AuditShare: Sensitive Data Sharing with Reliable Leaker Identification

Zhiyi Zhang  
UCLA  
zhiyi@cs.ucla.edu

Yu Guan  
Peking University  
shaxiyg@pku.cn

Xinyu Ma  
UCLA  
xinyu.ma@cs.ucla.edu

Lixia Zhang  
UCLA  
lixia@cs.ucla.edu

Abstract—As Personally Identifiable Information (PII) data sharing among multiple parties becomes increasingly common, so does the potential for data leakage. As required by new data protection regulations and laws, when PII leakage occurs, one must be able to reliably identify the leaking sources. Existing solutions utilize watermark technologies or data object allocation strategies to differentiate the data shared with different parties to identify potential leakers. However, these solutions lose their effectiveness under several attack scenarios, e.g., a data sender may leak the data and a receiver may deny the reception of certain shared data. Worse yet, multiple receivers might collude and apply a set of operations such as intersection, complement, and union to their received datasets before leaking them, making the task of leaker identification even more difficult.

In this paper, we propose AuditShare, a PII dataset sharing system with reliable leaking source identification. Firstly, taking advantage of the intrinsic properties of PII data, AuditShare allocates data objects to individual sharing parties by PII attributes. Secondly, AuditShare obliviously transfer data between the sender and each receiver and uses a Merkle Tree as an immutable record of the sharing. Thirdly, a knowledge-based identification algorithm is proposed to identify a guilty sender or colluding/non-colluding receivers. Through our evaluation, we show that: (i) With a modest amount of leaked data, AuditShare can accurately (accuracy > 99.99%) and undeniably identify all the guilty parties in different cases; (ii) It only takes 0.5 second to share 100,000 data objects in AuditShare, which is practical in real-world deployment.

I. INTRODUCTION

As the society becomes increasingly online, sharing Personally Identifiable Information (PII) [17] becomes increasingly needed. For example, according to the US National Coordinator for Health Information Technology (ONC), more than 60% hospitals in the US share patient information with health care providers [6] to benefit patients, e.g., to avoid duplicate tests. Another common use case is schools sharing student information with online educational applications for personalized services, according to the U.S. Department of Education [28].

PII data contains PII attributes, e.g., full name, email address, etc., which can be used to identify a particular person, so the protection of PII data has become a major concern. For example, according to [29], 342 breaches of health information have been reported in the first nine months in 2019, affecting more than 39 million individuals in the US. Under this background, Europe’s General Data Protection Regulation (GDPR) [9] and California’s Consumer Privacy Act (CCPA) [7] all contain provisions and requirements for PII data protection. If one is identified as a PII leaker, he/she will face serious punishment.

As a result, reliable leaking source identification in PII dataset sharing helps to prevent data breaches and is a key enabler for the enforcement of new regulations. The problem can be formally described as follows: a data owner (called the sender) shares its PII dataset with a number of agents (called receivers), where both sides agree that the shared data should never be disclosed to any third party. When a PII dataset leakage happens, offenders must be identified.

A number of solutions to the leaker identification problem in dataset sharing have been developed in recent years. At this time, the basic approach is to differentiate the datasets received by different receivers in the data sharing process. For example, the sender can allocate different portions of data across receivers [18][19][10][16], or watermark the original data [2][25][21][5] before sharing it with each receiver. In this way, the sender can trace the leaker(s) by matching the data entries or checking the watermark of the leaked data.

However, the traceable differentiation proposed in the existing works loses its effectiveness in identifying leakers in several cases, making these solutions less practical.

1) Because of dishonest employees [27][26] or vulnerable firewalls [5], the sender itself could be the leaker. In this case, existing solutions do not work because they rely on the sender to create differentiation among receivers. Worse yet, it is even possible for the sender to frame a receiver because the sender has the upper hand by virtue of being fully aware of the differentiation.

2) Collusion can help leakers to figure out the differentiation by comparing their received copies, so that leakers can manipulate the differentiation (e.g., data transformation, numeric value randomization) or only leak the overlapping data in order to circumvent leaker identification.

3) The differentiation can be denied if there is a lack of evidence for which data was received by which receiver.

In this paper, we propose AuditShare, a data-sharing system with reliable leaker identification in PII dataset leakage. Intuitively, AuditShare allocates data objects by data entry’s PII attribute(s) across the receivers in such a way that no single party is fully aware of the allocation results; at the same time, nobody can deny or lie about which data objects it has received. More specifically, the design of AuditShare includes
three basic components, addressing the issues in the existing solutions.

- **Obliviously Allocating Objects by PII Attributes.** The sender organizes the raw dataset into data objects by PII attribute(s) and then obliviously allocate data objects among receivers. With Oblivious Transfer (OT) \cite{22}, data objects will be partially shared with each receiver in such a way that the sender cannot know which objects have been received and the receiver is not aware of the remaining objects. To compensate the data loss caused by OT, AuditShare leverages fake but indistinguishable data objects \cite{19} to ensure that each receiver can receive all the original data objects eventually.

- **Identifying Leakers based on Leakers’ Knowledge.** When a leakage happens, AuditShare can identify the leaker or a number of colluding leakers by inferring the knowledge of the leaker(s) from the leaked data objects.

- **Immutably Recording the Data Sharing.** During the data sharing, AuditShare immutably records the result of the data sharing, preventing any party from lying or denying what data objects have been transferred/received.

We implement a prototype system of AuditShare based on 1-out-of-2 OT extension \cite{3} and evaluate it with real-world datasets. It only takes about 0.5 seconds to transfer 100,000 data objects (each object is about 90 Bytes) to a receiver, showing AuditShare’s practicality. By checking a modest number of leaked data objects, AuditShare obtains > 99.99% identification accuracy in leakages where there exists (i) a guilty sender, (ii) a non-colluding guilty receiver, or (iii) a number of colluding receivers.

**Contributions** The contributions of our work are threefold.

1) We propose a OT-based sharing system for PII data and a way to immutably record sharing results. To the best of our knowledge, AuditShare is the first allocation based sharing system that is capable of identifying a guilty sender.

2) We propose a knowledge-based leaker identification algorithm that is able to identify colluding leakers.

3) We implemented a prototype of AuditShare and evaluated its efficiency and effectiveness, showing that AuditShare can accurately and undeniably detect all leakers.

**Roadmap** In the rest of the paper, we use an example scenario where a school shares student information with several online educational services to show how AuditShare works. We introduce the system model in Section II and provide an overview of AuditShare in Section III. After that, we introduce the design of each component in AuditShare in Section IV and VI. We evaluate our work in Section VII and have a discussion in Section VIII. We introduce the related work in Section IX and conclude our work in Section X.

---

1When the leaker entirely erases PII attributes from the leaked dataset, our proposed knowledge-based algorithm cannot identify the leakers; nevertheless, in this case, the leaked dataset becomes non-PII data according to data protection regulations.

---

**II. System Model**

In this work, we consider a real world example described as followed to illustrate AuditShare. A school $S$ as the sender wants to share its student information with $N$ different online educational services $R_1, R_2,...,R_N$ as receivers so that students can enjoy personalized services based on their own situation. The dataset being shared is a structured dataset that contains PII, such as student ID, student name, etc. When student information from $S$ is found unexpectedly (e.g., found in public Internet), the goal is to identify the party/parties responsible for the leakage.

**A. Threats Model**

In this paper, we consider both the unintentional and intentional data leakage. So, based on the data sharing scenario, we define the threats model as follows:

- **A Guilty Sender.** The data sender may accidentally disclose its information, e.g. due to dishonest employees \cite{27} or a vulnerable campus firewall. It is also possible that, when a leakage occurs, the sender may frame a receiver by taking advantage of its complete knowledge about the data sharing.

- **Guilty receivers with/without collusion.** We assume that a receiver may leak part or all of its received data, and there may be more than one guilty but non-colluding receivers at the same time. Moreover, multiple parties may collude to gain advantages over the leakage identification system and escape the responsibility. Specifically, colluding parties can share their received dataset and disclose a combination of their datasets, where we assume the combination includes: (i) taking the union, intersection, or complement of any number of available datasets and (ii) randomly picking a number of data objects. We consider the cases when the sender also participates in the collusion (Section VIII-D).

- **Data Manipulation.** To avoid being caught by watermark-based identification approaches, malicious receivers can manipulate their received dataset, e.g., changing students’ numeric attributes like grades, ages, etc. However, we assume such manipulation will not erase the PII, otherwise such leakage may not be considered a crime by laws.

- **Repudiation.** After data leakage, guilty parties may deny the facts of what has been sent and received. For example, to escape the responsibility, a receiver who leaked the data can deny the receipt of a specific portion of the data and a guilty sender may lie about what has been sent.

Specifically, in respect of the collusive leakage, our proposed system aims to capture each colluding receiver who contributed to the leakage. If a corrupted receiver $R$ does not contribute to a leakage, that is, if other colluding receivers can have sufficient knowledge to make such a leaked dataset even without $R$’s participation, AuditShare cannot identify $R$ as a leaker.
B. Assumptions

We make the following assumptions when designing AuditShare:

- **Leaked Data contains Identifying Information.** We assume the leaked data is PII data, which means the leaked data contains PII attributes (e.g., Social Security Number, full name). AuditShare cannot identify leaking sources from leaked data with no PII attributes. In fact, such a leakage is not considered as a PII leakage crime by regulations like GDPR; identifying leaking sources in this scenario is out of this paper’s scope.

- **Trusted Third Parties.** In AuditShare, we assume an honest-but-curious Notary who will certify the sharing process and a fully-trustworthy Arbitrator who will run the leaker identification algorithm in the case of a leakage. In AuditShare, the Notary cannot gain any information about the data being shared. If there is no leakage, the Arbitrator is not needed; however, when there is a leakage, the Arbitrator will is able to learn all the data objects.

- **Adequate Number of Leaked Data Objects.** We assume an adequate number of leaked objects are known to AuditShare’s identification algorithm in the event of a leakage. A very small number leaked objects are not sufficient for AuditShare to identify leakers.

- **Fake Data’s Influence on Receiver Applications.** We assume these fake but indistinguishable data objects will not affect receiver applications much, and thus AuditShare does not apply to application scenarios that strictly require no fake objects at receiver side. A more detailed discussion on this assumption is in Section VIII-B.

III. AuditShare Overview

Our proposed system functions by allocating data objects by their PII attributes across receivers with 1-out-of-2 Oblivious Transfer (OT). 1-out-of-2 OT is a cryptographic protocol in which a sender sends two messages and the receiver will receive one; at the same time, the sender cannot know which message has been received and the receiver is oblivious to the value of the other message. The use of OT prevents the sender from fully knowing the allocation result, so the sender cannot frame a receiver. During the sharing, the whole process will be proved immutably by a Merkle-Tree-based credential signed by the notary, preventing any party from denying or lying about the data allocation. In the event of a data leakage, AuditShare can identify all leakers, including colluding receivers or a guilty sender, with high accuracy by reversely inferring leakers’ knowledge from the leaked data.

AuditShare consists of three main steps.

1. **Data Preprocessing.** In AuditShare, data preprocessing does two main tasks. (i) A sender groups original data entries with same PII attributes into data objects, each of which is associated to an individual and will be shared as a whole in a later sharing process. (ii) Then the sender prepares a number of fake data objects, which are used for potential leaker detection. A sufficient number of fake data objects are required to guarantee a high accuracy of leaker identification; the more fake data objects are used, the higher the accuracy can be achieved.

2. **Data Sharing.** After Data Preprocessing, a 1-out-of-2 OT is applied for the sender to share the data objects. Importantly, each receiver will get all the real data objects and a random subset of fake data objects while each receiver’s collection of fake data objects keeps unknown to both the sender and the receiver. In this process, a trust-but-curious Notary will witness the transfer and publish a certificate as an immutable proof of the sharing result.

3. **Knowledge-based Leaker Identification.** In a data leakage, an Arbitrator (e.g., a court, a government agent, or even a smart contract) will trigger the leaker identification process. All the involved parties will provide the Arbitrator with their received data objects and the Notary will disclose the confidential part of sharing certificates to the Arbitrator. With all the claimed received data objects verified, the Arbitrator then triggers the algorithm to identify all the guilty parties. In detail, the algorithm starts with the hypothesis that a party is not guilty. Under this hypothesis, the algorithm uses inferential statistics to evaluate the probability of having such a leaked dataset, thus judging whether the party is guilty or not.

Through these three steps, the auditability is built into the sharing system: By employing 1-out-of-2 OT, each received copy is a different collection from all others; even a combination of several received datasets is also distinguishable from other combinations. Additionally, no single party is aware of the allocation among receivers. With the certificate published by the Notary at the end of a sharing process, no party is able to cheat in a later arbitration. Due to these properties, in the case of a data leakage, our knowledge-based identification algorithm is able to effectively and undeniably identify all leaking sources. By introducing fake but indistinguishable fake objects, each received dataset contains all the data objects in the sender’s original dataset. This ensures data receivers are able to provide services to all involved users (e.g., all students of school S).

IV. AuditShare: Dataset Preprocessing

The purpose of data preprocessing is to prepare data objects for a later sharing process. Each data object is a collection of data associated to an individual, e.g., each student in school S. In order to allocate different data to receivers, the sender will also generate fake data objects, e.g., data associated with nonexistent students. Such a process can be illustrated by Figure 2.
A. Identifying Data Objects

A sender first groups raw data entries (e.g., database of student profiles) with the same PII attribute into one data object. Therefore, each object may contain one or more data entries; to combine multiple entries into one object, the sender can simply concatenate multiple entries or perform join operations among tables on PII attributes. When sharing data with a receiver, a data object is the smallest unit of the allocation; that is, all data entries in each object will be shared as a whole.

To be more specific, the sender will need to select PII attributes as the basis to group the data entries. Considering a PII dataset, there are two possible scenarios: there exists either a single PII attribute (e.g., student ID) or multiple PII attributes (e.g., student ID and full name). In the former case, the single attribute will be selected as the basis, while in the second case, a concatenation of all PII attributes will be used (Figure 3). In a potential leaker identification process, in case a leaker strips or changes some of PII attributes in leaked data, a leaked object is matched if any of its PII attributes matches the original object.

B. Preparing Fake Data Objects

In AuditShare, the leaker identification is based on data allocation. To eliminate data loss caused by the allocation and improve the identification accuracy, the sender will generate fake data objects for the purpose of allocation. These fake objects do not correspond to real entities but appear realistic to data receivers. The fake objects can be generated by different means (e.g., differential privacy approaches) and the details are out of the scope of this paper.

Since fake objects are used for allocation, in general, the more fake objects are generated, the higher the leaker identification accuracy can be achieved. In AuditShare, we define the lower bound of the quantity of fake objects to be twice the size of real data objects. This is to ensure no real data loss in a later OT as shown in Section [V]. However, a minimum quantity of fake objects may not ensure a good identification accuracy. On the other hand, too many fake objects are not needed because the accuracy is high enough and extra objects will lead to a high sharing overhead.

In AuditShare, the proper fake object quantity depends on the total number of receivers and the expectation on identification accuracy. For example, with 6 receivers and 20,000 fake objects, when 5% of the received data is leaked, a 99.99% accuracy in leaker identification can be guaranteed (Section [VII]). However, when there are 9 receivers in total, to achieve the same level of accuracy, about 200,000 fake objects are needed. Section [VII-C] will discuss how to decide the size of fake objects when an expected identification accuracy and the total number of receivers are given.

V. AuditShare: Data Sharing

After the sender finishes preprocessing data, all original and fake data objects will be shared through 1-out-of-2 Oblivious Transfer (OT), and as a result, each receiver will obtain all real objects and a random part of fake objects while the allocation of fake objects among receivers keeps unknown to the sender, preventing the sender from framing an innocent receiver.

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 selected information. In a classic 1-out-of-2 OT, the sender inputs two messages $m_0$ and $m_1$ while the receiver has a secret bit $b \in \{0, 1\}$. The receiver will learn $m_b$ while $b$ remains unknown to the sender at the end of the OT.

In this paper, we use the 1-out-of-2 OT. Since the size of data objects in a PII dataset can be very large, a large number of OTs may be involved in AuditShare. Therefore, in our prototype implementation and real-world deployment, an OT extension will be used for better efficiency. As pointed in [A], the performance bottleneck of oblivious transfer has been in the network rather than the computation overhead.

B. Data Sharing with OT

In a situation when fake objects are exactly twice the size of real objects, the sender binds each real data object $d_i$ with two different fake objects $f_{i,0}$ and $f_{i,1}$. After that, the sender creates two tuples $m_0 = <d_i, f_{i,0}>$ and $m_1 = <d_i, f_{i,1}>$ that will be transferred obliviously to the receiver. As noticed, $m_0$ and $m_1$ have the same real object but contain different fake objects. As a result of 1-out-of-2 OT, a receiver can only get $m_b$ ($b = 0$ or 1 represents the receiver’s choice). $m_b$
contains two objects that are indistinguishable to the receiver. All remaining data objects will be shared in the same way.

When fake data objects are more than twice the size of real objects, extra fake objects should be uniformly distributed into each pair of \( m_0 \) and \( m_1 \).

In AuditShare’s OT-based sharing, all messages should have the same length in order to prevent side channel attacks. For this purpose, simple bit padding mechanisms can be used.

\section*{C. Provable Sharing}

To make an immutable proof of a data sharing process, AuditShare requires a trusted third party as a Notary to generate a credential, so that in a potential arbitration phase, no party can deny the result of the sharing.

In fact, having a notary in commercial data sharing has become a common practice. In AuditShare, a Notary will generate a certification on what data objects have been transferred without gaining any knowledge of the exact PII information. To be more specific, the sender, all receivers, and the Notary will obey the following protocol during a data sharing process.

1) Before a sharing, the sender creates a three-layer Merkle tree, called a Sending Tree, which is defined as,
\[
SendingTree = H(\ldots||H(H(m_{i,0})\|H(m_{i,1}))||\ldots)
\]
for each message pair \( i \) sent by the sender. 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 its Sending Tree, which are sufficient to snapshot all the OT messages sent.

2) Before the sharing, a receiver provides the Notary with choices that will be used in the OT.

3) After the sharing, the receiver calculates a two-layer Merkle tree, called a Receiving Tree, on all the received messages. A Receiving tree is defined as,
\[
RecivingTree = H(\ldots||H(m_{i,b})||\ldots)
\]
for each message pair \( i \) sent by the sender, where \( b \) is receiver’s choice in the 1-out-of-2 OT of \( i \). The receiver then sends the root node value of its Receiving Tree to the Notary.

4) The Notary computes another Receiving Tree based on the Sending 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.

If the comparison in step 4 is successful (i.e., two identical values), the sharing is finished and the Notary will issue a certificate. Such a certificate contains (i) an encrypted root node value of the Sending Tree, along with (ii) an encrypted list of choices made by the receiver. 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 Notary as a witness and a new sharing process should be started.

In the sharing process, a Notary obtains zero knowledge of the actual data being transferred because of the one-way hash function used in Merkle tree.

\section*{VI. LEAKER IDENTIFICATION AFTER DATA LEAKAGE}

AuditShare identifies leaks by analyzing the leaked data and finding the involved parties whose knowledge is sufficient to create such a leaked dataset. Specifically, since all receivers will have the complete set of real objects, all the objects discussed in this section refer to fake data objects. The notations used in this section are listed in Table I.

To provide some intuitive ideas of how AuditShare’s identification works, we show four leakage examples in Figure 5. The dots in the figure are the leaked fake data objects while each circle represents all fake data objects 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, as in Figure 5a, in receiver \( R_3 \)’s circle, we can easily infer \( R_3 \) to be the leaker. When leaked objects are like Figure 5b, 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 5c it is trivial to deduce the leaked data is from \( \text{Recv}_1 \cap \text{Recv}_3 \cap \text{Recv}_2^{0} \), which means all three receivers collude in the leakage. Figure 5d is a more complicated scenario where leaked objects are all from \( \text{Recv}_1 \cap \text{Recv}_3 \) 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) **Confirming Sharing.** The involved sender and receivers provide the Arbitrator with their datasets sent/received. The Notary also sends decryption keys to the Arbitrator to access the confidential part of certificates. In this way, the Arbitrator is able to ensure no party can deny or lie about the result of sharing.

2) **Identifying Leakers.** Once the sharing result is confirmed, the Arbitrator knows the exact allocation of fake data objects among receivers. According to the knowledge used to construct the leaked data, AuditShare evaluates the innocence of every involved party and identifies the leaker(s) with a very low error rate. The first step is to ensure that the Arbitrator can get the correct information of the sender and each receiver’s knowledge while the second step is to identify the leakers based on each involved party’s knowledge.

The first step is straightforward: the Arbitrator checks the Merkle tree and receiver choices against the datasets provided by the sender and receivers. Therefore, in the remaining of

| Table I: Notation Table |
|--------------------------|
| \( N \) | number of data receivers |
| \( D \) | collection of all the fake data objects |
| \( D_{\text{leak}} \) | leaked fake data objects |
| \( \text{Unique}_i \) | fake data objects received by receiver \( R_i \) |
| \( \text{Drop}_i \) | fake data objects dropped by \( R_i \) in the OT |
Figure 5: Knowledge-based Leaker Identification (Circles are receivers’ knowledge and red points are leaked data objects)

In this section, we will describe the identification process, that is, given the exact data allocation, how to identify (i) a non-colluding leaker (the sender or a receiver), and (ii) all colluding leakers.

### A. Identifying Non-colluding Leakers

**Algorithm Summary** AuditShare takes the following steps to identify each non-colluding leaker, which can be the sender or a receiver.

1. The algorithm first figures out each receiver’s (i) unique data objects that are never received by other receivers and (ii) the objects that are dropped during the OT.
2. For each receiver, the algorithm detects (i) the overlap between leaked objects and unique objects and (ii) the overlap between leaked objects and dropped objects. If the first is nonempty but the second is empty, we consider this receiver is guilty.
3. Finally, to verify the sender’s honesty, the algorithm calculates the objects that are only known to the sender (i.e., dropped by all the receivers during OT). 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 fake objects that are never received by any other parties, and therefore these objects can serve as the clue of non-colluding leakers (the sender or non-colluding receivers). We use $Unique_i$ to represent the objects that are only received by a receiver $R_i$ and $SenderOnly_i$ to refer to those objects that are not received by any receiver. Moreover, by the property of OT, objects dropped by each receiver $R_i$ during the OT, which are denoted by $Drop_i$, are not known to any party.

Therefore the following statements hold when there is no collusion. Firstly, $R_i$ is a leaker by a large probability if leaked data objects $Data'_{i}$ contain objects that are also in $Unique_i$, but no objects from $Drop_i$. Specifically, $Unique_i$ can be calculated by

$$Unique_i = Recev_i \cap \left( \bigcup_{j \neq i} Recev_j \right)^c \quad (6.1.1)$$

Secondly, $S$ must be a leaker if $Data'_{i}$ contains objects from $SenderOnly_i$. Specifically, from the data allocation process, we have:

$$SenderOnly_i = (Drop_1 \cap Drop_2 \cap \ldots \cap Drop_N) \quad (6.1.2)$$

where $EO_i$ represents the $u$ exclusive objects allocated to $R_i$.

**Pseudo Code and Proof** Algorithm 1 shows the pseudo-code of AuditShare’s algorithm of non-colluding leaker identification.

**Algorithm 1** Identifying a Non-colluding Leaker

**Require:**
1. Leaked fake data objects $Data'$
2. Each receiver $R_i$’s fake data objects, $Recv_i$
3. Fake data objects dropped by $R_i$ in the sharing, $Drop_i$

**Ensure:** Non-colluding leaker set $L$;

1. Calculate each receiver’s unique objects $Unique_i$ from $Recv_1, Recv_2, \ldots, Recv_N$
2. for $i$ from 1 to $N$
3. 
4. 
5. if $Data' \cap Unique_i \neq \emptyset$ then
6. 
7. if $Data' \cap Drop_i \neq \emptyset$ then
8. 
9. Add $S$ into $L$
10. end if
11. end for
12. Calculate objects known to the sender only: $SenderOnly$
13. if $Data' \cap SenderOnly \neq \emptyset$ then
14. Add $S$ into $L$
15. end if

We prove the effectiveness of Algorithm 1 here. Specifically, we first prove that in a non-collusive leakage, each leaker’s unique objects can be observed by a high probability (Lemma VI.1 and Lemma VI.2) with an adequate number of leaked data objects. Based on this, we prove the algorithm can successfully identify the non-colluding leakers with a high success rate.

**Lemma VI.1.** If there exists a non-colluding guilty receiver $R_i$, given $Data'$ as the leaked fake data objects, $Data'$ will contain objects from $Unique_i$ with a probability $P$ as shown in Equation (6.1.3).
Therefore, we have the probability of “at least one object of leaked dataset Data’ is in Unique’” to be:

\[
P(\text{Data’} \cap \text{Unique’} \neq \emptyset) = 1 - \frac{|\text{Recv}_i| - |\text{Unique}_i|}{|\text{Data’}|}
\] (6.1.3)

According to the data allocation process with 1-out-of-2 OT, we have

\[
|\text{Unique}_i| = \left(\frac{1}{2}\right)^N \times |\text{Data}|
\]

Importantly, \(P\) is large enough even when a guilty receiver \(R_i\) only leaks a small portion of data objects from \(\text{Recv}_i\). For example, with 3 receivers in total, when \(|\text{Data}| = 20000\), the probability \(P > 1 - 3 \times 10^{-13}\) even when \(R_i\) leaks 1% of its received data (i.e., 100 fake data objects).

**Lemma VI.2.** If there exists a non-colluding guilty sender \(S\), given Data’ as the leaked fake data objects, Data’ will contain objects from SenderOnly with a probability \(P\) as shown in Equation (6.1.4).

**Proof.** The proof is the same as Lemma VI.1. The probability of an non-empty intersection of Data’ and SenderOnly is

\[
P(\text{Data} \cap \text{SenderOnly} \neq \emptyset) = \frac{|\text{Data} - |\text{SenderOnly}|}{|\text{Data’}|}
\] (6.1.4)

According to the property of 1-out-of-2 OT, we have:

\[
|\text{SenderOnly}| = \left(\frac{1}{2}\right)^N \times |\text{Data}|
\]

Similarly, \(P\) is large enough even when \(S\) only leaks a small portion of data objects from Data.

**Theorem VI.3.** Algorithm [7] can identify all non-colluding leakers with success rate shown in Equation (6.1.3) and Equation (6.1.4).

**Proof.** Code line 4-11 is to identify non-colluding guilty receivers and line 12-15 is to identify a guilty sender. According to Lemma VI.1 and Lemma VI.2, Algorithm [1] can detect the sender with success rate shown in Equation (6.1.4) and detect the non-colluding receiver with success rate shown in Equation (6.1.3).

**B. Identifying Colluding Leakers**

**Algorithm Summary** AuditShare identifies colluding leakers by the following steps.

1) The algorithm first represents all the data objects with a vector, in which each index represents a description (called a pattern) of each receiver’s state of receipt and the value of each element represents the number of objects that fall into this set. Similarly, the algorithm transforms all leaked objects into another vector.

2) For each receiver, the algorithm then infers whether this receiver contributed to the collusive leakage by comparing the leaked-object vector with the all-object vector with a statistical test called binomial test. Intuitively, the object distribution in the leaked-object vector should not be significantly different from that in the all-object vector if this receiver did not contribute to the leakage.

**Algorithm Rationales** In a more complicated scenario where multiple receivers collude, the leaked data can still reflect leakers’ knowledge because the leakers cannot know the data objects received by honest receivers. Following this principle, AuditShare 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 \(\text{Recv}_i\) should be close to statistically identical, due to the 1-out-of-2 OT used in the sharing process. If they are not, then \(R_i\) is likely a colluding leaker.

To facilitate the explanation of the algorithm, we first introduce the Pattern Matrix \(\text{Pat}\), where each row represents a type of allocation among receivers. Given a number of receivers \(N\), \(\text{Pat}\) is a \(2^N \times N\) matrix whose value is:

\[
\text{Pat} = \begin{bmatrix}
0 & 0 & \ldots & 0 & 0 \\
1 & 0 & \ldots & 0 & 0 \\
0 & 1 & \ldots & 0 & 0 \\
0 & 0 & \ldots & 1 & 1 \\
1 & 1 & \ldots & 1 & 1 \\
\end{bmatrix}
\]

The value of each row is a reverse of row index’s binary value. In each row \(j\), we define each element \(\text{Pat}[j][i] = 1\) to represent that \(R_i\) has received the set of data objects and \(\text{Pat}[j][i] = 0\) if this object has been dropped by \(R_i\) in OT. For example, we have:

\[
\text{Pat}[0] \rightarrow \text{Drop}_1 \cap \text{Drop}_2 \cap \ldots \cap \text{Drop}_N
\]

which represents the pattern of “dropped by all receivers”. \n
\[
\text{Pat}[2^N] \rightarrow \text{Recv}_1 \cap \text{Recv}_2 \cap \ldots \cap \text{Recv}_N
\]

which represents the pattern of “received by all receivers”. In the Pattern Matrix, we call two row indexes (i.e., patterns) \(k\) and \(k’\) are paired on \(R_i\) if they are only different at \(i\)’s slot. For example, \(\text{Pat}[0]\) and \(\text{Pat}[1]\) are paired rows on \(R_0\).

Since each leaked fake data object can be categorized into a pattern, which is a row index in the Pattern Matrix, all leaked fake objects can be represented by a Dataset Vector, denoted by \(V\).

\[
V = [a_0, a_1, \ldots, a_i, \ldots, a_{2^N}]
\]

where each element \(V[i]\) represents the number of data objects that fall into the \(i\)th pattern in the Pattern Matrix. 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 \(R_0\). Similarly, in a Dataset Vector, we call two
elements $V[k]$ and $V[k']$ are paired if they are indexed by two paired patterns $k$ and $k'$.

Importantly, for two paired indexes (i.e., patterns) $k$ and $k'$ on receiver $R_i$, if $R_i$ makes no contribution to the leakage, a leaked data object of pattern $k$ or $k'$ should have:

$$P(d \text{ is of pattern } k) = 0.5$$

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 whether the difference is significant enough, we employ a statistic test, binomial test \cite{23}, to obtain a p-value \cite{30}. In AuditShare, a binomial test can be represented as follow.

$$p\text{-value} = \text{Binomial\_Test}(V[k], V[k'], 0.5)$$

To be specific, p-value indicates the probability of having a distribution equal to or more extreme than $(V[k], V[k'])$, under the hypothesis that $R_i$ is honest, i.e., $P(d \in V[k]) = 0.5$. For example, when $(V[k], V[k']) = (14, 6)$, the p-value is 0.115, meaning 11.5% of the time one would expect to have a distribution equal to (14, 6) or more extreme (e.g., (15, 5)) if $R_i$ is honest. When p-value is less than a pre-defined value $\alpha$ (e.g., $\alpha = 0.05$), it is considered that the hypothesis does not hold and thus $R_i$ is one of the leakers.

A noteworthy fact is that AuditShare cannot identify a corrupted receiver when this receiver makes no contribution to the leakage, that is, other corrupted parties can still have sufficient knowledge to make such a leaked dataset even without this receiver’s participation.

**Pseudo Code and Proof** Algorithm 2 shows the pseudo code of the AuditShare’s identification algorithm for colluding leakers.

**Algorithm 2** Leakers identifying algorithm.

```
Require:
1: Leaked fake data objects
2: The allocation of all the fake data objects
3: A significance level: $\alpha$, $\alpha = 0.05$ by default
Ensure: Leakers set $L$ with an accuracy value $A$
4: Transform leaked data into a Dataset Vector $V$
5: Initialize a vector $PValues[N] = \{1, 1,...,1\}$
6: for $i$ from 1 to $N$ do
7:     for $k$ from 1 to $2^N$ do
8:         Find $k'$ which is paired with $k$
9:         $p\_value_i = \text{Binomial\_Test}(V[k], V[k'], P = 0.5)$
10:        if $p\_value_i < \alpha$ then
11:            Add $R_i$ into $L$
12:            $PValues[i] = \min(PValues[i], p\_value_i)$
13:        end if
14:     end for
15: end for
16: Calculate accuracy as $A = \prod_{i \in L} (1 - PValues[i])$
```

Now we prove the effectiveness of Algorithm 2 with a brief proof. Specifically, we first prove that when a receiver contributed to the leakage, its contribution can be reflected by the object distribution of two paired elements in Dataset Vector (Lemma VI.4). After that, we can prove the correctness of Algorithm 2 (Theorem VI.5).

Let $V$ be the Data Vector for the leaked dataset.

**Lemma VI.4.** If there exist any paired patterns $k$ and $k'$ on $R_i$, where $V[k]$ has a significant difference from $V[k']$, we draw the conclusion that receiver $R_i$ contributed to the leakage with an error rate of 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 $R_i$. Therefore, $V[k]$ and $V[k']$ are expected to be the same because corrupted receivers cannot tell objects of pattern $k$ and $k'$ apart. This contradicts the statement that $V[k]$ has a significant difference from $V[k']$, so receiver $R_i$ is corrupted and has contributed to the leakage.

The error 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.

**Theorem VI.5.** Algorithm 2 can identify all the colluding receivers with the accuracy obtained in code line 18.

**Proof.** In Algorithm 2 line 6 to line 15 check for each receiver $R_i$, whether there exists paired $V[k]$ and $V[k']$ such that $V[k]$ is significantly different from $V[k']$. According to Lemma VI.4, each corrupted receiver who contributed to the leakage can be identified with an error rate to be the p-value. Therefore, for each identified corrupted leaker $j$, the probability of giving a correct result is $1 - PValues[j]$, where $PValues[j]$ is the smallest p-value among all the p-values for each $V[k]$, $V[k']$ pair in which a significant difference is observed. The accuracy for the whole algorithm is the probability that the algorithm makes a right identification on each captured party, so the accuracy should be

$$\prod_{j \in L} (1 - PValues[j])$$

where $L$ is the set of all leakers captured by Algorithm 2.

**VII. Evaluation**

In this section, we evaluate AuditShare’s performance in the data sharing phase and the effectiveness of the knowledge-based identification algorithm when leakage happens. To be more specific:

- We measure the time overhead in AuditShare’s data sharing phase in terms of the number of data objects and receivers. The goal is to show how the cryptographic operations of OT will affect the performance. From the evaluation, we show AuditShare is able to finish heavy-load PII database sharing tasks in a short time.
- We evaluate AuditShare’s effectiveness in identifying leakers in different scenarios: a non-colluding guilty receiver, a guilty sender, and a group of colluding parties. Our evaluation shows that with total 20,000 objects and 6 receivers,
AuditShare is able to identify all the leakers with accuracy more than 99.99% in any scenarios if more than 5% of the objects are leaked and used in the identification algorithm.

A. Sharing Performance

We first evaluate AuditShare’s time overhead in OT-based data sharing process. Since AuditShare requires OT operations in the transfer of each data object, directly applying raw OT can be highly time-consuming. However, the time complexity caused by heavy-load OT operations can be greatly reduced by utilizing OT extensions [13], [15], [3]. We implement a prototype of AuditShare with the OT extension proposed by Ishai et. al [13] and all the evaluation results were obtained from this prototype implementation.

Evaluation Settings We evaluate AuditShare’s performance using a PII database extended from a public dataset called Heart Disease Data [14]. The original data contains patients’ SSNs and full names, but these PII attributes have been stripped in the publicly available version for privacy concerns. For the purpose of evaluation, we extend the size of the dataset and add back these two PII attributes with randomly generated values. The size of each data object used in the evaluation is 88 Bytes. The evaluation platform is a 4-core 2.2 GHz Intel Core i7 and the communication channel is a UDP connection between local host ports (so the time consumption is mainly from cryptographic operations).

![Figure 6: Time Consumption of Sharing in AuditShare](image)

Evaluation Results Our evaluation on AuditShare prototype implementation shows its efficiency in a real-world deployment. Though the time consumption is linear to the number of receivers and the total objects to share, it only takes about 580 milliseconds to share 100,000 data objects with each receiver. As shown in Figure 6, even with 1,000,000 data entries and 6 receivers, AuditShare is able to finish data sharing processes in about 30 seconds. We omitted the time consumption on the preparation and verification in this paper because they take much less time than the sharing process and do not need to be performed in real-time.

B. Accurate Leaker Identification

We then evaluate the identification accuracy of AuditShare’s knowledge-based identification algorithm through data leakages simulations. To be specific, we allow the leaker(s) to be (i) a guilty receiver, (ii) multiple guilty receivers without collusion, (iii) a guilty sender, and (iv) multiple guilty receivers with collusion, respectively, and simulate the leaked datasets by randomly picking up a certain percentage of objects from the leakers. In fact, we find the identification accuracy of a non-colluding guilty receivers is independent from how many non-colluding guilty receivers there are (i.e., case (i) and case (ii)), so we merge them into a single case.

In the collusive leakage, we randomly select three combinations; for example, in a leakage simulation of 3 colluding leakers out of 6 receivers, we create the leaked dataset to be a random subset of following sets respectively.

\[
\text{Recv}_1 \cap \text{Recv}_2 \cap \text{Recv}_3, \text{Recv}_1 \cap \text{Recv}_2 \cap \text{Recv}_3
\]

\[
(\text{Recv}_1 \cap \text{Recv}_2 \cap \text{Recv}_3) \cup (\text{Recv}_1 \cap \text{Recv}_2 \cap \text{Recv}_3)
\]

In the evaluation, we change the number of leaked objects by setting a different percentage when generating the leaked dataset. At the same time, we fix the total numbers of data objects and receivers in the sharing.

Evaluation Settings We let the value of \(|\text{Data}|\) be 20,000, and the total number of receivers \(N\) be 6. In the collusive leakage scenario, we evaluate the detection when there are 2, 3, and 6 colluding receivers out of total 6 receivers, and in each case, we randomly try three different types of dataset combinations, which consist of one or more intersection, complement, and union operations.

Plot Notation The \(x\)-axis indicates the fraction of the leaked data objects over a leaker’s dataset or a processed dataset made by colluding receivers. Specifically, we formally describe the meaning of \(x\) using \(x = 0.01\) as an example.

- In a non-collusive leakage, it means 1% of guilty receiver’s objects \(\text{Recv}_i\) are leaked and used in AuditShare to identify the leakers.
- In the case of a guilty sender, it means 1% of the common objects are leaked and used in AuditShare to identify the leakers.
- As for a collusive leakage, \(x = 0.01\) means 1% objects from the combined dataset (e.g., after a set of collusive operations) are leaked and used in AuditShare.

Notice that with the same fraction, the size of a leaked dataset in collusive leakage can be much smaller than that in a non-collusive leakage, e.g., \(|\text{Recv}_i| \times 0.01 < |\text{Recv}_1 \cap \text{Recv}_2| \times 0.01\). The \(y\)-axis indicates the error rate of the detection results. Error rate 1 means the number of leaked objects is not sufficient for the identification algorithm to identify the leaker(s) while error rate 0 indicates that the algorithm can identify all the leakers undoubtedly. In the plots, we let the maximum accuracy precision to 99.999999%, in other words, the minimum error rate is \(1 \times 10^{-8}\).

Evaluation Results As shown in Figure 7 the accuracy becomes 1 when 6%, 9%, and 10% of the received objects are leaked in the case of a non-colluding guilty receiver, a guilty sender, and a number of guilty receivers, respectively.
The more data is leaked and used in AuditShare, the higher the identification accuracy will be. In Figure 7a and Figure 7b we can observe similar identification accuracy in both cases. This is because $|\text{SenderOnly}| = |\text{Unique}|$ for any $i < N$ and the difference is because of the definition of leakage ratio: a receiver’s ratio denominator is $|\text{Recv}_i|$ while the sender’s ratio denominator is the quantity of all data objects. Figure 7c illustrates that when the number of colluding receivers increases, there is a decrease in the detection accuracy. This is because when more receivers contribute to the leakage, the combined dataset usually becomes smaller in size, e.g., $\text{Recv}_1 \cap \text{Recv}_2$ has a bigger size than $\text{Recv}_1 \cap \text{Recv}_2 \cap \text{Recv}_3$, and thus a larger number of leaked objects are needed to identify leakers. Moreover, we find the accuracy will not change much when different types of combination operations are made in a collusive leakage.

VIII. DISCUSSION

A. Basic Trade-off in AuditShare

The main approach of AuditShare is to allocate data objects among different receivers, which inevitably leads to a fundamental trade-off between AuditShare’s performance and the data loss in the allocation. For example, if each receiver receives the same whole dataset, there is no differentiation that can be used to identify leakers; on the other hand, if each receiver receives a totally different subset, then identification becomes much easier. Therefore, the larger the data loss is, the more differentiation each receiver’s data objects will carry and consequently the more accurate the identification will be. However, data loss will greatly limit the practical use of a sharing system. For example, in our example use case, a school is supposed to share all students’ information to receivers because the service should serve all the students.

In order to circumvent the limitation, AuditShare introduces fake objects. This changes the basic trade-off to be fake data object size versus the accuracy of leaker identification. This is because the data loss in normal allocation based approaches is replaced by fake data loss in AuditShare. On the other hand, the high accuracy is at the cost of having fake objects, which at least double the size of each received dataset. By enlarging the number of fake data objects used in AuditShare, identification accuracy can be further improved.

B. Impact of Fake Data Objects

PII data is usually shared for two reasons. Sometimes it is shared to enable the data receiver to provide certain services for each individual user in the dataset. For example, online educational services provide personalized services to students based on student information shared by the school. Other times it is shared to let data receivers analyze the data and dig out useful statistical conclusions.

AuditShare does not bring up negative effects for the former because AuditShare ensures that each receiver will receive all data entries in the original dataset, so all users will be served just as when AuditShare is not applied. We acknowledge that a data receiver may have to prepare services for non-existing users (corresponding to the fake data entries that are indistinguishable to the data receiver), but this will not influence the service quality for real users. Regarding the second use case, we argue that the dataset can be anonymized before data analysis. In addition, previous works also show that properly adding fake data entries basically will not decrease the quality of data analysis.

C. AuditShare with a Large Number of Receivers

AuditShare allocates fake data objects among receivers for leaker identification. To ensure a high identification accuracy, adequate fake objects should be allocated to each receiver. Inadequate objects will degrade AuditShare’s performance in leaker identification. Using the simulation setting as in Section VII we set the total fake object number to 20,000. As shown in Figure 8 when there are totally 3, 6, and 9 receivers, the error rate increases as the total receiver number grows. With 3 receivers, AuditShare can identify leakers accurately when about 5% of data objects are leaked (as shown in Section VII-B), so 20,000 fake objects are sufficient. When sharing with 6 receivers, AuditShare’s accuracy is close to 1 when 10% of data objects are leaked. This means 20,000 fake objects can still lead to a relatively good accuracy but is worse relative to the 3-receiver scenario. However, this quantity of fake objects are inadequate when there are 9 receivers. As shown, to reach an error ratio less than $10^{-8}$, AuditShare requires 50%, 50%, and 80% of fake data objects leaked from guilty receivers in three leakage cases.
Therefore, in real-world use of AuditShare, the sender should have more fake data objects when sharing data with more receivers. To achieve an accuracy > 99.99%, it requires about 200,000 fake objects in a 9 receiver scenario.

D. Collusion Between a Sender and Receivers

We have not elaborated the case when a sender colludes receivers, but in fact, AuditShare can cope with it naturally. Specifically, there are two possible situations.

1) A sender contributes to the leakage. In this case, the only contribution the sender can make are those objects not received by its associates (i.e., colluding receivers). By Lemma VI.2, there is a great chance that leaked objects overlap SenderOnly, so the sender will be caught by Algorithm 1. At the same time, all corrupted receivers who contributed to the leakage will be captured by Algorithm 2.

2) The sender does not contribute to the leakage, under which, there is no need to capture the sender. Leakers who contributed to the leakage can be captured by Algorithm 2.

Importantly, in either case, the sender has no clue of data received by honest receivers due to the use of OT, so a collusion between the sender and corrupted receivers cannot frame an honest receiver.

IX. RELATED WORK

The data leakage problem has been explored for years [24], [8]. 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 PII 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) data watermarking, and (ii) data object allocation across receivers.

A. Watermarking

In the first category [12], the data sender alters the noise-tolerant attributes or embeds hidden information into the data before sending it to each receiver so that each distributed copy is watermarked. By associating the watermark contained in the leaked data with the receivers, the data sender is able to identify the guilty party. This scheme was first proposed by Agrawal et al. [2], [1] and was further explored in later works [25], [31], [20], [21], [32], [11], [4].

However, as pointed out in many existing works [2], [12], [4], it is hard to effectively make a watermark on structured data, 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). For example, an attacker may undermine the watermark by simply removing or adding noise to certain attributes in the dataset. Even worse, a group of colluding parties can compare their received copies and figure out the watermarked data or the hidden information. Consequently, these will lead to unexpected detection results, thus making it hard to be the evidence used in a court of law.

To this end, AuditShare does not follow the watermarking direction; instead, AuditShare takes the data allocation approach and utilizes the property of PII data – PII attributes of a dataset should be kept unmodified in order to keep the property of PII. For example, if leakers randomize or erase PII attributes, it becomes harder (i.e., requires much more work) to link leaked information to each individual and the leaked data itself is not PII data any more. Compared with watermarking approaches, AuditShare has a stronger resistance to data manipulation made by leakers.

B. Data Object Allocation

First proposed in Panagiotis and Hector’s work [19], [18], the second category allocates data objects among receivers so that each receiver will obtain a different data object collection from others. When leaked data is found, the sender can utilize the receiver-objects mapping to evaluate each receiver’s probability of data leaking. Fake but realistic data objects can also be created to facilitate detection. Following this direction, Ishu and Ashutosh proposed a guilt detection mechanism [10] that uses a directed bigraph to allocate data objects to receivers.

However, existing allocation mechanisms also have several issues when used in the real world.

- An allocation can be disrupted by a collusion among receivers, by which corrupted receivers can learn the allocation and escape being captured by leaking common objects.
Furthermore, they can strip special data objects that are potentially used to uniquely identify each receiver.

- Existing allocation approaches didn’t consider the scenario where a data sender itself may leak the data intentionally or unintentionally, which hurts the usability and reliability of existing systems. For example, the sender can frame a receiver because it knows this receiver’s data and thus can leak those data on purpose. In turn, a guilty receiver may also frame the sender when a data breach happens.

- Furthermore, rare existing systems provide immutable proof of sharing, and thus involved parties may deny what has been shared when data is leaked.

In AuditShare, we address these three issues by the use of oblivious transfer, knowledge-based identification algorithm, and undeniable proof of sharing.

X. CONCLUSION

In this work, we study the leaker identification problem in PII data sharing and propose AuditShare, a data-sharing system with reliable leaker identification. Unlike existing solutions, AuditShare allocates data objects obliviously across receivers, immutably records the sharing process, and accurately identifies leakers with AuditShare’s knowledge-based algorithms. In AuditShare, we also utilize fake data objects to compensate the data loss caused by the allocation through 1-out-of-2 OT, thus ensuring a lossless data sharing.

Not limited to PII data sharing, AuditShare can apply to sharing of other data that contains identifiable attributes, e.g., movie name, product ID, etc.

Our work shows that proactively building auditability into the data sharing can greatly facilitate leaker identification in case of a data leakage. This will in turn prevent malicious parties from illegally selling or disclosing sensitive information.

REFERENCES

[1] 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.

[2] 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.

[3] G. Asharof, 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.

[4] 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.

[5] BBC News. (2019) British airways faces record £183m fine for data breach. [Online]. Available: https://www.bbc.com/news/business-48905907

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

[7] Californians for Consumer Privacy. (2019) California consumer privacy act (ccpa). [Online]. Available: https://www.captiv.org/clip/

[8] I. J. Cox, M. L. Miller, J. A. Bloom, and C. Honsinger, Digital watermarking. Springer, 2002, vol. 53.

[9] EU GDPR.ORG. (2019) The eu general data protection regulation (gdpr). [Online]. Available: https://eugdpr.org/

[10] 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.

[11] R. Halder and A. Cortesi, “Persistent watermarking of relational databases,” in Proceedings of the IEEE International Conference on Advances in Communications, Network, and Computing (CNC 2010), October. Citeseer, 2010, pp. 4–5.

[12] 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.

[13] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, “Extending oblivious transfers efficiently,” in Annual International Cryptology Conference. Springer, 2003, pp. 145–161.

[14] A. Janosi, W. Steinbrunn, M. Pfisterer, and R. Detrano. (2019) Uci machine learning repository: Heart disease data set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/heart+disease

[15] M. S. Kiraz, B. Schoenmakers, and J. Villegas, “Efficient committed oblivious transfer of bit strings,” in International Conference on Information Security. Springer, 2007, pp. 130–144.

[16] 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.

[17] 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.

[18] 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.

[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] Z. Qin, Y. Ying, L. Jia-jin, and L. Yi-shu, “Watermark based copyright protection of outsourced database,” in 2006 10th International Database Engineering and Applications Symposium (IDEAS’06), Dec 2006, pp. 301–308.

[21] Z. Qin, Y. Ying, L. Jia-jin, and L. Yi-shu, “Watermark based copyright protection of outsourced database,” in 2006 10th International Database Engineering and Applications Symposium (IDEAS’06). IEEE, 2006, pp. 301–308.

[22] M. O. Rabin, “How to exchange secrets with oblivious transfer.” JACR Cryptology ePrint Archive, vol. 2005, p. 187, 2005.

[23] K. H. Rosen and K. Krithivasan, Discrete mathematics and its applications: with combinatorics and graph theory. Tata McGraw-Hill Education, 2012.

[24] A. Shabtai, Y. Elovici, and L. Rokach, “Rights protection for categorical data,” IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 7, pp. 912–926, 2009.

[25] 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

[26] 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-1557106401

[27] US Department of Education. (2014) Protecting student privacy while using online educational services: Requirements and best practices. [Online]. Available: https://studentprivacy.ed.gov/resources/protection-student-privacy-while-using-online-educational-services-requirements-and-best-practices

[28] US Department of Health and Human Services Office for Civil Rights. (2019) Hhs breach of unsecured protected health information. [Online]. Available: https://ocr.report.jsf

[29] Wikipedia contributors, “p-value — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 23-May-2019]. [Online]. Available: https://en.wikipedia.org/wiki/P-value

[30] Y. Zhang, X. Niu, D. Zhao, J. Li, and S. Liu, “Relational databases watermark technique based on content characteristics,” in First International Conference on Innovative Computing, Information and Control-Volume 1 (ICICIC’06), vol. 3. IEEE, 2006, pp. 677–680.

[31] X. Zhou, M. Huang, and Z. Peng, “An additive-attack-proof watermarking mechanism for databases’ copyrights protection using image,” in Proceedings of the 2007 ACM symposium on Applied computing. ACM, 2007, pp. 254–258.