ReLiShare: Reliable Leaker Identification in Sensitive Dataset Sharing

***Abstract*—A wide spectrum of applications share data online today, bringing the need for reliably identifying responsible parties whenever a leaking of the shared data is detected. Existing solutions to this problem assume that there is a clear understanding on the dataset each sharing party receives, and a single culprit is responsible for each data leaking (i.e., no collusion), which must not be the data owner. Unfortunately, multiple evidences show that these assumptions may not always hold true. In this paper, we present ReLiShare, a data sharing system that removes the above assumptions. Our key idea is to share data via Oblivious Transfer, which prevents anyone, including the data owner, from knowing the exact dataset shared with a specific party. We further make use of a Merkle Tree based credential to record the resulting shared dataset, and develop a novel algorithm to identify leakers based on their knowledge of data objects. As a result, ReLiShare enables oblivious data sharing in a non-repudiable way, and, in the event of a data leakage, can reliably identify the culprits independent from whether it is a single party or a set of colluding parties, without excluding the data owner itself. The results of our experiments with real world datasets demonstrate the efficiency of ReLiShare's sharing system and the effectiveness of ReLiShare's identification. It only takes about one minute to share a dataset containing more than 2 million data objects to a receiver. In different simulated data leakage events, ReLiShare can accurately (with accuracy$>$99.99%) identify all guilty parties.**

I. Introduction

As the world becomes increasingly online, a wide spectrum of applications need to share datasets containing sensitive information (e.g., Personally Identifiable Information (PII) [1]) with multiple parties. Examples include (i) public Medicare and Medicaid Services (CMS) share data that contains people's identifiable information, including addresses with laboratories and organizations for research purposes across the country [2]. (ii) More than 60% of hospitals in the US share patient information with different health care providers [3] to benefit patients, e.g., avoiding duplicate tests. (iii) It is also increasingly common that schools or employers share student/employee information with online educational services for specialized training.

To address privacy concerns, many efforts have been devoted to preserving data confidentiality through fine-grained access control, or secured data query from untrusted platforms (e.g., cloud servers). However, most, if not all, of these systems entrust the authorized parties with keeping data confidentiality, a trust that has seen increasing violations recently [4]–[8]. For example, in 2019, a data breach affecting approximately 4.9 million users was caused by the leakage from third party services [6]. As required by new regulations and laws such as GDPR [9] and CCPA [10], whenever a data leakage occurs, the leaker(s) must be reliably identified.

The leaker identification (also called traitor identification or guilty agent detection) problem can be defined as follows. A data owner (called a sender) sharing a set of data objects with data users (called receivers), where the parties on both sides agree that the shared data shall not be disclosed to any third party. If any, or all, of the shared data objects are found in an unauthorized place (*e.g.,* the public Internet, personal laptop), the leaking source(s) should be identified.

A number of solutions [11]–[23] have been proposed to identify leakers in dataset sharing, however several critical challenges remain unsolved.

1) First, most of them assume a *trusted data owner* who would never leak data. The solutions with this assumption cannot detect leakage from the data owner, for example, by its employees intentionally or unintentionally [4], [5], [8] or due to vulnerable firewalls [7]. Worse yet, since the data owner knows the exact dataset shared with each receiver, it could frame a data receiver by intentionally leaking a dataset that is unique to an innocent receiver.
2) Second, most existing approaches cannot handle leakage by a set of *colluding* parties. Through collusion, multiple leakers can figure out the distinction in datasets by comparing each one's received data, then erase the distinction, for example, by only leaking common data objects [24].
3) Third, non-repudiation of multiparty data sharing remains a challenge. In the absence of immutable evidence showing which receiver possesses which dataset, a leaker may deny its possession of certain data pieces.

In this paper, we propose ReLiShare, a data sharing system that overcome the above shortages in data leaker identification. Compared with the existing results, ReLiShare provides non-repudiation of data possessions of every party in a data sharing agreement; and given a data leakage, ReLiShare can reliably identify the responsible party, or a responsible set of colluding parties, even in the case of the data owner being the guilty party or among them. More specifically, the ReLiShare design includes the following three basic approaches.

- *Oblivious Data Sharing.* The sender shares data objects through Oblivious Transfer (OT) [25] with each receiver, so that the sender does not know the exact set of data objects is transferred, and the receiver does not know the the complete set of data objects sent by the sender.
- *Non-repudiable Receipt based on Merkle Tree.* During the data sharing process, ReLiShare immutably records the result of the data sharing with Merkle Trees and generates a digital credential, preventing any party from lying or

denying what data objects have been transferred/received. This step involves the use of an honest third party, called a notary.

- *A Knowledge based Identification Algorithm.* We propose a novel identification algorithm to identify leakers by inferring the knowledge of the leaker(s) from the leaked data objects using a statistical method called the binomial test. With this identification algorithm, ReLiShare can effectively pinpoint guilty parties, including (i) a sender, (ii) non-collusive receivers, or (iii) collusive receivers, with a false positive rate and/or a false negative rate.

Our approach follows the direction of utilizing data allocation strategies [19], [20] to identify leakers, with the limitation that no receiver will be able to use the entire dataset. ReLiShare offers two options for different application scenarios.

1) Sharing real data objects only. This works when data receivers do not need the entire dataset, or the use of synthetic data objects is not acceptable.
2) Sharing both real and synthetic data objects. This is suitable for application systems where the inclusion of synthetic data has negligible negative impact.

Although the use of synthetic objects have been widely adopted in leaker identification solutions [19], [20], [26], [27], how to generate synthetic data objects was not addressed by previous work. In §V-C, we analyze the challenges of generating synthetic data and identify a Generative Adversarial Network (GAN) based approach, called CTGAN [28], which can serve as a generic approach for synthetic data generation. Since the evaluation results in the original CTGAN work are more about machine learning efficiency, to evaluate the quality of generated synthetic objects for leaker identification, we utilize t-Distributed Stochastic Neighbor Embedding (t-SNE) [29] and conduct experiments with real world datasets in §VIII-E to show its effectiveness.

We implement a prototype of ReLiShare sharing system in C++ and the knowledge-based leaker identification algorithm in Python [1]. Our experiments using real world datasets show the efficiency and effectiveness of ReLiShare. With 1-out-of-2 OT extension [30] and the HospitalCharge dataset [31], it only takes about one minute for a sender to share more than 2 million data objects to a receiver. The computation and network overhead for generating Merkle Tree based credentials are also practical to be deployed. The identification evaluation results show that ReLiShare can provide $> 99.99\%$ identification accuracy in different leakage scenarios with a relatively small number of data for allocation.

Contributions: While OT and Merkle Tree have been used in solving many problems, to our knowledge, using them in addressing the challenges of data sharing and leaker identification has not been tried before. To the best of our knowledge, ReLiShare is the first leaker identification mechanism capable of identifying a guilty sender and colluding receivers in dataset sharing. In addition, we provide an approach to immutably record the data sharing results in the granularity of data entry. Furthermore, we formally analyze our identification algorithm and formulate its false positive rate and/or false negative rate. Last but not least, we analyze the challenges in synthetic data generation, identify a GAN-based model which can also be adopted by existing and future works based on data allocation strategy, and evaluate its effectiveness.

Application Scenarios and Limitations: ReLiShare makes use of differentiated data allocation and provides two options for different application scenarios. However, it does not apply to application systems where data receivers must receive the complete dataset and do not tolerate synthetic data objects at the same time.

[1]The source code for ReLiShare's sharing system and identification algorithm will be released with the final version of the paper.

II. Preliminaries

We present a brief description of oblivious transfer and Merkle Tree data structure that are used in ReLiShare, and ReLiShare's cryptographic assumptions.

A. Oblivious Transfer

An OT protocol is a cryptographic protocol in which the sender transfers many pieces of information to a receiver but remains oblivious to which piece has been received. At the same time, the receiver learns nothing but the information it selected. In a classic 1-out-of-2 OT, the sender inputs two messages m_0 and m_1 while the receiver has a choice bit $c \in \{0, 1\}$. The receiver will learn m_c while c remains unknown to the sender at the end of the OT.

ReLiShare uses 1-out-of-2 OT as a black box and thus does not specify which construction of OT should be used. Since the size of data objects in a dataset can be very large, a large number of OTs may be involved in ReLiShare. For the sake of simplicity, in the rest of the paper, we denote n-time OT operations transferring a pair of data objects by

$$d_c = \text{1-2-OT}(d_0, d_1, c), \quad c = \{0, 1\}$$

where d_0 and d_1 are data objects sent by the sender, c is the choice made by the receiver, and d_c is the object received by the receiver. In our prototype implementation and real-world deployment, OT extension mechanisms [30] will be used for better performance.

B. Merkle Tree

A Merkle Tree is a tree data structure, where each leaf node contains the hash value of a message and each non-leaf node has the hash value of its child nodes concatenated together. For example, a parent node of two leaf node $h(m_0)$ and $h(m_1)$ has a value of $h(h(m_0)||h(m_1))$ where m_0 and m_1 represents plain text messages. By hashing all the data objects in a dataset into leaf nodes and constructing a Merkle Tree, the root value of the tree can uniquely identify this dataset. In addition, the one-way cryptographic hash function prevents one from learning the original data objects by parsing leaf or non-leaf nodes.

C. Cryptographic Assumptions in ReLiShare

ReLiShare inherits the cryptographic assumptions from underlying cryptographic schemes. The sender's agnostic of each receiver's dataset relies on the property of OT that prevents the sender from knowing which object is selected by the receiver. After the dataset transfer, a third party called a Notary uses its private key to sign the Merkle Trees of the data being shared with the receiver. The Notary cannot learn the original dataset in plaintext because of the underlying one-way cryptographic hash function used in Merkle Tree. Therefore, ReLiShare shares the cryptographic assumptions from the selected 1-out-of-2 OT construction, the one-way hash function used in Merkle Tree, and the digital signature scheme used in the credential generation.

III. System Model and Design Goals

There are four parties in our data sharing system. They are (i) a data sender who owns the original dataset, (ii) some receivers with whom the dataset will be shared, (iii) a notary who will record the sharing process with a credential, and (iv) an arbitrator who will investigate the leakage and identify leakers in the event of a data leakage.

Specifically, we let a data sender own a dataset $D = \{d_1, d_2, ..., d_m\}$ of data objects. Each object d is data that can be identified by a key (e.g., an explicit primary key in a rational or key-value database or an implicit key which is a combination of attributes) and can be in any data type (e.g., rows in database, image files). When sharing data with a receiver, a data object is the smallest unit in the dataset; for example, if an object contains more than one row in the database, these rows will be sent as a whole. The sender will share the dataset D with a number of receivers $R_1, R_2, ..., R_n$, in which both the sender and receivers agree that the data should never be leaked to unauthorized parties.

By adjusting the scope of D, the model can easily be extended to scenarios where only a subset of data is shared.

A. Security Consideration

In this work, we treat all unintentional data leakage (e.g., leakage caused by vulnerable firewall) as intended operations from an adversary.

Sender and receivers: We consider the sender and receiver to be *conditionally malicous*, namely, they will follow ReLiShare's protocol and finish the data sharing successfully, but when data is leaked, the leaker(s) will try to escape the leaker identification. This is reasonable in a real-world setting because at the phase of data sharing, the sender and receivers trust each other otherwise the sharing will not take place in the first place. However, if one party or a group of parties leak the data, intentionally or unintentionally, they will try to conceal their guilt [2]. We consider a receiver may collude with other parties so as to circumvent a later leaker identification process. Specifically, we assume colluding receivers can (i) apply a series of set operations (*i.e.,* union, intersection, or complement) to datasets from different receivers and (ii) select a subset of data objects from a processed dataset.

Notary: In our work, we employ an honest-but-curious Notary in the data sharing process to certify that a dataset has been shared with a receiver R_i. In this process, the notary is not supposed to learn any data objects in plaintext.

Arbitrator: We assume an honest arbitrator because (i) when a data leakage happens, the dataset is already revealed and (ii) an arbitrator is usually from an authorized agent (*e.g.,* court or government agencies).

B. Goal and Target Properties for ReLiShare

The primary goal of ReLiShare is to identify leakers in the event of a data leakage after sharing. To be more specific, ReLiShare should be able to identify a guilty sender, colluding receivers, and non-colluding receivers.

Note that it is not a goal to identify a party when this party does not contribute to a collusive leakage. By saying a party did not contribute, we mean that other corrupted parties can still have sufficient knowledge to make such a leaked dataset even without this receiver's participation. We argue that it is almost impossible to detect such a party because there is no "footprint" left.

At the same time, ReLiShare aims to provide the following properties at the same time.

Non-repudiation: A data sharing process between a sender and receiver cannot be denied afterward.

Non-frameability: A leaker, regardless of whether it is the sender or a receiver, cannot escape the leaker identification by framing any innocent party.

Data Confidentiality: Before a data leakage happens, the dataset should only be known to the sender and receivers.

Efficiency: In order to be used in practice, ReLiShare must be efficient, causing acceptable computation and network overhead to system participants.

IV. Overview of ReLiShare

The high-level idea of ReLiShare is to identify leakers based on system participants' different knowledge of the dataset: In the data sharing process, ReLiShare will let every party, including the sender, have a different view of the dataset. Importantly, each party's knowledge is unknown to others unless there is collusion. In addition, ReLiShare will immutably record the data sharing process to ensure non-repudiation of the identification results because leakers cannot deny or lie about their knowledge of the dataset.

To meet the target properties of ReLiShare, we propose three main technical approaches (Figure 1).

- **Oblivious Data Sharing:** For each receiver, the sender shares a randomly-permuted dataset to the receiver with 1-out-of-2 OT and the receiver will randomly generate the choices used in OT (❶). By properly preparing the

[2] In this work, we do not consider the case where the sender colludes with all receivers.

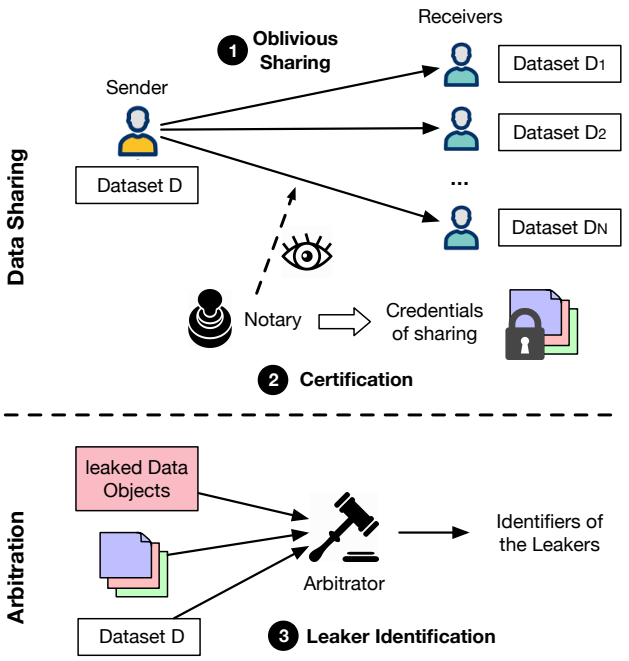


Figure 1: An Overview of ReLiShare

OT messages, each receiver will receive a distinct dataset unknown to the sender and other receivers.

- **Indisputable Receipt based on Merkle Tree of data sharing:** Each data sharing process will be proved immutably by a Merkle-Tree based credential generated by a Notary (❷). Such a credential logs the data objects that have been sent and received, preventing any party from denying or lying about the data allocation. In this process, the Notary cannot access the plaintext dataset because all data objects will be hashed into the Merkle Tree.
- **Knowledge-based Leaker Identification:** In the event of a data leakage, by reversely inferring leakers' knowledge from the leaked data using binomial test, ReLiShare can identify all leakers with high accuracy (❸).

After the sharing process, by employing OT, each received copy is a different collection from all others; even a combination of several received datasets is also distinguishable from other combinations. Importantly, no single party is fully aware of the allocation among the receivers. In addition, with credentials published by the Notary, no party is able to cheat in a later arbitration. Due to these properties, in the case of a data leakage, the knowledge-based identification algorithm is able to effectively and indisputably identify all leaking sources.

To differentiate the knowledge of a dataset among the sender and all receivers, in ReLiShare, some data objects are not commonly shared, namely dropped during the sharing process. This is intended and usually inevitable because the goal is to add difference to each received dataset by doing data object distribution [19], [20], which can potentially limit the use cases. To allow a wider adoption, ReLiShare provides two options for the sender: the sender can (i) sacrifice the completeness of real data objects, i.e., each receiver only receives a subset of the genuine dataset, or (ii) add synthetic data objects into the dataset so that each receiver can receive all genuine data objects.

In the rest of the paper, we first describe the oblivious data sharing process of ReLiShare in §V. Then we explain how Notary functions in ReLiShare and utilizes Merkle Tree to generate credential of sharing in §VI. In §VII, we introduce ReLiShare's knowledge based leaker identification algorithm. We present the evaluation results of ReLiShare in §VIII and compare ReLiShare with related works in §IX.

V. Oblivious Data Sharing

ReLiShare's oblivious data sharing allocates sender's data objects among receivers while each received dataset is only known to its receiver. Specifically, in §V-A, we discuss the two choices in preparing data objects used for allocation. In §V-B, we explain how OT based sharing works in ReLiShare. In some application scenarios where synthetic data objects are acceptable, to provide an approach to generate these synthetic objects, in §V-C, we present our synthetic data generation engine based on a machine learning framework called generative adversarial network (GAN).

A. Datasets for Transfer and Allocation

Though we do not make any assumptions on the authentic data objects or the use of synthetic data objects, ReLiShare does require the sender to decide the data objects that will be allocated among receivers; namely, each receiver will only obtain a subset of these objects.

For the sake of explanation, we let $Sent$ represent the whole dataset that will be sent by the sender, in which, we denote the data objects that will be distributed among receivers by $Dist$ and the data objects that will be received by all receivers by Com. Let

$$Sent = Dist \cup Com$$

Intuitively, all receivers can commonly receive all data objects in Com but each receiver will get a random subset of $Dist$. By the definition of data allocation and the use of 1-out-of-2 OT, in ReLiShare, half of the $Dist$ will be dropped individually in each sharing with a receiver.

- **Use real data for $Dist$:** When the application's use of a received dataset is sensitive to synthetic data objects, the sender can let
$$Sent = D, \quad Dist \subseteq Sent$$
This means the original dataset D will be transferred directly and the whole or a specified subset of D will be allocated among receivers.
- **Use synthetic data for $Dist$:** When synthetic data objects are acceptable, we have
$$Dist = \{s_1, s_2, ..., s_m\}, \quad Com = D$$
where s_i is a synthetic data object that cannot be distinguished from the real data object. This means while both real and synthetic data objects will be transferred, only synthetic

N	total number of data receivers
D	original dataset to be shared
$Sent$	the dataset sent by the sender in the data sharing process
Com	common data objects that will be received by all receivers
$Dist$	data objects that will be distributed among receivers
$Recv_i$	$Dist$ data objects received by receiver R_i
$Unique_i$	$Dist$ data objects *only* received by R_i
$Drop_i$	$Dist$ data objects *not received by* R_i in the OT based sharing
$SenderOnly$	$Dist$ data objects *not received by any receiver* in the OT based sharing
$SenderTree$	a three layer Merkle Tree of $Sent$ dataset where each two leaves are the hash values of each message pair in OT
$ReceiverTree$	a two layer Merkle Tree of $Receive$ dataset where each leaf is the hash value of each selected message in OT

Table I: Notation

data objects are used for allocation and each receiver can get all real data objects.

The size of $Dist$ is the key parameter to the system because the effectiveness of ReLiShare's identification algorithm depends on the number of leaked $Dist$ objects. Intuitively, the larger the size of $Dist$, the better chance that ReLiShare can identify leakers in the case of a leakage. To decide the size of $Dist$, the sender or the system operator should consider (i) total number of receivers and (ii) the expected identification accuracy. In § VII, we will present the relationship between these factors and the size of $Dist$ with equations when proving the correctness of our identification algorithm. In § VIII, we also provide the experiment results of identification accuracy when different number of $Dist$ are leaked from leaker(s).

The notations used in this paper is shown in Table I.

B. OT based Sharing

The sender will share the dataset with each receiver through ReLiShare's OT-based sharing. In each share process, besides the sender and a receiver, the notary will also be present to record the process of sharing. We focus on the OT sharing between the sender and receiver in this section and explain how notary generates the certificate in the next section.

Sharing Setup: Before sharing, the receiver randomly generates a bit array C as the choices used in later 1-out-of-2 OTs.

On the other side, the sender will need to prepare the dataset $Sent$ by deciding data objects in $Dist$ and Com according to the use case.

After that, the sender needs to allocate all data objects into pairs because each OT takes two input from the sender. To be more specific, assume each OT message only contains one data object, for data objects in Com, the sender can prepare OT input as

$$\text{1-2-OT}(d_i,\ d_i,\ c) \text{ for } d_i \in Com$$

meaning that the sender take the same real data object as two inputs to OT. In this case, the receiver will receive the real object regardless the value of choice c. As for objects from $Dist$, in contrast, the sender should make two inputs different for the purpose of object distribution

$$\text{1-2-OT}(d_i,\ d_{i+1},\ c) \text{ for } d_i, d_{i+1} \in Dist$$

Note that to reduce the total number of OTs, each OT message can contain more than one data objects. Importantly, the sender must randomly permute the data objects from Com and $Dist$ to prevent multiple receivers from colluding and using the same choice array C.

Sharing: After preparing the input of OTs, the sender then obliviously transfer these message pairs to the receiver until all data objects in $Sent$ are sent. During the sending process, the notary will also generate Merkle Trees from each received dataset $Recv_i$, which will be explained in the following section.

Through a OT based sharing, each receiver can receive the entire Com and a random subset of size $\frac{1}{2}|Dist|$ of $Dist$. At the same time, several desired properties can be achieved.

- After OT based sharing, the receiver only received data objects with regard to her choices C, which are not known to the sender. Therefore, the sender cannot predict the received dataset and the receiver cannot infer the dataset being sent. This property allows ReLiShare to prevent a guilty sender from framing innocent receivers.
- Since receiver's choices are randomly generated and the data objects are permuted by the sender before sharing, in the sharing process, half of objects in $Dist$ will be dropped unpredictably. Consequently, each sender i will have a different dataset, which contains data objects (denoted by $Unque_i$) that are only received by i and dropped by the other receivers. In addition, some data objects (denoted by $SenderOnly$) are dropped by all receivers and only known to the sender. The difference in each party's knowledge of dataset is the foundation of ReLiShare's knowledge-based leaker identification algorithm.

C. Synthetic Data Object Generation

When using synthetic data for $Dist$, the user is required to make sure they cannot be distinguished from real data objects. This is because otherwise the leaker(s) can intentionally filter out all $Dist$ to escape the identification. In ReLiShare, we utilize GAN to generate synthetic data objects since by setting up two neural networks contesting with each other, GAN is good at generating new dataset with similar properties as the original dataset.

We first highlight some of the important challenges we noticed when studying real world datasets.

- *Diverse Attribute Types.* Many real world datasets contain various types of attributes. For example, these attributes can be numeric (*e.g.,* age) with or without bound, categorical

(*e.g.,* gender), or strings (*e.g.,* credit card, address) with or without certain format.

- *Non-Normal Distributions.* Often the attributes that have continuous value have a non-Gaussian distribution. For example, the usual use case of GAN-based Neural Network works on iamge with range from 0 to 255, but in our tabular data, the range of some attributes (*e.g.,* transaction time) ranges from 1 to ∞. This leads to difficulty in selecting the last layer of a usual GAN-based network.
- *Imbalanced Data.* It is possible that a high percentage of data objects from a dataset all belong to a major attribute value (*e.g.,* "Female") while only a small percent of data objects belong to other values of that column (*e.g.,* "Male" and "Other").

To address the challenges mentioned above, we apply CTGAN [28], a data synthesizer based on conditional GAN. CTGAN is able to generate synthetic tabular data with high fidelity and performs particularly well on imbalanced datasets with various data types.

Although the authors of CTGAN have conducted evaluations on some realistic datasets, we decide not to use its evaluation results directly because their metric, *i.e.,* machine learning efficacy, depends largely on specific machine learning tasks. For example, for a simple binary classification task, the resulting accuracy would be high for authentic and synthetic data, but the accuracy is low for a more complicated task. Thus, presenting these two results together would lead to confusion. Importantly, ReLiShare's usage of the synthetic dataset is not related to machine learning tasks, and so machine learning efficacy could not help us to understand the quality of the synthetic dataset in our case.

As a result, we decide to use a more general, straightforward, and visual evaluation metric to evaluate the data property. For this purpose, we conduct experiments to evaluate the quality of generated synthetic data in §VIII. Notice that as a general approach, the model used for synthetic data generation may need to further tuned because of the diversity of datasets.

VI. Non-repudiable Proof of Sharing

To indisputably record the sharing process, a Notary is involved to produce a credential of each sharing process, in which the sharing result will be recorded into Merkle Trees and signed by the Notary. The goal for this step is to make sure that no party can deny what data objects they send or receive.

Specifically, the Notary will verify the data shared from the sender to receiver with the Merkle Trees derived from sender and receiver's input.

1) Before the sharing, the sender creates a three-layer Merkle Tree, called an *Sender Tree*, which is defined as,

$$SenderTree\ Root\ Value = H(...||\ H(H(m_{j,0})||H(m_{j,1}))\ ||...)$$

for each message pair j sent by the sender S. Each two paired OT messages share a parent node and all parent nodes are children of the root node. The sender provides the Notary with all nodes in the *Sender Tree*, which are sufficient to snapshot all the OT messages sent (❶).

2) Before the sharing, a receiver provides the Notary with choices that will be used in the OT (❷).
3) After the sharing, the receiver calculates a two-layer Merkle Tree, called a *Receiver Tree*, on all the received messages. A Receiving Tree is defined as,

$$ReceiverTree\ Root\ Value = H(...||\ H(m_{j,b})\ ||...)$$

for each message pair j sent by the sender, where b is receiver's choice in the 1-out-of-2 OT of j. The receiver then sends the root node value of its *Receiver Tree* to the Notary (❸).

4) The Notary computes another *Receiver Tree* based on the *Sender Tree* and choices provided by the receiver. It then compares the root node with the root node value of the *Receiving tree* provided by the receiver in step 3 (❹).

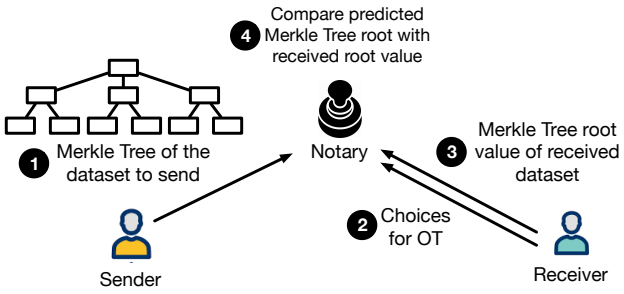


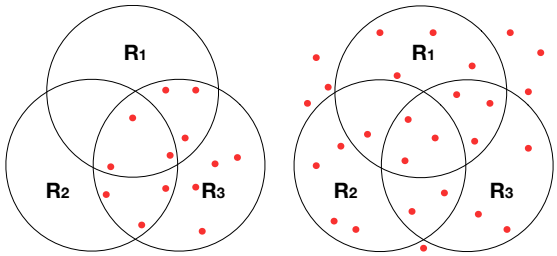
Figure 2: Sharing under the witness of a Notary

If the comparison in step 4 is successful (i.e., same root value), the sharing is finished and the Notary will issue a credential. Such a credential contains (i) the root node value of the *Sender Tree*, along with (ii) the root node value of the *Receiver Tree* and (iii) the timestamp. The Notary will sign the credential and permanently keep it; for example, it can serialize the credential to paper copies or insert it into an immutable ledger. On the other hand, if the comparison fails, the receiver should delete all the received data with the sender and Notary as witnesses and a new sharing process should be started. Such a process can potentially be facilitated by trusted hardware [32], [33], which are out of the scope of this paper.

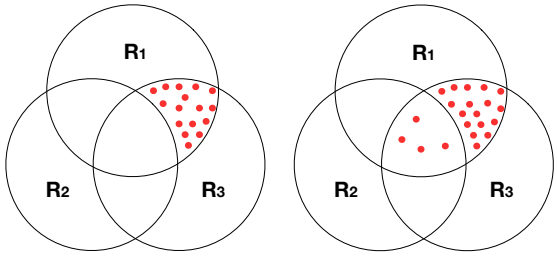
The credential immutably records the data being sent with the Merkle Tree and the data being received with the choices. In a later arbitration process, the arbitrator can compare the claimed dataset with the certificate to ensure no party, including the sender, can cheat. In the sharing process, a Notary cannot see the actual transferred data because of the one-way hash function used in Merkle Tree.

VII. Leaker Identification

ReLiShare identifies leakers by analyzing the leaked data and finding the involved parties whose knowledge is sufficient to create such a leaked dataset.



(a) Leaker Is R_3 (b) Leaker is S



(c) All Receiver Colludes (d) All Receiver Colludes

Figure 3: Intuitive explanation of ReLiShare knowledge-based identification

Specifically, since all receivers will have the complete Com, in this section, *all data objects we discussed refer to data objects in* $Dist$.

To provide some intuitive ideas of how ReLiShare's identification works, we show four leakage examples in Figure 3. The dots in the figure are the leaked data objects while each circle represents data objects of $Dist$ received by each receiver. The intersection of circles are the objects that are received by multiple receivers. As shown, if leaked data objects are uniformly distributed in receiver R_3's circle, as in Figure 3(a), we can easily infer R_3 to be the leaker. When leaked objects are like Figure 3(b), the sender is guilty because some leaked objects have never been received by any receiver (dots not falling into any circle). If objects are distributed in a specific section as shown in Figure 3(c), it is trivial to deduce the leaked data is from $Recv_1 \cap Recv_3 \cap Recv_2^{\complement}$, which means all three receivers collude in the leakage. Figure 3(d) is a more complicated scenario where leaked objects are all from $Recv_1 \cap Recv_3$ but with a non-uniform distribution. Under this scenario, we can still infer that R_2 participates in the collusion because R_1 and R_3 should not have the knowledge to distinguish objects that belong to R_2 without R_2's help.

Following this idea, the leaker identification phase in AuditShare consists of two simple steps:

1) **Confirmation:** The involved sender and receivers provide the Arbitrator with their datasets sent/received in the same order as in the sharing. By calculating the Merkle Trees of these datasets and checking against credentials provided by the Notary, the Arbitrator is able to ensure no party can deny or lie about the result of sharing.
2) **Identification:** Once the sharing process is revealed, the Arbitrator knows the exact allocation of synthetic data objects among receivers. According to the knowledge used to construct the leaked data, ReLiShare evaluates the innocence of every involved party and identifies the leaker(s) with a very low error rate.

The confirmation process is relatively straight forward comparing with the identification process because the Arbitrator can directly check the Merkle Tree root values from the credential against the datasets claimed by each party. Therefore, in the remaining section, we focus on the identification process, that is, given the exact data allocation, how to identify (i) a non-colluding leaker (the sender or a receiver), and (i) all colluding leakers.

In an arbitration, the Arbitrator will first run non-colluding leaker identification algorithm introduced in §VII-A and then run colluding leaker identification algorithm as shown in §VII-B. As a result, ReLiShare will generate a list of guilty parties with the success rate of each party.

A. *Identifying Non-Colluding Leaker*

A non-colluding leaker can either be a guilty sender or one of the receivers.

Algorithm Summary: ReLiShare takes the following steps to identify each non-colluding leaker.

1) If there is no $Dist$ data objects (*i.e.,* all leaked objects are from Com), consider the sender is the leaker.
2) The algorithm first figures out each receiver R_i's (i) $Unique_i$, unique data objects that are never received by all the other receivers and (ii) $Drop_i$, the objects that are not received by this receiver.
3) For each receiver R_i, the algorithm then calculates the overlap between (i) leaked objects and $Unique_i$ and between (ii) leaked objects and $Drop_i$. If the first set is nonempty but the second is empty, we consider this receiver is a leaker.
4) Finally, to verify the sender's honesty, the algorithm calculates $SenderOnly$, the objects that are only known to the sender (*i.e.,* not received by all the receivers). If there exists an overlap between sender-only objects and leaked objects, we consider the sender guilty too.

Algorithm Rationales: Based on the data objects allocation, both the sender and each receiver have a number of data objects from $Dist$ that are never received by any other parties. Thus these objects can serve as the clue of identifying non-colluding leakers.

Therefore the following statements hold when there is no collusion. First, R_i is a leaker by a large probability if leaked data objects D_{ls} contain objects that are also in $Unique_i$ but no objects from $Drop_i$ because only R_i has received $Unique_i$. If leaked objects contain objects from $Drop_i$, R_i is not a non-colluding leaker as R_i cannot leak data objects that have not been received (however, this does not exclude R_i from being a collusive leaker). Specifically, $Unique_i$ can be calculated by

$$Unique_i = Recv_i \cap \left(\bigcup_{j \neq i} Recv_j \right)^{\complement} \qquad (7.1.1)$$

Algorithm 1 Identifying a Non-colluding Leaker

Require:
1: Leaked Dist data objects: $Dist_L$
2: Each receiver R_i's synthetic data objects: $Recv_i$
3: Dist data objects that were not received by R_i: $Drop_i$
Ensure: Non-colluding leaker set L;
4: Calculate each receiver's unique objects $Unique_i$ from $Recv_1, Recv_2, ..., Recv_N$
5: **for** i from 1 to N **do**
6: **if** $Dist_L \cap Unique_i \neq \varnothing \wedge Dist_L \cap Drop_i = \varnothing$ **then**
7: Add R_i into L
8: Compute false positive rate by Equation 7.1.3
9: Compute false negative rate by Equation 7.1.4
10: **end if**
11: **end for**
12: Calculate objects known to the sender only: $SenderOnly$
13: **if** $Dist_L \cap SenderOnly \neq \varnothing$ **then**
14: Add S into L
15: Compute false negative rate by Equation 7.1.5
16: **end if**

In addition, S must be a leaker if D_{ls} contains objects from $SenderOnly$ because only S possesses $SenderOnly$. Specifically, from the data allocation process, we have:

$$SenderOnly = (Drop_1 \cap Drop_2 ... \cap Drop_N) \quad (7.1.2)$$

Since the sender knows which objects are in $Dist$, the sender may only leak data from Com dataset. However, since receivers cannot tell them apart, the leaker must be the sender. In a more complicated scenario where the sender can leak a small number of objects from $Dist$, the sender will be caught by the colluding leaker identification algorithm as discussed in §VII-B.

Pseudo Code and Proof: Algorithm 1 shows the pseudo-code of ReLiShare's algorithm of non-colluding leaker identification and we prove its effectiveness by proving Lemma VII.1 and Lemma VII.2.

Lemma VII.1. *Given $Dist_L$ as the leaked Dist data objects, when $Dist_L$ contains objects from $Unique_i$ but not from $Drop_i$, the algorithm can detect a non-colluding guilty receiver R_i with false positive rate $FP_{nocollusion}$ and false negative rate $FN_{nocollusion}$ as shown in Equation 7.1.3 and Equation 7.1.4.*

Proof. See Appendix A.

$$FP_{nocollusion} = \frac{\binom{|Dist|-|Drop_i|}{|Dist_L|} - \binom{|Dist|-|Drop_i|-|Unique_i|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \approx \frac{\binom{\frac{1}{2}|Dist|}{|Dist_L|} - \binom{(\frac{1}{2}-\frac{1}{2^N})|Dist|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \quad (7.1.3)$$

$$FN_{nocollusion} = \frac{\binom{|Recv_i|-|Unique_i|}{|Dist_L|}}{\binom{|Recv_i|}{|Dist_L|}} \approx \frac{\binom{(\frac{1}{2}-\frac{1}{2^N})|Dist|}{|Dist_L|}}{\binom{\frac{1}{2}|Dist|}{|Dist_L|}} \quad (7.1.4)$$

□

Lemma VII.2. *Given $Dist_L$ as the leaked Dist data objects, when $Dist_L$ contain objects from $SenderOnly$, the algorithm can detect a non-colluding guilty sender with false positive rate $FP_{sender} = 0$ and false negative rate $FN_{nocollusion}$ as shown in Equation 7.1.5.*

Proof. See Appendix A.

$$FN_{sender} = \frac{\binom{|Dist|-|SenderOnly|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \approx \frac{\binom{(1-\frac{1}{2^N})|Dist|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \quad (7.1.5)$$

□

Importantly, $FP_{nocollusion}$ and $FN_{nocollusion}$ is small enough even when a guilty receiver R_i only leaks a small portion of data objects from $Recv_i$. For example, with 3 receivers in total, when $|Dist| = 200$, both false positive and negative rate is $< 1 \times 10^{-18}$ even when R_i randomly leaks half of its received data, which contains about 50 data objects from $Dist$. Similarly, FN_{sender} is negligible even when S only leaks a small portion of data objects from $Data$.

As shown in Algorithm 1, code line 4-11 is for non-colluding guilty receiver identification and line 12-16 is for a guilty sender.

B. Identifying Colluding Leakers

Algorithm Summary: After identifying all non-colluding parties with Algorithm 1, ReLiShare identifies colluding leakers by the following steps.

1) The algorithm first generates a vector to represent the distribution of all $Dist$ data objects, in which each index represents a pattern of distribution, called a *distribution pattern*, and the value of each element represents the number of objects falling into that pattern.
2) Similarly, another vector will be generated from all leaked $Dist$ data objects. We denote the vector for $Dist$ by V_D and the vector for leaked objects by V_L.
3) For each receiver, the algorithm then infers whether this receiver contributed to the collusive leakage by comparing V_L and V_D with a statistical test called binomial test. Intuitively, the object distribution in the V_L should not be significantly different from that in V_D if this receiver did not contribute to the leakage.

Algorithm Rationales: Where multiple receivers collude, the leaked data can still reflect leakers' knowledge due to the fact that these leakers cannot figure out which objects have been received by other honest receivers. Following this direction, ReLiShare is able to identify all leakers with the following procedure. If a given party R_i did not contribute to

the collusion, the leaked objects in and out of $Recv_i$ should be statistically identical, considering the 1-out-of-2 OT used in the sharing process. If not, then there is a great chance that R_i participated in the leakage. If no receiver is identified in the above process, it means all data objects in $Dist$ are uniformly leaked, indicating the leaker does not have knowledge of the distribution of $Dist$ objects for any receiver. Therefore, it is the sender that leaked the data objects.

To facilitate the explanation of the algorithm, we first introduce the Pattern Matrix Pat, where each row represents a pattern of allocation among receivers. Given a number of receivers N, Pat is a $2^N \times N$ matrix whose value shown in Figure 4. In each row j, we define each element $Pat[j][i] = 1$ to represent that R_i has received the set of data objects and $Pat[j][i] = 0$ if this object has been dropped by R_i because of OT. For example, as shown in the figure, $Pat[0]$ represents the distribution pattern of "dropped by all receivers" or $SenderOnly$ and $Pat[2^N]$ denotes the pattern of "received by all receivers".

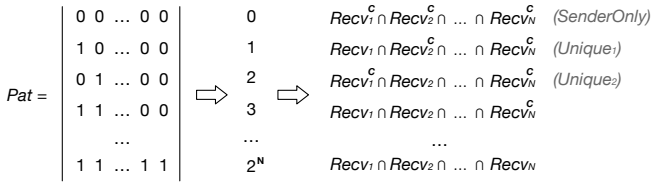


Figure 4: Relationship between pattern matrix, index, and data object distribution

Consider the leftmost element in each row to be the least significant bit, each row of Pat can be transformed into an integer (the middle part in Figure 4). We will then use these integers as the *index* of the vector. For the sake of explanation, we call two integers (*i.e.,* two rows in Pat) k and k' are *paired* on R_i if they are only different at i's slot. For example, $Pat[0]$ and $Pat[1]$ are paired rows on R_0.

Since each leaked $Dist$ data object can be categorized in to a pattern, which is a row in the Pattern Matrix, all leaked objects can be represented by a *dataset vector*, denoted by V, where each element $V[i]$ represents the number of data objects that fall into the distribution pattern of index i, *i.e.,* the i-th row in the Pattern Matrix Pat. For example, $V[0]$ is the number of objects that are never received by any receiver and $V[1]$ is the number of objects that are only received by R_1. Similarly, in a Vector, we call two elements $V[k]$ and $V[k']$ are paired if they are indexed by two paired indexes k and k'.

Importantly, for two paired indexes k and k' on receiver R_i, if R_i does not contribute to the leakage, a leaked data object d should have:

$$P(d\ is\ of\ pattern\ k) = \frac{1}{2}$$

This is because leakers have no knowledge to distinguish whether the object d has been received by R_i or not.

In order to measure how likely a party does not have the knowledge, we employ a statistic test, binomial test [34], to obtain a p-value [35]. In ReLiShare, a binomial test for two paired indexes k and k' on receiver R_i can be represented as:

$$p\_value = Binomial\_Test(V[k], V[k'], \frac{1}{2})$$

To be specific, a P Value indicates the probability of having a distribution equal to or more extreme than $(V[k],\ V[k'])$ given $P(d \in V[k]) = \frac{1}{2}$ (*i.e.,* R_i cannot tell objects from two patterns). For example, when $(V[k], V[k'])$ = (14, 6), a p-value of 0.115 means that 11.5% of the time one would expect to have a distribution equal to (14, 6) or more extreme (*e.g.,* (15, 5)). When p-value is less than a pre-defined value α (e.g., $\alpha = 0.05$), it is considered that the party is likely to have the knowledge to distinguish objects in these two paired patterns. In §X-A, we justify that ReLiShare does not abuse P Value in identifying leakers.

Algorithm 2 Leakers identifying algorithm.

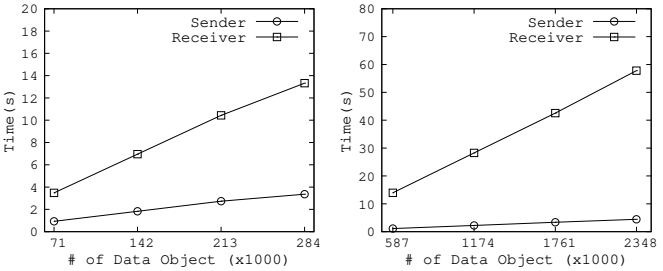
Require:
Leaked $Dist$ data objects: $Dist_L$
The allocation of $Dist$ objects: V_D
A significance level: α, α = 0.05 by default
Ensure: Leakers set L with an accuracy value A
Transform $Dist_L$ into a dataset vector V_L
Initialize a vector $Pmin[N]$ = {1,1,...,1}
for i from 1 to N **do**
  for k from 1 to 2^N **do**
    Find k' which is paired with k
    Calculate P = $V_D[k]$ / $V_D[k']$
    p_value_i = Binomial_Test($V_L[k]$, $V_L[k']$, P)
    if $p\_value_i < \alpha$ **then**
      Add R_i into L
      Compute false positive rate to be min($Pmin[i]$, p_value_i)
    end if
  end for
end for
Calculate overall specificity $\prod_{R_i \in L}(1 - Pmin[i])$

Pseudo Code and Proof: Algorithm 2 shows the pseudo code of the ReLiShare's identification algorithm for colluding leakers and we prove its effectiveness by proving Lemma VII.3, which says that when a receiver contributed to the leakage, its contribution can be reflected by the object distribution of two paired elements in Dataset Vector.

Let V_L be the dataset vector for the leaked $Dist$ objects and V_D be the dataset vector for all $Dist$ objects.

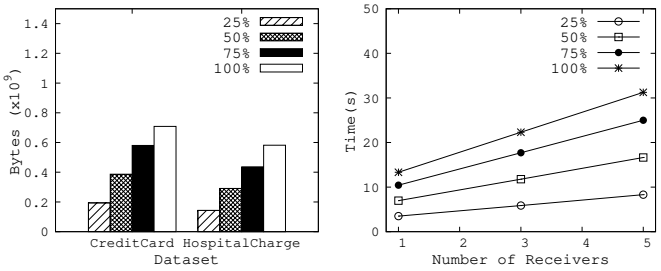
Lemma VII.3. *If there exist any paired patterns k and k' on R_i where $V_L[k]$ has a significant difference from $V_L[k']$, we draw the conclusion that receiver R_i contributed to the leakage with false positive rate being the p-value obtained in the binomial test.*

Proof. See Appendix A. □



(a) Sharing CreditCard (b) Sharing HospitalCharge

Figure 5: Time overhead of OT based sharing



(a) Bandwidth overhead (b) Multiple receivers

Figure 6: Bandwidth overhead of OT based sharing and sharing with multiple receivers

In Algorithm 2, line 6 to line 16 check for each receiver R_i, whether there exists paired $V_L[k]$ and $V_L[k']'$ such that $V_L[k]$ is significantly different from $V_L[k']'$. The overall specificity is the probability when the algorithm makes a true negative identification on each captured party. In §X-B, we discuss the false negative rate in identifying colluding leakers.

VIII. Evaluations

In this section, we present the evaluation of ReLiShare. To be more specific:

- We evaluate the performance of ReLiShare by measuring (i) the time overhead in ReLiShare's data sharing process in terms of the number of data objects and receivers, (ii) the time and bandwidth overhead in the credential generation. From the evaluation result, we show ReLiShare is able to finish heavy-load dataset sharing tasks and credential generation within a short time.
- We evaluate ReLiShare's effectiveness in identifying leakers in different scenarios. To be more specific, our evaluation shows that, when the leaker(s) leak a random half of their received dataset or jointly leak 50% objects from a combined dataset, ReLiShare only requires about 1500 $Dist$ objects to be sufficient to identify non-colluding and colluding leakers with error rate $< 1\times10^{-7}$ and $< 1\times10^{-2}$, respectively. Considering today's dataset can contain a huge number of data objects (*e.g.,* two real world datasets used in our experiments contains more than 280K objects and 2M objects), the number of objects required by ReLiShare for allocation is reasonable.
- We also evaluate our synthetic data generation algorithm by measuring the similarity between synthetic data objects and authentic data objects.

A. Experiment Setup

We perform our experiments over two real world datasets: CreditCard [36] and HospitalCharge [31] (Table II), which were downloaded from public sources. CreditCard is users' credit card information and we manually append three sensitive attributes: SSN, job title, and name. HospitalCharge is charges information of patients in the hospital and we first select the gender, length of stay, total charges, and total costs from the original dataset and manually add two sensitive attributes: name and ssn. We present our results on datasets with different scales: for each of the datasets we use 25%, 50%, 75%, and 100% of its data for experiments (*e.g.,* 25% of CreditCard is simply 0.28M * 0.25 = 0.07M data objects).

Name	Average Length	Cardinality	Size (MB)
CreditCard	568	0.28 M	155
HospitalCharge	49	2.3 M	114

Table II: Two real world datasets used in evaluation

In our experiment, IKNP OT extension [37] is used in ReLiShare's OT sharing. In credential generation, cryptographic hash function Sha256 [38] is applied in Merkle Tree generation and Ed25519 [39] is used by the Notary to sign the credential.

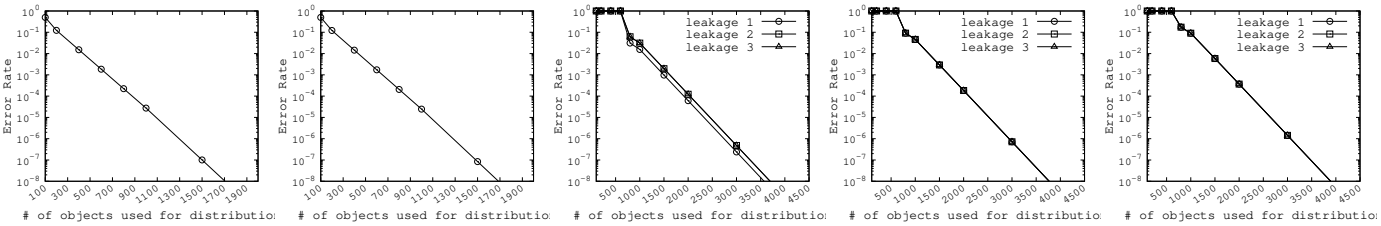
All experiments are conducted on a server with an Intel Xeon(R) CPU processor, 32GB RAM, running Debian GNU/Linux 10. The system except leaker identification algorithms (written in Python) are compiled with GCC 8.3.0.

B. Overhead of OT based Sharing

To evaluate the overhead of OT based sharing on different scales of data and observe the relationship between dataset size and time/bandwidth overhead, we conduct extensive experiments on different scales of data. To be more specific, for each of the datasets we use 25%, 50%, 75%, and 100% of its data. Note that, in measuring the overhead, without loss of generality, we use synthetic data objects for $Dist$.

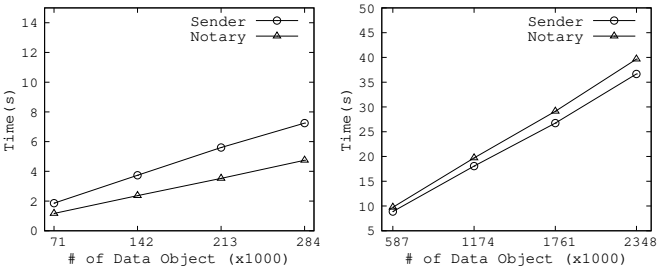
The results of time and bandwidth overhead evaluation are shown in Figure 5 and Figure 6(a). From the graphs, the relationship between time/bandwidth overhead and size of the dataset is linear, indicating that ReLiShare is capable of handling sharing tasks with large datasets.

In addition to evaluating the relationship between OT overhead and the size of the dataset being used, we also evaluate the relationship between OT overhead and the number of receivers on dataset CreditCard. To be more specific, we evaluated the time overhead on the sender side with different scales of the dataset used: we looked at 1, 3, and 5 receivers and 25%, 50%, 75%, and 100% of the CreditCard. Since the increase in the number of receivers does not affect the time/bandwidth overhead in the individual receiver, we ignore the OT overhead on the receivers' side. The results are shown



(a) Only Sender (b) Non-colluding Receiver (c) 2 Colluding Receivers (d) 3 Colluding Receivers (e) 6 Colluding Receivers

Figure 7: Identification error rate in different scenarios



(a) For CreditCard (b) For HospitalCharge

Figure 8: Time overhead of Merkle Tree generation and verification

in Figure 6(b). Note the linear relationship between the number of receivers and OT overhead. It implies that our proposed system works with multiple receivers.

C. Credential Generation Overhead

For a Notary to generate a credential, a sender first needs to prepare a three-layer Merkle Tree and then the Notary needs to reconstruct a two-layer Merkle Tree. We measure the time overhead in Merkle Tree generation on the sender side and Notary side (Figure 8). It shows that the time to generate a Merkle Tree grows linearly with the size of dataset and the time overhead for the two datasets used in our experiments are about 7s and 40s, respectively. Note that a sender can generate the Merkle Tree offline and does not take time in the sharing. Also note that for CreditCard the time required for the sender to generate the Merkle Tree is greater than that for the Notary, while it is the opposite case for HospitalCharge. This is due to the fact that the average length of CreditCard's data objects is greater than that of HospitalCharge's data objects. Consequently, it requires more time for the sender to build the Merkle Tree from plaintext than for the Notary to build from hashed leaves.

In addition, we also measure the time used for Notary to prepare and sign the credential (Table III). We observe that the time remains almost the same regardless of the sizes of data being transferred during the OT. This is because the size of Merkle Tree root values remains the same.

Containing two Merkle tree root values and other information (*e.g.,* Ed25519 signature, timestamp, Notary's identity information), a credential is about 470 Bytes. Note that the size can change if a different hash function for Merkle Tree or digital signature scheme for signing is used.

Size of total data objects (MB)	38.8	77.5	155
Avg. time to sign the credential (ms)	0.014	0.016	0.013

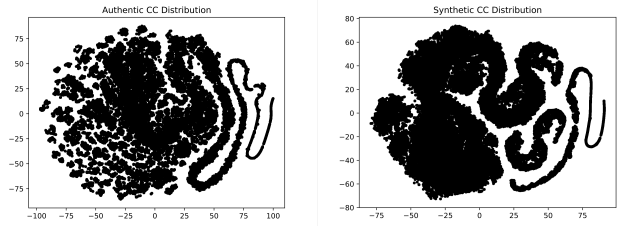
Table III: Credential Generation Time

D. Identification Effectiveness

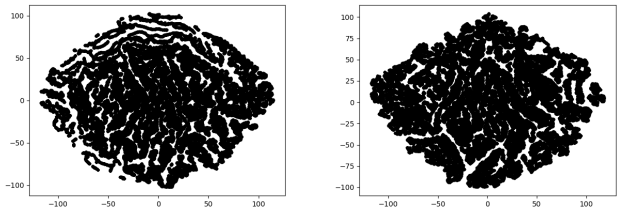
In this section we evaluate ReLiShare's effectiveness in identifying leakers in three cases: (i) a guilty sender, (ii) a non-colluding guilty receiver, and (iii) a group of colluding receivers. The objects used for distribution can be either real objects or synthetic objects, depending on whether synthetic data is acceptable. Specifically, we measure the false positive rate with difference sizes of $Dist$, that is, the number of data objects used for distribution. We do not give false negative rate is because it is much smaller than false negative rate, which is confirmed by our mathematical proof and experiments.

We first investigate the cases of a single leaker, which is the sender or a receiver. Specifically, we let the sender and the receiver randomly leak half data objects from the whole dataset and the dataset received by the receiver, respectively. The false negative error rate is presented in Figure 7(a) and Figure 7(b). As shown, when $Dist$ is more than 400, the false negative rate of identifying a guilty sender and a non-colluding guilty receiver is already lower than 0.01. When about 1700 $Dist$ objects are used in sharing, the chance of false negative is negligible (*i.e.,* $< 1 \times 10^{-8}$) in two cases.

When simulating leakage from multiple colluding receivers, we did multiple experiments for different colluding mechanisms by randomly flipping participant's received set between $Recv$ and $Recv^{\complement}$ in $Recv_i \cap ... \cap Recv_j$ where colluding leakers are $R_i, ..., R_j$ and applying set union between different combinations. The leaked objects are a random half of the combined dataset (*i.e.,* after a set of collusive operations). We present three representative experiment results (marked as "leakage 1", "leakage 2", and "leakage 3") for 2-party, 3-party, and 6-party leakage in Figure 7(c), 7(d), and 7(e), respectively. For a detailed description of the combination mechanisms, check Appendix B. Compared with non-collusive leakage, to identify colluding leakers, a larger $Dist$ is needed. For example, while about 1700 $Dist$ objects can result in a $< 1 \times 10^{-8}$ false negative rate in non-colluding leaker identification, to achieve



(a) Authentic CreditCard (b) Synthetic CreditCard



(c) Authentic HospitalCharge (d) Synthetic HospitalCharge

Figure 9: Comparison of Authentic and Synthetic Datasets

the same level of accuracy, about 4000 objects are needed for a 6-party leakage. This is expected because instead of relying on a relatively large unique dataset (*e.g.,* $SenderOnly$ or $Unique_i$) for identification, the binomial tests required a comparison between data objects falling in different small groups. Therefore, a larger sample set is needed to obtain accurate results in colluding leaker identification.

E. Quality of Synthetic Data

As noted in §V-C, the evaluation results of Machine learning efficiency in the original CTGAN work do not promise its effectiveness in generating synthetic data for leaker identification. Therefore, besides manually check the dataset, we also apply dimensionality reduction and use t-Distributed Stochastic Neighbor Embedding (t-SNE) [29] to visualize the two datasets we used in experiments and corresponding synthetic datasets. We present our result in Figure 9. Note that since the high-dimensional datasets are reduced to two-dimension, there are no meaningful labels for both horizontal or vertical axis, and thus we ignore these labels for the figures.

From the graphs we see, though there are some discrepancies between the synthetic and real data objects (because synthetic data objects cannot be identical to real ones), the nearest neighbor distance distribution of the synthetic data is very close to the real data. However, as mentioned, since each dataset is very different from others and each attribute can have its own pattern (*e.g.,* credit card number cannot be random because there are pre-defined conventions), our model may need to be modified or manually corrected when used in specific use cases.

IX. Related Work

The data leakage problem has been explored for years [40], [41]. Past works in this domain cover leakage prevention, leakage detection, and leaking source identification. In terms of the media of information, existing works have covered structured data (e.g., rational database, spreadsheets), textual data (e.g., contracts, reports), and multimedia (e.g., images, video). Our work focuses on the leaker identification problem in dataset, which is typically under the category of structured data. Existing leaker identification solutions to the same issue can be divided into two big categories: (i) fingerprinting (watermarking), and (ii) data object allocation.

Fingerprinting: While most digital watermarking mechanisms [24], [41], [42] for dataset is for integrity and right protection, watermarking can also be used for leaker identification in dataset sharing, which is usually referred to as fingerprinting. First proposed by Agrawal *et. al* [11], [12], dataset fingerprinting was further extended in later works [13]–[18], where the fingerprint is applied to multiple types of attributes and generated by different means. The main idea of applying fingerprinting for leaker identification is to let the data sender alter the noise-tolerant attributes or embeds distinct hidden information into the dataset for each receiver. Later, when unauthorized dataset is found, by associating the watermark contained in the leaked data with the receivers, the data sender is able to identify leakers in the case of a leakage.

Among these works, to prevent a data sender from framing innocent receivers in a leaker identification process, Xian *et. al* proposed to utilize a trusted watermark server and take input from both sender and receiver to derive watermarking [18]. As pointed out in existing works [11], [24], [43], it is hard to effectively make a watermark on structured data in general, because it is trivial for attackers to further alter the structured data to spoil the watermark compared with other types of media (e.g., video, image), especially when multiple receivers collude together and compare their received copies. A detailed comparison between ReLiShare and watermark based works can be found in Table IV.

Data Object Allocation: First proposed in Panagiotis *et. al*'s work [19], [20], data object allocation mechanisms allocate data objects among receivers so that each receiver will obtain a distinct dataset from others. After finding unauthorized data, the sender can utilize the receiver-objects mapping to evaluate each receiver's probability of data leaking. Synthetic but realistic data objects can also be created to facilitate detection. Following this direction, Ishu *et. al* proposed a guilt detection mechanism [21] that uses a directed bigraph to allocate data objects to receivers. A threshold based scheme [22] was proposed to utilize threshold measure in distribution strategy to improve identification accuracy. Detailed comparisons between ReLiShare and these works are presented in Table IV.

Nevertheless, most existing allocation mechanism rely on the data sender to allocate data objects, enabling the sender to become a traitor without being caught or even to frame innocent receivers. In addition, most allocation strategies can be disrupted by a collusion among receivers, by which corrupted receivers can learn the allocation and escape being captured by stripping special data objects that are potentially used to

Work	Main approach	Corrupt sender	Collusive leakage	Indisputable sharing	Impact on dataset
Agrawal and Kiernan *et. al* [11], [12]	Distortion based fingerprint on numerical attributes	✗	✗	✗	Embedding random bits to numerical attributes
Li *et. al* [13], [14]	Distortion based fingerprint on numerical attributes	✗	✗	✗	Embedding meaningful bits to numerical attributes
Liu *et. al* [15]	Distortion based fingerprint	✗	✗	✗	Embedding receiver ID and sender's secret into attributes
Guo *et. al* [16]	Two-levels fingerprinting	✗	✗	✗	Embeding a unique fingerprint for each receiver
Tardos *et. al* [17]	Randomized scheme to generate codewords for each receiver	✗	✓	✗	Embedding fingerprint to attributes
Xian *et. al* [18]	Trusted third party for fingerprinting derived from sender and receiver's key	✓	✗	✗	Embedding fingerprint to attributes
Papadimitriou *et. al* [19], [20]	Data object allocation	✗	✗	✗	Adding synthetic data objects
Gupta *et. al* [21]	Data object allocation using Bigraph	✗	✗	✗	Adding synthetic data objects
Gupta *et. al* [22]	Allocating data object with threshold	✗	✗	✗	Reducing size of received dataset
Singh *et. al* [23]	Allocation data objects dynamically	✗	✗	✗	Reducing size of received dataset
ReLiShare	Data object allocation with OT and indisputable proof of sharing	✓	✓	✓	Reducing size of received dataset or adding synthetic data objects

Table IV: Comparison between ReLiShare and some representative related works

uniquely identify each receiver (e.g., leaking common objects only). Another issue is that most existing works assume the leakers cannot deny the recipient of dataset and will not lie on the received dataset; however, such assumption can only be held true if there exist an indisputable proof of the sharing.

X. Discussion

A. The use of P Value in ReLiShare

Recently, there have been criticisms about abusing P Value to conclude there is difference or no association just because a P Value is larger than a threshold in certain application scenarios [44]–[46]. In ReLiShare, we utilize P Value to measure how likely a given receiver is honest when identifying colluding leakers. However, in the process of identifying leakers based on data object allocation, the distribution of leaked data objects itself is a dichotomization and the P Value in our applications does reflect the probability of a potential leaker's knowledge of telling whether data objects belong to a certain receiver. Furthermore, in ReLiShare, instead of directly claiming guilty parties, we compute the probability of false positive and/or negative rate for each suspect, providing useful input for further out-of-band leaker investigations. Therefore, the use of P Value in ReLiShare does not match the scenarios mentioned in recent criticisms and is not an abuse.

B. False Negative in Identifying Colluding Leakers

We do not give the false negative rate for each identified colluding leaker for two reasons. First, we cannot obtain it from binomial tests. Second, there is a false negative result for R_i when R_i joined the collusion but leaked data objects cannot reflect colluding party's knowledge of being able to distinguish between data objects that R_i received and not received As noted in §III-B, we do not need to identify such a party because it makes no or little contribution to the leakage so that even without its participation, other colluding receivers still have sufficient knowledge to leak the same objects. In other words, this party leaves no "footprint" in a leakage, meaning R_i shouldn't be considered guilty and at the same time, it is almost impossible to trace R_i by solely analyzing leaked objects.

XI. Conclusion

In this work, we study the leaker identification problem in sensitive data sharing and propose ReLiShare, a data-sharing system with reliable leaker identification. ReLiShare obliviously allocates data objects across receivers, indisputably records the sharing process, and accurately identifies leakers with ReLiShare's knowledge-based algorithms. Compared with existing works, ReLiShare provides desirable features of providing nonrepudiation and detecting guilty senders and colluding leakers, making ReLiShare more practical to handle real world leakage events.

Our work shows that by proactively building leaker traceability into the data sharing process, one can significantly facilitate the process of leaker identification in case of data leakage. This can in turn push participants involved in a sensitive data sharing to better protect their data.

References

[1] E. McCallister, T. Grance, and K. Scarfone, *Guide to protecting the confidentiality of personally identifiable information*. National Institute of Standards and Technology (NIST) Special Publication 800-122, 2010.

[2] Centers for Medicare and Midcaid Services. (2020) Data.cms.gov. [Online]. Available: https://data.cms.gov/

[3] M. Blackman. (2019) Hospital health IT interoperability. [Online]. Available: https://www.mckesson.com/blog/rise-in-patient-data-exchanged/

[4] The Wall Street Journal. (2019) Amazon investigates employees leaking data for bribes. [Online]. Available: https://www.wsj.com/articles/amazon-investigates-employees-leaking-data-for-bribes-1537106401

[5] The New York Times. (2019) Capital one data breach compromises data of over 100 million. [Online]. Available: https://www.nytimes.com/2019/07/29/business/capital-one-data-breach-hacked.html

[6] DoorDash Blog. (2020) Important security notice about your doordash account. [Online]. Available: https://blog.doordash.com/important-security-notice-about-your-doordash-account-ddd90ddf5996#46h35gr24e
[7] BBC News. (2019) British airways faces record £183m fine for data breach. [Online]. Available: https://www.bbc.com/news/business-48905907
[8] DoorDash Blog. (2020) Important security notice about your doordash account. [Online]. Available: https://blog.doordash.com/important-security-notice-about-your-doordash-account-ddd90ddf5996#46h35gr24e
[9] EU GDPR.ORG. (2019) The eu general data protection regulation (gdpr). [Online]. Available: https://eugdpr.org/
[10] Californians for Consumer Privacy. (2019) California consumer privacy act (ccpa). [Online]. Available: https://www.caprivacy.org
[11] R. Agrawal and J. Kiernan, "Watermarking relational databases," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 155–166.
[12] R. Agrawal, P. J. Haas, and J. Kiernan, "Watermarking relational data: framework, algorithms and analysis," *The International Journal on Very Large Data Bases*, vol. 12, no. 2, pp. 157–169, 2003.
[13] Y. Li, V. Swarup, and S. Jajodia, "Constructing a virtual primary key for fingerprinting relational data," in *Proceedings of the 3rd ACM workshop on Digital rights management*, 2003, pp. 133–141.
[14] ——, "Fingerprinting relational databases: Schemes and specialties," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 34–45, 2005.
[15] S. Liu, S. Wang, R. H. Deng, and W. Shao, "A block oriented fingerprinting scheme in relational database," in *International conference on information security and cryptology*. Springer, 2004, pp. 455–466.
[16] F. Guo, J. Wang, and D. Li, "Fingerprinting relational databases," in *Proceedings of the 2006 ACM symposium on Applied computing*, 2006, pp. 487–492.
[17] G. Tardos, "Optimal probabilistic fingerprint codes," *Journal of the ACM (JACM)*, vol. 55, no. 2, pp. 1–24, 2008.
[18] H. Xian and D. Feng, "Leakage identification for secret relational data using shadowed watermarks," in *2009 International Conference on Communication Software and Networks*. IEEE, 2009, pp. 473–478.
[19] P. Papadimitriou and H. Garcia-Molina, "A model for data leakage detection," in *2009 IEEE 25th International Conference on Data Engineering*, March 2009, pp. 1307–1310.
[20] P. Papadimitriou and H. Garcia-Molina, "Data leakage detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 51–63, Jan 2011.
[21] I. Gupta and A. K. Singh, "A probabilistic approach for guilty agent detection using bigraph after distribution of sample data," *Procedia Computer Science*, vol. 125, pp. 662–668, 2018.
[22] ——, "Dynamic threshold based information leaker identification scheme," *Information Processing Letters*, vol. 147, pp. 69–73, 2019.
[23] A. K. Singh and I. Gupta, "Online information leaker identification scheme for secure data sharing," *Multimedia Tools and Applications*, pp. 1–18, 2020.
[24] R. Halder, S. Pal, and A. Cortesi, "Watermarking techniques for relational databases: Survey, classification and comparison," *J. UCS*, vol. 16, no. 21, pp. 3164–3190, 2010.
[25] M. O. Rabin, "How to exchange secrets with oblivious transfer." *IACR Cryptology ePrint Archive*, vol. 2005, p. 187, 2005.
[26] N. Kumar, V. Katta, H. Mishra, and H. Garg, "Detection of data leakage in cloud computing environment," in *2014 International Conference on Computational Intelligence and Communication Networks*. IEEE, 2014, pp. 803–807.
[27] A. Kumar, A. Goyal, A. Kumar, N. K. Chaudhary, and S. S. Kamath, "Comparative evaluation of algorithms for effective data leakage detection," in *2013 IEEE Conference on Information & Communication Technologies*. IEEE, 2013, pp. 177–182.
[28] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," in *Advances in Neural Information Processing Systems*, 2019.
[29] G. E. Hinton and S. Roweis, "Stochastic neighbor embedding," *Advances in neural information processing systems*, vol. 15, pp. 857–864, 2002.
[30] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 535–548.
[31] N. Y. S. D. of Health. (2019) Hospital inpatient discharges (sparcs de-identified): 2015. [Online]. Available: https://health.data.ny.gov/Health/Hospital-Inpatient-Discharges-SPARCS-De-Identified/82xm-y6g8
[32] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell *et al.*, "{SCONE}: Secure linux containers with intel {SGX}," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 689–703.
[33] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using arm trustzone to build a trusted language runtime for mobile applications," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, 2014, pp. 67–80.
[34] K. H. Rosen and K. Krithivasan, *Discrete mathematics and its applications: with combinatorics and graph theory*. Tata McGraw-Hill Education, 2012.
[35] Wikipedia contributors, "p-value — Wikipedia, the free encyclopedia," 2019, [Online; accessed 23-May-2019]. [Online]. Available: https://en.wikipedia.org/wiki/P-value
[36] Machine Learning Group of ULB. (2018) Credit card fraud database. [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud
[37] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Annual International Cryptology Conference*. Springer, 2003, pp. 145–161.
[38] D. Eastlake and T. Hansen, "Us secure hash algorithms (sha and sha-based hmac and hkdf)," Internet Requests for Comments, RFC Editor, RFC 6234, May 2011, http://www.rfc-editor.org/rfc/rfc6234.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6234.txt
[39] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," Internet Requests for Comments, RFC Editor, RFC 8032, January 2017.
[40] A. Shabtai, Y. Elovici, and L. Rokach, *A survey of data leakage detection and prevention solutions*. Springer Science & Business Media, 2012.
[41] I. J. Cox, M. L. Miller, J. A. Bloom, and C. Honsinger, *Digital watermarking*. Springer, 2002, vol. 53.
[42] M. Kamran and M. Farooq, "A comprehensive survey of watermarking relational databases research," *arXiv preprint arXiv:1801.08271*, 2018.
[43] M. Backes, N. Grimm, and A. Kate, "Data lineage in malicious environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 178–191, 2016.
[44] V. Amrhein, S. Greenland, and B. McShane, "Scientists rise up against statistical significance," 2019.
[45] R. L. Wasserstein, A. L. Schirm, and N. A. Lazar, "Moving to a world beyond p ¡ 0.05," *The American Statistician*, vol. 73, no. sup1, pp. 1–19.
[46] H. Pike, "It's time to talk about ditching statistical significance," *Nature*, vol. 567, p. 283, 2019.

APPENDIX A
PROOFS OF LEMMAS

We give the full proofs of the lemmas listed in §VII.

Lemma A.1. *(Lemma VII.1) Given $Dist_L$ as the leaked Dist data objects, when $Dist_L$ contains objects from $Unique_i$ but not from $Drop_i$, the algorithm can detect a non-colluding guilty receiver R_i with false positive rate $FP_{nocollusion}$ and false negative rate $FN_{nocollusion}$ as shown in Equation 7.1.3 and Equation 7.1.4.*

Proof. When leaked objects contains $Unique_i$ objects and does not contain data from $Drop_i$, R_i is a leaker because a non-collusive guilty receiver R_i does not have knowledge of $Drop_i$ and cannot tell which objects are in $Unique_i$, so the $Dist_L$ should contain objects uniformly from $Unique_i$ and $Recv_i$. However, there are false positive cases when the sender is a leaker and the false positive rate is the probability

of "sender leaks objects just containing at least one $Unique_i$ object but no $Drop_i$ objects", which is:

$$FP_{nocollusion} = \frac{\binom{|Dist|-|Drop_i|}{|Dist_L|} - \binom{|Dist|-|Drop_i|-|Unique_i|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \approx \frac{\binom{\frac{1}{2}|Dist|}{|Dist_L|} - \binom{(\frac{1}{2}-\frac{1}{2^N})|Dist|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \tag{7.1.3}$$

The false negative is the probability of "R_i leaks objects just containing no data from $Unique_i$", which is:

$$FN_{nocollusion} = \frac{\binom{|Recv_i|-|Unique_i|}{|Dist_L|}}{\binom{|Recv_i|}{|Dist_L|}} \approx \frac{\binom{(\frac{1}{2}-\frac{1}{2^N})|Dist|}{|Dist_L|}}{\binom{\frac{1}{2}|Dist|}{|Dist_L|}} \tag{7.1.4}$$

where by 1-out-of-2 OT, $|Unique_i|$ is about $(\frac{1}{2})^N|Dist|$ and $|Drop_i| = |Recv_i|$ is strictly $\frac{1}{2}|Dist|$. □

Lemma A.2. *(Lemma VII.2) Given $Dist_L$ as the leaked Dist data objects, when $Dist_L$ contain objects from $SenderOnly$, the algorithm can detect a non-colluding guilty sender with false positive rate $FP_{sender} = 0$ and false negative rate $FN_{nocollusion}$ as shown in Equation 7.1.5.*

Proof. Since only the sender knows $SenderOnly$, when an object from it is found in the $Dist_L$, the sender must be a leaker and false positive rate $FP_{sender} = 0$. However, there is a chance that when sender leaks data, the leaked objects just do not contain any data from $SenderOnly$, leading to false negative result. The false negative rate is the probability of "sender leaks object just containing no data from $SenderOnly$"

$$FN_{sender} = \frac{\binom{|Dist|-|SenderOnly|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \approx \frac{\binom{(1-\frac{1}{2^N})|Dist|}{|Dist_L|}}{\binom{|Dist|}{|Dist_L|}} \tag{7.1.5}$$

where by 1-out-of-2 OT, $|SenderOnly|$ is about $(\frac{1}{2})^N|Dist|$. □

Lemma A.3. *(Lemma VII.3) If there exist any paired patterns k and k' on R_i where $V_L[k]$ has a significant difference from $V_L[k']$, we draw the conclusion that receiver R_i contributed to the leakage with false positive rate being the p-value obtained in the binomial test.*

Proof. We give a proof by contradiction. Assuming R_i does not contribute to the leakage, since the corrupted receivers cannot know $Recv_i$, their leaked data randomly contains both objects received and not received by R_i. In other words, corrupted receivers cannot tell objects of pattern k and k' apart.

Therefore, $V_L[k]$ and $V_L[k']$ are expected to be statistically close to the distribution between $V_D[k]$ and $V_D[k']$, which is about $1:1$ because of the 1-out-of-2 OT. This contradicts the statement that $V_L[k]$ has a significant difference from $V_L[k']$, so receiver R_i is corrupted and has contributed to the leakage.

The false positive rate is the probability that a significant difference exists when R_i is honest, which is represented by the p-value obtained from the binomial test. □

APPENDIX B
EXPERIMENT OF COLLUSIVE LEAKAGE

In this appendix section, we give the full description of the combination mechanisms used in collusive leakage experiments.

For 2-receiver leakage, in the "leakage 1" of Figure 7(c), the leaked objects are randomly selected from

$$(Recv_2 \cap Recv_3^{\complement}) \cup (Recv_2 \cap Recv_3),$$

where R_2 and R_3 collude. In "leakage 2" and "leakage 3", the combinations are as follows

$$Recv_3 \cap Recv_4, Recv_4 \cap Recv_5$$

where R_3 and R_4, R_4 and R_5 are leakers, respectively.

For 3-receiver leakage (Figure 7(d)), leakers perform a relatively complex set operation and the resulting leaked data set can be described as:

$$(Recv_2 \cap Recv_3 \cap Recv_5^{\complement}) \cup (Recv_2^{\complement} \cap Recv_3 \cap Recv_5),$$

$$Recv_2 \cap Recv_3 \cap Recv_4,$$

$$(Recv_2 \cap Recv_3^{\complement} \cap Recv_4) \cup (Recv_2^{\complement} \cap Recv_3 \cap Recv_4),$$

in leakage event 1, 2, and 3, respectively. As shown, the leakers are receivers corresponding to the received datasets used in each leakage.

For 6-receiver leakage (Figure 7(e)), the leaked data objects are randomly selected from the datasets after following operations in "leakage 1", "leakage 2" and "leakage 3", respectively.

$$(Recv_1 \cap Recv_2 \cap Recv_3 \cap Recv_4^{\complement} \cap Recv_5^{\complement} \cap Recv_6^{\complement})$$

$$\cup (Recv_1 \cap Recv_2^{\complement} \cap Recv_3^{\complement} \cap Recv_4 \cap Recv_5 \cap Recv_6)$$

$$Recv_1 \cap Recv_2 \cap Recv_3 \cap Recv_4 \cap Recv_5 \cap Recv_6$$

$$Recv_1 \cap Recv_2^{\complement} \cap Recv_3 \cap Recv_4^{\complement} \cap Recv_5 \cap Recv_6^{\complement}$$