selecting undesirable attribute values in a histogram and aggregating the number of occurrences. This approach readily extends to validity evaluation which is concerned with counting the number of occurrences on a range of values. Thus, it is essential to have an efficient privacy-preserving primitive for selective aggregation that can be leveraged for evaluating various data quality metrics.

3.2.1 Select and Aggregate Protocol

We present a protocol that enables the client to obliviously obtain the summation of selected elements from the server's vector. The protocol ensures that the client and server do not discover elements of each other's vectors. There are many ways of achieving this; we consider an efficient approach that uses additively homomorphic encryption. This approach is closely related to the method of implementing Oblivious Transfer (OT), and secure inner products using an additively homomorphic cryptosystem.

Inputs: The client has a vector $\mathbf{a} \in \{0,1\}^t$. The server has an integer vector $\mathbf{b} \in \mathbb{Z}^t$. The client possesses the public/private key pair of an additively homomorphic cryptosystem, while the server only has the client's public key.

Output: The client receives $\sum_{j=1}^t a_j b_j$, but discovers nothing about the b_j. The server discovers nothing.

Protocol: $\mathsf{SelectAndAggregate}(\mathbf{a}, \mathbf{b})$

1. The client encrypts each $a_j \in \mathbf{a}$ to obtain $E(a_j)$. It sends the encrypted vector to the server.
2. For each $b_j \in \mathbf{b}$, the server computes:
$$E(a_j b_j) = E(a_j)^{b_j} \tag{2}$$
3. The server then computes:
$$E(\gamma) = E\left(\sum_{j=1}^t a_j b_j\right) = \prod_{j=1}^t E(a_j b_j) \tag{3}$$
Note that the server operates on encrypted data and does not see value of γ. The server transmits this result to the client.
4. Using its private key, the client decrypts γ.

Correctness: The correctness of the protocol is based on the properties of the additively homomorphic cryptosystem. Intuitively, the protocol selects b_j for which $a_j = 1$, adds them up, and reveals the result to the client.

Privacy: The protocol ensures that the server does not see the query vector, as it operates only in the encrypted domain. Note that the semantic security property of the homomorphic cryptosystem ensures that the server cannot make inferences on the binary vector $\mathbf{a}$ based on $E(a_j)$. The client, on the other hand, obtains no information about the server's data vector $\mathbf{b}$, other than the aggregate summation γ.

Cost: The protocol clearly incurs a computational and communication overhead that is linear in the length of the vectors $\mathbf{a}$ and $\mathbf{b}$. Iin order to ensure that our data quality testing protocols are more efficient than the PSI-CA-based implementation described earlier, the inputs $\mathbf{a}, \mathbf{b}$ to the SelectAndAggregate protocol must have lengths considerably smaller than the number of dataset elements mn.

Intuitively, the protocols that we present below can be understood as consisting of two main tasks: (1) Finding a low-dimensional representation of the client's query and server's data that is suitable for testing the quality metric under consideration; (2) Inputing the low-dimensional representation to the SelectAndAggregate protocol.

3.2.2 Completeness Evaluation Protocol

We propose to use hash maps together with the SelectAndAggregate protocol to efficiently assess data completeness. The protocol is illustrated in Fig. 2.

Inputs: The client has a set $\mathcal{U}$ consisting of values to test such as "NULL", the empty string "", and so on. It also possesses the public/private key pair of an additively homomorphic cryptosystem. The server possesses a dataset $\mathcal{D}_s$, whose completeness is to be tested. The server also has the client's public key.

Output: The client obtains the sum of the number of occurrences in the server's dataset of each element $u \in \mathcal{U}$. This allows the client to compute a measure of completeness. The server discovers nothing.

Protocol: The protocol proceeds as follows:

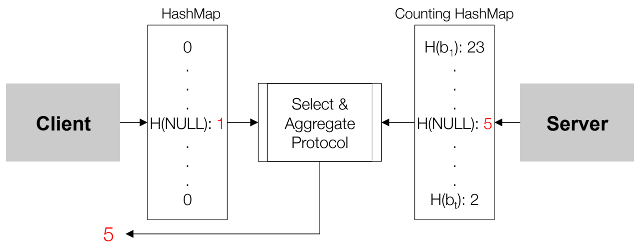


Figure 2: Illustration of the completeness evaluation protocol. The client computes a hash map vector full of 0's with 1's only for undesirable values such as NULL. The server computes a counting hash map of its dataset $\mathcal{D}_s$.

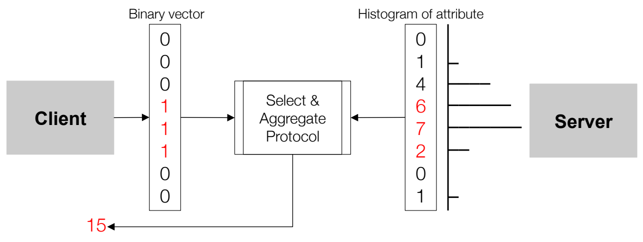


Figure 3: Illustration of the validity evaluation protocol. The client computes a binary vector full of 0's with 1's only corresponding to the range of interest. The server computes a histogram of each attributes in its dataset $\mathcal{D}_s$.

1. The server computes the number of unique elements $t = \mathsf{Unique}(\mathcal{D}_s)$ and shares t with the client.

2. The client maps each element $u \in \mathcal{U}$ onto a hash map given by $\mathbf{a} = \mathsf{Hashmap}(\mathcal{U}, t)$.

3. For each attribute A_i where $i \in 1, ..., m$, the server computes a counting hash map $\mathbf{b}_i = \mathsf{CountingHashmap}(\mathcal{D}_s[, A_i], t)$. Note that the hash map $\mathbf{a}$ and the counting hash map $\mathbf{b}$ use the same hashing functions and have equal length that depends on t.

4. For each attribute A_i where $i \in 1, ..., m$, the client and server execute the SelectAndAggregate protocol with inputs $\mathbf{a}$ and $\mathbf{b}_i$ respectively. At the end of the protocol, the client receives $\gamma_i = \mathsf{SelectAndAggregate}(\mathbf{a}, \mathbf{b}_i)$.

5. The client computes the completeness metric can be computed as:

$$\mathsf{Completeness} = 1 - \frac{\sum_{i \in \mathcal{M}} \gamma_i}{n|\mathcal{M}|}$$

   where $\mathcal{M}$ is the set of attributes whose completeness the client wants to test. For example, if the client wants to test the completeness of the whole dataset, then $\mathcal{M} = \{1, 2, ..., m\}$.

Correctness: The client's hash map $\mathbf{a}$ is a binary vector with ones corresponding to the elements of $\mathcal{U}$. Based on the SelectAndAggregate protocol, the dot product of $\mathbf{a}$ with $\mathbf{b}$ only preserves the membership count of the elements of $\mathcal{U}$. For each attribute A_i, the client is finally provided with the sum γ_i.

Privacy: Privacy is inherited from the SelectAndAggregate protocol. Note that, if the client chooses, he can test the completeness of only a few attributes, without revealing which attributes are being tested. Because the server returns γ_i for each $i = 1, 2, ..., m$, the client has the flexibility to choose the set of interesting attributes $\mathcal{M}$ and consider only those γ_i for which $i \in \mathcal{M}$. Thus, the protocol provides the client with query privacy in that the server discovers neither the attribute set $\mathcal{M}$, nor the values being tested (e.g., NULL entries), nor the value of the completeness metric. The client also obtains no information about the server's data, other than the number of unique elements t, and the number of occurrences of the elements of $\mathcal{U}$.

Cost: The protocol has cost linear in the size of the hash maps, i.e., the computational and communication overhead is $O(t)$. Recall that t is the number of unique elements in $\mathcal{D}_s$, which is the sum of the number of unique values taken by each attribute A_i of $\mathcal{D}_s$. This number tends to be much smaller than the number of dataset tuples n for large datasets. For e.g., If we have a dataset of students in a large country with 2 attributes, viz., integer scores from 1 to 100 and the type of school attended (public/private), then $t = 200$, whereas the number of students can be in the hundreds of thousands. The protocol thus consumes significantly less overhead than the PSI-CA-based protocol above.

3.2.3 Validity Evaluation Protocol

Validity testing relies on checking range constraints, which can be tested using histograms and indicator vectors constructed on the desired ranges [36]. We propose a validity evaluation protocol in which the client builds such an indicator vector over the range of interest and the server builds a histogram on its data. The protocol is illustrated in Fig. 3.

Inputs: The client has a set of attributes $\mathcal{M}$ whose validity it is interested in testing. For each attribute $A_i \in \mathcal{M}$, the client posses a range of valid values to be tested. These ranges can be contiguous or disjoint sets, which we denote by $\mathcal{R}_i$. It also possesses the public/private key pair of an additively homomorphic cryptosystem. The server possesses a dataset $\mathcal{D}_s$, whose validity is to be tested. The server also has the client's public key.

Output: For each attribute $A_i \in \mathcal{M}$, the client obtains the number of occurrences of valid entries, i.e., entries that belong to the validity sets $\mathcal{R}_i$. This allows the client to compute a measure of validity. The server discovers nothing.

Protocol: For each attribute $A_i, i = 1, 2, ..., m$:

1. The client and server agree on reasonable range $\mathcal{Q}_i$ of values corresponding to the maximum and minimum values that the attribute can take, a bin size ℓ_i and the number of bins t_i. For example, if the i^{th} attribute is age, then a reasonable range of values is $[0, 110]$, the bin size can be 1, which means that the ages are quantized to the nearest integer, and the number of bins can be 111. Thus, $\mathcal{Q}_i = \{0, 110\}, \ell_i = 1, t_i = 111$. Note that these ranges are based on public-domain knowledge about the attribute under consideration, and thus do not leak information about the server's data.

2. The server computes a histogram of the attribute A_i with the given bin size ℓ_i, and number of bins t_i and the ranges specified in $\mathcal{Q}_i$. Specifically, the server computes $\mathbf{b}_i = \mathsf{Histogram}(\mathcal{D}_s[, A_i], t_i, \ell_i, \mathcal{Q}_i)$.

3. The client generates a binary vector $\mathbf{a}_i$ of size t_i, where each entry of $\mathbf{a}_i$ corresponds to a bin of the histogram of A_i. If it is not interested in testing the validity of attribute A_i, i.e., if

$A_i \notin \mathcal{M}$, then the client initializes all elements of $\mathbf{a}_i$ to 0. If it is interested in testing the validity of attribute A_i, it locates the bins that intersect with $\mathcal{R}_i$, and sets the corresponding entry in $\mathbf{a}_i$ to 1 and leaves the rest to 0. For example, if the client is interested in the age range [0, 35] in the above example, it sets the corresponding 36 entries of $\mathbf{a}_i$ to 1, and all others to 0.

4. The client and server execute the SelectAndAggregate protocol with inputs $\mathbf{a}_i$ and $\mathbf{b}_i$ respectively. At the end of the protocol, the client receives $\gamma_i = \mathsf{SelectAndAggregate}(\mathbf{a}_i, \mathbf{b}_i)$.

Finally, the client computes the validity metric as:

$$\mathsf{Validity} = \frac{\sum_{i \in \mathcal{M}} \gamma_i}{n|\mathcal{M}|} \quad (4)$$

Correctness: The client's binary vector $\mathbf{a}$ contains 1's only at places corresponding to valid values of the attribute being tested, while the server's histogram vector $\mathbf{b}$ contains the number of occurrences of all observed values for that attribute. The dot product of $\mathbf{a}$ with $\mathbf{b}$ thus returns the number of valid items.

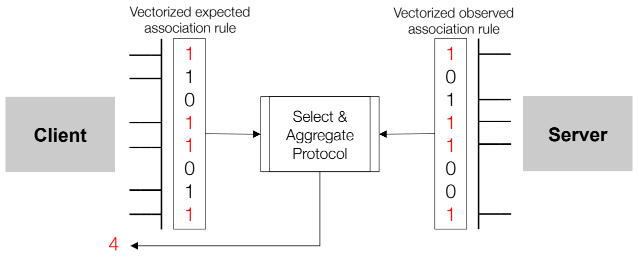
Privacy: Again, privacy is inherited from the SelectAndAggregate protocol. If the client chooses, he can test the validity of only a few attributes, without revealing which attributes are being tested. This is because the server returns γ_i for each $i = 1, 2, ..., m$, while the client has the flexibility to choose the set of interesting attributes $\mathcal{M}$, forcing $\gamma_i = 0$ when $A_i \notin \mathcal{M}$. The protocol provides the client with query privacy because the server discovers neither the interesting attribute set $\mathcal{M}$, nor the validity ranges $\mathcal{R}_i$, nor the value of the validity metric. The client also obtains no information about the server's data, other than the number of occurrences of valid quantities in the attributes of interest.

Cost: The protocol has cost linear in the number of histogram bins, i.e., the computational and communication overhead is $O(\sum_{i=1}^{m} t_i)$. As in the case of completeness, the number of bins in the histogram of A_i tends to be much smaller than the number of dataset tuples n. Again, the protocol consumes significantly less overhead than the PSI-CA protocol.

Observation: Since timeliness is a validity constraint, the validity evaluation protocol can also be used to evaluate timeliness by choosing appropriate constraint parameters for attributes that are associated with time. The correctness, privacy and cost guarantees remain the same as above.

3.2.4 Consistency Evaluation Protocol

Recall that testing for consistency involves determining the number of dataset tuples that satisfy a dependency constraint specified by the client. The dependency constraint may involve two or more attributes. We develop a privacy-preserving protocol for evaluating the consistency with respect to a given dependency constraint. The intuition is that the client can define the dependency constraint in terms of an association rule involving two or more attributes, and the server computes the dependency that is observed among those attributes in its dataset. Then the overlap between the association rule specified by the client, and the observed dependency in the server's dataset is a measure of consistency. This intuition is depicted in Fig. 4, wherein the multidimensional association rule at the client's end and the multidimensional attribute dependency observed at the server's end are vectorized so that the SelectAndAggregate protocol can be reused.



Vectorized expected association rule
Vectorized observed association rule
Client
Server
Select & Aggregate Protocol
1
1
0
1
1
0
1
1
1
0
1
1
1
0
0
1
4

Figure 4: Illustration of the consistency evaluation protocol. This protocol can be thought of as a multidimensional extension of the validity evaluation protocol with the caveat that the server's data structure is a binary vector rather than a histogram.

Inputs: The client is interested in a set of attributes $\mathcal{M} \subseteq \{A_1, ..., A_m\}$. It also possesses the public/private key pair of an additively homomorphic cryptosystem. The server possesses a dataset $\mathcal{D}_s$, whose consistency is to be tested with respect to some attributes chosen privately by the client. The server also has the client's public key.

Outputs: The client obtains the number of entries for which the attributes in the set $\mathcal{M}$ are consistent. This allows it to measure the consistency of those attributes. The server discovers nothing.

Protocol: The protocol proceeds as follows:

1. For each attribute A_i, the client and server agree on reasonable range $\mathcal{Q}_i$ of values corresponding to the maximum and minimum values that the attribute can take, a bin size ℓ_i and the number of bins t_i As in the validity evaluation protocol, these ranges are based on public-domain knowledge about the attribute under consideration, and thus do not leak information about the server's data.

2. The client generates an association rule involving the attributes $A_i \in \mathcal{M}$, and expresses the association rule as a binary valued mapping $f(d_1, ..., d_m)$ where d_k is a value taken by attribute A_k and f takes a value 1 if the tuple $(d_1, ..., d_m)$ is a legitimate collection according to the association rule. Otherwise, it takes a value 0. As an example, considering a three-attribute dataset with zip codes, state names, and political party affiliations, where the client is only interested in checking the consistency of zip codes and states and does not care about political party affiliations. Then, $|\mathcal{M}| = 2$, $f(94043, \mathsf{CA}, \cdot) = 1$, but $f(01000, \mathsf{WA}, \cdot) = 0$.

3. The server reads the values of the attribute tuples $(d_1, ..., d_m)$ and computes a binary valued mapping $g(d_1, ..., d_m)$ where the mapping g takes a value 1 if the tuple $(d_1, ..., d_m)$ exists in the dataset. Otherwise, it takes a value 0. As there are n tuples, the map g has n ones and $\prod_i^m t_i - n$ zeros.

4. The client and server vectorize their mappings f and g according to an indexing scheme that is agreed upon. As a result, the client has the vector $\mathbf{a}$, while the server has the vector $\mathbf{b}$, where $\mathbf{a}, \mathbf{b}$ each have length $\prod_i^m t_i$.

5. The client and server execute the SelectAndAggregate protocol with inputs $\mathbf{a}$ and $\mathbf{b}$ respectively. At the end of the protocol the client receives $\gamma = \mathsf{SelectAndAggregate}(\mathbf{a}, \mathbf{b})$.

6. The client computes the consistency metric as:

$$\mathsf{Consistency}(\mathcal{M}) = \frac{\gamma}{n} \quad (5)$$

Correctness: To see that the protocol is correct, observe that the client generates a binary map representing an association rule, while the server generates a binary map representing associations observed in its dataset. Then, the protocol privately computes the element-wise logical AND function on the corresponding elements of the binary maps and adds up the result using an additively homomorphic cryptosystem.

Privacy: Once again, privacy in the computation of γ is inherited from the $\mathsf{SelectAndAggregate}$ protocol. The protocol provides the client with query privacy in that the server discovers neither the attribute set of interest to the client $\mathcal{M}$, nor the client's association rule, nor the value of the consistency metric. The client also obtains no information about the server's data, other than the number of tuples that are consistent with respect to the attributes of interest.

Cost. The ciphertext computation and communication cost of the scheme is $O(\prod_i^m t_i)$. In comparison, a protocol that adapts PSI-CA for consistency testing using a method similar to the one described in Section 3.1 would incur an overhead of $O(n^m)$.

3.2.5 Uniqueness Evaluation Protocol

Testing for uniqueness involves counting the number of unique values taken by an attribute in a range specified by the client. Rather than counting values at arbitrary precision, it may be advisable for the client and server to agree on a quantization of the attribute space, i.e., a constant or variable bin size. Then uniqueness is redefined as the number of populated bins in the histogram of the data attribute held by the server.

A privacy-preserving protocol for determining uniqueness is readily obtained by a small modification to the second step in the validity evaluation protocol. The modification involves replacing the non-zero entries in the vector $\mathbf{b} = \mathsf{Histogram}(\mathcal{D}_s(A_i), t_i, \ell_i, \mathcal{Q})$ by 1. It can be verified, that owing to this modification, the client obtains the number of bins in the histogram of the server's data, that contain at least one attribute value. Correctness, privacy and cost can be determined using exactly the same procedure as that used for the validity evaluation protocol.

4. RELATED WORK

In this paper, we have proposed specialized protocols for privacy preserving computation of specific data quality metrics. Below, we review several related proposals.

4.1 Privacy-Preserving Cloud Auditing

The literature on data integrity checking contains work on public auditability of remotely stored data [37–40]. Public auditability allows an external party, in addition to the data owner himself, to verify the correctness of data remotely stored in the cloud. Wang *et al.* [41] proposed a privacy-preserving auditing protocol. Their scheme supports an external auditor to audit outsourced data in the cloud without gaining knowledge about the data.

These approaches were developed in the context of secure outsourced data storage. Data quality assessment is a related problem but the focus is on verifying semantic notions of data quality rather than on the integrity or retrievability of third party data.

4.2 Privacy-Preserving Data Mining

The data mining community has contributed a number of a number of seminal papers proposing secure protocols to compute clusters, and perform regression and classification across vertically partitioned databases while protecting the privacy of participants [42–44]. For example, it is possible to conduct statistical analysis on joint datasets, while preserving the confidentiality of each participant [45]. Existing work focused on data mining computations which are not directly applicable to the data quality assessment problem. Data quality is concerned with simpler tests, such as equality, comparisons and conjunctions. Our work could be used in conjunction with data mining approaches, in order to assess the quality of data held by an untrusted party, prior to running privacy-preserving data mining algorithms. This helps increase confidence in the quality of results.

Among the specialized protocols that could help with privacy-preserving data quality assessment, Private Set Intersection (PSI) has generated a lot of interest. PSI involves two parties, a server with a set $\mathcal{S}$, and a client with a set $\mathcal{C}$. At the end of an interactive protocol, the latter only learns $\mathcal{S} \cap \mathcal{C}$, whereas the former learns nothing beyond client's set size. State-of-the-art implementations of PSI include garbled circuit-based techniques [34, 46] as well as specialized protocols [27, 31, 32, 47, 48], and are characterized by different computational assumptions and complexity guarantees. However, as discussed, PSI is a symmetric data sharing problem and does not solve the data quality assessment problem.

With Private Set Intersection Cardinality (PSI-CA) [27–30], we show that it is possible to assess data quality privately. Yet, the overhead is large, and thus not satisfactory in real-world scenarios. Our specialized protocols are inspired from recent developments to scale PSI to large datasets. A number of researchers [34, 49–51] have proposed the use of Bloom filters to reduce the size of input sets and increase the scalability of PSI. Other related efforts to speed up PSI include [33, 35].

Finally, the literature on search over encrypted data and privacy-preserving querying of databases also makes use of techniques for obliviously computing equality, comparison and range queries [36, 52, 53]. Those efforts, however, seek to provide search results, in contrast with data quality metrics, many of which involve data aggregation. This difference is significant as it allows us to rely on dimensionality reducing representations, such as histograms, that dramatically reduce overhead.

4.3 Private Sharing of Security Data

Large-scale data sharing for cyber threat mitigation raises numerous challenges, including efficiency, data protection, and sanitization [17, 19]. Consequently, the research community has attempted to balance data utility with privacy protection via anonymization and data encryption techniques. Lincoln *et al.* [21] suggested sharing sanitized security data for collaborative analysis of security threats. Specifically, they removed sensitive data — such as IP addresses — prior to sharing. Other mechanisms were proposed to anonymize traces, ranging from prefix-preserving anonymization of IP addresses [54, 55] to statistical obfuscation [56]. However, previous work showed that inference attacks can de-anonymize network traces [57], and that it is difficult to maintain data utility [58–61]. Alternatively, Applebaum *et al.* [20] presented cryptographic protocols for private data aggregation, and leveraged the computed aggregates for anomaly detection. Burkhart *et al.* [22] proposed a distributed solution based on secure multi-party computation and secret sharing that supports aggregation of security alerts and traffic measurements among peers, e.g., to estimate global traffic volume. Finally, Freudiger *et al.* [19] estimate benefits of data sharing prior to collaboration by computing in a privacy-preserving way the similarity between different data sets.

Our work complements secure and private data sharing solutions. Specifically, our protocols help organizations assess data quality before deciding whether or not to engage in secure data sharing.

5. CONCLUSION AND FUTURE WORK

Our motivation is to facilitate high fidelity data collaboration environments by mitigating the cost associated with exchange or acquisition of dirty data. We identify challenges in privacy-preserving data quality assessment. The problem consists in protecting data and query privacy while enabling assessment of data quality held by untrusted parties. We present an overview of principal data quality metrics and show that existing secure protocols are not effective for data quality computation. To overcome this problem, we propose secure two-party protocols for computation of various data quality metrics: Completeness, Validity, Uniqueness, Consistency and Timeliness. We design the protocols so that they operate on reduced dimensionality descriptions. As a result, they can scale to large datasets. Our results show that it is possible to efficiently verify data quality in a privacy-preserving manner.

Our next step is to evaluate our protocols on real-world public datasets. From a protocol design standpoint, we are interested in investigating the impact of attribute value quantization for the more computationally intensive tasks such as consistency testing. In some cases, our protocols can be applied to strings by considering their ASCII representations. An interesting extension consists in supporting private data quality testing with general non-numeric attributes. In the consistency evaluation protocol, we presented a general method for investigating dependencies among attributes, however, it may be possible to design more efficient protocols for certain kinds of functional dependencies [9]. Another related area of interest is the investigation of *mutual* data quality in data integration scenarios (such as Extract, Transform, Load (ETL)), where data quality also depends on the compatibility of the untrusted server's data with the client's own databases [3]. Finally, a relevant but extremely difficult problem is to privately evaluate the reliability, i.e., truthfulness of data held by an untrusted party.

References

[1] J. Plansky, J. Solomon, R. Karp, and C. Drisko. The Data Gold Rush, Strategy Report 2013. http://www.strategyand.pwc.com/media/file/Strategyand_The-Data-Gold-Rush.pdf, 2013.

[2] Thomas C Redman. The impact of poor data quality on the typical enterprise. *Communications of the ACM*, 41(2), 1998.

[3] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[4] Wayne W Eckerson. Data quality and the bottom line. *TDWI Report, The Data Warehouse Institute*, 2002.

[5] Yang W Lee, Diane M Strong, Beverly K Kahn, and Richard Y Wang. AIMQ: a methodology for information quality assessment. *Information & management*, 40(2), 2002.

[6] Diane M Strong, Yang W Lee, and Richard Y Wang. Data quality in context. *Communications of the ACM*, 40(5), 1997.

[7] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

[8] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. The llunatic data-cleaning framework. *VLDB Endowment*, 6(9), 2013.

[9] Wenfei Fan. Dependencies revisited for improving data quality. In *PODS*, 2008.

[10] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.

[11] An Act. Health insurance portability and accountability act of 1996. *Public Law*, 104:191, 1996.

[12] Fabio Soldo, Anh Le, and Athina Markopoulou. Predictive blacklisting as an implicit recommendation system. In *INFOCOM*, 2010.

[13] Sachin Katti, Balachander Krishnamurthy, and Dina Katabi. Collaborating against common enemies. In *IMC*, 2005.

[14] Ernesto Damiani, S De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. P2P-based collaborative spam detection and filtering. In *P2P*, 2004.

[15] George Oikonomou, Jelena Mirkovic, Peter Reiher, and Max Robinson. A framework for a collaborative DDoS defense. In *ACSAC*, 2006.

[16] Brent Tzion Hailpern, Peter Kenneth Malkin, Robert Jeffrey Schloss, Steve R White, Philip Shi-Lung Yu, and Charles Campbell Palmer. Collaborative server processing of content and meta-information with application to virus checking in a server network, 2001. US Patent 6,275,937.

[17] Phillip Porras and Vitaly Shmatikov. Large-scale collection and sanitization of network security data: risks and challenges. In *Workshop on New security paradigms*, 2006.

[18] Jian Zhang, Phillip A Porras, and Johannes Ullrich. Highly predictive blacklisting. In *USENIX Security*, 2008.

[19] Julien Freudiger, Emiliano De Cristofaro, and Alex Brito. Privacy-friendly collaboration for cyber threat mitigation. *arXiv preprint arXiv:1403.2123*, 2014.

[20] B. Applebaum, H. Ringberg, M.J. Freedman, M. Caesar, and J. Rexford. Collaborative, privacy-preserving data aggregation at scale. In *PETS*, 2010.

[21] Patrick Lincoln, Phillip Porras, and Vitally Shmatikov. Privacy-preserving sharing and correction of security alerts. In *USENIX Security*, 2004.

[22] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *Usenix Security*, 2010.

[23] Auguste Kerckhoffs. *La Cryptographie Militaire*. University Microfilms, 1978.

[24] Phillip Cykana, Alta Paul, and Miranda Stern. DoD Guidelines on Data Quality Management. In *IQ*, pages 154–171, 1996.

[25] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[26] I. Damgård and M. Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Workshop on Practice and Theory in Public Key Cryptosystems*, pages 119–136, 2001.

[27] Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[28] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *SIGMOD*, 2003.

[29] S. Hohenberger and S. Weis. Honest-verifier private disjointness testing without random oracles. In *PETS*, 2006.

[30] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and Private Computation of Cardinality of Set Intersection and Union. In *CANS*, 2012.

[31] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *CRYPTO*, 2005.

[32] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, 2010.

[33] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Scaling private set intersection to billion-element sets. In *FC*, 2014.

[34] Changyu Dong, Liqun Chen, and Zikai Wen. When Private Set Intersection Meets Big Data: An Efficient and Scalable Protocol. In *CCS*, 2013.

[35] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on ot extension. In *USENIX Security*, 2014.

[36] Dan Boneh and Brent Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. In *TCC*, 2007.

[37] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *TISSEC*, 14(1):12, 2011.

[38] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *ESORICS*. 2009.

[39] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *ASIACRYPT*. 2008.

[40] Ari Juels and Burton S Kaliski Jr. PORs: Proofs of retrievability for large files. In *CCS*, 2007.

[41] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM*, 2010.

[42] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. In *CRYPTO*, 2000.

[43] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. *ACM Sigmod Record*, 29(2), 2000.

[44] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *S&P*, 2013.

[45] Wenliang Du and Mikhail J Atallah. Privacy-preserving cooperative statistical analysis. In *ACSAC*, 2001.

[46] Y. Huang, D. Evans, and J. Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *NDSS*, 2012.

[47] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *FC*, 2010.

[48] E. De Cristofaro and G. Tsudik. Experimenting with fast private set intersection. In *TRUST*, 2012.

[49] Marcin Nagy, Emiliano De Cristofaro, Alexandra Dmitrienko, N Asokan, and Ahmad-Reza Sadeghi. Do I know you?: efficient and privacy-preserving common friend-finder protocols and applications. In *ACSAC*, 2013.

[50] Dilip Many, Martin Burkhart, and Xenofontas Dimitropoulos. Fast private set operations with SEPIA. Technical report, 2012.

[51] Florian Kerschbaum. Public-key encrypted Bloom filters with applications to supply chain integrity. In *Data and Applications Security and Privacy*. 2011.

[52] Steven Michael Bellovin and William R Cheswick. Privacy-enhanced searches using encrypted Bloom filters. 2007.

[53] Femi Olumofin and Ian Goldberg. Privacy-preserving Queries over Relational Databases. In *PETS*, 2010.

[54] Adam Slagell and William Yurcik. Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization. In *Security and Privacy for Emerging Areas in Communication Networks*, 2005.

[55] Jun Xu, Jinliang Fan, Mostafa H Ammar, and Sue B Moon. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *ICNP*, 2002.

[56] Eytan Adar. User 4xxxxx9: Anonymizing query logs. In *Query Log Analysis Workshop*, 2007.

[57] Scott E Coull, Charles V Wright, Fabian Monrose, Michael P Collins, Michael K Reiter, et al. Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Network Traces. In *NDSS*, 2007.

[58] Erin Kenneally and Kimberly Claffy. Dialing privacy and utility: a proposed data-sharing framework to advance Internet research. *IEEE S&P*, 8(4), 2010.

[59] Kiran Lakkaraju and Adam Slagell. Evaluating the utility of anonymized network traces for intrusion detection. In *SECURECOMM*, 2008.

[60] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. BotGrep: Finding Bots with Structured Graph Analysis. In *Usenix Security*, 2010.

[61] Titan Threat Intelligence System. http://www.gtresearchnews.gatech.edu/titan-threat-intelligence-system/, 2013.