Secure Multi-Party Functional Dependency Discovery

Chang Ge
University of Waterloo
c4ge@uwaterloo.ca

Ihab F. Ilyas
University of Waterloo
ilyas@uwaterloo.ca

Florian Kerschbaum
University of Waterloo
fkerschb@uwaterloo.ca

ABSTRACT

Data profiling is an important task to understand data semantics and is an essential pre-processing step in many tools. Due to privacy constraints, data is often partitioned into silos, with different access control. Discovering functional dependencies (FDs) usually requires access to all data partitions to find constraints that hold on the whole dataset. Simply applying general secure multi-party computation protocols incurs high computation and communication cost.

This paper formulates the FD discovery problem in the secure multi-party scenario. We propose secure constructions for validating candidate FDs, and present efficient cryptographic protocols to discover FDs over distributed partitions. Experimental results show that solution is practically efficient over non-secure distributed FD discovery, and can significantly outperform general purpose multi-party computation frameworks. To the best of our knowledge, our work is the first one to tackle this problem.

PVLDB Reference Format:
Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. Secure Multi-Party Functional Dependency Discovery. *PVLDB*, 13(2): 184-196, 2019.
DOI: https://doi.org/10.14778/3364324.3364332

1. INTRODUCTION

Data profiling refers to the general activities of mining metadata about an input dataset, including finding keys, dependencies, statistics and correlations. Profiling is a key prerequisite for many big data tasks, such as master data management [1], data exploration [26], and data integration [25]. Among these metadata, functional dependency (FD) describes specific correlations between attributes. Informally, an FD $A \rightarrow B$ states that for any two tuples that agree on Attribute set A, they also agree on the Attribute B. FDs have been widely used for schema normalization [32], error detection [34] and repair [7], and query optimization [13].

While current FD discovery algorithms assume a single dataset, data is often owned and stored by multiple parties in silos, usually with different access controls. This distributed separation is often required by business or data governance rules (e.g., GDPR [2]) to meet data security and privacy standards. For example, Figure 1 shows the two partitions of an employee dataset for an organization: Partition D_1, which records EU employees is owned and stored by its European subsidiary, and D_2, which records US employees is owned and stored in US.

D_1

	edu	edu_num	marital	occupation	ethnicity	income
t_1	Bachelors	13	Single	Exec-mag	Caucasian	>50K
t_2	Bachelors	13	Married	Exec-mag	Caucasian	>50K
t_3	HS-grad	9	Divorced	Exec-mag	Caucasian	>50K
t_4	11th	7	Married	Handlers	African	≤50K
t_5	Bachelors	13	Married	Sales	African	≤50K

D_2

	edu	edu_num	marital	occupation	ethnicity	income
t_1	HS-grad	9	Divorced	Sales	Caucasian	≤50K
t_2	HS-grad	9	Single	Craft-repair	Caucasian	≤50K
t_3	As-acdm	12	Single	Server	Caucasian	≤50K
t_4	Bachelors	13	Divorced	Exec-mag	Caucasian	≤50K
t_5	11th	7	Married	Exec-mag	African	>50K

Figure 1: Two partitions of employee dataset.

Given these partitioning requirements on the data, the question is how to efficiently discover FDs that hold on the whole dataset, while minimizing the data leakage in the lack of a single trusted party. The question presents a 3-way tradeoff between *soundness*, *privacy* and *efficiency*: soundness requires exchanging information among partitions to corretly compute the global set of FDs; privacy dictates not revealing more than necessary data to other partitions and without revealing certain information under some acceptable definition of privacy; and efficiency necessitates the overall process to run in reasonable (e.g., polynomial) time. Naïvly applying current techniques can be either insecure, incorrect or too expensive. For example, simply computing FDs from each partition separately and intersecting the FD sets might respect the privacy of each partition, but can lead to invalid FDs on the whole dataset; the following example illustrates the problem with this simple approach.

Example: Consider the following FDs:
$f_1 : edu \rightarrow edu\_num$ $\quad f_2 : ethnicity \rightarrow income$

f_1 states that for any two persons with the same *edu*, they must have the same *edu_num*. f_2 states *ethnicity* determines

Proceedings of the VLDB Endowment, Vol. 13, No. 2
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3364324.3364332

income. By examining the tuples from individual partitions, it is clear that f_1 and f_2 are valid FDs on both D_1 and D_2.

However, by examining the tuples from $D = D_1 \cup D_2$, f_2 is not valid on D, due to violations such as $D_1.t_1[ethnicity] = D_2.t_1[ethnicity]$, but $D_1.t_1[income] \neq D_2.t_1[income]$. □

On the other hand, exchanging information between partitions to validate global FDs in an encrypted way might meet both privacy and soundness, but suffers from severe inefficiency. For instance, validating one candidate FD on a dataset requires comparing all needed evidence from all the partitions. Securely comparing all tuples from all partitions is expensive due to the lack of scalable cryptographic constructions. General purpose secure multi-party computation protocols such as garbled circuit [38], homomorphic encryption [29] and oblivious polynomial evaluation [27] incur high cost already when securely computing one function among multiple parties [22], rendering an FD discovery algorithm prohibitively inefficient.

Therefore, to solve the problem of discovering global FDs among multiple parties securely and efficiently, this work presents efficient cryptographic protocols to support discovering FDs in semi-honest multi-party scenarios. We highlight the main contributions of this paper as follows:

- We define the FD discovery problem in the secure multi-party scenario and propose a top-down based framework to validate FDs (Section 2).
- We formulate the distributed FD validation problem over multiple partitions, and provide an efficient solution for discovering FDs in the multi-party scenario (Section 4).
- As the building blocks of our solution, we design efficient mix networks to enable secure equality testing against a semi-honest adversary (Section 5).
- We also propose a relaxed version of FD (referred to as congenial FD) and show that our framework is able to efficiently and securely discover these FDs (Section 6).
- With extensive evaluations using real world datasets, we show that our solution is practically efficient over non-secure distributed FD discovery, and can significantly outperform general purpose multi-party computation frameworks (Section 7).

2. PROBLEM STATEMENT AND SOLUTION OVERVIEW

2.1 Problem Statement

Assume a dataset D in schema R, which is horizontally partitioned into D_1 to D_m: $D = \cup_{i=1}^{m} D_i$. Let S denote the set of all FDs that are valid on the dataset D: $S = \{f \mid D \models f\}$, and let S_i be the set of all FDs that are valid on a partition D_i: $S_i = \{f \mid D_i \models f\}$. Let $\hat{S}$ represent the intersection of all S_i: $\hat{S} = \cap_{i=1}^{m} S_i$. It is clear that $S \subseteq \hat{S}$, as we showed in the example from Section 1.

We call an FD $A \rightarrow B$ *trivial* when $B \in A$. We also call an FD *minimal* if $\nexists K \subseteq A$ such that $A \setminus K \rightarrow B$. Let $\sum$ be the set of all non-trivial and minimal FDs on D, $\sum \models S$, i.e., any $f \in S$ can be either in $\sum$ or implied by $\sum$.

The problem is to securely find $\sum$ against semi-honest adversaries, where each honest-but-curious party $\mathbb{P}_i (i \in [1, m])$ owns a private D_i and follows the protocol honestly, but tries to infer information from other parties.

2.2 Solution Overview

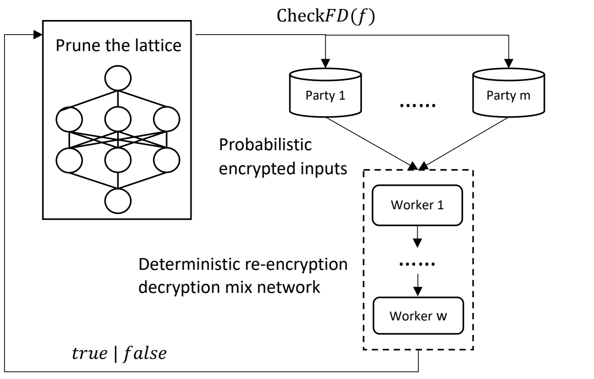


Figure 2: Architecture of secure multi-party FD discovery.

Figure 2 depicts the architecture of our solution. Our framework adopts a top-down approach [23] to prune the FD search space. Assume there are m parties numbered from 1 to m, each holding exactly one database partition $D_i (i \in [1, m])$, and a mix network chain consisting of w workers sequentially from worker 1 to w.

We first construct a set containment lattice. Each node in the lattice represents a set of attributes, and every edge in the lattice represents a candidate FD [17]. In order to prune the space of possible FDs, for each candidate FD f, all the m parties jointly validate the candidate FD, which is represented by the functionality of $CheckFD(f)$. The validation process involves two types of actions: 1) all the parties prepare necessary inputs and encrypt using probabilistic encryption; and 2) the encrypted messages are sent to a deterministic re-encryption decryption mix network. A mix network which consists of a chain of w workers, securely computes on the inputs and and eventually returns the validity of f, which will be used to prune the lattice in the next round. The process runs until all the edges have been properly traversed.

Note that traversing the lattice can be guided by any schema driven search algorithms [3, 17, 28, 39], and is not the contribution of this work. The scope of this work focuses on the fundamental functionality $CheckFD(f)$ that is used to prune the candidate space.

The details of the solution will be discussed as follows: Section 4 explains the $CheckFD(f)$ flow and the end-to-end solution. Two deterministic re-encryption decryption mix networks, which are used together to securely validate candidate FDs will be introduced in Section 5.

3. BACKGROUND

Table 1 lists the notations used in this paper.

3.1 Functional Dependency

DEFINITION 1. *(Functional Dependency) Given a data instance D in schema R, let A be a set of attributes $A \subseteq R$ and B be an attribute $B \in R$, a functional dependency $f : A \rightarrow B$ holds on D if for all tuples $t_1, t_2 \in D, t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]$.*

For $K \subseteq R \setminus A$, $f' : A, K \rightarrow B$ is called a specialization of f, i.e., $f' \in Spec(f)$.

Table 1: Summary of Notation

symbol	meaning
A	an attribute set
B	an attribute
D, D_i	a relational table instance, instance on $\mathbb{P}_i$
S, S_i	a set of all valid FDs on D, D_i
$\hat{S}$	the intersection of all S_i
$\sum$	a set of all non-trivial and minimal FDs on D
$\sum_c$	a set of complete congenial FDs on $\forall i D_i$
t	a tuple in D
$Spec(f)$	a specialization of FD f
π_A^D	a partition over A on data D
V_A^D	the values of equivalent classes over A on D
$e(A^D)$	attribute partition error of A on D
$\mathbb{P}_i$	the i'th party
$\mathbb{W}_i$	the i'th worker
p	a large prime
g	a generator
$\mathbb{G}_p$	a cyclic group of order p using generator g
$\mathbb{Z}_p$	a set of p-adic integers
m	number of parties / partitions
w	number of workers in a mixnet
$H(\cdot)$	a hash function to $\mathbb{Z}_p$ domain
x, y, z	private key for $\mathbb{W}_1$, $\mathbb{W}_2$, $\mathbb{W}_3$
k, K	public key, set of public keys
r	random value sampled by a party
x', y', z'	secret re-encryption key of $\mathbb{W}_1$, $\mathbb{W}_2$, $\mathbb{W}_3$
M	the product of secret re-encryption keys
$f_v(\cdot)$	a boolean function to compare inputs equality
Π_v	a multiplicative deterministic mixnet
$f_s(\cdot)$	a boolean function to compare sum of inputs
Π_s	an additive deterministic mixnet

Definition 2. *(Attribute Partition) Given an attribute set A and a relational instance D, the attribute partition of D under A is a disjoint set as $\pi_A^D = \{[t]_A | t \in D\}$, where $[t]_A = \{t' \in D | t[A'] = t'[A'], \forall A' \in A\}$*

$|\pi_A^D|$ represents the cardinality of the set π_A^D. Since π_A^D is a set of tuple sets, we use $\|\pi_A^D\|$ to denote the cardinality of tuples, which in other words, is equivalent to the cardinality of D. We use V_A^D to represent the set of values of the attribute partition of A on D.

Definition 3. *(Attribute Partition Error) The attribute partition error $e(A)$ of an attribute set A with respect to an instance D is defined as the minimal number of rows that need to be removed in order to make A a super key. Mathematically, $e(A^D) = \|\pi_A^D\| - |\pi_A^D|$.*

3.2 ElGamal Encryption

ElGamal encryption [15] is a probabilistic asymmetric encryption scheme. ElGamal can be defined over a cyclic group $\mathbb{G}_p$ of order p with a generator g. p is usually a large prime number and g is a primitive element of the group. ElGamal encryption consists of three algorithms: key generation, encryption and decryption, which are formally described in Algorithm 1.

The KGen algorithm takes two input values: a large prime number p, and the generator g. The private key x is a random integer uniformly drawn from $\mathbb{Z}_p$, and the public key k is an element in $\mathbb{G}_p$ computed from x and g. By the discrete logarithm assumption [12], it is difficult to infer x from k.

Algorithm 1 ElGamal Scheme

```
Input: $p, g$                                  ▷ a large prime, generator
    procedure KGen($p, g$)
        $x \leftarrow_\$ \mathbb{Z}_p$                         ▷ $x$ is the private key
        $k \leftarrow g^x$                          ▷ $k$ is the public key
        return $(x, k)$
    end procedure

Input: $v, k$                                  ▷ plain value, public key
    procedure Enc($v, k$)
        $r \leftarrow_\$ \mathbb{Z}_p$                         ▷ sample a random value
        $c_1 \leftarrow g^r$
        $c_2 \leftarrow v \cdot k^r$
        return $(c_1, c_2)$
    end procedure

Input: $(c_1, c_2), x$                         ▷ cipher text, private key
    procedure Dec($(c_1, c_2), x$)
        $c_1' \leftarrow c_1^x$                    ▷ updating random parameter
        $c_2 \leftarrow c_2 \cdot (c_1')^{-1}$                       ▷ decrypt
        return $c_2$
    end procedure
```

The Enc algorithm takes inputs of a value v and the public key k, and outputs a cipher text pair (c_1, c_2). The Dec algorithm decrypts the cipher text pair using private key x.

3.3 Mix Network

A mix network (mixnet, in short) [9] is a cryptographic construct where a chain of *workers* establishes hard-to-trace communications between the senders and receivers. The senders encrypt each message to each worker using public key cryptography and send encrypted messages to the mixnet, where each worker strips off an encryption layer, does some operations and shuffles them before enrouting to the next worker sequentially. At the output, receivers receive cryptographically transformed and randomly permuted messages, making the end-to-end communications untraceble.

4. SECURE FD DISCOVERY

Recall in Section 2.2 that our solution is based on validating candidate FD to prune the search space. We first formulate the distribuetd FD validation in Section 4.1, and then introduce the FD discovery protocol in Section 4.2.

4.1 Distributed FD Validation

The main observation is that in order to validate a candidate FD f, one needs to compare all the attributes relevant to f across the partitions. Based on traditional FD validation on a single dataset [17], we can derive the FD validation rule on distributed partitions as follows:

Lemma 1. *(FD Validation [17]) A dataset D in schema R is partitioned into D_1 to D_m: $D = \cup_{i=1}^m D_i$. An FD $A \rightarrow B$ holds on D if and only if $e(A^D) = e((A \cup B)^D)$.*

Example: Consider $D = D_1 \cup D_2$ in Figure 1. For the candidate FD f_1: $\pi_{edu} = \{\{D_1.t_1, D_1.t_2, D_1.t_5, D_2.t_4\}, \{D_1.t_3, D_2.t_1, D_2.t_2\}, \{D_1.t_4,$

$D_2.t_5\}, \{D_2.t_3\}\}$, and hence $e(edu^D) = 10-4 = 6$. Similarly, $e((edu, edu\_num)^D) = 6$. Therefore, $D \models f_1$.

However, for the candidate FD f_2: $e(ethnicity^D) \neq e((ethnicity, income)^D)$; therefore, $D \not\models f_2$. $\square$

Based on Lemma 1, we can further formalize the attribute partition error over multiple parties. For any attribute set $A \subseteq R$, the cardinality of m-set union[1] satisfies:

$$\|\pi_A^D\| = \sum_{i=1}^{m} \|\pi_A^{D_i}\| \tag{1}$$

Meanwhile, the size of attribute partitions satisfies:

$$\begin{aligned} |\pi_A^D| = &\sum_{i=1}^{m} |\pi_A^{D_i}| - \sum_{1\le i<j\le m} |V_A^{D_i} \cap V_A^{D_j}| \\ &+ \sum_{1\le i<j<k\le m} |V_A^{D_i} \cap V_A^{D_j} \cap V_A^{D_k}| + \cdots \\ &+ (-1)^{m-1} |\bigcap_{i=1}^{m} V_A^{D_i}| \end{aligned} \tag{2}$$

Where $V_A^{D_i}$ is the set of values of attribute partition of attribute set A on partition D_i. Let PSI-CA(A^D) $= \sum_{i=1}^{m} |\pi_A^{D_i}| - |\pi_A^D|$. Equation (1) $-$ (2) leads to:

$$e(A^D) = \sum_{i=1}^{m} e(A^{D_i}) + \text{PSI-CA}(A^D) \tag{3}$$

From Equation 3, the final attribute partition error consists of two parts: 1) the sum of attribute partition errors on each partition $\sum_{i=1}^{m} e(A^{D_i})$; and 2) the power set intersection cardinality (PSI-CA). Every party $\mathbb{P}_i$ is able to compute $e(A^{D_i})$ and $V_A^{D_i}$ locally since D_i belongs to $\mathbb{P}_i$. To validate an FD $A \to B$, one simply needs to evaluate the expression $e(A^D) - e((A \cup B)^D) == 0$ based on Equation 3.

Example: Continue the example from Figure 1. Consider $f_1 : edu \to edu\_num$. $e(edu^{D_1}) = 3$, $e(edu^{D_2}) = 4$, $e((edu, edu\_num)^{D_1}) = 3$, and $e((edu, edu\_num)^{D_2}) = 4$.
$V_{edu}^{D_1}$ ={`Bachelors,HS-grad,11th`}
$V_{edu}^{D_2}$ ={`HS-grad,As-acdm,Bachelors,11th`}
$V_{edu,edu\_num}^{D_1}$ = {`(Bachelors,13),(HS-grad,9),(11th,7)`}
$V_{edu,edu\_num}^{D_2}$ = {`(HS-grad,9),(As-acdm,12),(11th,7),(Bachelors,13)`}
PSI-CA(edu^D)=$|V_{edu}^{D_1} \cap V_{edu}^{D_2}|$=3, and PSI-CA$((edu, edu_num)^D)$=$|V_{edu,edu_num}^{D_1} \cap V_{edu,edu_num}^{D_2}|$=3. Hence,
$e(edu^D)$-$e((edu, edu\_num)^D)$=
$e(edu^{D_1})$-$e((edu, edu\_num)^{D_1})$+
$e(edu^{D_2})$-$e((edu, edu\_num)^{D_2})$+
PSI-CA(edu^D)-PSI-CA$((edu, edu_num)^D)$=0.
Therefore, $D \models f_1$. $\square$

4.2 A Secure FD Discovery Protocol

We arbitrarily choose one party to act as the *moderator*, who triggers the execution of FD validation and receives the output from mixnet. The moderator is a role for a party, and learns only the output of the mixnet as other parties do.

Figure 3 illustrates the protocol to securely fulfil the *CheckFD*(f) function: given a candidate FD $f : A \to B$ as input, the protocol consists of three steps: 1) on attribute sets A and AB, it computes the power set intersection cardinality (PSI-CA) if not done yet. Each party encrypts and sends its set of values for the attribute partition via a multiplicative deterministic re-encryption decryption mixnet (Section 5.1). The output of the mixnet is the encrypted set of values for the attribute partition, based on which, the moderator intersects all sets directly over encrypted values for both attribute sets A and AB, and returns the cardinalities; 2) each party computes local attribute partition errors on A and AB and encrypts them to send the difference to an additive deterministic re-encryption decryption mixnet (Section 5.2). Meanwhile, the moderator also encrypts and sends the difference of PSI-CA of attribute sets A and AB as input. The output of the mixnet is the encrypted error difference; finally 3), the moderator concludes the validity of candidate FD based on the output of second mixnet: $A \to B$ is true if and only if the encrypted value equals to 1.

The two mixnets in above protocol will be explained in details later in Section 5. In the rest of this section, we present an efficient approach to compute PSI-CA.

Algorithm 2 Power set intersection cardinality

```
Input: $\mathbb{S}$                         ▷ set of $m$ sets to be intersected
 1: procedure PSI-CA($\mathbb{S}$)
 2:     $card \leftarrow 0$, $imap < int, int >$
 3:     for $t \in [1, m], \forall k \in \mathbb{S}[t]$ do
 4:         $imap[k] + +$                    ▷ increase the counter
 5:     end for
 6:     $cmap < int, int >$          ▷ (occurrance, contribution)
 7:     for all $occ \in imap.values$ do
 8:         if $occ \geq 2$ then        ▷ element occurs than once
 9:             if $\neg cmap.contains(occ)$ then
10:                 $cntrbtn \leftarrow 0$
11:                 for $i \leftarrow occ; i \geq 2; --i$ do
12:                     $c \leftarrow C^i_{occ}$ ▷ $i$-combination: occ-choose-i
13:                     if $i$ mod 2 == 0 then ▷ flipping signs
14:                         $cntrbtn \leftarrow cont + c$
15:                     else
16:                         $cntrbtn \leftarrow cont - c$
17:                     end if
18:                 end for
19:                 $cmap(occ) \leftarrow cntrbtn$
20:             end if
21:             $card \leftarrow card + cmap.get(occ)$
22:         end if
23:     end for
24:     return $card$
25: end procedure
```

A naïve way to compute PSI-CA is that for each set intersection, we intersect the encrypted values directly from the sets. Given there are m partitions, the number of intersections is $2^m - m - 1$, which is expensive. To reduce the exponential cost of PSI-CA, Algorithm 2 presents a linear approach to efficiently compute PSI-CA for a set of m sets. The intuition is that rather than computing intersections, we count the contribution of each value to PSI-CA, which is determined by the number of its occurrence in all partitions. For example, if an value occurs three times in any of the three partitions, then that value will contribute to PSI-CA by being intersected in one 3-set intersection and three 2-set intersections. This method simply requires traversing all

[1]We assume each tuple has its own tuple id and partition id; therefore no identical tuples exist.

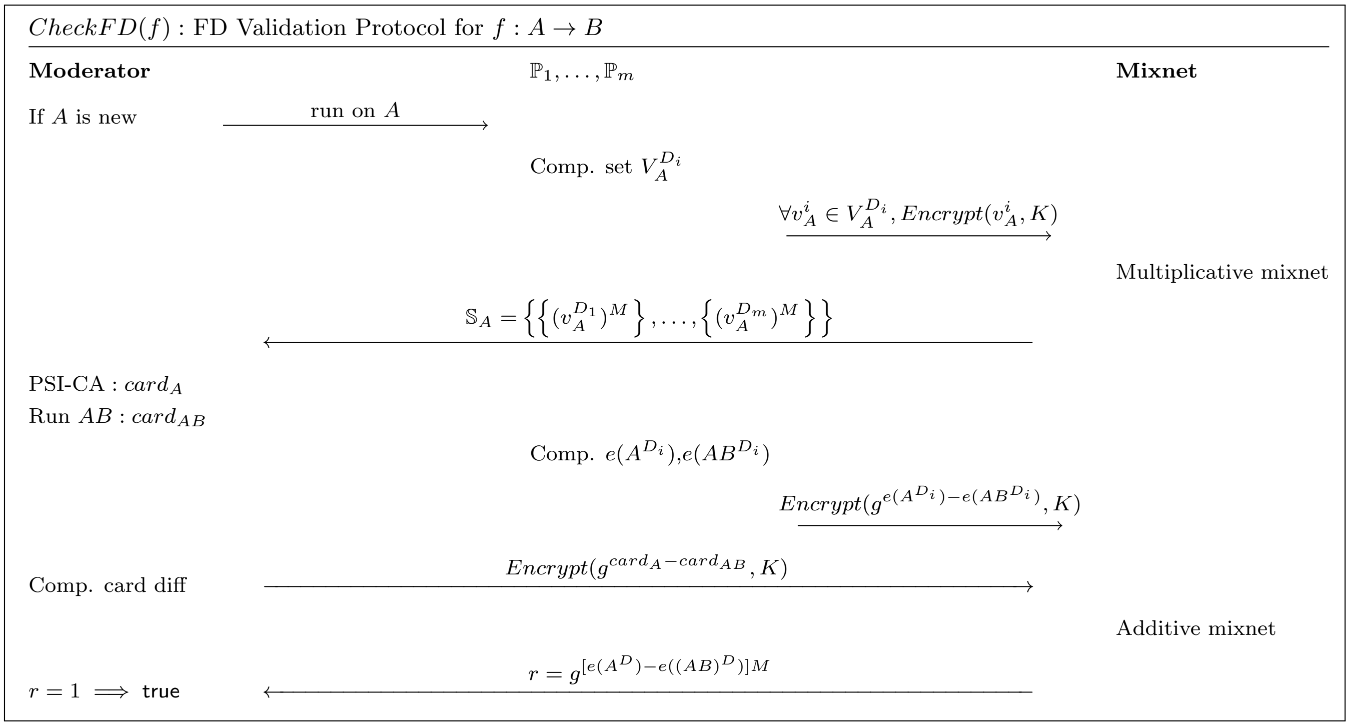
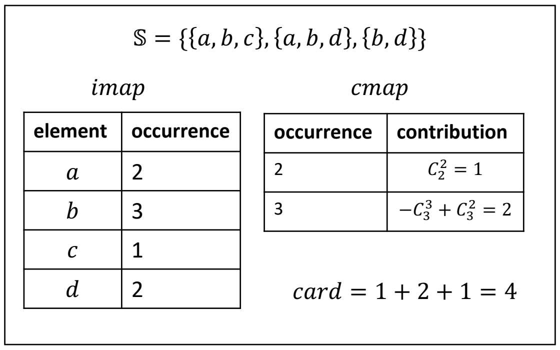

$CheckFD(f)$: FD Validation Protocol for $f : A \rightarrow B$
Moderator
$\mathbb{P}_1, \ldots, \mathbb{P}_m$
Mixnet
If A is new
run on A
Comp. set $V_A^{D_i}$
$\forall v_A^i \in V_A^{D_i}, Encrypt(v_A^i, K)$
Multiplicative mixnet
$\mathbb{S}_A = \left\{\left\{(v_A^{D_1})^M\right\}, \ldots, \left\{(v_A^{D_m})^M\right\}\right\}$
PSI-CA : $card_A$
Run AB : $card_{AB}$
Comp. $e(A^{D_i})$,$e(AB^{D_i})$
$Encrypt(g^{e(A^{D_i})-e(AB^{D_i})}, K)$
Comp. card diff
$Encrypt(g^{card_A - card_{AB}}, K)$
Additive mixnet
$r = g^{[e(A^D)-e((AB)^D)]M}$
$r = 1 \implies$ true

Figure 3: The FD validation protocol for securely validating a candidate FD using mixnets.

the elements only once. We use two simple map structures to keep counting: 1) *imap* stores the mapping of encrypted elements as the keys, and values are the counter of key occurrences; 2) *cmap* stores the occurrence of an encrypted element as the key, and its contribution to the set intersection cardinalities in Equation 3 (Section 4.1).



$\mathbb{S} = \{\{a, b, c\}, \{a, b, d\}, \{b, d\}\}$

imap

element	occurrence
a	2
b	3
c	1
d	2

cmap

occurrence	contribution
2	$C_2^2 = 1$
3	$-C_3^3 + C_3^2 = 2$

$card = 1 + 2 + 1 = 4$

Figure 4: An example of computing PSI-CA.

Figure 4 gives an example to compute PSI-CA using Algorithm 2 with an input set $\mathbb{S}$ consisting of three sets, each of which is a set of encrypted values from a partition. In Section 5, we will explain in details how each value is probabilistically encrypted but still can be deterministically compared for equality. Given the example input of $\mathbb{S}$ in Figure 4, we can easily reason out that the PSI-CA equals to $card = 4$ by Equation 3 (Section 4.1). With Algorithm 2, the process starts with a single scan on each elements in order to build the *imap* structure (Line 3–5). After the traverse, *imap* stores the occurrences for all values from partitions. For example, element a appears twice in $\mathbb{S}$. Then, *imap* is sequentially visited to build the *cmap* (Line 6–23), which stores the contribution of each occurrence to the final PSI-CA. For instance, the occurrence of element b is 3 in S, then it contributes to the overall cardinalities by showing up in one 3-set intersection ($-C_3^3$), and three 2-set intersections (C_3^2). The *cmap* is adaptively built and can be reused. For instance, $(2, 1)$ was inserted when it visited $(a, 2)$ in *imap*, and can be reused when visiting $(d, 2)$. In the meantime of building *cmap*, the final cardinality is iteratively calculated (Line 21): $card = 1 + 2 + 1$.

The *imap* stores the occurrence and its size can be at most the dataset cardinality. Since each element can appear in m parties, the size of *cmap* is at most m. Once m is fixed, the *cmap* can be built offline. Both *imap* and *cmap* are built on the last worker and incur no communication cost. Although the elements of each run are different, the cardinality only depends on the occurrences regardless of the elements. Hence, computing PSI-CA only requires one-pass scan of all encrypted elements to construct the *imap*, while the *cmap* serves as a constant lookup table. Therefore, the complexity of PSI-CA is linear to the number of elements.

5. EQUALITY-AWARE MIXNET

In this section, we propose two versions of mixnets that enable secure equality testing on the input values. The first mixnet (Section 5.1) enables value-level equality check, which allows comparing the equality of input messages without revealing the messages themselves. The second mixnet (Section 5.2) supports comparing the sum of integer set to a given value. We will also prove the security of both mixnets.

5.1 Value-Level Equality Testing

The first mixnet is designed to achieve the following goal: given a set of input values $\{v_1, \ldots, v_m\}$, where v_i is the confidential message from party $\mathbb{P}_i$, and a boolean functionality $f_v(\cdot)$ taking two input values and outputting true if inputs are equal, the mixnet can securely fulfil $f_v(\cdot)$ on all pairs from the input set.

We now introduce the *multiplicative deterministic re-encryption decryption mixnet* using examples. Figure 5 shows one such mixnet with 3 workers labelled from $\mathbb{W}_1$ to $\mathbb{W}_3$ sequentially. There are 2 parties numbered as $\mathbb{P}_1$ and $\mathbb{P}_2$, with input values a' and b' respectively, which are to communicate via the mixnet. The input values are usually hashed into $\mathbb{G}_p$ using a hash function $H(\cdot)$. Initially (step 0.1), each worker generates a pair of keys (described in Section 3.2) and distributes the public keys to all parties and workers. Then (step 0.2), each party fully encrypts its values using all the public keys before sending the encrypted values to the first worker of the mixnet (step 0.3).

Algorithm 3 Fully encrypting a single value

```
Input: $v$                                  ▷ an integer to be encrypted
Input: $K$                                  ▷ the set of all public keys
1: procedure ENCRYPT($v, K$)
2:     $(c_1, c_2) \leftarrow \mathsf{Enc}(c_2, \prod_{k \in K} k)$
3:     return $(c_1, c_2)$
4: end procedure
```

Algorithm 3 describes the encryption process that occurs on each party. All operations are performed in $\mathbb{G}_p$, and for simplicity we omit this in Algorithm 3 and in the rest of the description. The encrypted value is represented as a pair in the form of (c_1, c_2), which is encrypted under the product of the public keys. Intuitively, the encryption process is to encrypt multiply layers where each layer is secured by a key from one of the mixnet worker.

Each worker in the mixnet does the following operations:
0) Initialization (step .0). For each run, every worker generates a fresh secret re-encryption key that will be used for re-encryption in step .2.
1) Decryption (step .1). Since the value was encrypted with the worker's public key, the worker decrypts it using its private key. Consider the encrypted value from $\mathbb{P}_1$ in Figure 5, $\mathbb{W}_1$ uses its private key x and the random variable g^{r_1} and constructs a new random variable $g^{r_1 x}$, which is then used to divide the encrypted value $ag^{r_1 x}g^{r_1 y}g^{r_1 z}$ as a denominator. Hence, the encrypted value is decrypted to $ag^{r_1 y}g^{r_1 z}$.
2) Re-encryption (step .2). The decrypted value is deterministically re-encrypted by the secret re-encryption key. Continue above example, $\mathbb{W}_1$ re-encrypts $ag^{r_1 y}g^{r_1 z}$ by exponentiating with its secret re-encryption key x', to $a^{x'}g^{r_1 x' y}g^{r_1 x' z}$. Similarly, the random value is also updated.
3) Re-randomize the encrypted value (step .3). This is simply realized by freshly encrypting a value of 1 and homomorphically multiplying it to the cipher text.
4) Permute the values, and send to the next worker (step .4). This step adds an additional protection to hide the ordering. Each worker does the same set of operations until the values flow out from the last worker.

Algorithm 4 describes the mixnet operations that occur on the workers. As the output of the mixnet, the input values are permuted and deterministically encrypted by the secret re-encryption keys of all workers. Thus, equality checking is realized by directly comparing the encrypted values. □

Algorithm 4 Mixneting a single value

```
Input: $(c_1, c_2)$                  ▷ cipher text to be re-encrypted
Input: $x$                           ▷ private key of the running party
Input: $x'$   ▷ secret re-encryption key of the running party
1: procedure RE-ENCRYPT($(c_1, c_2)$)
2:     $c_2 = \mathsf{Dec}((c1, c2), x)$                        ▷ decrypt
3:     $(c_1, c_2) = (c_1^{x'}, c_2^{x'})$ ▷ encrypt with re-encryption key
4:     $(c'_1, c'_2) = \mathsf{Enc}(1, \Pi)$     ▷ $\Pi$: product of next pubkeys
5:     $(c_1, c_2) = (c_1 \times c'_1, c_2 \times c'_2)$            ▷ re-randomize
6:     return $(c_1, c_2)$
7: end procedure
```

Let Π_v denote a multiplicative deterministic re-encryption decryption mixnet, we have the following theorem:

THEOREM 1. *Π_v securely implements the functionality $f_v(\cdot)$ in the presence of semi-honest adversary given at least one party and one worker are not controlled by the adversary.*

PROOF. The functionality $f_v(\cdot)$ is fulfilled by comparing the equality of encrypted values, and outputs true or false as the final output of the protocol. Correctness is immediate, so we proceed to the privacy proof. We consider that both the parties and the workers can be controlled by the semi-honest adversary. Let Sim be a simulator, $\mathcal{A}$ be a semi-honest adversary, and the input information regarding to the group be $I = \{\mathbb{G}_p, H(\cdot)\}$. In the ideal world, parties and workers only learn the output of $f_v(\cdot)$. The task of Sim is to output messages that are indistinguishable from actual view of $\mathcal{A}$.

Suppose adversary $\mathcal{A}$ controls party $\mathbb{P}_1$, workers $\mathbb{W}_1$ and $\mathbb{W}_2$ using Figure 5 as the running example. The actual view of $\mathcal{A}$ consists of four messages m_{1-4}. In Figure 5, m_1 and m_2 correspond to the encrypted value $(g^{r_2}, bg^{r_2 x}g^{r_2 y}g^{r_2 z})$ from honest $\mathbb{P}_2$, and m_3 and m_4 represent a value from the permuted, re-encrypted set $\{a^{x'y'z'}, b^{x'y'z'}\}$ from honest worker $\mathbb{W}_3$ (step 3.2), respectively.

For (m_1, m_2), $\mathcal{A}$ can further decrypt m_2 using the private keys of $\mathbb{W}_1$ and $\mathbb{W}_2$, leaving the value only encrypted with $g^{r_2 z}$. Both g^{r_2} and g^z are known to $\mathcal{A}$; however, according to the DDH assumption [8], for the triplet $(g^{r_2}, g^z, g^{r_2 z})$, $\mathcal{A}$ is not able to distinguish $g^{r_2 z}$ from a random element, which is generated by Sim. In addition, for $\{m_3, m_4\}$, given the output of $f_v(\cdot)$, Sim can construct a simulated set $\mathbb{S}$ by drawing two random elements (or, drawing one random element if $f_v(\cdot) =$ false). The adversary $\mathcal{A}$ is not able to distinguish $\mathbb{S}$ from the real set $\{m_3, m_4\}$, except with negligible probability. During the execution, $\mathcal{A}$ only learns input (from colluding $\mathbb{P}_1$) and output of $f_v(\cdot)$, but nothing else. Therefore, the simulator Sim is able to output messages that are indistinguishable given only its input. To formally present the view of adversary $\mathcal{A}$, we have $\{\mathsf{Sim}(I, g^z, a', x, y, x', y', f_v(a', b'))\} \overset{c}{\equiv} \{View_{\mathcal{A}}^{\Pi_v}(I, a', b', x, y, z, x', y', z')\}$.

The other two cases where worker $\mathbb{W}_1$ and worker $\mathbb{W}_2$ respectively remains honest follow a similar simulation algorithm, and hence we omit them here in the proof. □

Let n represent the average number of values per party from a total of m parties, and let w workers form the multiplicative deterministic re-encryption decryption mixnet. The

	Party $\mathbb{P}_1$	Party $\mathbb{P}_2$
0.1, Sync $K\{g^x, g^y, g^z\}$	$value: a = H(a')$	$value: b = H(b')$
0.2, Encrypt$(value, K)$	$(g^{r_1}, ag^{r_1x}g^{r_1y}g^{r_1z})$	$(g^{r_2}, bg^{r_2x}g^{r_2y}g^{r_2z})$
0.3, Send values to mixnet		$\Downarrow$
..........Worker $\mathbb{W}_1$. Keys: (x, g^x)..........		
1.0, Initialize	secret re-encryption key: $x' \leftarrow_\$ \mathbb{Z}_p$	
1.1, Decrypt	$(g^{r_1}, ag^{r_1y}g^{r_1z})$	$(g^{r_2}, bg^{r_2y}g^{r_2z})$
1.2, Re-encrypt	$(g^{r_1x'}, a^{x'}g^{r_1x'(y+z)})$	$(g^{r_2x'}, b^{x'}g^{r_2x'(y+z)})$
1.3, Re-randomize	$s_1^1 \leftarrow_\$ \mathbb{Z}_p, (g^{r_1x'+s_1^1}, a^{x'}g^{(r_1x'+s_1^1)(y+z)})$	$s_1^2 \leftarrow_\$ \mathbb{Z}_p, (g^{r_2x'+s_1^2}, b^{x'}g^{(r_2x'+s_1^2)(y+z)})$
1.4, Permute and send to next		$\Downarrow$
..........Worker $\mathbb{W}_2$. Keys: (y, g^y)..........		
2.0, Initialize	secret re-encryption key: $y' \leftarrow_\$ \mathbb{Z}_p$	
2.1, Decrypt	let $m_1^1 = r_1x' + s_1^1, (g^{m_1^1}, a^{x'}g^{m_1^1z})$	let $m_1^2 = r_2x' + s_1^2, (g^{m_1^2}, b^{x'}g^{m_1^2z})$
2.2, Re-encrypt	$(g^{m_1^1y'}, a^{x'y'}g^{m_1^1y'z})$	$(g^{m_1^2y'}, b^{x'y'g^{m_1^2y'z}})$
2.3, Re-randomize	$s_2^1 \leftarrow_\$ \mathbb{Z}_p, (g^{m_1^1y'+s_2^1}, a^{x'y'}g^{(m_1^1y'+s_2^1)z})$	$s_2^2 \leftarrow_\$ \mathbb{Z}_p, (g^{m_1^2y'+s_2^2}, b^{x'y'}g^{(m_1^2y'+s_2^2)z})$
2.4, Permute and send to next		$\Downarrow$
..........Worker $\mathbb{W}_3$. Keys: (z, g^z)..........		
3.0, Initialize	secret re-encryption key: $z' \leftarrow_\$ \mathbb{Z}_p$	
3.1, Decrypt	let $m_2^1 = m_1^1y' + s_2^1, (g^{m_2^1}, a^{x'y'})$	let $m_2^2 = m_1^2y' + s_2^2, (g^{m_2^2}, b^{x'y'})$
3.2, Re-encrypt	$a^{x'y'z'}$	$b^{x'y'z'}$

Figure 5: A multiplicative, deterministic re-encryption decryption mixnet with 3 workers and 2 parties.

communication cost in the mixnet is $O(nmw)$, and the computation cost is $O(nmw)$ in terms of modular exponentiation.

The proposed multiplicative deterministic re-encryption decryption mixnet provides a powerful protocol for many useful computations over encrypted data. These include: 1) equality comparison directly on encrypted data, which is one fundamental primitive in numerous applications. If two values are equal, then their encrypted values are also equal. We showed this when computing PSI-CA in Section 4.2; 2) set union and intersection cardinalities. For instance, when $\mathbb{P}_1$ and $\mathbb{P}_2$ encrypt a set of values, this re-encryption mixnet can securely compute set union cardinality and intersection cardinality; 3) homomorphic multiplication and division operations. For example, $a^{x'y'z'} \times b^{x'y'z'} = (a \times b)^{x'y'z'}$.

The multiplicative deterministic re-encryption decryption mixnet can be easily extended as well. For example, after the process shown in Figure 5, the workers can initiate a second round of decryption-only process, to decrypt the value $(a \times b)^{x'y'z'}$ for multiplicative operations, or the union/intersected elements for set union/intersection problems. We leave these straightforward extensions to readers for exercise.

5.2 Set-Level Equality Testing

The second mixnet is designed to achieve the following goal: given a set of input integers $\{v_1, \ldots, v_m\}$, where v_i is the confidential integer from party $\mathbb{P}_i$, and a boolean functionality $f_s(\cdot)$ taking input set and outputting true if the sum of inputs is 0, $f_s(\cdot)$ can be securely fulfilled.

We now present the second mixnet, called *additive deterministic re-encryption decryption mixnet* to achieve above set-level equality testing. An additive mixnet shares many properties with multiplicative mixnet, and is more efficient for additive operations.

Figure 6 shows an example of such additive deterministic re-encryption decryption mixnet using three workers and two parties. We highlight the differences from the multiplicative deterministic re-encryption decryption mixnet (Section 5.1): 1) instead of encrypting an integer directly, each party encrypts its value v to g^v; 2) worker $\mathbb{W}_1$ multiplies the encrypted values from parties. This step constructs the sum of input values (on the exponent); 3) each worker goes through an re-encryption process for the additive value, and eventually the last worker gets 1 if the sum equals to 0.

Let Π_s denote an additive deterministic re-encryption decryption mixnet, we can derive the following theorem:

THEOREM 2. *Π_s securely implements the functionality $f_s(\cdot)$ in the presence of semi-honest adversary given at least one party and one worker are not controlled by the adversary.*

PROOF. The last worker fulfils the functionality $f_s(\cdot)$. If the sum of set is zero, then the encrypted value equals to one. In the ideal world, parties and workers only learn the output of $f_s(\cdot)$. Consider a simulator Sim, input information $I = \{\mathbb{G}_p, H(\cdot)\}$ a worst case scenario where the adversary $\mathcal{A}$ controls $\mathbb{P}_1$, workers $\mathbb{W}_1$ and $\mathbb{W}_2$ using Figure 6 as the example. The actual view of $\mathcal{A}$ consists of three messages m_{1-3}. In Figure 6, m_1 and m_2 correspond to the encrypted value $(g^{r_2}, g^b g^{r_2x}g^{r_2y}g^{r_2z})$ from honest $\mathbb{P}_2$, and m_3 corresponds to the re-encrypted value $g^{(a+b)x'y'z'}$ from honest worker $\mathbb{W}_3$ (step 3.2). Similar to the proof for Theorem 1, $\mathcal{A}$ is not able to distinguish (m_1, m_2) because of DDH assumption. As for m_3, given the output of $f_s(\cdot)$, Sim can output a random value

	Party $\mathbb{P}_1$		Party $\mathbb{P}_2$
0.1, Sync $K\{g^x, g^y, g^z\}$	$value : a = H(a')$		$value : b = H(b')$
0.2, Encrypt(g^{value}, K)	$(g^{r_1}, g^a g^{r_1 x} g^{r_1 y} g^{r_1 z})$		$(g^{r_2}, g^b g^{r_2 x} g^{r_2 y} g^{r_2 z})$
0.3, Send values to mixnet		$\Downarrow$	
		Worker $\mathbb{W}_1$. Keys: (x, g^x)	
1.0, Initialize		secret re-encryption key: $x' \leftarrow\$ \mathbb{Z}_p$	
1.1, Multiply		let $r = r_1 + r_2, (g^r, g^{a+b} g^{rx} g^{ry} g^{rz})$	
1.2, Decrypt		$(g^r, g^{a+b} g^{ry} g^{rz})$	
1.3, Re-encrypt		$(g^{rx'}, g^{(a+b)x'} g^{rx'(y+z)})$	
1.4, Re-randomize		$s_1 \leftarrow\$ \mathbb{Z}_p, g^{rx'+s_1}, g^{(a+b)x'} g^{(rx'+s_1)(y+z)})$	
1.5, Send to next		$\Downarrow$	
		Worker $\mathbb{W}_2$. Keys: (y, g^y)	
2.0, Initialize		secret re-encryption key: $y' \leftarrow\$ \mathbb{Z}_p$	
2.1, Decrypt		let $m_1 = rx' + s_1, (g^{m_1}, g^{(a+b)x'} g^{m_1 z})$	
2.2, Re-encrypt		$(g^{m_1 y'}, g^{(a+b)x'y'} g^{m_1 y' z})$	
2.3, Re-randomize		$s_2 \leftarrow\$ \mathbb{Z}_p, (g^{m_1 y' + s_2}, g^{(a+b)x'y'} g^{(m_1 y' + s_2)z})$	
2.4, Send to next		$\Downarrow$	
		Worker $\mathbb{W}_3$. Keys: (z, g^z)	
3.0, Initialize		secret re-encryption key: $z' \leftarrow\$ \mathbb{Z}_p$	
3.1, Decrypt		let $m_2 = m_1 y' + s_2, (g^{m_2}, g^{(a+b)x'y'})$	
3.2, Re-encrypt		$g^{(a+b)x'y'z'}$	

Figure 6: An additive, deterministic re-encryption decryption mixnet with 3 workers and 2 parties.

(or, 1 if $f_s(\cdot) = \mathsf{true}$). The adversary cannot distinguish this from m_3, since x', y', and z' are freshly chosen for each run.

Therefore, the simulator Sim is able to generate messages that are indistinguishable given only its input. To form the view of adversary $\mathcal{A}$, we have $\{\mathsf{Sim}(I, g^z, a', x, y, x', y', f_s(a', b'))\} \overset{c}{\equiv} \{View_{\mathcal{A}}^{\Pi_s}(I, a', b', x, y, z, x', y', z')\}$. $\square$

The additive deterministic re-encryption decryption mixnet has a linear computation and communication cost. Both the computation and communication cost are $O(m+w)$ in terms of modular exponentiation.

The proposed additive deterministic re-encryption decryption mixnet can efficiently empower a category of additive comparisons. These include: 1) value-to-value equality check. To compare whether $a = b$, we can construct a set of $\{a, -b\}$ as the input to the mixnet. If the encrypted output is 1, then it implies $a = b$; 2) value-to-set evaluation. The mixnet provides a way to evaluate the sum of the set to a value t. Consider a set of size n, the input $n+1$ values (n values from the set in addition to $-t$) can be evaluated using the mixnet; 3) set-to-set sum comparison. Comparing the equality of the sum of two sets can be securely implemented by adding all values from one set and all inverted value from the other.

5.3 Parallelizing MixNet

While the mixnets incur linear costs, they can be further scaled out to multiple chains for performance improvement.

Consider two chains $\mathbb{C}_1$ and $\mathbb{C}_2$, each of them consists of w_1 and w_2 workers respectively. w_1 and w_2 are not necessarily to be equal, but for simplicity, we assume two chains are of same length. For the multiplicative deterministic re-encryption decryption mixnet, continuing the example in Figure 5, $\mathbb{C}_1$ consists of 3 workers with secret re-encryption keys x'_1, y'_1 and z'_1 respectively. Let $\mathbb{C}_2$ have 3 workers with secret re-encryption keys x'_2, y'_2 and z'_2. Let $\mathbb{P}_1$ go with $\mathbb{C}_1$, and $\mathbb{P}_2$ go with $\mathbb{C}_2$. The output of the two chains would be $a^{x'_1 y'_1 z'_1}$ and $b^{x'_2 y'_2 z'_2}$. If the exponents are equivalent (i.e., $x'_1 y'_1 z'_1 = x'_2 y'_2 z'_2$), then the equality check of input values does work across different chains. In the following, we describe the pre-processing model to generate secret re-encryption keys.

Consider a trusted oracle $\mathcal{O}$ and a cyclic group $\mathbb{Z}_p$. At the beginning, $\mathcal{O}$ randomly fix a value $M \in \mathbb{Z}_p$. Given w, which is the length of a chain $\mathbb{C}$, $\mathcal{O}$ randomly samples $w-1$ values $s_1, ..., s_{w-1} \in \mathbb{Z}_p$, and sets $s_w = M \times (s_1 \times ... \times s_{w-1})^{-1}$. s_i corresponds to the secret re-encryption key on worker $\mathbb{W}_i$. $\mathcal{O}$ sends s_i to worker $\mathbb{W}_i$ via a secure channel. $\mathcal{O}$ reuses M when it needs to generate a new set of keys. Obviously an value v is always encrypted to v^M regardless of chains.

The above process also works for the additive deterministic re-encryption decryption mixnet. In practice, $\mathcal{O}$ can be easily implemented using secure computations [6].

6. SECURE CONGENIAL FD DISCOVERY

So far, we adopted the traditional definition of FD (Definition 1) in the secure multi-party settings, and required the discovered FDs to hold on the whole dataset. In this section, we relax that requirement; we define f_c as an FD that holds on any of the partition. Note that f_c might not be an valid FD on the whole dataset. For example, in Figure 1, f_2 is valid on both D_1 and D_2, but is not an valid FD on

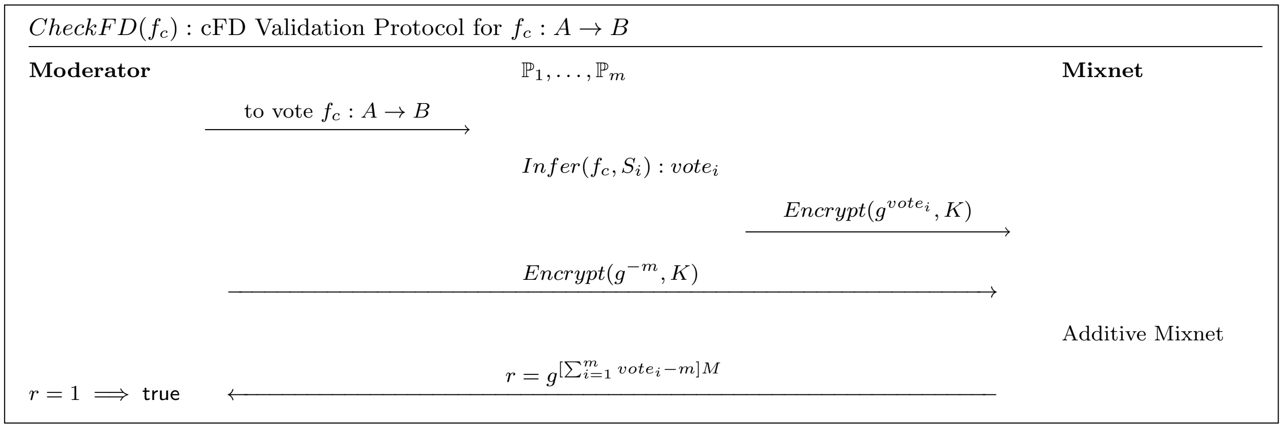

$CheckFD(f_c)$: cFD Validation Protocol for $f_c : A \to B$
Moderator
$\mathbb{P}_1, \ldots, \mathbb{P}_m$
Mixnet
to vote $f_c : A \to B$
$Infer(f_c, S_i) : vote_i$
$Encrypt(g^{vote_i}, K)$
$Encrypt(g^{-m}, K)$
Additive Mixnet
$r = g^{[\sum_{i=1}^{m} vote_i - m]M}$
$r = 1 \implies$ true

Figure 7: The cFD validation protocol for securely validating a candidate cFD using mixnet.

D. Formally, let $\sum_c$ represent the set of all non-trivial and minimal FDs that are valid on every partition $D_i (i \in [1, m])$: $\sum_c \models \hat{S}$. For the purpose of distinction, we call an FD $f \in \sum$ the union FD (uFD, in short), and an FD $f_c \in \sum_c$ the congenial FD (cFD, in short).

To contrast the utility of cFD with uFD, consider the following: after learning the set of uFDs, each party learns two pieces of profiling information: 1) the set of uFDs, which holds globally on the entire dataset; and 2) all the local FDs that are invalid on other partitions, because they were not part of uFDs. On the other hand, by learning the set of cFDs, each party will know: 1) a superset of uFDs; and 2) a subset of local FDs that is invalid on other partitions. Both pieces are useful profiling information (especially the pruned local FDs). Another way to leverage cFD is in pruning the search space of the FD discovery algorithm (Section 2.2): $f \in \sum$ is true either $f \in \sum_c$ or $\exists f_c \in \sum_c$ that $f \in Spec(f_c)$. All the parties do not need to validate a candidate FD f if it has no chance of being an uFD given the set of cFDs (e.g., if the left hand side attributes of f does not intersect with that of any $f_c \in \sum_c$).

6.1 Distributed cFD Validation

In the case of cFD, rather than validating all data across partitions, discovering cFD only requires to validate the *agreement* by all parties. Intuitively, for a given candidate cFD, if every party agrees its validity on its data partition, then the candidate is a valid cFD.

In general, for each candidate cFD $f_c : A \to B$, a naïve way to validate it is that each party $\mathbb{P}_i$ can compute attribute partition errors $e(A^{D_i})$ and $e((AB)^{D_i})$ locally on their data, and determine the validity using Lemma 1 (Section 4.1). But this general approach would require to compute the attribute partitions every time for each validation. In order to avoid such cost, we propose that each party first computes its FD set S_i using its data partition, and then infers the validity of a given candidate cFD f_c using S_i. This approach is inspired by the following property of the cFD:

Property 1. *Given a set of m FD sets $\mathbb{S} = \{S_1, ..., S_m\}$, where S_i is the set of all minimal, non-trivial FDs on partition D_i. A congenial FD f_c on $\mathbb{S}$ satisfies that $\forall S \in \mathbb{S}, \exists f \in S, f_c \in Spec(f)$. A congenial FD f_c is minimal if $\nexists f'_c$ that $f_c \in Spec(f'_c)$ and f'_c is a congenial FD.*

Example: Consider the following FDs in Figure 1:
f_3 : *occupation* $\to$ *income* f_4 : *martial* $\to$ *income*
f_5 : *occupation*, *marital* $\to$ *income*

By examining D_1, $f_3 \in S_1$, and $f_5 \in Spec(f_3)$. Similarly on D_2, $f_4 \in S_2$, and $f_5 \in Spec(f_4)$. Hence, f_5 in a congenial FD. Furthermore, f_5 is also minimal because neither f_3 nor f_4 is an valid congenial FD. ☐

Algorithm 5 cFD inference

```
Input: $f : A \to B$                ▷ candidate cFD to be evaluated
Input: $S$  ▷ set of all minimal non-trivial FDs of the party
1: procedure INFER($f, S$)
2:     for all $A' \to B \in S$ do
3:         $A^* = A \cup A'$
4:         if $A^* \equiv A$ then            ▷ check specialization
5:             return 1
6:         end if
7:     end for
8:     return 0
9: end procedure
```

Algorithm 5 utilizes Property 1 to do inference using a party's FDs only. The idea is to check whether the candidate cFD can be specialized by any of its FDs which share the same right-hand side attribute B. If the candidate cFD can be specialized by all the parties, then the candidate is a true cFD. This leads to the solution for securely verifying candidate cFDs: every party securely votes either 1 if the candidate is a valid FD on its data partition, or 0 if not; and then we can construct an additive deterministic re-encryption decryption mixnet to securely check whether the sum of all votes is equal to the total number of parties.

6.2 A Secure cFD Discovery Protocol

The prerequisite in cFD protocol is to discover the set of all minimal and non-trivial FDs S_i by each party from its data D_i. This can be done locally by every party running an existing FD discovery algorithm, e.g., [17]. Similarly to the uFD discovery protocol in Section 4.2, we arbitrarily choose a moderator to initialize the search space and drive pruning. To prune the search space (Section 2.2), Figure 7 shows such a protocol to implement $CheckFD(f_c)$ for a given candidate

Table 2: End to end comparison between SMFD and the plaintext-based distributed FD validation

				Comput. (Sec)			Comm. (MB)				Comput. (Sec)			Comm. (MB)		
Dataset	# of Col	# of Row	# of uFD	SM FD	Plain Text	Over head	SM FD	Plain Text	Over head	# of cFD[2]	SM FD	Plain Text	Over head	SM FD	Plain Text	Over head
iris	5	150	4	7.6	0.5	15	9.8	0.1	98	4	3.0	0.3	10	0.4	0.01	40
balance-scale	5	625	1	9.0	0.5	18	13.8	0.1	138	1	3.5	0.4	9	0.5	0.01	50
chess	7	28056	1	420.5	2.9	143	1304.9	11.5	113	1	20.4	2.0	10	2.7	0.06	45
abalone	9	4177	137	1819.5	11.6	157	5644.7	104.7	54	159	64.1	6.2	10	8.4	0.19	44
nursery	9	12960	1	532.5	31.5	17	1333.6	65.8	20	9	94.6	9.1	10	12.4	0.29	43
breast-cancer	11	699	46	1508.1	98.8	15	3190.6	35.4	90	77	421.0	69.5	6	55.0	1.36	40
bridges	13	108	142	1726.8	197.8	9	2130.0	47.2	45	127	753.0	130.0	6	98.6	2.53	39
echocardiogram	13	132	538	1342.3	137.0	10	1958.0	30.5	64	420	506.2	92.3	5	66.2	1.70	39

cFD f_c. First, the moderator instructs all parties to vote either 1 if the candidate is valid on their data, or 0 otherwise. Each party runs Algorithm 5 on the candidate f_c and their own FD set S_i, and sends its encrypted vote to an additive deterministic re-encryption decryption mixnet (Section 5.2). In addition, the encrypted inverse of the number of parties (i.e., $-m$) is also sent as an input. The output r of the mixnet is an encrypted value, indicating whether all parties voted 1 or not. If $r = 1$, then f_c is a true cFD.

7. EVALUATION

7.1 Experiment Setup

Datasets. We choose 8 datasets from the UCI ML repository [11]. These datasets have different number of columns, rows and FDs. The details are listed in Table 2.

To simulate the data partitions that are held on parties, we horizontally chunk the datasets into m disjoint and even parts where m is the number of parties. For reproducibility purposes, a scale factor SF is used to represent the number of chunks that each partition has. Every partition is composed of SF number of consecutive chunks in a round-robin sequence. By default, SF is 1 for all datasets. Although data size is evenly distributed, the values of attribute partitions (used in uFD discovery) and local FDs (used in cFD discovery) can have skewness.

Implementation details. We conducted all experiments on AWS platform. Each party or worker was deployed on an EC2 instance locating in spread networks across us-west regions. Each instance has 72 vCPU and 144 GB memory. The solutions are implemented in C++ with gcc-5.4.0, and run on the Ubuntu-14.04.

Security parameters are fixed as follows: both the length of generator g in ElGamal and the length of keys are set to be 256 bits. p has a length of 3072 bits.

In additional, we also compared our solution (tagged by *SMFD*) with distributed FD validation using two general purpose MPC protocols: 1) *MultipartyPSI* [20], for multi-party private set intersection. We integrate their code[3] by replacing the mixnet with set intersections among parties. Specifically, for uFD, we compute the set intersections in Equation 3. Due to the limitation that MultipartyPSI requires all parties have same-sized sets, we assign each party to own the full dataset to avoid the overhead of padding. cFD protocol is implemented by intersecting votes of all parties. 2) *SMCQL* [5], for secure SQL querying over the union of its source databases, and serves as a general approach to discover uFDs. Due to the limitation of the current implementation[4], we compute the attribute partition errors $e(A)$ and $e(AB)$ by issuing two SQL queries to select `count(distinct A)` and `count(distinct AB)`, respectively. SMCQL was tested with two databases on the same host.

Metrics. We measure two end-to-end costs: 1) computation cost, measured by the overall FD discovery time; 2) communication cost, counted by the bytes which are transferred through all the nodes. Every setting reports the mean and standard error of 3 runs.

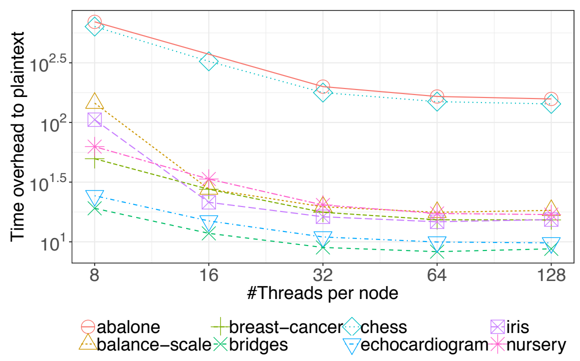


Figure 8: uFD computation overhead to the plaintext decreases with more threads per instance.

7.2 Overhead of SMFD

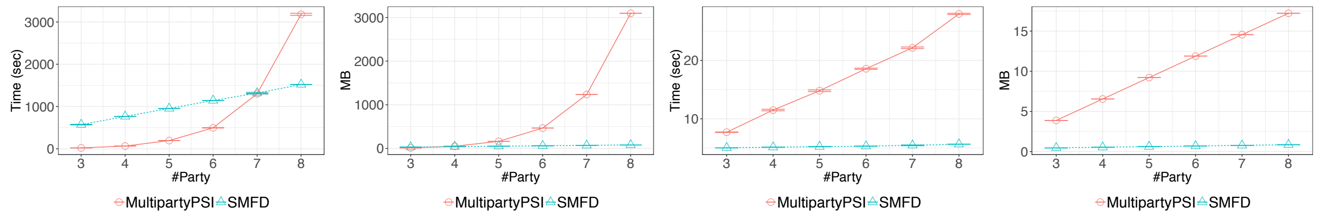
Table 2 shows the end to end cost comparison between the proposed SMFD and the non-secure plaintext-based distributed FD validation, in a setting of 3 parties, 3 workers, where each party or worker node uses 128 threads for public key operations. The security overhead using all datasets is often within one order of magnitude for both uFD and cFD discoveries, for both computation and communication costs.

The computation overhead stems from public key operations, and the communication overhead is due to ciphertext expansion. As Table 2 shows, FD discovery costs are highly data dependent. The overall cost is dominated by two factors: 1) the cost of validating one FD, and 2) the number of FDs that require validation. The first factor is determined by the size of attribute partition values for uFD, which usually but not necessarily increases with the number of rows; while for cFD, the cost is independent because we process only votes from parties. Meanwhile, the second factor is determined by

[2]based on the partition strategy in Section 7.1.

[3]`https://github.com/osu-crypto/MultipartyPSI`

[4]`https://github.com/smcql/smcql`



(a) Computation Cost uFD (b) Communication Cost uFD (c) Computation Cost cFD (d) Communication Cost cFD

Figure 9: Cost comparison between SMFD and distributed FD discovery using MultipartyPSI, with increasing number of parties on the balance-scale dataset: a) and b) show the computation and communication cost for uFD, respectively. SMFD has a linearly cost, while MultipartyPSI incurs an exponential cost. c) and d) show the costs for cFD. MultipartyPSI requires larger costs and is more sensitive to the number of parties.

the number of columns and FDs. Larger number of columns lead to more complex lattice with more FD candidates; while a true FD can prune out all its specialization from the lattice.

The overhead of discovering uFD is usually heavier than cFD, because in uFD discovery, the values of attribute partitions pass through the mixnet, and number of values can be large. Therefore, the computation cost of uFD can be reduced by using multiple threads, where multiple threads can work on values in parallel.

To show the benefit of parallel processing, Figure 8 illustrates the uFD computation overhead using different threads per party and worker node. Using more threads reduces the overhead, implying that our solution can benefit more from modern hardware. Even using 8 threads, the overhead is still often within one order of magnitude. The overhead can be further reduced by scaling out to multiple mixnet chains (Section 5.3) and here we omit because of similar behaviour. In the rest of experiments, we show results using one thread per node and one mixnet chain.

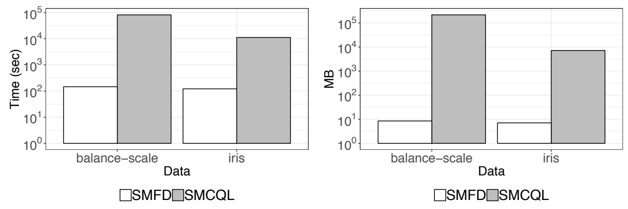
7.3 Efficiency of SMFD

Comparison to MultipartyPSI. We compare SMFD with distributed FD vaidation using MultipartyPSI [20]. Note that MultipartyPSI has a weaker security model since parties learn the intersected values.

Figure 9a and Figure 9b illustrate the costs for discovering uFD. Recall Section 4.1 that for each uFD validation, there are $2^m - m - 1$ intersections for m parties. Although one set intersection using MultipartyPSI is cheaper because of the use of symmetric key cryptography instead of public key cryptography, the exponential number of sets overwhelms the overall performance. For example, in Figure 9a, when the number of parties is small, MultipartyPSI runs faster. With more parties, the benefit of faster processing per intersection is soon diminished by the exponential number of intersections. When there are 8 or more parties, SMFD starts to outperform MultipartyPSI. In contrast, SMFD costs increase since more parties are sending data to the mixnet; however, computing PSI-CA only incurs linear complexity (Section 4.2).

Figure 9c and Figure 9d show the costs of discovering uFD. MultipartyPSI requires larger costs than SMFD and is more sensitive to the number of parties.

Comparison to SMCQL. Figure 10a compares the execution time of discovering uFDs. On balance-scale dataset, SMCQL takes more than 22 hours to finish validating all uFDs, while SMFD completes in 146 seconds. On iris dataset,



(a) Computation Cost (b) Communication Cost

Figure 10: Cost comparison between SMFD and SMCQL.

SMCQL takes 3.2 hours, and SMFD finishes in 121 seconds. On both datasets, SMFD is two orders of magnitude faster.

Figure 10b shows the comparison of communication cost. SMCQL intensively inserts dummy tuples to pad intermediate results during its execution, so it is expected that communication cost is expensive. On balance-scale dataset, SMFD costs 8MB, while SMCQL transfers more than 200GB. On iris dataset, SMFD consumes 7MB, outperforming SMCQL's 7178MB by three orders of magnitude.

Note that SMCQL was tested on the same host, and the difference of computation cost in distributed scenario is expected to be amplified by its communication cost.

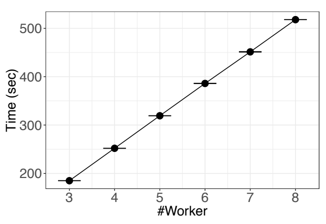
7.4 Scalability of SMFD

Scalability of varying the number of workers. Figure 11 shows the costs of varying the number of workers. The costs of computation and communication increase linearly with the number of workers, for both types of FDs.

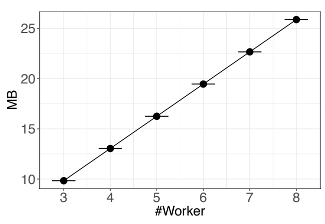
Adding more workers does not affect FD candidates, but only affects the performance of mixnet. Recall that in a mixnet chain (Section 5), all workers collaborate in a sequential way, and they share the same type of job. Hence, the cost increases linearly with the number of workers.

Scalability of varying data volume. Figure 12 measures the cost of changing data volume per party. Figure 12a and Figure 12b show that costs increase with SF for discovering uFDs. Generally, the larger data volume on each party, the larger the attribute partition set is expected. Therefore, both the execution time and transferred bytes increase.

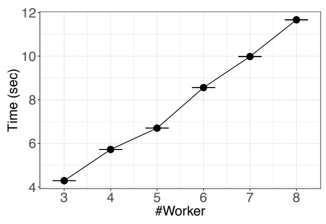
However, uFD highly correlates with the FDs that are discovered by each party, and the size of candidate uFD is not static. When SF is small, increasing the data volume per party invalidates more uFDs, and consequently triggers more inference (Algorithm 5). When $SF > 4$, increasing data volume per party does not provide more insights, and



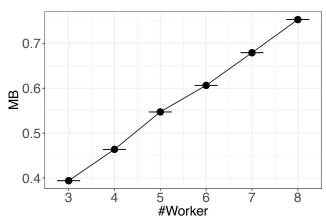
(a) Computation Cost uFD



(b) Communication Cost uFD

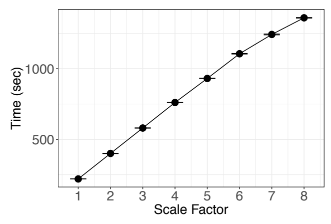


(c) Computation Cost cFD

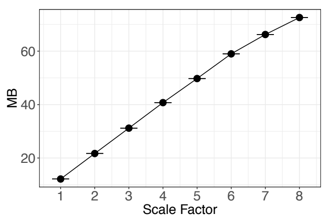


(d) Communication Cost cFD

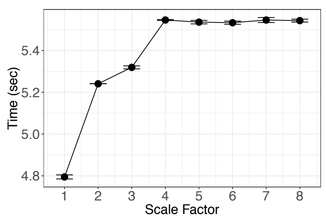
Figure 11: Cost of varying the number of workers using 3 parties on the iris dataset.



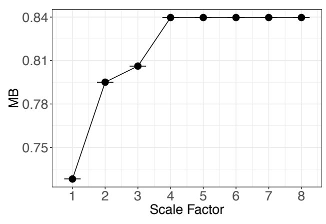
(a) Computation Cost uFD



(b) Communication Cost uFD



(c) Computation Cost cFD



(d) Communication Cost cFD

Figure 12: Cost of varying the data volume using 8 parties and 3 workers on the iris dataset.

hence it is expected the cost remains the same. Figure 12c and Figure 12d illustrate such trend.

8. RELATED WORK

There has been extensive literature on discovering functional dependencies [23]. These approaches can be categorized into three classes: 1) top-down schema driven approaches which generate candidate FDs first and then remove invalid FDs [3, 17, 28, 39]; 2) bottom-up data driven algorithms which compare the data to find agree or difference sets and induct FDs from observation [14, 24, 37]; 3) a mix of both [31]. This work belongs to the first class of approach. Traditional practice [30] generally assumes a single dataset without considering distributed scenarios. Distributed FastFD [36] is to the best of our knowledge the only effort considering FD discovery over partitions, which adopts a bottom-up data driven approach and is orthogonal to our work. Distributed FastFD focuses on minimizing communication cost by computing a full self-join of the dataset and computing evidence from all tuple pairs, and is expensive to implement in the secure multi-party scenario.

Our FD validation protocols relies on mix networks to securely re-encrypt and decrypt inputs. A mixnet provides anonymity for a batch of inputs, by changing their appearance and shuffling the order. Based on the cryptographic transformations that workers do, a recent survey [35] classifies mixnets into different types such as decryption mixnet [18], and re-encryption mixnet [33] to fit different applications. Our proposed mixnets extend the literature by hybriding the decryption and re-encryption operations per mix on each worker using the ElGamal encryption [15].

Our FD validation and discovery protocols also depend on set operations, especially set intersection cardinality and sum of a set of integers. There exists a class of research on solving private set intersection e.g., [19, 20] and cardinalities e.g., [10]. To validate one FD requires set intersections for all subset of parties, and using any private set intersection techniques would require $O(2^m)$ intersections given m parties. However, our approach computes the set intersection cardinality for all subsets in $O(m)$, which is optimal, since a sub-linear solution cannot exist. In addition, using private set intersection would leak more information, since each party will learn the values in intersection. Our approach only outputs the set intersection cardinality for all subsets.

This work is closely related to the large body of literature on secure multi-party computation [4, 22]. We follow the same model to define security and simulate the proofs. There exists extensive research on studying general purpose protocols [38, 29, 27], and from that, building complex protocols for privacy-preserving data mining [21]. It is not clear how to use these protocols to efficiently compute PSI-CA in $O(m)$. Note that our mixnets use shuffling—a technique that requires at least a circuit of size $n \log n$—in $O(n)$ public key operations and compare n elements from a large domain to each other, but as plaintext operations on deterministically encrypted ciphertexts. Using generic secure computation protocols, these operations would have to be performed on secret shared data.

Recent work on generic query answering provides a practical way to answer SQL queries securely. SMCQL [5] evaluates queries obliviously using ORAM [16], and exhaustively pads dummy values for each query operator in the query plan. Due to operator cascading, padding values leads to significant output size and hence, loss in query performance. SMCQL is a generic system, while our work is more specialized for the operations in FD discoveries.

9. CONCLUSION

This paper focuses on discovering FDs in the secure multi-party scenario against semi-honest adversaries. We formulate discovering FDs, design secure constructions for FD validation, and present efficient protocols to enable secure multi-party FD discovery. Experimental results show that solution is practically efficient over non-secure distributed FD discovery, and can significantly outperform general purpose multi-party computation frameworks.

10. REFERENCES

[1] IBM infosphere master data management. `https://www.ibm.com/ca-en/marketplace/ibm-infosphere-master-data-management`.

[2] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). *OJ*, 2016-04-27.

[3] Z. Abedjan, P. Schulze, and F. Naumann. DFD: efficient functional dependency discovery. In *CIKM*, pages 949–958, 2014.

[4] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright. From keys to databases - real-world applications of secure multi-party computation. *Comput. J.*, 61(12):1749–1771, 2018.

[5] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers. SMCQL: secure query processing for private data networks. *PVLDB*, 10(6):673–684, 2017.

[6] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[7] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *PVLDB*, 3(1-2):197–207, Sept. 2010.

[8] D. Boneh. The decision diffie-hellman problem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, pages 48–63, 1998.

[9] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[10] E. D. Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *CANS*, pages 218–231, 2012.

[11] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[13] M. Eich, P. Fender, and G. Moerkotte. Faster plan generation through consideration of functional dependencies and keys. *PVLDB*, 9(10):756–767, 2016.

[14] P. A. Flach and I. Savnik. Database dependency discovery: A machine learning approach. *AI Commun.*, 12(3):139–160, Aug. 1999.

[15] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

[16] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *STOC*, pages 182–194, 1987.

[17] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[18] A. Jerichow, J. Müller, A. Pfitzmann, B. Pfitzmann, and M. Waidner. Real-time mixes: a bandwidth - efficient anonymity protocol. *IEEE Journal on Selected Areas in Communications*, 16(4):495–509, 1998.

[19] L. Kissner and D. X. Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[20] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *CCS*, pages 1257–1272, 2017.

[21] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

[22] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *IACR Cryptology ePrint Archive*, 2008:197, 2008.

[23] J. Liu, J. Li, C. Liu, and Y. Chen. Discover dependencies from data - A review. *IEEE Trans. Knowl. Data Eng.*, 24(2):251–264, 2012.

[24] S. Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *EDBT*, 2000.

[25] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, 2001.

[26] K. Morton, M. Balazinska, D. Grossman, and J. D. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *PVLDB*, 7(6):453–456, 2014.

[27] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.

[28] N. Novelli and R. Cicchetti. FUN: an efficient algorithm for mining functional and embedded dependencies. In *ICDT*, pages 189–203, 2001.

[29] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[30] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome. *PVLDB*, 2015.

[31] T. Papenbrock and F. Naumann. A hybrid approach to functional dependency discovery. In *SIGMOD*, 2016.

[32] T. Papenbrock and F. Naumann. Data-driven schema normalization. In *EDBT*, pages 342–353, 2017.

[33] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*, pages 248–259, 1993.

[34] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[35] K. Sampigethaya and R. Poovendran. A survey on mix networks and their secure applications. *Proceedings of the IEEE*, 94(12):2142–2181, 2006.

[36] H. Saxena, L. Golab, and I. F. Ilyas. Distributed discovery of functional dependencies. In *ICDE*, pages 1590–1593, 2019.

[37] C. Wyss, C. Giannella, and E. L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *DaWaK*, 2001.

[38] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[39] H. Yao, H. J. Hamilton, and C. J. Butz. Fdmine: discovering functional dependencies in a database using equivalences. In *ICDM*, pages 729–732, 2002.