2011

Detecting Inconsistencies in Private Data with Secure Function Evaluation

Nilothpal Talukder
Purdue University, ntalukde@cs.purdue.edu

Mourad Ouzzani
Purdue University, mourad@cs.purdue.edu

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Mohamed Yakout
Purdue University, myakout@cs.purdue.edu

Report Number:

11-006

Detecting Inconsistencies in Private Data with Secure Function Evaluation

Nilothpal Talukder, Mourad Ouzzani, Ahmed K. Elmagarmid, and Mohamed Yakout
Purdue University, West Lafayette, IN, USA
{ntalukde,mourad,ake,myakout}@cs.purdue.edu

Abstract— **Erroneous and inconsistent data, often referred to as 'dirty data', is a major worry for businesses. Prevalent techniques to improve data quality consist of discovering data quality rules, identifying records that violate those rules, and then modifying the data to either remove those violations. Most of the work described in the literature deals with cases where both the data and the rules are visible to the party that is in charge of cleaning the data. However, consider the case where two parties with data and data quality rules wish to cooperate in data cleaning under two restrictions: (1) neither of the parties is willing to share their data due to its sensitive nature, and (2) the data quality rules may reveal information about the content of the data and may be considered as a private asset to the business. The question then is how to clean the data without having to share the data or the rules. While the data cleaning process involves several phases, our focus in this paper is on detecting inconsistent data. We propose a novel inconsistency detection protocol that preserves the privacy of both the data and the data quality rules without the use of a third party. Inconsistent data is defined as all records in a database that violate some conditional functional dependencies or CFDs. Our approach is based primarily on the secure multi-party computation framework. We present complexity analysis of our protocol and a series of experiments about its performance.**

I. Introduction

It is no secret that 'dirty data' has continued to trouble businesses costing staggering \$600 billion per annum [1]. In most cases, data quality problems arise from failure to enforce integrity and domain constraints. Errors in the data propagate from initial data acquisition to archival [2] opening the avenue for more errors in the future ('Garbage In! Garbage out!'). 'Dirty data' can have serious effects for healthcare industry where wrong treatments due to erroneous medical history cost lives. On the other hand, data cleaning is a labor-intensive process costing 30-80% of development time and effort in a typical data-warehouse project [3].

Recently researchers have been looking into automatic discovery of data quality rules from the data [4], [5], [6], [7], [8], which are mainly extensions to traditional functional dependency (FD). After discovering these rules, they are enforced on the data to identify and correct the records that violate the rules i.e., dirty or inconsistent records, to improve the overall quality. In this paper, we consider conditional functional dependency (CFD) based inconsistency detection [9], [5]. The main difference between traditional FDs and CFDs is that FDs are used for schema design and integrity constraints definition, whereas CFDs enforce binding of semantically consistent values in a database. This property can be leveraged to detect inconsistencies in data [5]. Given a database instance D and a set of CFD rules Σ, the inconsistency detection problem is to find the set of records $D' \subseteq D$ which violate Σ. While there have been several techniques, either CFD-based or otherwise, to detect dirty data, there have been little work on data quality when the data is private.

Let us consider a scenario where two parties with private databases wish to cooperate in improving their data quality. They want to check the quality of the data in one database with the rules discovered in the other database as there may not be enough evidence (e.g., support, conviction) [9] present in the data of one party to discover all data quality rules. To better understand this situation let us take a look at an example.

Example 1.1: We consider a relation CUSTOMER(Name, CC, Zip, Street, State), where CC stands for country code. The tables I and II show CUSTOMER relations from two different databases D_1 and D_2.

As shown in Table I, the following CFD rule can be discovered in D_1: $\phi_{1,1} : ([CC = 44|zip] \rightarrow street)$. This is an FD that is applied to a subset of records $t_i, 1 \leq i \leq |D'|, D' \subseteq D$ that are semantically equivalent in the attribute 'CC' (having the same value '44'). This CFD is interpreted as: in the United Kingdom (CC = 44), the zip code determines the street address. Records t_1-t_5 are consistent with the above rule and no inconsistencies are detected in D_1. However, it is not possible to discover the same rule from D_2, since fewer (two entries for consistent value 'Mayfield') or none (just one entry for 'Princess') of the records are consistent with each other. The reverse is true for the CFD rule $\phi_{2,1} : ([CC = 01, Zip = 46825] \rightarrow [City = FortWayne, State = IN])$ detected in D_2 with records $t_8 - t_9$. The same rule cannot be detected in D_1 due to inconsistent records $t_6 - t_7$. Based on the above example, there can be two different settings for the problem:

Setting 1: Relations can be horizontally or vertically (or both) partitioned and distributed across different sites. For example, if databases D_1 and D_2 represent horizontal partitions of a database D, the problem becomes inconsistency detection in distributed database [10]. Detecting inconsistencies in D_2 requires data to be shipped to one partition (such as, D_1) where a local inconsistency detection is performed. The choice of the partition where the data would be shipped is based on optimal communication cost [10].

Setting 2: In this setting, one party will assist another party with its locally detected rules to find inconsistencies. In example 1, the owner of D_2 (D_1 resp.) requires rules

TABLE I

CUSTOMER RELATION AND CFD RULES IN D_1

id	Name	CC	Zip	Street	City	State
t_1	Smith	44	EH4 8LE	Mayfield	EDI	n/a
t_2	Anne	44	EH4 8LE	Mayfield	EDI	n/a
t_3	Chris	44	EH4 8LE	Mayfield	EDI	n/a
t_4	Ravi	44	EH2 4HF	Princess	EDI	n/a
t_5	Robin	44	EH2 4HF	Princess	EDI	n/a
t_6	Ismail	01	46825	Bell Avenue	Fort Wayne	IN
t_7	Gyle	01	46825	Bell Avenue	Ft Wayn	MI

CC	Zip	Street
44	-	-

$\phi_{1,1}$: (CC, Zip $\rightarrow$ Street, $\{44, -||-\}$)

TABLE II

CUSTOMER RELATION AND CFD RULES IN D_2

id	Name	CC	Zip	Street	City	State
t_1	Shawn	44	EH4 8LE	Mayfield	EDI	n/a
t_2	Dave	44	EH4 8LE	Mayfild	EDI	n/a
t_3	Brian	44	EH4 8LE	Mayfield	EDI	n/a
t_4	Bret	44	EH4 8LE	Mayfeld	EDI	n/a
t_5	Alice	44	EH2 4HF	Princess	EDI	n/a
t_6	Ray	44	EH2 4HF	Prncs	EDI	n/a
t_7	Rachel	44	EH2 4HF	Royal	EDI	n/a
t_8	Ron	01	46825	Bell Avenue	Fort Wayne	IN
t_9	Zach	01	46825	Bell Avenue	Fort Wayne	IN
t_{10}	Jim	01	47906	Northwestern	West Lafayette	IN
t_{11}	Joe	01	47906	State	West Lafayette	IN

CC	Zip	City	State
01	46825	Fort Wayne	IN
01	47906	West Lafayette	IN

$\phi_{2,1}$: (CC, Zip $\rightarrow$ City, State, $\{01, 46825||FortWayne, IN\}$)
$\phi_{2,2}$: (CC, Zip $\rightarrow$ City, State, $\{01, 47906||WestLafayette, IN\}$)

from the owner of D_1 (D_2 resp.) to detect inconsistencies in its own data. Another motivation for this setting would be commercial data quality assessment service such as fixing and standardizing customer addresses. For example, such service providers may want to use their prior experience (assessment on D_1's data) and apply their already discovered data quality rules to detect inconsistencies in D_2.

In both settings, the database may contain sensitive information, like healthcare records and credit card information, which the data owner cannot share with others. Furthermore, the data quality rules may involve constant values which may reveal partial information about the data. In addition, the rules constitute an asset to the business and hence must also remain private. In summary, for legal and business reasons the parties must operate on private data and rules.

Problem Statement: The privacy-preserving inconsistency detection problem can be stated as finding inconsistent records in a private database with the assistance of data quality rules discovered in another private database where the rules need also to remain private. Without loss of generality, we can reduce this problem to the case where one party owns the rules, namely 'rules owner', and the other party owns the data 'data owner'. This problem statement refers to the general case of privacy-preserving inconsistency detection with any data quality rules [4], [5], [6], [7], [8]. However, in this paper, we focus on CFD-based inconsistency detection approaches and enhance them with privacy preservation constraints.

The simple SQL-based detection technique used with CFDs [6] groups the records that match on the left hand side (LHS) attributes in a CFD rule and determines inconsistencies based on the mismatch on the right hand side (RHS) attribute values. Our protocol for inconsistency detection in private data uses the secure multi-party computation (SMC) framework proposed by Lindell et. al. [11] and uses Yao's [12] secure function evaluation (SFE). Our protocol operates in multiple steps and each step involves specific techniques to ensure privacy guarantees of the data and the rules. A high level description of our solution is shown in Fig. 1. Using SFE, our approach securely performs the oblivious grouping of records based on the LHS attribute value match (secure blocking and oblivious grouping steps). Likewise, we identify inconsistent records (inconsistency detection steps) based on the oblivious grouping information and provide them only to the data owner. Furthermore, we use XOR and additive share of values to generate oblivious intermediate results. To provide privacy guarantee when the number of rules is small, we obfuscate the subset of records relevant to the rules and ensure that it is sufficiently large.

Introducing privacy constraints in the data quality problem has received some attention in the literature [13] [14]. However, most of these approaches either use a third party or will allow the rules owner to know how many records are linked to a particular data value in a rule and are subject to k-anonymity based re-identification attacks [15]. Furthermore, the data owner may be able to infer the values in a rule which constitute a private asset for the business. Anonymizing [16] the records before performing private record linkage will reduce the utility of the data and result in poor inconsistency detection.

Our main contributions in this paper are:

- We propose the first formulation of the problem of privacy-preserving inconsistency detection involving two parties where both the data and the data quality rules need to remain private.
- We propose a cryptographic approach for CFD-based inconsistency detection in private databases without the use of a third party.
- We give computational complexity analysis, and a series of experiments of our protocol.

II. Problem Definition

In this section, we present the formal definition of the private inconsistency detection problem with CFDs.

A. Private Inconsistency Detection with CFD

Given a set of private data records D (owned by the data owner) and a set of private CFD rules Σ (owned by the rules owner), the private inconsistency detection query will return the set of inconsistent records $D' \subseteq D$ only to the data owner such that D' violates the rules Σ, also expressed as : $D' \not\models \Sigma$.

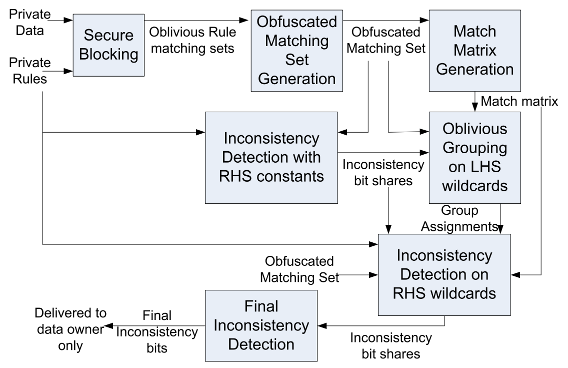

Private Data
Private Rules
Secure Blocking
Oblivious Rule matching sets
Obfuscated Matching Set Generation
Obfuscated Matching Set
Match Matrix Generation
Match matrix
Inconsistency Detection with RHS constants
Oblivious Grouping on LHS wildcards
Inconsistency bit shares
Group Assignments
Obfuscated Matching Set
Inconsistency Detection on RHS wildcards
Delivered to data owner only
Final Inconsistency bits
Final Inconsistency Detection
Inconsistency bit shares

Fig. 1. Privacy Preserving Inconsistency Detection Protocol

The data owner and the rules owner are assumed to have the same schema R.

The final result of the private inconsistency detection, only visible to the data owner, is a bit array with a size equal to the number of records in the database. Each bit indicates whether the record is inconsistent or not: $C_i = 1$, if t_i is inconsistent and 0, otherwise.

B. Framework

Our solution to the private inconsistency detection problem uses cryptographic secure multi-party protocols (SMC) [12]. In our setting, the notion of privacy means that 1) neither the data nor the rules can be visible to the rules and data owner respectively and 2) at any stage of the protocol, the rules owner would not be able to infer the content of the data and the data owner would not be able to generate the rules. We henceforth refer to the data owner as $\mathcal{A}$ and to the rules owner as $\mathcal{B}$.

A trivial solution to this problem, known as the ideal model, is to use a trusted third party (TTP), where $\mathcal{A}$ gives her private data and $\mathcal{B}$ gives his private rules to TTP. TTP determines the inconsistent records with the help of the SQL-based detection techniques and send the result to $\mathcal{A}$. In contrast, our protocol does not use any TTP. It uses secure two-party computation which functions as if a trusted external party exists. We assume a semi-honest model (also known as 'honest but curious' model). Therefore, $\mathcal{A}$ ($\mathcal{B}$) honestly follows the protocol steps, but tries to infer rules (data) from the available information from the other party. Our proposed privacy preserving inconsistency detection protocol has the following properties:

- When comparing a data value in a record and a constant in a rule, the identities of the attributes involved in the rules are not leaked to $\mathcal{A}$. Likewise, $\mathcal{B}$ learns nothing about the actual content of the data records as it is additively shared between $\mathcal{A}$ and $\mathcal{B}$ before the comparison. This is achieved by the adoption of oblivious attribute selection protocol [17]
- The intermediate results of the protocol (rule matching set membership, inconsistency bits and group numbers) are made oblivious by either XOR sharing or additive sharing between $\mathcal{A}$ and $\mathcal{B}$ which restricts both parties from inferring anything about the content of data and rules.

III. Preliminaries

A. Conditional Functional Dependencies

For a relational schema R, a CFD ϕ is defined as $(R : X \to Y, T_\phi)$ where (1) X, Y are sets of attributes of R and $X \cap Y = \emptyset$, (2) T_ϕ is a pattern tableau for a CFD ϕ with attributes $A_i \in (X \cup Y)$, where for each tuple $r \in T_\phi$, $r[A_i]$ is either a constant or an unspecified value '-' (denoted as wildcard); the constant is assumed to be drawn from the discrete domain of attribute A_i, or simply $dom(A_i)$, and (3) $X \to Y$ refers to a standard FD with a set of attributes $A \subseteq (X \cup Y)$ such that $|A| \geq 1$ and $\forall A_i \in A, r \in T_\phi$, $r[A_i]$ is semantically bound by a constant value.

The pattern tableau is used for uniform representation of both the data and constraints involved in CFD rules. For example in Table I (II) the pattern tableau for CFD rules $\phi_{1,1}$ ($\phi_{2,1}$ and $\phi_{2,2}$) in D_1 (D_2) is shown to the right of the CUSTOMER relation. The pattern tableau T_ϕ contains one or more records r_k representing different rules for CFDs. A data record $t_i \in D$ and a rule $r_k \in T_\phi$ are considered a match (denoted as $t_i \asymp r_k$) when the following is satisfied: if $\forall A \in X \cup Y$, $r_k[A]$ is a constant then $t_i[A] = r_k[A]$.

B. Inconsistency Detection with CFD

A relation D of schema R satisfies the CFD ϕ (denoted by $D \models \phi$) when the following holds: for each record $r_k \in T_\phi$ and $t_1, t_2 \in D$ if $t_1[X] = t_2[X] \asymp r_k[X]$, then $t_1[Y] = t_2[Y] \asymp r_k[Y]$. The notation $t_1[X] = t_2[X] \asymp r_k[X]$ denotes that for attribute $X_l \in X$, if $r_k[X_l]$ is a constant then $t_1[X_l]$, $t_2[X_l]$ and $r_k[X_l]$ are equal, otherwise (when $r_k[X_l]$ is a wildcard) just $t_1[X_l]$ and $t_2[X_l]$ are equal. Finally, if Σ is a set of CFDs, we can say that $D \models \Sigma$ if $D \models \phi$ for each CFD $\phi \in \Sigma$. On the other hand, if some records do not satisfy the CFD ϕ (or violate CFD ϕ), those records are said to be inconsistent w. r. t. ϕ.

Data cleaning is a two step process. The first step is to find inconsistent records. The second step is to provide value modification suggestion(s) to remove these inconsistencies [5].

A set of CFDs Σ can be accommodated in the same pattern tableaux having the same set of attributes. This is known as merged pattern tableaux [5] (denoted as T_Σ). The merged pattern tableaux can be split into two parts, one for LHS (T_Σ^X) and the other for RHS (T_Σ^Y). The attribute that does not apply to a rule is considered a don't care value '@' for that specific rule. Table III shows the merged pattern tableaux for CFDs ϕ_1 and ϕ_2 from Example 1.1 with newly assigned rule ids.

Definition 1: **Rule matching set**: The rule matching set $S_k \subseteq D$ of a CFD rule $r_k \in T_\phi$ having the form $X \to Y$, is the set of data records $t_i \in D$ that match the rule r_k on the left hand side attributes X; in other words, $\forall X_l \in X$ the following holds $t_i[X_l] \asymp r_k[X_l]$. Also note that if $S_k \models r_k$, then $D \models r_k$.

TABLE III

MERGED PATTERN TABLEAUX FOR CFDS ϕ_1 AND ϕ_2

	CC	Zip
r_1	44	-
r_2	01	46825
r_3	01	46906

Tableau T_{Σ}^{X}

Street	City	State
-	@	@
@	Fort Wayne	IN
@	West Lafayette	IN

Tableau T_{Σ}^{Y}

Definition 2: **Inconsistent records set**: The inconsistent records set Υ_k is the set of data records in S_k that violate the CFD rule $r_k \in T_\phi$. Formally, $\Upsilon_k \subseteq S_k$ and $\forall t_i \in \Upsilon_k, t_i \not\models r_k$.

Example 3.1: In table I, the CFD rule $\phi_{1,1}$ matches records $t_1 - t_5$ in D_1 (a match on CC='44'). Therefore, we can write $S_{1,1}(D_1) = \{t_1, t_2, t_3, t_4, t_5\}$ If we apply the CFD rule $\phi_{1,1}$ on D_2, we get $S_{1,1}(D_2) = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ Furthermore, $\forall t_i, t_j \in S_{1,1}(D_1), r_k \in T_\phi$ the following holds $t_i[CC, Zip, Street] = t_j[CC, Zip, Street] \asymp r_k[CC, Zip, Street]$. That means, all the records $t_i \in S_{1,1}(D_1)$ satisfy $\phi_{1,1}$. Therefore, we can write $\Upsilon_{1,1}(D_1) = \emptyset$ and $D_1 \models \phi_{1,1}$. Meanwhile, for D_2, $S_{1,1}(D_2) \not\models \phi_{1,1}$ and $\Upsilon_{1,1}(D_2) = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$

C. Cryptographic Primitives

In order to hide the intermediate results of the protocol from both $\mathcal{A}$ and $\mathcal{B}$, we use additive sharing and xor sharing of values. In additive sharing, the value x is additively split between $\mathcal{A}$ and $\mathcal{B}$ such that they have x_A and x_B respectively, and $x = x_A + x_B$. Likewise, xor sharing is done as: $x = x_A \oplus x_B$.

In the 1 out of N (OT_1^N) protocol, originally proposed by Rabin [18], one party (sender) holds an array of values $x = x[0], \ldots, x[N]$ and the other party (receiver) holds an index j. The receiver learns the value $x[j]$ and the sender learns nothing.

The oblivious attribute selection protocol [17] is used to generate additive shares of attribute value $t_i[X]$ of data record t_i for $\mathcal{A}$ and $\mathcal{B}$. The attribute X remains oblivious to $\mathcal{A}$. This is used for secure comparison of values between data records and rules in secure blocking and inconsistency detection with constants steps.

We use Yao's secure function evaluation (SFE) [12] to securely evaluate all the expressions in our protocol and generate oblivious intermediate results. $\mathcal{B}$ generates 'garbled circuit' with the expression and send the circuit to $\mathcal{A}$ to evaluate it with the encrypted inputs of both $\mathcal{A}$ and $\mathcal{B}$. More details can be found in appendix.

IV. PRIVACY-PRESERVING INCONSISTENCY DETECTION PROTOCOL

Out protocol for privacy-preserving inconsistency detection consists of six major steps as shown in Fig. 1:

- Secure Blocking
- Obfuscated Matching Set Generation
- Match Matrix Generation
- Inconsistency Detection with RHS Constants
- Oblivious Grouping with LHS wildcards
- Inconsistency Detection with RHS wildcards
- Final Inconsistency Detection

The secure blocking step identifies the records that match the constant LHS part of a CFD rule, and thus generates the rule matching sets. The matching set is the union of all rule matching sets. The obfuscated matching set generation step securely adds more record-ids to the matching set in order to prevent k-anonymity based re-identification attack. The oblivious grouping step considers the wildcard LHS attributes of a CFD rule and assign group numbers to records based on the match. Records belong to the same group w.r.t. a rule, if they match on those wildcard attributes. Inconsistency detection with RHS constants marks a record inconsistent if there is a mismatch on RHS constant attribute value of a CFD rule. Likewise, inconsistency detection with RHS wildcard step identifies inconsistent records having the same group number when they have a mismatch on RHS wildcard attributes. The intermediate results in secure blocking and inconsistency detection steps are XOR shared between $\mathcal{A}$ and $\mathcal{B}$. Similarly, the results of oblivious grouping are additively shared between them. The notations used for shared results in our algorithms are described in the Appendix (Table IV).

We make the following assumptions: (i) $\mathcal{A}$ and $\mathcal{B}$ have the same schema R, (ii) $\mathcal{B}$ has union compatible CFD rules [5] and a merged pattern tableaux T_Σ (split into tableaux T_Σ^X andT_Σ^Y), and (iii) $\mathcal{A}$ and $\mathcal{B}$ agree on a set of attributes to operate on during the whole procedure. This set includes all the attributes X and Y in the merged pattern tableau T_Σ.

For notational convenience, we denote each entry in the pattern tableau as $r_\phi \in T_\Sigma$. For additional privacy protection of the rules, $\mathcal{B}$ might choose to include extra attributes besides all the attributes in the merged pattern tableau. For an entry in the pattern tableau r_ϕ of the form $X \to Y$, where $X, Y \subseteq attr(R)$. We use notations X' and Y' for LHS and RHS attributes with constant values and X'' and Y'' for LHS and RHS attributes with wildcards. Finally, $X - (X' \cup X'')$ and $Y - (Y' \cup Y'')$ are the attributes with '@'.

A. Secure Blocking

The objective of the secure blocking step is to identify the records that exactly match the constant part of the LHS of a CFD rule. These records are members of the rule matching set, S_ϕ. The set membership for the i-th record, t_i is denoted by $t_i[r^\phi] \in \{0, 1\}$ and this information is obliviously shared between $\mathcal{A}$ ($t_i[r_A^\phi]$) and $\mathcal{B}$ ($t_i[r_B^\phi]$) such that

$$t_i[r_A^\phi] \oplus t_i[r_B^\phi] = \begin{cases} 1, & \text{if record } t_i \in S_\phi \\ 0, & \text{otherwise} \end{cases}$$

We perform matching between all the data records and the constant part of the LHS of a rule and hence generate one partition per rule referred to as rule matching set. The partitions or sets can also be overlapping across different rules. XOR sharing ensures that neither $\mathcal{A}$ nor $\mathcal{B}$ learns the result of the matching. For additional protection, the oblivious attribute selection protocol is used so that the attributes used in the protocol are hidden from $\mathcal{A}$.

Algorithm 1 shows the secure blocking protocol. For each entry $r_\phi : X \rightarrow Y$ in the pattern tableau T_Σ, the protocol is initiated by $\mathcal{B}$. Initially, we consider that all records fall into the rule, i.e., $\forall i, t_i \in S_\phi$ and the XOR shares of set membership, $t_i[r_A^\phi]$ and $t_i[r_B^\phi]$ are set to 1 and 0 respectively. The oblivious attribute selection protocol is used (described in line (a) and (b) of Algorithm 1) to generate random additive shares u and v for $t_i[x_k]$. $\mathcal{A}$ and $\mathcal{B}$ then participate in SFE protocol to determine if $(u + v) = r_\phi[x_k]$ for constant attributes $x_k \in X$. The order of attribute x_k in each iteration is randomly chosen by $\mathcal{B}$ and is hidden from $\mathcal{A}$ through this protocol. Lines (c)-(e) describe the 'garbled circuit' generation by $\mathcal{B}$ and evaluation by $\mathcal{A}$. The Boolean expression used in the protocol has two parts. The first part $((t_i[r_A^\phi](old) \oplus t_i[r_B^\phi](old))$ checks if t_i is still a valid member of S_ϕ in the current iteration (the membership becomes invalid after the first mismatch is found). The second part $(u + v) = r_\phi[x_k])$ evaluates the match/mismatch between the rule and the record on attribute x_k. In order for t_i to be a member of S_{phi}, both conditions must be true. $\mathcal{B}$ chooses a random share $t_i[r_B^\phi](new)$ and therefore generates the 'garbled circuit' for the complete expression: $((t_i[r_A^\phi](old) \oplus t_i[r_B^\phi](old)) \wedge ((u+v) = r_\phi[x_k])) \oplus t_i[r_B^\phi](new)$ which would actually generate the XOR share for $\mathcal{A}$, i.e., $t_i[r_A^\phi](new)$. The new XOR shares become old shares (shown as new and old in the expression) in the next iteration with the next chosen LHS constant attribute of r_ϕ. Fig. 2 shows an example of XOR shares of the rule matching set membership w. r. t. rule $r_1 \in T_\Sigma$ on CUSTOMER relation of D_2 in Example 1.1.

Algorithm 1: Secure Blocking Protocol

```
Input: $D$ ($\mathcal{A}$), and $X'$ of $T_\Sigma^X$ ($\mathcal{B}$)
Output: $t_i[r_A^\phi]$ and $t_i[r_B^\phi]$, $\forall t_i \in D$
/*initialize membership (all records are members)*/
foreach $t_i \in D$ do
    $t_i[r_A^\phi] \leftarrow 1$, $t_i[r_B^\phi] \leftarrow 0$ ;
end
foreach $x_k \in X'$ (Chosen by $\mathcal{B}$ from a random order of $k$) do
    foreach $t_i \in D$ do
        (a) $\mathcal{A}$ and $\mathcal{B}$ use the oblivious attribute selection
        protocol to generate random shares $u$ and $v$ such that
        $(u + v) = t_i[x_k]$; $\mathcal{A}$ retains $u$ and $\mathcal{B}$ retains $v$.
        (b) $\mathcal{B}$ generates the random XOR share for the rule
        matching set membership, $t_i[r_B^\phi](new)$.
        (c) $\mathcal{B}$ creates the garbled circuit with the following
        expression:
        $t_i[r_A^\phi](new) \leftarrow ((t_i[r_A^\phi](old) \oplus t_i[r_B^\phi](old)) \wedge ((u+v) =$
        $r_\phi[x_k])) \oplus t_i[r_B^\phi](new)$
        (d) $\mathcal{B}$ sends $\mathcal{A}$ keys for inputs $v$, $r_\phi[x_k]$ and $t_i[r_B^\phi]$; $\mathcal{A}$
        performs $OT_1^2$ for keys of inputs $u$ and $t_i[r_A^\phi]$
        (e) $\mathcal{B}$ sends $\mathcal{A}$ the garbled circuit, $\mathcal{A}$ evaluates it and
        obtains the bit $t_i[r_A^\phi](new)$
    end
end
```

B. Obfuscated Matching Set Generation

After the secure blocking step, the records are partitioned into rule matching sets, $S_\phi, \forall r_\phi \in T_\Sigma$. We call the union of all rule matching sets, the *Matching Set*, denoted as $S = \bigcup_\phi S_\phi$. The Matching Set can be determined by the following secure circuit evaluated on $\mathcal{A}$'s side: $\forall r_\phi \in T_\Sigma, \bigvee(t_i[r_A^\phi] \oplus t_i[r_B^\phi])$. Therefore, the subsequent steps in the protocol can consider just the records in the matching set S instead of the whole database D. We also notice that the set membership of t_i would be visible to both $\mathcal{A}$ and $\mathcal{B}$.

The above naïve approach for determining the matching set is subject to privacy attacks. For example, the fewer the number of rules, the higher the chances for $\mathcal{B}$ to be able to link records to the constant parts of the rule. The worst case is: if $\mathcal{B}$ has a single rule that matches a single database record on $\mathcal{A}$! Consider another situation where all the rules have constant values and no wildcards. If this information along with the list of attributes in the pattern tableaux is known to $\mathcal{A}$, she will be able to accurately regenerate the rules on her side. In order to eliminate this problem, random records-ids are added to the matching set and thus the obfuscated matching set is generated.

The objective of generating an obfuscated Matching Set is to protect both $\mathcal{A}$ and $\mathcal{B}$ from the privacy concerns that may arise primarily due to a small number of rules The obfuscated matching set, $\mathcal{S}$ can be generated by adding extra record-ids to the existing members of the Matching Set, S. We define an obfuscation parameter, η as the percentage of records that need to be added to S to generate $\mathcal{S}$. η provides a bound for the number of records to be added to $\mathcal{S}$. However, in order to do this, $\mathcal{A}$ and $\mathcal{B}$ must learn the current size of S, $|S|$, which can be securely obtained by having $\mathcal{A}$ and $\mathcal{B}$ evaluate the following expression in a secure fashion: $|S| \leftarrow \forall r_\phi \in T_\Sigma, \sum_{i=0}^{|D|}(t_i[r_A^\phi] \oplus t_i[r_B^\phi])$.

The result of secure count of matching set records remains oblivious to $\mathcal{B}$. To provide sufficient privacy for a small number of records in the matching set (such as, $|S| = 1$) $\mathcal{A}$ can set η to a higher value ($\eta >> 1.0$). On the other hand, for a sufficiently large number of records it can be set to a smaller value (such as, $\eta << 1.0$). The set membership of $\mathcal{S}$ can be obliviously obtained by simply OR-ing a randomly generated bit '1' (for $|S| \times (1+\eta)$ records) with the Boolean expression of the secure matching set generation. The modified expression to determine whether t_i is a member of the obfuscated matching set, $\mathcal{S}$ would be: $\forall r_\phi \in T_\Sigma, (\bigvee(t_i[r_A^\phi] \oplus t_i[r_B^\phi])) \vee b_i$, where b_i is randomly chosen from $\{0, 1\}$. Thus, we can write $\mathcal{S} = \{t_i \in D | b_i \vee (t_i[r_A^\phi] \oplus t_i[r_B^\phi]), 1 \leq i \leq |D|\}$ and $|\{b_i = 1, 1 \leq i \leq |D|\}| = |S| \times (1 + \eta)$. Therefore, the bound for the number of records in $\mathcal{S}$ would be: $|S| \times (1+\eta) \leq |\mathcal{S}| \leq |S| \times (2+\eta)$. After securely evaluating the expression both $\mathcal{A}$ and $\mathcal{B}$ become aware of the members of the obfuscated matching set, $\mathcal{S}$. Henceforth, the protocol considers only the records of $\mathcal{S}$ instead of D. For simplicity reasons we consider the records in $\mathcal{S}$ are indexed from 1 through $|\mathcal{S}|$. Fig. 2 shows an example of an obfuscated

rid	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
$t_i[r_A^1]$	0	1	0	1	0	0	1	0	1	1	0
$t_i[r_b^1]$	1	0	1	0	1	1	0	0	1	1	0
b_i	0	1	1	0	1	1	0	1	1	1	1
$t_i[\mathcal{S}]$	1	1	1	1	1	1	1	1	1	1	1

Fig. 2. The XOR shares of rule matching set membership and obfuscated matching set of D_2 w.r.t. $r_1 \in T_\Sigma$ and $\eta = 0.3$

rid	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
$t_i[G_A^\phi]$	1	10	55	16	6	-61	-71	30	65	-81	87
$t_i[G_B^\phi]$	0	-9	-54	-15	-1	66	76	-22	-56	91	-76
sum	1	1	1	1	5	5	5	8	9	10	11

Fig. 3. Additive shares of group numbers of D_2 after Oblivious Grouping performed with attribute 'ZIP' w.r.t. $r_1 \in T_\Sigma$

rid	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
$t_i[\Upsilon_A^1]$	1	0	0	1	0	1	0	1	0	1	0
$t_i[\Upsilon_B^1]$	0	1	1	0	1	0	1	1	0	1	0
$t_i[\Upsilon_1]$	1	1	1	1	1	1	1	0	0	0	0

Fig. 4. XOR shares of inconsistency in D_2 records w.r.t. $r_1 \in T_\Sigma$

matching set for D_2 considering just one rule $r_1 \in T_\Sigma$ and $\eta = 0.3$.

C. Match Matrix Generation

We denote the exact match (using edit distance) of two records on some attribute with the value 0 and mismatch with the value 1. This information is hidden from $\mathcal{B}$ and is used as input by $\mathcal{A}$ to the 'garbled circuit' in oblivious grouping and inconsistency detection with wildcards steps. Before these steps, $\mathcal{A}$ generates $|\mathcal{S}| \times |\mathcal{S}|$ distance matrices for all m attributes agreed upon by $\mathcal{A}$ and $\mathcal{B}$ prior to the protocol. An entry $d_{ij}^k = 0/1$ in the k-th matrix indicates a match/mismatch between t_i and t_j on the k-th attribute. The space complexity of match matrices is $O(m \times |\mathcal{S}|^2)$.

D. Inconsistency Detection with RHS const.

The secure blocking step obliviously generates one rule matching set, S_ϕ per rule ($r_\phi \in T_\Sigma$). All records $t_i \in S_\phi$ exactly match the constants on the LHS in the rule r_ϕ, and S_ϕ membership is obliviously shared between $\mathcal{A}$ and $\mathcal{B}$. If any RHS attribute of the rule is a constant, the record $t_i \in S_\phi$ must exactly match that value. This step is very similar to the secure blocking step; here the secure comparison is performed between a data record ($t[y_k]$) and a constant RHS (instead of LHS) attribute ($r[y_k]$) in a rule. During initialization the record is considered consistent w. r. t. the rule ($t[\Upsilon_A^\phi] = t[\Upsilon_A^\phi] = 0$). The inconsistency/consistency information is then XOR shared between $\mathcal{A}$ and $\mathcal{B}$ such that:

$$t_i[\Upsilon_A^\phi] \oplus t_i[\Upsilon_B^\phi] = \begin{cases} 1, & \text{if record } t_i \in S_\phi \text{ is inconsistent} \\ 0, & \text{otherwise} \end{cases}$$

Like the secure blocking protocol, $\mathcal{B}$ keeps his share fo inconsistency bits, $t_i[\Upsilon_B^\phi]$ and $t_j[\Upsilon_B^\phi]$. $\mathcal{A}$ obtains the following Boolean circuit from $\mathcal{B}$ and securely evaluates it to determine her share of inconsistency bits, $t_i[\Upsilon_A^\phi]$ and $t_j[\Upsilon_A^\phi]$:

$t_i[\Upsilon_A^\phi](new) \leftarrow ((t_i[r_A^\phi] \oplus t_i[r_B^\phi]) \wedge (((u+v) \neq r_\phi[y_k]) \vee ((t_i[\Upsilon_A^\phi](old) \oplus t_i[\Upsilon_B^\phi](old))))) \oplus t_i[\Upsilon_B^\phi](new)$

The inconsistency bits obtained from the previous iteration (shown as old in the expression) are used as inputs in the current iteration to check if t_i and t_j are still consistent.

E. Oblivious Grouping with LHS wildcards

The goal of this step is to perform a grouping of the records in the obfuscated matching set, $\mathcal{S}$, that exactly match on all LHS attributes $x_i \in X, 1 \le i \le m$ of a specific rule. Here, m is the number of attributes in T_Σ^X. Each relevant record will be assigned a group number and that number is additively shared between $\mathcal{A}$ and $\mathcal{B}$. For a CFD rule in the pattern tableau $r_\phi \in T_\Sigma$, the assigned group number for tuple t_i is denoted as $t_i[G^\phi]$. The expression $t_i[G^\phi] = t_j[G^\phi]$ denotes that for a rule $\phi : X \to Y, \forall X_l \in X : t_i[X_l] = t_j[X_l] \asymp r_\phi[X_l]$. The basic idea of this algorithm is to compare each record t_i with every other record t_j that appears latter in $\mathcal{S}$, and assign a group number to t_j based on the match/mismatch on LHS wildcard attribute in a CFD rule. Initially, for any record the group number is the same as the record index ($t_j[G^\phi] = j$). The first iteration of the oblivious grouping (Algorithm 2) assigns the group number of t_i to t_j (i.e., $t_j[G^\phi](new) \leftarrow t_i[G^\phi]$), when t_i and t_j are in the same rule matching set ($t_i[r^\phi] = t_j[r^\phi]$) and match on the first wildcard attribute x_{k_1} chosen by $\mathcal{B}$ ($d_{ij}^{k_1} = 0$). Otherwise, the group number of t_j remains the same (i.e., $t_j[G^\phi](new) \leftarrow t_j[G^\phi](old)$). Algorithm 2 shows that the additive shares of group number for t_j, $t_j[G_A^\phi](new)$ and $t_j[G_B^\phi](new)$ are always generated irrespective of whether there is a new assignment or not. Fig. 3 shows additive shares of group numbers assigned to records in D_2 on attribute 'ZIP' w.r.t. $r_1 \in T_\Sigma$.

If there is more than one wildcard in the LHS of a rule, successive iterations are necessary for the oblivious grouping algorithm. We briefly describe the necessary modifications to the secure expression of oblivious grouping first iteration to handle these successive iterations. Basically, the expression evaluates three things: 1) are t_i and t_j in the same rule matching set ($t_i[r^\phi] = t_j[r^\phi]$), 2) do t_i and t_j belong to the same group after the previous iteration ($t_i[G^\phi](old) = t_j[G^\phi](old)$), and 3) is there a mismatch between values $t_i[x_k]$ and $t_j[x_k]$ (i.e., $d_{ij}^k = 1$). If all of the above conditions are true, t_j will be assigned group number j. The uniqueness of this group number assignment ensures that t_j is not sharing the group number with any other record. Otherwise, the old group number is retained, but new shares $t_j[G_A^\phi](new)$ and $t_j[G_B^\phi](new)$ are generated. The detailed algorithm for the subsequent iterations is provided in the Appendix.

F. Inconsistency Detection with RHS wildcards

After performing the oblivious grouping of the records per rule, the next phase of the protocol is to find inconsistent records based on the RHS wildcards. Like the Oblivious Grouping step , this step has also to consider pair-wise match between records. Plainly, the records in the same group are

Algorithm 2: Oblivious Grouping with LHS wildcards (First Iteration)

```
Input: $\mathcal{S} \subseteq D$,   $t_i[r_A^\phi]$ and $t_i[r_B^\phi], 1 \le i \le n$,   attribute
    index $k_1$
Output: $t_i[G_A^\phi]$ and $t_i[G_B^\phi]$
//initialization
foreach $t_i \in S_\phi$ do
    $t_i[G_A^\phi] \leftarrow i$ , $t_i[G_B^\phi] \leftarrow 0$;
end
$\mathcal{B}$ chooses a random attribute index $k_1, 1 \le k_1 \le m$
foreach $t_i \in \mathcal{S}$ do
    foreach $t_j \in \mathcal{S}, 1 \le i < j \le n$ do
        (a) $\mathcal{B}$ generates a random share $t_j[G_B^\phi](new)$ and
        keeps it
        (b) $\mathcal{B}$ creates the garbled circuit with the following
        exp.:
        if
        $(t_i[r_A^\phi] \oplus t_i[r_B^\phi] = 1) \wedge (t_j[r_A^\phi] \oplus t_j[r_B^\phi] = 1) \wedge (d_{ij}^{k_1} = 0)$
        then
            $t_j[G_A^\phi](new) \leftarrow t_i[G_A^\phi] + t_i[G_B^\phi] - t_j[G_B^\phi](new)$
        else
            $t_j[G_A^\phi](new) \leftarrow$
            $t_j[G_A^\phi](old) + t_j[G_B^\phi](old) - t_j[G_B^\phi](new)$
        end
        (c) $\mathcal{B}$ sends $\mathcal{A}$ keys for inputs $t_i[r_B^\phi]$, $t_j[r_B^\phi]$, $t_i[G_B^\phi]$,
        $t_j[G_B^\phi](old)$, $t_j[G_B^\phi](new)$, and $k_1$; $\mathcal{A}$ performs $OT_1^2$
        for keys of $t_i[r_A^\phi]$, $t_j[r_A^\phi]$, $t_i[G_A^\phi]$, $t_j[G_A^\phi](old)$, and
        $d_{ij}^1$,...,$d_{ij}^m$
        (d) $\mathcal{B}$ sends $\mathcal{A}$ the garbled circuit, $\mathcal{A}$ evaluates it and
        obtains the value $t_j[G_A^\phi](new)$
    end
end
```

inconsistent, if at least one of their values mismatch on the RHS attribute(s) in the rule [5]. The initial inconsistency XOR shares, $t_i[\Upsilon_A^\phi]$ and $t_i[\Upsilon_B^\phi]$ are obtained from the inconsistency detection with RHS constants step. If there are no constants on RHS, the shares are initialized to 0.

Algorithm 3 shows the inconsistency detection with RHS wildcards step. In this step, for each wildcard attributes in the RHS, each record t_i is compared against the records t_j such that $i < j$. The Boolean expression assigned to $temp1$ actually checks whether both records t_i and t_j are members of rule matching set, S_ϕ, and if they have the same group number. The inconsistency bits modified in the previous iteration are shown as old shares in the expression of current iteration. The expression $temp2$ checks whether at least one of t_i or t_j is inconsistent. Records t_i and t_j will be inconsistent if any of them are currently inconsistent or there is a mismatch on the k-th attribute, i.e., $t_i[y_k] \neq t_j[y_k]$. Therefore, in the expression both t_i and t_j are marked inconsistent if $temp1$ is 1 and either $temp2$ is 1, or t_i and t_j mismatch on the k-th attribute, i.e., $d_{ij}^k = 1$. If the above condition does not hold, the old values are retained, but new shares, $t_i[\Upsilon_A^\phi](new)$ and $t_j[\Upsilon_B^\phi](new)$ are generated for $\mathcal{A}$ and $\mathcal{B}$. Fig. 4 shows an example of XOR shared inconsistency bits of records in D_2 w. r. t. $r_1 \in T_\Sigma$.

Algorithm 3: Inconsistency Detection with RHS wildcards

```
Input: $\mathcal{S} \subseteq D$,   $t_i[r_A^\phi]$ and $t_i[r_B^\phi], 1 \le i \le n$,   $t_i[G_A^\phi]$ and
    $t_i[G_B^\phi]$,   $t_i[\Upsilon_A^\phi](old)$ and $t_i[\Upsilon_B^\phi](old), \forall t_i \in D'$ from
    inconsistency detection with RHS constants step
Output: $t_i[\Upsilon_A^\phi](new)$ and $t_i[\Upsilon_B^\phi](new), \forall t_i \in D'$
foreach $x_k \in X, 1 \le k \le m$ do
    foreach $t_i \in \mathcal{S}$ do
        foreach $t_j \in \mathcal{S}, 1 \le i < j \le n$ do
            (a) $\mathcal{B}$ generates random XOR shares $t_i[\Upsilon_B^\phi](new)$
            and $t_j[\Upsilon_B^\phi](new)$
            (b) $\mathcal{B}$ creates the garbled circuit with the following
            exp.:
            $temp1 \leftarrow (t_i[r_A^\phi] \oplus t_i[r_B^\phi]) \wedge (t_j[r_A^\phi] \oplus t_j[r_B^\phi]) \wedge$
            $(t_i[G_A^\phi] + t_i[G_B^\phi] = t_j[G_A^\phi] + t_j[G_B^\phi])$
            $temp2 \leftarrow (t_i[\Upsilon_A^\phi](old) \oplus t_i[\Upsilon_B^\phi](old)) \vee$
            $(t_j[C_A^\phi](old) \oplus t_j[\Upsilon_B^\phi](old))$
            $temp3 \leftarrow temp1 \wedge (d_{ij}^k \vee temp2)$
            if $temp3 = 1$ then
                $t_i[\Upsilon_A^\phi](new) \leftarrow temp3 \oplus t_i[\Upsilon_B^\phi](new)$
                $t_j[\Upsilon_A^\phi](new) \leftarrow temp3 \oplus t_j[\Upsilon_B^\phi](new)$
            else
                $t_i[\Upsilon_A^\phi](new) \leftarrow$
                $(t_i[\Upsilon_A^\phi](old) \oplus t_i[\Upsilon_B^\phi](old)) \oplus t_i[\Upsilon_B^\phi](new)$
                $t_j[\Upsilon_A^\phi](new) \leftarrow$
                $(t_j[\Upsilon_A^\phi](old) \oplus t_j[\Upsilon_B^\phi](old)) \oplus t_j[\Upsilon_B^\phi](new)$
            end
            (c) $\mathcal{B}$ sends $\mathcal{A}$ keys for inputs $t_i[r_B^\phi]$, $t_j[r_B^\phi]$,
            $t_i[G_B^\phi]$, $t_j[G_B^\phi]$, $t_i[\Upsilon_B^\phi](old)$,
            $t_j[\Upsilon_B^\phi](old)$,$t_i[\Upsilon_B^\phi](new)$, $t_j[\Upsilon_B^\phi](new)$ and $k$; $\mathcal{A}$
            performs $OT_1^2$ for keys of inputs $t_i[r_A^\phi]$, $t_j[r_A^\phi]$,
            $t_i[G_A^\phi]$, $t_j[G_A^\phi]$, $t_i[\Upsilon_A^\phi](old)$,
            $t_j[\Upsilon_A^\phi](old)$,$t_i[\Upsilon_A^\phi](new)$, $t_j[\Upsilon_A^\phi](new)$, and
            $d_{ij}^1$,...,$d_{ij}^m$
            (d) $\mathcal{B}$ sends $\mathcal{A}$ the garbled circuit, $\mathcal{A}$ evaluates it
            and obtains the value $t_i[\Upsilon_A^\phi]$ and $t_j[\Upsilon_A^\phi]$
        end
    end
end
```

G. Final Inconsistency Detection

The final step of the protocol is to securely detect the records that are inconsistent w. r. t. at least one rule $r_\phi \in T_\Sigma$. The information that whether a record is inconsistent or not is only available to $\mathcal{A}$ and $\mathcal{B}$ learns nothing about it. Recall from section III that the inconsistency of a record is denoted by a bit, C_i for $t_i \in \mathcal{S} \subseteq D$. Thus, $\mathcal{A}$ simply evaluates the following secure expression to determine the inconsistency for record t_i:

$$t_i[C^\phi] \leftarrow \forall r_\phi \in T_\Sigma, \bigvee_\phi (t_i[\Upsilon_A^\phi] \oplus t_i[\Upsilon_B^\phi])$$

V. Experiments

We have implemented a prototype of our privacy-preserving inconsistency detection protocol on top of the fairplay secure two-party computation platform [19]. Developed in Java, fairplay allows two parties to perform TCP/IP socket communication and securely evaluate a Boolean expression using a 'garbled' representation of a circuit. The pseudo-random generator function used in the fairplay implementation is SHA-1 which generates digests of length 160 bits that are used for circuit communication. We used the 'Adult' dataset from UCI Machine Learning Repository [20] namely and 6 rules (5 constant rules and 1 rule with wildcards) to conduct the experiments.

The goal of the experiment is to measure and report the performance of the individual major steps of the protocol (secure blocking, oblivious grouping, and inconsistency detection with constants and wildcards). All of these major steps can be performed in parallel for individual rules. The timing shown here is the average for all rules and attributes used. Fig. 5a shows the time required by the secure blocking and inconsistency detection with RHS constants steps with varying numbers of records (100-1000) from the original dataset and matching set respectively. Since the secure expressions of these two steps are very similar, they demonstrate almost the same performance and the time grows linearly w.r.t. the number of records. Fig. 5d shows the time required by the oblivious grouping and inconsistency detection with RHS wildcards steps with varying numbers of records (100-500). The time grows quadratically w.r.t. the number of records. From the detailed time shown in fig. 5b and 5e, it is clear that most of the time in our approach is spent on oblivious transfer of inputs. It supports the fact that the communication complexity dominates the overall protocol. Fig. 5c shows the sizes of matching set and obfuscated matching sets generated for $\eta = 0.1, 0.4$ and 0.8 with varying numbers of records (100-1000) in our dataset. Since the number of records in the matching set is sufficiently large (e.g., for 600 records 364 matching set entries), setting η to a large value such as 0.4 and 0.8 would include the same number of records as the original recordset and no performance will be gained. Fig. 5f shows the significant reduction in time for oblivious transfer when the group number input share is reduced from 32 bits to 16 bits in the circuit. However, the 'garbled circuit' communication time remains the same because of the fixed length of SHA-1 function.

The quadratic algorithms in the protocol, namely, oblivious grouping and inconsistency detection with wildcards (the complexity analysis is reported in the Appendix) incur high cost on large datasets. However, hash partitioning the dataset on the attributes agreed upon between the data and rules owner can significantly contribute to the performance gain, while maintaining the correctness of the protocol. The hash partitioning can be done on the attributes that have either constants or wildcards on the LHS of all rules, but do not have wildcards on the RHS. In cases where such attributes cannot be found, the rules can be partitioned to satisfy these criteria and thus different hash partitioning can be used on the same dataset. However, as a trade-off, the rules owner may have to reveal some information about the attributes (having wildcards or constants in the rules), but still no data content will be leaked.

VI. Related Work

Much work has been done on data quality [21], record linkage [22], and extending functional dependencies [5], [6], [8], [9] to identify data quality rules that will be used to clean the data. However, there have been very few efforts on data quality and data cleaning in private settings. Existing efforts on private record linkage protocols include [23] and [13]. Scannapieco et. al. [13] transform the records to a vector space (with the SparseMap approach) before performing the schema and data matching by a third party. Yakout et. al. [23] use the same approach for embedding records and eliminate third party by using secure dot product protocol. The privacy preserving data quality assessment [14] embeds data and domain look-up table values with Borugain Embedding. The encrypted data is verified against encrypted look-up table values by a third party (certifier). The use of this protocol will allow the parties to know the statistics of records after matching with the records. Privacy preserving data mining [11] and data imputation [24]use cryptographic protocols. However, they do not provide solution to the data cleaning problem.

VII. Conclusions

We proposed a novel algorithm for identifying inconsistent records, detected based on CFD rules, from private data without compromising both data and rules. To the best of our knowledge, this is the first work that solves this problem for rule-based private data cleaning without the use of a third party. The key idea is to obliviously perform all the steps involved in identifying inconsistent records, providing the final result only to the appropriate party. In our case, the final result is an indicator of inconsistency for each record provided only to the data owner. The use of Yao's combinatorial garbled circuits for secure function evaluation (SFE) provides sufficient privacy guarantees. The high computational and communication cost due to SFE is minimized by limiting the number of input bits (to reduce the overhead of OT protocol). The obfuscated matching set eliminates the need for considering the whole dataset (from oblivious grouping step and onwards) while providing sufficient privacy guarantee.

Introduced mainly for data cleaning, CFD rules can only detect inconsistent records if the records exactly match the LHS of the rule. This requires that the LHS of the record must have to be clean to detect the inconsistent records. This requirement may not be necessary with other forms of data quality rules such as conditional inclusion dependencies (CIND) and Matching Dependencies (MD). Two important extensions to our privacy-preserving inconsistency detection framework is to include different data quality rules [8], [25], [7] as well as provide repair suggestions for dirty data to complete the picture of the privacy-preserving data cleaning.

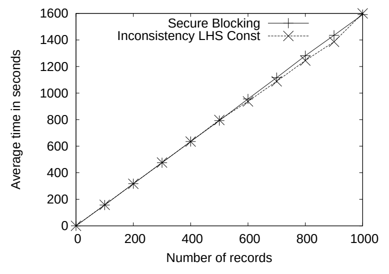
References

[1] W. Eckerson, "Data warehousing special report: Data quality and the bottom line," http://www.tdwi.org/research/display.aspx?ID=6064, 2002.

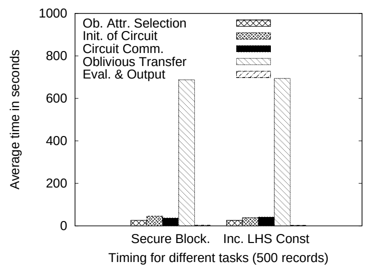
[2] J. M. Hellerstein, "Quantitative data cleaning for large databases," *Report for UNECE,2008*.

[3] C. C. Shilakes and J. Tylman, "Enterprise information portals," New York, USA, 1998.

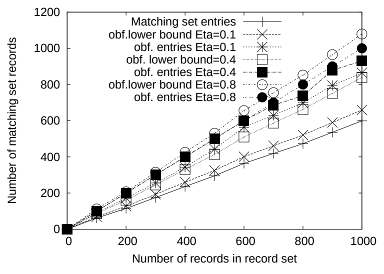
[4] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga, "Cords: Automatic discovery of correlations and soft functional dependencies," in *SIGMOD*, 2004.



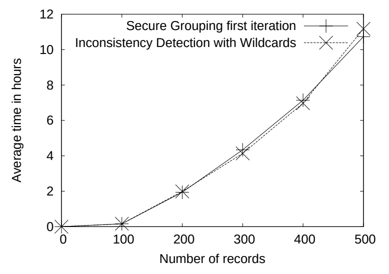
(a) Secure Blocking and Inconsis. Detection LHS Const. overall



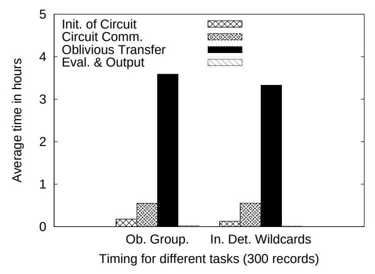
(b) Secure Blocking and Inconsis. Detection LHS Const. details(500 records)



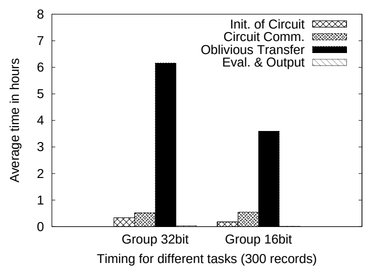
(c) Number of records with different η values



(d) Ob. Grouping and Inconsis. Detection RHS wildcards overall



(e) Ob. Grouping and Inconsis. Detection RHS wildcards details (300 records)



(f) Ob. Grouping 16 and 32 bit inputs

Fig. 5. Performance of privacy-preserving inconsistency detection

[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," vol. 0, 2007.

[6] L. Bravo, W. Fan, and S. Ma, "Extending dependencies with conditions," in *VLDB*, 2007.

[7] S. Song and L. Chen, "Discovering matching dependencies," Tech. Rep. arXiv:0903.3317, Mar 2009.

[8] A. Arasu, C. Ré, and D. Suciu, "Large-scale deduplication with constraints using dedupalog," in *ICDE*, 2009.

[9] F. Chiang and R. J. Miller, "Discovering data quality rules," *Proc. VLDB Endow.*, vol. 1, no. 1, 2008.

[10] W. Fan, F. Geerts, S. Ma, and H. Müller, "Detecting inconsistencies in distributed data," in *ICDE*, 2010.

[11] Y. Lindell and B. Pinkas, "Privacy preserving data mining," *J. Cryptology*, vol. 15, no. 3, 2002.

[12] A. C. Yao, "Protocols for secure computations," in *SFCS*, 1982.

[13] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid, "Privacy preserving schema and data matching," in *SIGMOD*, 2007.

[14] D. Barone, A. Maurino, F. Stella, and C. Batini, "A privacy-preserving framework for accuracy and completeness quality assessment," in *QDB, VLDB*, 2009.

[15] L. Sweeney, "k-anonymity: a model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, 2002.

[16] A. Inan, M. Kantarcioglu, E. Bertino, and M. Scannapieco, "A hybrid approach to private record linkage," in *ICDE*, 2008.

[17] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel, "Privacy-preserving remote diagnostics," in *CCS*, 2007.

[18] M. O. Rabin, "How to exchange secrets with oblivious transfer," Cryptology ePrint Archive, Report 2005/187, 2005.

[19] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay a secure two-party computation system," in *USENIX Security Symposium*, 2004.

[20] A. Frank and A. Asuncion, "UCI machine learning repository," http://archive.ics.uci.edu/ml, 2010.

[21] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*, ser. Data-Centric Systems and Applications. Springer, 2006.

[22] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 1, 2007.

[23] M. Yakout, M. J. Atallah, and A. Elmagarmid, "Efficient private record linkage," in *ICDE*, 2009.

[24] G. Jagannathan and R. N. Wright, "Privacy-preserving imputation of missing data," *Data K. Eng.*, vol. 65, no. 1, 2008.

[25] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," *Proc. VLDB Endow.*, vol. 1, no. 1, 2008.

[26] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: Single database, computationally-private information retrieval," in *FOCS*, 1997.

[27] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

Appendix

TABLE IV
NOTATIONS

Terms	Interpretations
$t_i[r_A^\phi]$ and $t_i[r_B^\phi]$	XOR shares of rule matching set membership bit assigned to $\mathcal{A}$ and $\mathcal{B}$ such that $t_i[r_A^\phi] \oplus t_i[r_B^\phi] = 1$, if $t_i \in S_\phi$, and 0, otherwise.
$t_i[G_A^\phi]$ and $t_i[G_B^\phi]$	Additive shares of group number assigned to $\mathcal{A}(\mathcal{B})$ such that $t_i[G_A^\phi] + t_i[G_B^\phi]$ denotes the group number of record t_i.
d_{ij}^k	An indicator of exact match on attribute values held by $\mathcal{B}$ such that $d_{ij}^k = 0$, if t_i and t_j exactly match on attribute k 1, otherwise.
$t_i[\Upsilon_A^\phi]$ and $t_i[\Upsilon_B^\phi]$	XOR shares of consistency bit assigned to $\mathcal{A}$ and $\mathcal{B}$ such that $t_i[\Upsilon_A^\phi] \oplus t_i[\Upsilon_B^\phi] = 1$, if $t_i \in \Upsilon_\phi$, and 0, otherwise.

The union-compatibility of two or more CFDs is defined as CFDs having the same set of LHS and RHS attributes in their pattern tableaux [5]. For example, for two CFDs, $\phi_1 : \{X \to Y, T\}$ and $\phi_2 : \{X' \to Y', T'\}$ to be union-compatible, the following must hold: $X = X'$ and $Y = Y'$. Using these union compatible rules, we can construct a unified or merged pattern tableau replacing two separate tableaux T and T' [5]. However, if the CFDs are not union-compatible, we need to extend the CFDs to have the same set of attributes. For example, we extend CFD ϕ_1 to have attributes $Z = (X \cup Y) - (X' \cup Y')$ and the value for $r_k[z], \forall z \in Z$ where $r_k \in T$ are set to 'don't care' symbol'@'. The merged pattern tableau is denoted by T_Σ for a set of CFDs Σ. The definition for CFD violation described in the previous section now considers only '@'-free attributes, $X^{free} \subseteq X$ and $Y^{free} \subseteq Y$ (attributes with '-' and constants only). Again, to distinguish between two sets of attributes X and Y, we can split the pattern tableau T_Σ into two tableaux T_Σ^X and T_Σ^Y. Recall from Table III, the merged pattern tableaux for CFDs ϕ_1 and ϕ_2 from Example 1.1.

A. *Random Shares*

Our protocol uses random shares of values between $\mathcal{A}$ and $\mathcal{B}$ to denote the rule matching set, inconsistency for a specific rule, and split data content. Two types of shares are used a) additive sharing and b) XOR sharing. For example, in additive sharing, the value x is additively split between $\mathcal{A}$ and $\mathcal{B}$ such that they have x_A and x_B respectively, and $x = x_A + x_B$. None of the parties learn x. The additions are done using modular arithmetic ($\mod N$) where N is a sufficiently large number. For simplicity, we omit the use of 'mod' throughout the paper. A bit z is XOR shared between $\mathcal{A}$ and $\mathcal{B}$ such that they have z_A and z_B respectively, and $z = z_A \oplus z_B$. None of the parties learn the actual value of z.

B. 1 *out of* N *Oblivious Transfer Protocol*

Also denoted as OT_1^N, 1 out of N oblivious transfer protocol was first introduced by Rabin [18]. In this protocol, one party (known as the sender) holds an array of values $\bar{x} = x[0], x[1], ..., x[N]$ from some domain X, and the other party (known as the receiver or chooser) holds an index $j, 1 \leq j \leq N$. As a result of the protocol the receiver obtains the value of $x[j]$ and nothing else, whereas the sender learns nothing. Efficient OT_1^N protocols are described under the heading of Private Information Retrieval (PIR) by Kushilevitz et. al. [26]. The communication complexity of this protocol is $O(N^{0.5+c})$, where the database is represented as an $N \times N$ bit matrix, $c = \log \chi$ is a constant and χ is the security parameter [26]. Latter described in this section, OT_1^2 is used in Yao's secure function evaluation (SFE) protocol [12].

C. *Homomorphic Encryption*

A homomorphic encryption is a form of encryption where it is possible to perform algebraic operations (such as, addition and multiplication) on plaintexts given their ciphertexts. In our protocol, we need an additive homomorphic encryption scheme for the oblivious attribute selection protocol described in the next subsection. With an additive homomorphic encyption, given two ciphertexts $E(x)$ and $E(y)$ of corresponding plaintexts x and y, it is possible to compute $E(x+y)$. Paillier cryptosystem [27] has this additive homomorphic property and can provide safeguard against semi-honest participants of a secure protocol.

D. *Oblivious Attribute Selection Protocol*

In the oblivious blocking step of our protocol, a match is performed between constant (LHS) attributes in the rule and data records. To hide the attributes included in the rule from $\mathcal{A}$ and hide the actual data value from B, the oblivious attribute selection protocol [17] is used. After performing this protocol, the value of a particular attribute (chosen from a rule by $\mathcal{B}$ and hidden from $\mathcal{A}$) is additively shared between $\mathcal{A}$ and $\mathcal{B}$; $\mathcal{A}$ and $\mathcal{B}$ learn nothing else. The protocol goes as follows: $\mathcal{A}$ generates public/private key pairs of additive homomorphic encryption and sends $\mathcal{B}$ the public key. $\mathcal{A}$ sends the ciphertexts for attribute values of record t_i, $E(t_i[X_1]), ..., E(t_i[X_m])$ to $\mathcal{B}$, where $A_k, 1 \leq i \leq m$ is the list of attributes initially agreed by $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{B}$ generates a random share y and computes $E(t_i[X_k] - y)$ using a homomorphic encryption scheme. This value is sent to $\mathcal{A}$ which then obtains the plaintext, $x = t_i[X_k] - y$. Therefore, $\mathcal{A}$ and $\mathcal{B}$ obtain the additive share of values for $t_i[X_k]$ such that $t_i[X_k] = x + y$.

E. *Secure Function Evaluation*

Our protocol extensively uses secure function evaluation with secure two-party computation which was originally proposed by Yao [12]. Yao uses a 'combinatorial garbled circuit' approach to securely compute a Boolean function represented as a circuit. Let us consider that $\mathcal{A}$ and $\mathcal{B}$ want to securely evaluate a function $C(x, y)$ with inputs (each with n bits) $x = x_1, ..., x_n$ and $y = y_1, ..., y_n$ owned by $\mathcal{A}$ and $\mathcal{B}$ respectively. $C(x, y)$ must be evaluated without sharing the inputs with each other. In the 'garbled circuit', a random key is generated for each input wire (0 or 1) of the gates. The length of the key is the output of a pseudo-random number generator used in the protocol. For a binary gate, all four outputs represented as random keys are encrypted using two corresponding input keys, thus generating the garbled truth table for a gate. Let $\mathcal{B}$ be the garbled circuit generator and $\mathcal{A}$ be the evaluator. $\mathcal{B}$ sends encrypted input bits to $\mathcal{A}$ and $\mathcal{A}$ obtains her encrypted input bits from $\mathcal{B}$ by OT_1^2 protocol. $\mathcal{A}$ evaluates the circuit one gate at a time and obtains the final output. For more details and complexity of the 'garbled circuit' evaluation see [12]. An efficient implement of a secure two-party computation is fairplay [19] which was latter extended to multi-party version named fairplayMP. For our setting with two parties, namely data and rules owner, we use the fairplay implementation to securely evaluate the Boolean expressions in our protocol.

In this section we provide the algorithms for inconsistency detection with RHS constants, and oblivious grouping (successive iterations).

Each step in our protocol requires $\mathcal{B}$ to prepare a 'garbled circuit' and send it to $\mathcal{A}$. $\mathcal{A}$ will then independently evaluate the circuit with encrypted inputs of both $\mathcal{A}$ and $\mathcal{B}$, and retain the output. Each of these circuits are made of gates and wires (bits). The input and output wires in the gate are encoded with random binary string or a 'garbled value'. The 'garbled value' of the output wire is encrypted using a pseudo-random generator function F keyed by the'garbled values' of the input

Algorithm 4: Inconsistency detection with RHS constants

```
Input: $\mathcal{S} \subseteq D$ ($\mathcal{A}$), attributes $Y'$ of $r_\phi$ $\mathcal{B}$, and ($t_i[r_A^\phi]$ and
    $t_i[r_B^\phi]$)
Output: $t_i[\Upsilon_A^\phi]$ and $t_i[\Upsilon_B^\phi], \forall t_i \in S$
/* initialization: all records are consistent */
foreach $t_i \in \mathcal{S}$ do
    $t_i[\Upsilon_A^\phi] \leftarrow 0$ ;
    $t_i[\Upsilon_B^\phi] \leftarrow 0$ ;
end
foreach $y_k \in Y'$ do
    foreach $t_i \in \mathcal{S}$ do
        (a) $\mathcal{A}$ and $\mathcal{B}$ use oblivious attribute selection and
        generate random shares $u$ and $v$ such that
        $(u + v) = t_i[y_k]$; $\mathcal{A}$ retains $u$ and $\mathcal{B}$ retains $v$.
        (b) $\mathcal{B}$ generates random XOR share for inconsistency
        bit, $t_i[\Upsilon_B^\phi](new)$.
        (c) $\mathcal{B}$ creates the garbled circuit with the following
        expression:
        $t_i[\Upsilon_A^\phi] \leftarrow ((t_i[r_A^\phi](old) \oplus t_i[r_B^\phi](old)) \wedge (((u + v) \neq$
        $r_\phi[y_k]) \vee ((t_i[\Upsilon_A^\phi](old) \oplus t_i[\Upsilon_B^\phi](old))))) \oplus$
        $t_i[\Upsilon_B^\phi](new)$
        (d) $\mathcal{B}$ sends $\mathcal{A}$ keys for inputs $v$, $t_i[r_B^\phi]$, $t_i[\Upsilon_B^\phi]$ and
        $r_\phi[y_k]$; $\mathcal{A}$ performs $OT_1^2$ for keys of inputs $u$, $t_i[r_A^\phi]$,
        $t_i[\Upsilon_A^\phi]$
        (e) $\mathcal{B}$ sends $\mathcal{A}$ the garbled circuit, $\mathcal{A}$ evaluates it and
        obtains the bit $t_i[\Upsilon_A^\phi](new)$
    end
end
```

Algorithm 5: Oblivious Grouping with LHS wildcards(subsequent iterations)

```
Input: $\mathcal{S} \subseteq D$, $t_i[r_A^\phi]$ and $t_i[r_B^\phi], 1 \leq i \leq n$, attribute index $k$,
    and $t_i[G_A^\phi]$ and $t_i[G_B^\phi]$
Output: $t_i[G_A^\phi]$ and $t_i[G_B^\phi]$
foreach $x_k \in X, 1 \leq k \leq m, k \neq k_1$ in the first iteration, (k's
are chosen in random order) do
    foreach $t_i \in \mathcal{S}$ do
        foreach $t_j \in \mathcal{S}, 1 \leq i < j \leq n$ do
            (a) $\mathcal{B}$ generates a random additive share
            $t_j[G_B^\phi](new)$
            (b) $\mathcal{B}$ creates the garbled circuit with the following
            expression:
            if $(t_i[r_A^\phi] \oplus t_i[r_B^\phi]) \wedge (t_j[r_A^\phi] \oplus t_j[r_B^\phi]) \wedge$
            $(t_i[G_A^\phi](old) + t_i[G_B^\phi](old) =$
            $t_j[G_A^\phi](old) + t_j[G_B^\phi](old)) \wedge (d_{ij}^k = 1)$ then
                $t_j[G_A^\phi](new) \leftarrow j - t_j[G_B^\phi](new)$
            else
                $t_j[G_A^\phi](new) \leftarrow$
                $t_j[G_A^\phi](old) + t_j[G_B^\phi](old) - t_j[G_B^\phi](new)$
            end
            (c) $\mathcal{B}$ sends $\mathcal{A}$ keys for inputs $t_i[r_B^\phi]$, $t_j[r_B^\phi]$,
            $t_i[G_B^\phi](old)$, $t_j[G_B^\phi](old)$, $t_j[G_B^\phi](new)$ and $k$; $\mathcal{A}$
            performs $OT_1^2$ for keys of inputs $t_i[r_A^\phi]$, $t_j[r_A^\phi]$,
            $t_i[G_A^\phi](old)$, $t_j[G_A^\phi](old)$ and $d_{ij}^1$,...,$d_{ij}^m$
            (d) $\mathcal{B}$ sends $\mathcal{A}$ the garbled circuit, $\mathcal{A}$ evaluates it
            and obtains the value $t_j[G_A^\phi](new)$
        end
    end
end
```

wires. For simplicity reasons, let us consider that all the gates are binary gates. Hence, for each gate, $\mathcal{B}$ has to send a garbled truth table with 4 entries to A. The communication overhead of the secure function evaluation (SFE) protocol is then $4ef$, where e is the number of gates in the circuit and f is the size in bits of the random number generated by F, e.g., the SHA-1 hash function generates 160 bit values. As discussed earlier, to evaluate the circuit, $\mathcal{A}$ needs to get her garbled inputs from $\mathcal{B}$ using OT_1^2 protocol. Therefore, each circuit evaluation involves n oblivious transfers of inputs, where n is the number of input bits in the circuit. The overhead of evaluating the circuit (computation cost) is negligible compared to the overhead of the oblivious transfers (communication cost) [19].

Now, let us take a look at the performance of the individual steps in our protocol. The performance of the 'garbled circuit' evaluation in each step depends on the size of the circuit (for communication of the 'garbled gates') and oblivious transfer of inputs. The 'garbled circuit' of each step have different size and have different number of inputs. Therefore, the performance is different for each step. First, we ignore the cost of evaluating the 'garbled circuit' required in each iteration of the steps and determine the complexity in terms of the number of iterations. The secure blocking and inconsistency detection with RHS constants steps take each a linear time in the number of records multiplied by the maximum number of LHS and RHS constant attributes. Let us consider that the maximum number of LHS and RHS constant attributes in the rules are m'_X and m'_Y) respectively. Therefore, the complexity of these steps are: $O(m'_X \times |D|)$ and $O(m'_Y \times |\mathcal{S}|)$ resp.

The oblivious grouping and inconsistency detection with RHS wildcards are the expensive steps in the protocol and cost $O(m''_X \times |\mathcal{S}|^2)$ and $O(m''_Y \times |\mathcal{S}|^2)$ respectively. Here, m''_X and m''_Y are the maximum numbers of wildcards LHS and RHS in a rule. All the above four steps can be performed in parallel for individual rules.

The secure obfuscated matching set generation step takes $O(|D| \times |\Sigma|)$, where it is assumed that $|D| >> |\Sigma|$. The final inconsistency detection step costs $O(|\mathcal{S}| \times |\Sigma|)$. However, these two steps can be performed in parallel by partitioning the records. Unlike the previous two steps where evaluation was done based on each rule, these two can be performed for individual records and all rules. The corresponding costs would be $O(\frac{|D|}{p} \times |\Sigma|)$ and $O(\frac{|\mathcal{S}|}{p} \times |\Sigma|)$, where p is the number of partitions.

As mentioned earlier, each iteration involves the cost of evaluating a 'garbled circuit', i.e. a large constant for a constant size input of individual circuits. For example, inconsistency detection with RHS wildcards step will take $O(\mathcal{C}m''_Y \times |\mathcal{S}|^2)$, where $\mathcal{C}$ is the computation and communication overhead of evaluating the corresponding 'garbled circuit'.

We extended the existing Java based implementation of two-party secure computation platform [19] to develop our privacy-preserving inconsistency detection prototype. We generated the secure circuits with SFDL (secure function Description

TABLE V

NUMBER OF CIRCUIT GATES, INPUTS AND OUTPUT BITS

Functions	Gates	$\mathcal{A}$ Input	$\mathcal{B}$ Input	$\mathcal{A}$ Output
Secure Blocking	1804	201	402	1
Oblivious Grouping (first)	383	39	66	16
Oblivious Grouping (succ.)	487	55	66	16
Inconsistency (constant)	1806	202	403	1
Inconsistency (wildcard)	340	41	54	2

Language) specification of fairplay compiler. The table V shows the number of gates and inputs required by each major circuit in the protocol. In our protocol, secure blocking securely compares the additive shares of the data value from a record and the constant value in the rule. We assume that the attributes do not exceed 25 characters and we need 200 bits for a character array converted into UTF-8 format. That is why the circuits for secure blocking and inconsistency detection with constant steps are fairly large compared to the others. However, they take linear time w.r.t. the number of records. The Oblivious Grouping (first and successive iterations) and inconsistency detection with wildcards steps use 16 bit integers as group number shares for the i-th and j-th records. These protocols take quadratic time and as we have seen from the experiments, most of the time is spent on oblivious transfer of inputs. Therefore, the number of input bits to the circuits should be as small as possible.

We performed all our experiments on a machine with 8 processors (3 Ghz each) having 32GB memory and running GNU/Linux OS and use MySQL database to store the data, rules and the intermediate results.

We are using a real dataset from UCI Machine Learning Repository [20] namely 'Adult' or 'Census Income Dataset'. We considered the following CFD rules discovered from the 'Adult' dataset [9]:

- {marital-status='Married-AF-spouse' $\rightarrow$ relationship='Husband', gender='Male'},
- {marital-status='Married-civ-spouse' $\rightarrow$ relationship='Husband', gender='Male'},
- {marital-status='Married-spouse-absent' $\rightarrow$ relationship='Husband', gender='Male'},
- {workclass='Never-married' $\rightarrow$ occupation='?', salary='<=50K.'},
- {relationship='Own-child' $\rightarrow$ marital-status='Never-married', salary='<=50K.'}, and
- { marital-status='Married', gender='-' $\rightarrow$ relationship='-'}.

For simplicity, we consider the same set of attributes for both LHS and RHS merged pattern tableaux and the attributes are as follows: {'age', 'workclass', 'education', 'marital-status', 'occupation', 'relationship', 'gender', 'salary' }.