Federated Learning Enhancement through Privacy-Preserving Deduplication

Abstract

Deduplication is one of the vital preprocessing steps for creating a machine learning model that enhances model's performance and also saves energy and time for the training. In this paper, we address the problem of deduplication in federated machine learning setup, where the participating nodes run the deduplication process before the training that reveals nothing except the duplicates. We use private set intersection which is a well known technique for doing multiparty computation.

1 Introduction

In today's world, machine learning is everywhere - it application ranges from online shopping, banking KYC to various other applications that are directly related to our daily life.

Machine learning is a data dependent process, and needless to say that there is always a risk of privacy leakage associated with this. Privacy preserving machine learning is currently getting attention which aim to preserve the privacy of the participating entity in a machine learning process. Federate machine learning (FedML) is one of the basic steps towards privacy preserving learning. FedML is a distributed machine learning setup, where there is a set of different nodes holding their local data and based on their local data they compute a local model, which the nodes later share with a common server that finally aggregates and derives the final model that captures the feature of all the local data held by the individual nodes [13]. As these nodes never share their local data, therefore, this provides some privacy guarantee to their data.

Machine learning is a data dependent process and model accuracy increases with higher number of training datasets. However, one has to be cautious to check the quality of the data too, for instance, training with too many outliers will significantly affect the accuracy of the model that results in inaccurate inferences. Therefore, it is important to preprocess the data to remove outliers to ensure high model accuracy.

In the following, we emphasize another reason for preprocessing the data. An unorganised (raw) dataset may contain duplicate entries. Deduplication means removing duplicates from a dataset. Recently, Lee et al. [11] have pointed out the importance of data deduplication in machine learning. Their case study

included language modeling datasets. They examined the dataset known as C4 (Colossal Clean Crawled Corpus) which had been introduced by Raffel et al [16]. They found a 61-word sequence in C4 that is repeated 61,036 times verbatim in the training dataset and 61 times in the validation set. They observed that train-test set overlap led to overestimating the model accuracy and a biased model selection. On the other hand, deduplication of datasets makes learning much more efficient – it saves CPU time and reduces the training cost in terms of time, money, and environment [1, 17, 15]. [**Sumanta:** these references are from [11], need to recheck before finalizing the paper.] Given that the world is now more focused in sustainable developments, thus energy efficient machine learning is quite in line with this theme. It was also observed in [11] that deduplication did not worsen perplexity (a metric to measure how well a model can predicts test samples, lower is better), rather in some cases, it reduced perplexity by 10%.

As per the discussions so far, it is now evident that deduplication brings benefit in many ways, and so it is important to do deduplication before the training. When the data owner runs deduplication on her data, the risk in terms of privacy is minimal as the data is completely possessed by the owner. This scenario completely changes when deduplication is considered in FedML setup. Two nodes which are participtaing in a federated learning process, may share common data (even after deduplicating their own data). If they want to deduplicate the union of their data, then one possible way is to share each other's data and check the intersection. However, this immediately breaks the privacy guarantee of FedML. In this paper, we show how we can run deduplication in FedML setup without harming privacy of the data of the individual nodes involved in the learning.

Overview of privacy preserving deduplication in FedML

Suppose there are n-nodes $D_1, \ldots, D_n$ involved in a FedML training having datasets $S_1, \ldots, S_n$, respectively. A FedML training will actually be done on the union of these datasets, i.e., $\cup_i^n S_i$. However, if there is a nonempty intersection between sets, this means the FedML training has been done on duplicates and the final model will suffer from the drawbacks due to duplicates in training datasets as discussed in the introduction. So, it is important to get rid of the duplicates from the training datasets without harming the users' data privacy.

Let us first assume that there are two nodes: D_1 and D_2 having datasets S_1 and S_2, respectively. If the intersection $S_1 \cap S_2$ is nonempty, then training individually on S_1 and S_2 will mean training twice on $S_1 \cap S_2$. Thus the duplicate $S_1 \cap S_2$ should be avoided, and that should be done without harming the data privacy of both D_1 and D_2. Our solution applies private set intersection (PSI) which securely finds intersection of S_1 and S_2 without revealing any other elements. Therefore, once D_1 and D_2 learns $S_1 \cap S_2$ after running a PSI algorithm, D_1 will train on $S_1' = S_1 \setminus S_1 \cap S_2$ and D_2 will train on S_2. Note that $S_1' \cup S_2 = S_1 \cup S_2$, therefore resulting FedML model will be the same as intended however the training is free from duplicates and avoids related drawbacks.

The case of two nodes appears to be straightforward, however, when more

than two nodes participate, it becomes a bit complicated. We now consider the n-nodes $D_1, \ldots, D_n$ having sets $S_1, \ldots, S_n$, respectively, where $n > 2$. Following the 2-nodes case, we can simply apply PSI among n-nodes and find their intersection I_n. Then D_1 will train on $S_1' = S_1 \setminus I_n$, and rest of the nodes will train on their own dataset S_i. This will of course get rid of the duplicates that exist across the nodes, however, this will not take care of the duplicates that exist among a subset of nodes. For instance, if there is k-sbuset ($k < n$) with a large intersection I_k, then I_k will remain in the full training dataset $S_1' \bigcup \cup_{i=2}^{n} S_i$ as duplicates. Therefore, a general PSI does not help in this case. So we need to consider nodes pairwise and remove the duplicates accordingly. To be precise, we first need to run PSI among all possible pair of nodes - $\binom{n}{2}$ in total. In this paper, we present Group-PSI that efficiently finds pairwise intersections among n-nodes, and at the end $S_1, \ldots, S_n$ are updated as $S_1', \ldots, S_n'$ such that $\cup_{i=1}^{n} S_i' = \cup_{i=1}^{n} S_i$, where $S_i' \cap S_j' = \Phi, i \neq j$.

[**Aydin:** It seems there is a research line called "privacy-preserving deduplication", e.g., [19, 10, 5]. The privacy-preserving deduplication techniques do not necessarily use PSIs. I think, we need to explain (1) why we cannot use the existing deduplication techniques and (2) why we want to use a PSI instead.]

[**Sumanta:** I checked these three references and appears that they are not relevant. I summarise the contributions of these papers below so that you can also verify if my understanding is correct.

[$ZMX + 22$]: Application of this work is Fog network (a variant of IoT network) having the frame work of fog-cloud network. Main goal of this work is to save storage by deduplication. They emphasize on compression based deduplication that allows deduplication of similar data. Data owner can only know the deduplication as the whole network is owned by the data owner who imposes privacy by encryption.

[$KH12$]: Deduplication in Fog computing for the purpose of storage management. Deduplication prohibits uploading same documents by others. They use hash to check the duplicates and file is encrypted to protect privacy. Fog devices has raw data from the sensors and server keeps encrypted data that will keep the privacy of the data.

[$DLW14$]: They use server as a deduplication service. Data D is transformed into D' and server runs deduplication on D'. Deduplicated D' is returned to the client from which client can reverse to the deduplicated original data. They apply locality-sensitive hashing (LSH) to map a file which is compared to check the duplicates. They also have proposed another mapping from string to points on a Euclidean plan.]

[**Vishnu:** This paper is relevant [9]. They propose multi-party private set intersection]

Application of deduplicated FedML

[**Sumanta:** Revisit this section: may need to be omitted as we are not any more talking about which node work more and which will work less.]

Deduplicated FedML enjoys all the benefits as mentioned earlier. On top of that in some applications it has other benefits too. We mention one such application here.

Suppose we want to run FedML in Internet of Things (IoT) networks. An IoT network is heterogeneous – it comprises of diverse types of devices that come with different computational capabilities and resource constraints. For example, IoT device powered by a Raspberry Pi can certainly handle much more complex computations than devices like microcontrollers (used in sensors). Therefore, any application in IoT domain should support the resource constrained devices, that is also true in case of machine learning applications.

Now consider a FedML application which intends to train on the local data of devices in an IoT network. The kind of computation that a device has to do to train on huge local data could become a burden to small devices in the network, whereas other devices which are more powerful can easily perform the task. So before the training, if IoT devices run a deduplication process to identify the intersection of the data, then devices with higher resources can train on their full data, whereas low-resourced devices can train on their data subtracting the intersection, resulting in much lighter training for these low-resourced devices.

2 Preliminaries

2.1 Notations and Assumptions

We define a wrapper function $\texttt{Update}(S, \hat{S}) \rightarrow S$ which takes two sets, S and $\hat{S}$. It updates S by removing from it the elements in set $\hat{S}$ and returns the updated set S. In this paper, by the sum of sets (e.g., $S_i + S_j$) we mean the concatenation of the sets which may result in a multi-set. We denote an empty set by $\emptyset$. We denote a size of vector $\vec{v}$ with $|\vec{v}|$.

We assume that the server and all the users have access to secure channels among them.

2.2 Federated Learning

Federated Learning was proposed by McMahan et al. [13] as a framework to train a machine learning model where the training data are distributed across multiple devices. In this framework there is a server who aggregates locally-computed models as follows. Suppose there are n users and each user U_i has a dataset $\mathcal{D}_i$. The server has the initial model $\boldsymbol{\theta}$. The server sends $\boldsymbol{\theta}$ to the users and each user performs the gradient descent computation on their local dataset $\mathcal{D}_i$ as

$J_i(\mathcal{D}_i, \theta) = \frac{1}{d_i} \sum_{(\mathbf{x},y)\in\mathcal{D}_i} C(\boldsymbol{\theta}, (\mathbf{x}, y))$, where C is the cost function and $d_i = |\mathcal{D}_i|$. The user U_i computes the local models as $\boldsymbol{\theta}^i \leftarrow \boldsymbol{\theta} - \eta \nabla J(\mathcal{D}_i, \boldsymbol{\theta})$, where η

is the learning rate. After receiving the local models from the users, the server performs an aggregated averaging on the local models to construct the global model as:

$$\boldsymbol{\theta} = \frac{1}{d}\left(d_1\boldsymbol{\theta}^1 + \ldots + d_n\boldsymbol{\theta}^n\right) \tag{1}$$

where $d = \sum_{i=1}^{n} d_i$ is the total size of the dataset $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$, and $\boldsymbol{\theta}^i$ is the locally trained model on the dataset $\mathcal{D}_i$. This computation is repeated until the model converges.

2.3 Security Model

In this paper, we use the simulation-based paradigm of secure multi-party computation [6] to define and prove the proposed protocol. Since we focus on the static passive (semi-honest) adversarial model, we will restate the security definition in this adversarial model.

2.3.1 Two-party Computation

A two-party protocol Γ problem is captured by specifying a random process that maps pairs of inputs to pairs of outputs, one for each party. Such process is referred to as a functionality denoted by $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f := (f_1, f_2)$. For every input pair (x, y), the output pair is a random variable $(f_1(x, y), f_2(x, y))$, such that the party with input x wishes to obtain $f_1(x, y)$ while the party with input y wishes to receive $f_2(x, y)$. When f is deterministic, then $f_1 = f_2$. The above functionality can be easily extended to $m > 2$ parties.

2.3.2 Security in the Presence of Passive Adversaries

In the passive adversarial model, the party corrupted by such an adversary correctly follows the protocol specification. Nonetheless, the adversary obtains the internal state of the corrupted party, including the transcript of all the messages received, and tries to use this to learn information that should remain private. Loosely speaking, a protocol is secure if whatever can be computed by a party in the protocol can be computed using its input and output only. In the simulation-based model, it is required that a party's view in a protocol's execution can be simulated given only its input and output. This implies that the parties learn nothing from the protocol's execution. More formally, party i's view (during the execution of Γ) on input pair (x, y) is denoted by $\mathsf{View}_i^{\Gamma}(x, y)$ and equals $(w, r^i, m_1^i, ..., m_t^i)$, where $w \in \{x, y\}$ is the input of i^{th} party, r_i is the outcome of this party's internal random coin tosses, and m_j^i represents the j^{th} message this party receives. The output of the i^{th} party during the execution of Γ on (x, y) is denoted by $\mathsf{Output}_i^{\Gamma}(x, y)$ and can be generated from its own view of the execution.

Definition 1. Let f be the deterministic functionality defined above. Protocol Γ security computes f in the presence of a static passive probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, if for every $\mathcal{A}$ in the real model, there exist PPT algorithms $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that:

$$\{\mathsf{Sim}_1(x, f_1(x,y))\}_{x,y} \stackrel{c}{\equiv} \{\mathsf{View}_1^{\mathcal{A},\Gamma}(x,y)\}_{x,y}$$
$$\{\mathsf{Sim}_2(y, f_2(x,y))\}_{x,y} \stackrel{c}{\equiv} \{\mathsf{View}_2^{\mathcal{A},\Gamma}(x,y)\}_{x,y}$$

Definition 1 can be easily extended to $m > 2$ parties.

2.4 Pseudorandom Function and Permutation

Informally, a pseudorandom function $\mathtt{PRF}(.)$ is a deterministic function that takes a key of length λ and an input of length u; and outputs a value of length v indistinguishable from that of a truly random function. More formally, a pseudorandom function can be defined as $\mathtt{PRF} : \{0,1\}^\lambda \times \{0,1\}^u \rightarrow \{0,1\}^v$, where λ is the security parameter. In practice, a pseudorandom function can be obtained from an efficient block cipher [7].

The definition of a pseudorandom permutation, $\mathtt{PRP} : \{0,1\}^\lambda \times \{0,1\}^u \rightarrow \{0,1\}^u$, is very similar to that of a pseudorandom function, with a difference; namely, it is required the keyed function $\mathtt{PRP}(k,.)$ to be indistinguishable from a uniform permutation, instead of a uniform function. In cryptographic schemes that involve PRP, sometimes honest parties may be required to compute the inverse of pseudorandom permutation, i.e., $\mathtt{PRP}^{-1}(k,.)$, as well. In this case, it would require that $\mathtt{PRP}(k,.)$ be indistinguishable from a uniform permutation even if the distinguisher is additionally given oracle access to the inverse of the permutation.

2.5 Oblivious Pseudorandom Function

An Oblivious Pseudorandom Function (OPRF) is a protocol that involves a client and a server. OPRF enables the client with $x \in \{0,1\}^u$, and the server with key $k \in \{0,1\}^\lambda$ to execute an instance of PRF. Informally, the security of an OPRF asserts that, by the completion of OPRF, the client only learns the output of PRFevaluated on inputs k and x, i.e., $\mathtt{PRF}(k,x)$ while the server gains no information, e.g., about the input of the client and the output of PRF. We refer readers to [4] for a survey of OPRF.

2.6 Trusted Execution Environments

Trusted Execution Environment ($\mathcal{TEE}$), also known as a secure enclave, constitutes a secure processing environment comprising processing, memory, and storage hardware units [14, 20]. Within this environment, the code and data residing in them remain isolated from other layers in the software stack, including the operating system. An ideal $\mathcal{TEE}$ guarantees the preservation of data

integrity and confidentiality. Moreover, $\mathcal{TEE}$ can provide remote attestation capabilities, allowing a party to remotely verify the execution of an enclave on a genuine $\mathcal{TEE}$ hardware platform. Given the assumption that the physical CPU remains uncompromized, enclaves are shielded from attackers with physical access to the machine, including the memory and system bus.

Side-channel attacks on different deployments of $\mathcal{TEE}$s have been demonstrated in the literature [18]. These attacks pose a threat as they could enable attackers to extract secrets from $\mathcal{TEE}$s. Nevertheless, $\mathcal{TEE}$s technologies have been evolving to address and mitigate side-channel attacks.

Our security, trust, and system assumptions regarding $\mathcal{TEE}$s are conservative. Specifically, as detailed in Sections 4.2.2 and 3.2, our solution (i) avoids disclosing any plaintext messages or private keys to $\mathcal{TEE}$ and (ii) does not expect $\mathcal{TEE}$ to maintain an extensive storage and memory space or possess strong processing resources.

Instead, we establish a formal model and construction under the assumption that $\mathcal{TEE}$ guarantees execution integrity (and authenticity), and ensures minimal confidentiality. Specifically, we formally demonstrate that $\mathcal{TEE}$ only learns the size of the encrypted computation result, namely, the intersections' cardinality. Moreover, considering the frequent constraints on resources in real-world implementations of $\mathcal{TEE}$s, our solution is tailored for scenarios in which (a) the storage and memory space of $\mathcal{TEE}$ is smaller than the combined size of all datasets and (b) cannot carry out (numerous) public-key cryptography operations. $\mathcal{TEE}$ in our work can be seamlessly substituted with any semi-honest server possessing constrained storage, memory, and computational resources, and lacking collusion with other entities.

3 Group PSI (G-PSI)

3.1 Formal Definition of G-PSI

In this section, we present the concept of Group PSI (G-PSI). In G-PSI, there are two groups of clients, $\mathcal{G}_0$ and $\mathcal{G}_1$, where each group contains m clients, $\mathcal{G}_j : \{\mathcal{C}_{j,1}, ..., \mathcal{C}_{j,m}\}$, $0 \leq j \leq 1$. Each client $\mathcal{C}_{j,1}$ has a set $S_{j,1}$, such that no pair of clients' sets in the same group share a common element.

Informally, G-PSI allows a client in one group to *efficiently* find the intersection it has with every client of the other group, without allowing those clients that do not share the common elements with that client to learn anything about the plaintext intersection. To achieve a high level of computational efficiency, in G-PSI, we will involve a third-party $\mathcal{TP}$ that assists the clients with computing the intersection. The functionality $f_{\text{G-PSI}}$ that G-PSI computes takes a set $S_{j,i}$ from every client $\mathcal{C}_{j,i}$ and no input from $\mathcal{TP}$. It returns (i) to every client $\mathcal{C}_{j,i}$ a vector $\vec{v}_{j,i}$ which contains the intersection that $\mathcal{C}_{j,i}$'s set has with every other client's set in the other group and (ii) to $\mathcal{TP}$ an empty set $\emptyset$. Hence, functionality $f_{\text{G-PSI}}$ can be formally defined as follows.

$$f_{\text{G-PSI}}\Big(\underbrace{(S_{0,1},...,S_{0,m})}_{\mathcal{G}_0},\underbrace{(S_{1,1},...,S_{1,m})}_{\mathcal{G}_1},\emptyset\Big)\rightarrow\Big(\underbrace{(\vec{v}_{0,1},...,\vec{v}_{0,m})}_{\mathcal{G}_0},\underbrace{(\vec{v}_{1,1},...,\vec{v}_{1,m})}_{\mathcal{G}_1},\emptyset\Big) \tag{2}$$

where, $\vec{v}_{j,i}=\Big[[S_{j,i}\cap S_{1-j,1}],...,[S_{j,i}\cap S_{1-j,m}]\Big]$, $0\leq j\leq 1$, and $1\leq i\leq m$.

[**Sumanta:** We can rewrite $f_{\text{G-PSI}}$ as follows:

$$f_{\text{G-PSI}}\Big(\underbrace{(S_{0,1},...,S_{0,m})}_{\mathcal{G}_0},\underbrace{(S_{1,1},...,S_{1,m})}_{\mathcal{G}_1},\emptyset\Big)\rightarrow\Big(\underbrace{(\vec{v}_{0,1},...,\vec{v}_{0,m})}_{\mathcal{G}_0},\underbrace{(S_{1,1},...,S_{1,m})}_{\mathcal{G}_1},\emptyset\Big) \tag{3}$$

Then in the protocol we mention only G_0 gets the intersection. Later we make a comment that if G_1 wants to have the intersection that can be shared by the clients of G_0, in the semi-honest setting this does not make any difference, right?]

Since the $\mathcal{TP}$ performs computation on all clients' encrypted sets, there is a possibility of leakage to $\mathcal{TP}$. Depending on the protocol that realizes $f_{\text{G-PSI}}$ this leakage could contain different types of information; for instance, it could contain (i) the size of the intersection of any two clients' sets, or (ii) the size of the intersection of all clients' sets, or (iii) nothing at all. Often such leakage is defined by a leakage function $\mathcal{L}$ that takes all parties (encoded) inputs and returns the amount of leakage.

We assert a protocol securely realizes $f_{\text{G-PSI}}$ if (1) it reveals nothing beyond a predefined leakage to $\mathcal{TP}$ and (2) whatever can be computed by a client in the protocol can be obtained from its input and output only. This is formalized by the simulation paradigm. We require a client's view during an execution of G-PSI to be simulatable given its input and output. Also, we require that the $\mathcal{TP}$'s view can be simulated given the leakage.

Definition 2 (Security of G-PSI)**.** Let $\mathcal{G}_0$ and $\mathcal{G}_1$ be two groups of clients, where each group contains m clients, $\mathcal{G}_j:\{\mathcal{C}_{j,1},...,\mathcal{C}_{j,m}\}$, $0\leq j\leq 1$. Let $\mathcal{L}$ denote a leakage function, $f_{\text{G-PSI}}$ be the functionality defined above (in Relation 2 on page 8), $S=\{S_{0,1},...,S_{0,m},S_{1,1},...,S_{1,m}\}$, and $S_{j,i}$ represent a set belonging to client $\mathcal{C}_{j,i}$. Then, a protocol Γ securely realises $f_{\text{G-PSI}}$ in the presence of a static semi-honest PPT adversary $\mathcal{A}$, if for every $\mathcal{A}$ in the real model, there exists a PPT adversary (simulator) Sim in the ideal model, such that for every $\mathcal{C}_{j,i}\in\{\mathcal{C}_{0,1},...,\mathcal{C}_{0,m},\mathcal{C}_{1,1},...,\mathcal{C}_{1,m}\}$ and $\mathcal{TP}$, Relations 4 and 5 hold respectively.

$$\{\mathsf{Sim}_{\mathcal{C}_{j,i}}(S_{j,i},\vec{v}_{j,i})\}_S\stackrel{c}{\equiv}\{\mathsf{View}^{\mathcal{A},\Gamma}_{\mathcal{C}_{j,i}}(S,\emptyset)\}_S \tag{4}$$

$$\{\mathsf{Sim}^{\mathcal{L}}_{\mathcal{TP}}(\emptyset,\emptyset)\}_S\stackrel{c}{\equiv}\{\mathsf{View}^{\mathcal{A},\Gamma}_{\mathcal{TP}}(S,\emptyset)\}_S \tag{5}$$

where $\vec{v}_{j,i}=\Big[[S_{j,i}\cap S_{1-j,1}],...,[S_{j,i}\cap S_{1-j,m}]\Big]$, $0\leq j\leq 1$, and $1\leq i\leq m$.

3.2 Efficient Construction of G-PSI

In this section, we introduce two efficient protocols that realize G-PSI. The first one, called EG-PSI$^{(\mathrm{I})}$, is highly efficient and based on symmetric key cryptography. The second one, called EG-PSI$^{(\mathrm{II})}$, is based on OPRF (in turn depends on public key cryptography) and discloses fewer amounts of information to $\mathcal{TEE}$ than the former does. Thus, these two protocols trade-off between performance and leakage amount.

3.2.1 EG-PSI$^{(\mathrm{I})}$

In this section, we present EG-PSI$^{(\mathrm{I})}$, a highly efficient protocol that realizes G-PSI and uses only symmetric key cryptography. At a high level, EG-PSI$^{(\mathrm{I})}$ operates as follows. Initially, each client in group $\mathcal{G}_0$ agrees with every client in group $\mathcal{G}_1$ on a secret key. Every client encrypts its set elements using every key it has agreed on with other clients (in a different group).

Each client, for every encrypted set element, temporally stores a triple that includes (i) the encrypted element, (ii) the key used to encrypt that element, and (iii) the index of the client with whom the key was shared. These triples will enable the client to efficiently (a) retrieve the correct key and (b) identify the client with whom it has the element in common when presented with an encrypted element. Once all elements are encrypted, each client transmits only its encrypted elements to $\mathcal{TEE}$.

Given the sets of encrypted elements from all clients, $\mathcal{TEE}$ aggregates these sets and identifies the encrypted elements that appear more than once. Subsequently, $\mathcal{TEE}$ forwards to a client those encrypted elements that (1) appeared more than once and (2) were among the messages that the client initially sent to $\mathcal{TEE}$. Upon receiving each encrypted element from $\mathcal{TEE}$, a client searches its local list of triples to locate the corresponding key and the index l representing a specific client. Utilising the key, the client decrypts the element and regards the resultant element as one of the elements within the intersection it shares the l-th client. For a comprehensive description of EG-PSI$^{(\mathrm{I})}$, refer to Figure 7.

Informally, the only information that the $\mathcal{TEE}$ learns in EG-PSI is the *size* of the intersection of any two clients' sets, which we called it *pair-wise intersection cardinally*. Below, we formally define it.

Definition 3 (pair-wise intersection cardinally)**.** Let $(\underbrace{S'_{0,1}, ..., S'_{0,m}}_{\mathcal{G}_0}), (\underbrace{S'_{1,1}, ..., S'_{1,m}}_{\mathcal{G}_1})$ be two groups $\mathcal{G}_0$ and $\mathcal{G}_1$ of (encrypted) sets. Then, vector $\vec{s}$ represents the pair-wise intersection cardinally: $\vec{s} = [\vec{s}_{0,1}, ..., \vec{s}_{0,m}, \vec{s}_{1,1}, ..., \vec{s}_{1,m}]$, where $\vec{s}_{j,i} = \Big[\big|[S_{j,i} \cap S_{1-j,1}]\big|, ..., \big|[S_{j,i} \cap S_{1-j,m}]\big|\Big]$, $0 \leq j \leq 1$, and $1 \leq i \leq m$.

Definition 4. Let $\vec{s}$ be a pair-wise intersection cardinality of two groups $\mathcal{G}_0$ and $\mathcal{G}_1$ of encrypted sets: $(\underbrace{S'_{0,1}, ..., S'_{0,m}}_{\mathcal{G}_0}), (\underbrace{S'_{1,1}, ..., S'_{1,m}}_{\mathcal{G}_1})$, w.r.t. Definition 3. Then, leakage function $\mathcal{L}$ is defined as follows: $\mathcal{L}\Big((S'_{0,1}, ..., S'_{0,m}), (S'_{1,1}, ..., S'_{1,m})\Big) \rightarrow \vec{s}$.

- *Parties.* Trusted Execution Environment $\mathcal{TEE}$, clients in group $\mathcal{G}_0$: $\{\mathcal{C}_{0,1}, ..., \mathcal{C}_{0,m}\}$, and clients in group $\mathcal{G}_1 : \{\mathcal{C}_{1,1}, ..., \mathcal{C}_{1,m}\}$.
- *Inputs.* Sets $S_{0,1}, ..., S_{0,m}, S_{1,1}, ..., S_{1,m}$, where each $S_{i,j}$ belongs to client $\mathcal{C}_{i,j}$, $0 \leq j \leq 1$ and $1 \leq i \leq m$.
- *Outputs.* $\vec{v}_{j,i}$ to $\mathcal{C}_{j,i}$, where $\vec{v}_{j,i} = \Big[[S_{j,i} \cap S_{1-j,1}], ..., [S_{j,i} \cap S_{1-j,m}]\Big]$.

---

1. *Setup.*
   (a) each client $\mathcal{C}_{0,i}$ in $\mathcal{G}_0$ agrees with every client $\mathcal{C}_{1,l}$ in $\mathcal{G}_1$ on a secret key $k_{i,l}$, by picking a random key $k_{i,l}$ and sending it to $\mathcal{C}_{1,l}$. Client $\mathcal{C}_{0,i}$ stores this key as $k_{i,l}$ while $\mathcal{C}_{1,l}$ stores this key as $k_{l,i}$.
   (b) each $\mathcal{C}_{j,i}$ takes the following steps:
      i. encrypts its set elements under keys $k_{i,l}$ $(\forall l, 1 \leq l \leq m)$ as follows: $\forall e \in S_i : e'_{i,l} = \mathtt{PRP}(k_{i,l}, e)$. Let set $S'_{j,i}$ contain the encrypted set elements of $\mathcal{C}_{j,i}$ and let set $T_{j,i}$ contains all triples of the form $(e'_{i,l}, k_{i,l}, l)$.
      ii. sends $S'_{j,i}$ to $\mathcal{TEE}$ and locally keeps $T_{j,i}$.
2. *Finding Encrypted Intersection.* $\mathcal{TEE}$ takes the following steps for each $\mathcal{C}_{j,i}$.
   (a) appends to an empty set, $R_{j,i}$, every ciphertext that satisfy the following conditions hold:
      - it appears more than once in the set $S = \sum_{j=0}^{1} \sum_{i=1}^{m} S'_{j,i}$
      - it appears in set $S'_{j,i}$
   (b) sends $R_{j,i}$ to $\mathcal{C}_{j,i}$.
3. *Extracting Plaintext Intersection.* Each $\mathcal{C}_{j,i}$ takes the following steps.
   (a) constructs a vector $\vec{v}_{j,i} = [\vec{v}_{j,i,1}, ..., \vec{v}_{j,i,m}]$, where each vector in $\vec{v}_{j,i}$ is initially empty.
   (b) decrypts each element of $R_{j,i}$ as follows. $\forall e' \in R_{j,i}$:
      i. retrieves decryption key $k_{i,l}$ and index l from $T_{j,i}$ using e'.
      ii. calls $\mathtt{PRP}^{-1}(k_{i,l}, e') = e$ and appends e to l-th vector in $\vec{v}_{j,i}$.
   (c) considers $\vec{v}_{j,i}$ as the result.

Figure 1: First Variant of Efficient Group PSI (EG-PSI$^{(\mathrm{I})}$).

Theorem 1. *Let $f_{G\text{-}PSI}$ be the functionality defined in Relation 2 (on page 8). Also, let $\mathcal{L}$ be the leakage function presented in Definition 4. If* PRP *is a secure pseudorandom permutation, then EG-PSI$^{(I)}$ (presented in Figure 7) securely realizes $f_{G\text{-}PSI}$, w.r.t. Definition 2.*

Appendix A presents the proof of Theorem 1.

3.2.2 EG-PSI$^{(II)}$

In this section, we present EG-PSI$^{(II)}$ which is the second variant of EG-PSI. As we see that $\mathcal{TEE}$ in EG-PSI will be able to learn the cardinality of the intersection of each pair of clients' sets. So we aim to reduce the leakage in EG-PSI$^{(II)}$, however this is no longer based on symmetric key cryptography, rather depends on OPRF, and to the best of our knowledge, all the constructions of OPRFs are based on public key cryptography, that is, their security depends on some hard problem assumptions, so is EG-PSI$^{(II)}$.

A brief overview of the construction of EG-PSI$^{(II)}$ is as follows. We have two groups of clients $\mathcal{G}_0$ and $\mathcal{G}_1$ having their own set that they want to find the pairwise intersection. There is a $\mathcal{TEE}$ that has a key k for PRF.

During the setup, each client engages with $\mathcal{TEE}$ to get their set encrypted under $\texttt{PRF}(k, \cdot)$ through an OPRF call. Then each client of the group $\mathcal{G}_1$ will share their encrypted set with every client of $\mathcal{G}_0$. Upon receiving an encrypted set from a client of $\mathcal{G}_1$, each client of $\mathcal{G}_0$ determines the intersection with their encrypted set, once the client finds intersection, they mark the index of the elements in the intersection and consider the elements with the same index in the unencrypted set as elements in the intersection. Similarly, each client of $\mathcal{G}_0$ shares their encrypted set with each client of $\mathcal{G}_1$ and follows the same steps as described just now. At the end, every client of $\mathcal{G}_0$ knows the intersection between their set and all the sets of belonging to the clients of $\mathcal{G}_1$, and the other way around. We give a detailed description of EG-PSI$^{(II)}$ in Figure 3.

In EG-PSI$^{(II)}$, $\mathcal{TEE}$ does not participate in the PSI protocol except the OPRF evaluation and that is also one time activity. As a result, $\mathcal{TEE}$ does not learn anything about the set intersection, it only learns the cardinality of each client's set.

[**Sumanta:** Aydin to write a table with comparison between EG-PSI$^{(I)}$ and EG-PSI$^{(II)}$]

Theorem 2. *Let $f_{G\text{-}PSI}$ be the functionality defined in Relation 2 (on page 8). Further, let $\mathcal{L}$ be the leakage function presented in Definition 4 with the empty output. If OPRF is secure, then EG-PSI$^{(II)}$ (presented in Figure 3) securely realises $f_{G\text{-}PSI}$, w.r.t. Definition 2.*

3.2.3 Extension

In certain cases (including the deduplication protocol proposed shortly), a simpler version of the proposed PSIs that output the intersection to the clients of *only one group* would suffice. This simpler variant would have lower communication and computation costs.

- *Parties.* Trusted Execution Environment $\mathcal{TEE}$, clients in group $\mathcal{G}_0$: $\{\mathcal{C}_{0,1}, ..., \mathcal{C}_{0,m}\}$, and clients in group $\mathcal{G}_1 : \{\mathcal{C}_{1,1}, ..., \mathcal{C}_{1,m}\}$.
- *Inputs.* Sets $S_{0,1}, ..., S_{0,m}, S_{1,1}, ..., S_{1,m}$, where each $S_{i,j}$ belongs to client $\mathcal{C}_{i,j}$, $0 \leq j \leq 1$ and $1 \leq i \leq m$.
- *Outputs.* $\vec{v}_{j,i}$ to $\mathcal{C}_{j,i}$, where $\vec{v}_{j,i} = \Big[[S_{j,i} \cap S_{1-j,1}], ..., [S_{j,i} \cap S_{1-j,m}]\Big]$.

---

1. *Setup.*

   $\mathcal{TEE}$ generates a secret key k for $\texttt{PRF}$ that will be used for an OPRF evaluation.

2. *Encryption of each client's set.*

   Each client $\mathcal{C}_{j,i}$ for $j = 0, 1$, and $1 \leq i \leq m$ and $\mathcal{TEE}$ run an OPRF protocol and obtains the encrypted set $\texttt{PRF}(k, S_{j,i})$.

3. *Finding Encrypted Intersections.*

   (a) Each client $\mathcal{C}_{1,i}$ for $1 \leq i \leq m$ shares $\texttt{PRF}(k, S_{1,i})$ with every client $\mathcal{C}_{0,i}$ $1 \leq i \leq m$.

   (b) Each client $\mathcal{C}_{0,i}$ for $1 \leq i \leq m$ shares $\texttt{PRF}(k, S_{0,i})$ with every client $\mathcal{C}_{1,i}$ $1 \leq i \leq m$.

   (c) Each client $\mathcal{C}_{j,i}$ $j = 0, 1$, $1 \leq i \leq m$, does the following.

      i. Creates an empty set $R_{j,i,t}$ for all $1 \leq t \leq m$.

      ii. Let e_ℓ represents an element of $\texttt{PRF}(k, S_{j,i})$, where $1 \leq \ell \leq |\texttt{PRF}(k, S_{j,i})|$.
      Performs $R_{j,i,t} \leftarrow R_{j,i,t} \cup \{\ell\}$ if e_ℓ appears both in $\texttt{PRF}(k, S_{j,i})$ and $\texttt{PRF}(k, S_{j-1,i})$.

4. *Extracting Plaintext Intersection.*

   Each $\mathcal{C}_{j,i}$ for $j = 0, 1$, and $1 \leq i \leq m$ does the following.

   (a) Constructs a vector $\vec{v}_{j,i} = [\vec{v}_{j,i,1}, ..., \vec{v}_{j,i,m}]$, where each vector in $\vec{v}_{j,i}$ is initially empty.

   (b) Appends the ℓ-th element of $S_{j,i}$ to $\vec{v}_{j,i,t}$ for all $\ell \in R_{j,i,t}$.

   (c) Returns $\vec{v}_{j,i}$.

Figure 2: Second Variant of Efficient Group PSI (EG-PSI$^{(\text{II})}$).

4 Privacy-Preserving Multi-Party Deduplication (P-MPD)

4.1 Formal Definition of P-MPD

Informally, P-MPD considers the setting where there are n parties $C = \{\mathcal{C}_1, ..., \mathcal{C}_n\}$ and each $\mathcal{C}_i \in C$ has a set. P-MPD enables the parties to (efficiently) remove duplications from their local sets such that after the deduplication the concatenation of all parties' local sets equals the union of their initial sets.

The functionality $f_{\text{P-MPD}}$ that P-MPD computes takes a set S_i from each P_i. It returns an updated (deduplicated) set S'_i to each P_i. Formerly, $f_{\text{P-MPD}}$ can be defined as follows:

$$f_{\text{P-MPD}}(S_1, ..., S_n) \rightarrow (S'_1, ..., S'_n) \tag{6}$$

where $\sum_{i=1}^{n} S'_i = \bigcup_{i=1}^{n} S_i = S_\cup$ and $S_\cup$ denotes the union of all initial sets and contains only unique elements.

To maintain generality, we define a leakage function, denoted as $\mathcal{W}$. The output of this function relies on the protocol implementing $f_{\text{P-MPD}}$. We assert that a protocol securely realises $f_{\text{P-MPD}}$ if whatever can be computed by a party in the protocol can be derived solely from its individual input and output, given the output of $\mathcal{W}$. Below, we formally state it.

Definition 5 (Security of P-MPD)**.** Let $S = \{S_1, ..., S_m\}$, and S_i represent a set belonging to party $\mathcal{C}_i$. Let $\mathcal{W}$ denote a leakage function and $f_{\text{P-MPD}}$ be the functionality defined in Relation 6. Then, a protocol Ψ securely realises $f_{\text{P-MPD}}$ in the presence of a static semi-honest PPT adversary $\mathcal{A}$, if for every $\mathcal{A}$ in the real model, there exists a PPT Sim in the ideal model, such that for every $\mathcal{C}_i \in \{\mathcal{C}_1, ..., \mathcal{C}_m\}$, Relation 7 holds.

$$\{\mathsf{Sim}^{\mathcal{W}}_{\mathcal{C}_i}(S_i, S'_i)\}_S \overset{c}{\equiv} \{\mathsf{View}^{\mathcal{A},\Psi}_{\mathcal{C}_i}(S)\}_S \tag{7}$$

4.2 Constructions of P-MPD

4.2.1 Basic Approach

In this section, we describe a basic approach that can be taken to construct P-MPD. This is a kind of *snowball* approach, where the i-th party waits for the input from the $(i+1)$-th party and the input size grows as the this flows down the parties. The high level idea is as follows. The party $\mathcal{C}_1$ computes $S_1 \cap \bigcup_{j=2}^{m} S_j$ and then $S'_1 = S_1 \setminus (S_1 \cap \bigcup_{j=2}^{m} S_j)$. So S'_1 does not have any intersection with $S_2, \ldots, S_m$. Similarly, $\mathcal{C}_2$ derives $S'_2 = S_2 \setminus (S_2 \cap \bigcup_{j=3}^{m} S_j)$, and so on so forth resulting in parties having sets which do not have pairwise intersections.

We now elaborate the method in the following. Apart from the m-parties, there is a trusted execution environment $\mathcal{TEE}$ having an OPRF key k. First, each party $\mathcal{C}_i$, where $1 \leq i \leq m$, runs an OPRF protocol with $\mathcal{TEE}$ to get

the encrypted set $\texttt{PRF}(k, S_i)$. Then parties update their sets as follows. It starts with the last party that is $\mathcal{C}_m$. They simply pass their encrypted set $\texttt{PRF}(k, S_m)$ to $\mathcal{C}_{m-1}$. In this case $G = \texttt{PRF}(k, S_m)$ which is transferred from $\mathcal{C}_m$ to $\mathcal{C}_{m-1}$. After receiving G from $\mathcal{C}_m$, the party $\mathcal{C}_{m-1}$ finds the intersection $E_{m-1} = \texttt{PRF}(k, S_{m-1}) \cap G = \texttt{PRF}(k, S_{m-1}) \cap \texttt{PRF}(k, S_m)$. Once this intersection E_{m-1} is available to $\mathcal{C}_{m-1}$, they find the corresponding subset $T_{m-1} \subset S_{m-1}$ whose encryption under the OPRF is E_{m-1}. One possible way to do this is by maintaining the indices of $\texttt{PRF}(k, S_{m-1})$, and if the j-th element of $\texttt{PRF}(k, S_{m-1})$ is in $\texttt{PRF}(k, S_m)$, then the j-th element $S_{m-1,j}$ of S_{m-1} is in T_{m-1}. Party $\mathcal{C}_{m-1}$ updates their set as $S'_{m-1} = S_{m-1} \setminus T_{m-1}$ and updates $G = \texttt{PRF}(k, S'_{m-1}) \cup G = \texttt{PRF}(k, S_{m-1}) \cup \texttt{PRF}(k, S_m)$. The updated set G is now sent to $\mathcal{C}_{m-2}$. Party $\mathcal{C}_{m-2}$ derives T_{m-2} in the same way as T_{m-1} was derived by $\mathcal{C}_{m-1}$ and obtains $S'_{m-2} = S_{m-2} \setminus T_{m-2}$. Next $\mathcal{C}_{m-2}$ updates G as $G = \texttt{PRF}(k, S'_{m-2}) \cup G$, that means $G = \texttt{PRF}(k, S'_{m-2}) \cup \texttt{PRF}(k, S'_{m-1}) \cup \texttt{PRF}(k, S'_m)$. Then G is transferred to $\mathcal{C}_{m-3}$. In the same way parties $\mathcal{C}_{m-3}, \ldots, \mathcal{C}_1$ update their local sets as $S'_{m-3}, \ldots, S'_1$, respectively. Once this completes, then it is easy to note that $S'_1 = S_1 \setminus (S_1 \cap \bigcup_{j=2}^{m} S_j)$, that means S'_1 does not have any intersection with $S_2, \ldots, S_m$. In general, $S'_i = S_i \setminus (S_i \cap \bigcup_{j=i+1}^{m} S_j)$. Therefore, $S'_i \cap S'_j = \phi$ for $i \neq j \in [1 \ldots m]$.

The reason we call this a snowball approach is that the procedure starts from the top, that is from $\mathcal{C}_m$, and the set G grows bigger as it rolls down from party $\mathcal{C}_{i+1}$ to $\mathcal{C}_i$.

This approach involves one-way communications from party $i+1$ to i where party i needs to wait for the input from party $i+1$. Therefore, it needs $m-1$ iterations for m-parties. In the next section, we present an efficient P-MPD construction that needs $O(\log_2 m)$ iterations.

4.2.2 Construction Based on EG-PSI

We introduce a pioneering protocol called Duplication Eliminator, DE, that realizes P-MPD. The innovative idea behind DE's design involves constructing a binary tree with leaf nodes representing the parties' indices. Subsequently, EG-PSI is recursively applied to the sets of parties sharing the same node until reaching the tree's root. After each invocation of EG-PSI, the parties with the lower indices update their sets by removing the intersection, returned by EG-PSI, from their local sets. These updated sets are then used as input in the subsequent EG-PSI invocation.

Next, we delve into further detail. We begin by sorting the parties' indices in ascending order. Following this, we construct a binary tree in a manner where the leaf nodes of the tree represent the parties' indices.

At level $d = 1$, commencing from the left-hand side, every two parties form a cluster. Within each cluster, the first party is assigned to group $\mathcal{G}_0$ and the second party to group $\mathcal{G}_1$. Then, the parties within the same cluster engage in an instance of EG-PSI. Upon the return of the sets' intersection by EG-PSI, only the party situated on the left-hand side, specifically the one with the smaller

- *Parties.* Trusted Execution Environment $\mathcal{TEE}$, and parties $\mathcal{C}_1, \ldots, \mathcal{C}_m$.
- *Inputs.* Sets $S_1, \ldots, S_m$, where each S_i belongs to party $\mathcal{C}_i$, where $1 \leq i \leq m$.
- *Outputs.* $S'_1, ..., S'_m$, where $\sum_{i=1}^{n} S'_i = \bigcup_{i=1}^{m} S_i$.

---

1. *Setup.*

   $\mathcal{TEE}$ generates a secret key k for $\texttt{PRF}$ that will be used for an OPRF evaluation.

2. *Encryption of each party's set.*

   Each party $\mathcal{C}_i$ for $1 \leq i \leq m$ and $\mathcal{TEE}$ run an OPRF protocol and obtains the encrypted set $\texttt{PRF}(k, S_i)$.

3. *Updating Individual Sets.*

   $G \leftarrow \phi$, an empty set

   For $i = m$ to 1, $\mathcal{C}_i$ does the following:

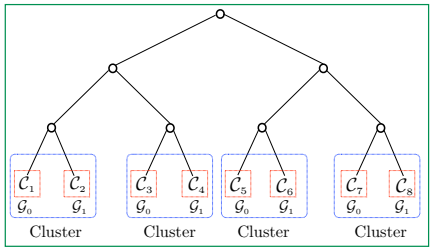
   (a) finds the encrypted intersection $\texttt{PRF}(k, S_i) \cap G$,
   (b) updates $S'_i \leftarrow S_i \setminus T_i$, where $\texttt{PRF}(k, T_i) \subset \texttt{PRF}(k, S_i) \cap G$,
   (c) updates $G \leftarrow \texttt{PRF}(k, S_i) \cup G$ and passes G to party $i - 1$.

Figure 3: A basic construction of P-MPD.

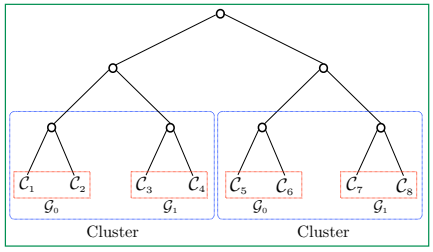
index, modifies its local set by eliminating the elements of the intersection from its sets. This process is outlined in Figure 4a.

At level $d = 2$, starting from the left-hand side, every set of 2^d parties form a cluster. Within each cluster, the initial $\frac{2^d}{2} = 2$ parties enter group $\mathcal{G}_0$ and the remainder (i.e., the remaining 2 parties) enter group $\mathcal{G}_1$. Subsequently, the parties within the same cluster engage in an instance of EG-PSI. Once again, upon the return of their sets' intersection by EG-PSI, the parties possessing the smaller indices update their local sets. This procedure is illustrated in 4b.

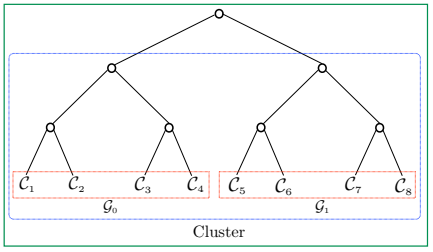
The process continues until level $d = \log_2 m$ is reached. At this point, all m parties collectively engage in an instance of EG-PSI. Each $\frac{2^d}{2} = \frac{m}{2}$ parties enter group $\mathcal{G}_0$ and the remaining $\frac{m}{2}$ parties enter group $\mathcal{G}_1$. Then, these m parties jointly participate in an instance of EG-PSI. Similarly to previous steps, the parties with the smaller indices update their local sets based on the output of EG-PSI. This procedure is outlined in 4c. Figure 5 presents a detailed description of DE.



(a) Illustration of how parties enter different groups and clusters at level 1.



(b) Demonstration of how parties join various groups and clusters on level 2.



(c) Depiction of how parties become part of diverse groups and clusters at level 3.

Figure 4: The process of how eight parties form clusters and groups at various levels of a binary tree. At each level, parties belonging to two groups $\mathcal{G}_0$ and $\mathcal{G}_1$ within the same cluster initiate EG-PSI, followed by the updating of their respective local sets.

4.2.3 A Direct Construction of P-MPD based on OPRF

It is now clear that given a EG-PSI construction we can build P-MPD. We now show that it is also possible to construct P-MPD directly that requires $\mathcal{TEE}$ involvement only during the setup phase. Our construction is based on OPRF..

Let there be clients $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and they have the sets $S_1, \ldots, S_m$ respectively. During the Setup phase, first $\mathcal{TEE}$ generates a secret key k for a $\texttt{PRF}$ that will be used for an OPRF evaluation. Then every client of $\mathcal{C}_1, \ldots, \mathcal{C}_m$ involves with $\mathcal{TEE}$ for an OPRF evaluation of their set and obtains the encrypted set $\texttt{PRF}(k, S_i)$ for $1 \leq i \leq m$.

Then clients follow the same tree based approach as taken in Section 4.2.2. That is at level d, two sets of 2^d clients form group $\mathcal{G}_0$ and $\mathcal{G}_1$ as per their rank of indices. Further, they form a cluster. Then all the clients from $\mathcal{G}_1$ send their encrypted sets to all the clients of $\mathcal{G}_0$. Since, both groups' clients have their sets encrypted by the same key, so if they have the same elements in their sets, their encryption will also be the same. So, the clients of $\mathcal{G}_0$ updates their sets by removing the intersections that they have with the clients of $\mathcal{G}_1$. This happens in parallel at the level for all the clusters. Once this level is complete they move to the next level $d+1$ and repeat the same where the group $\mathcal{G}_0$ and $\mathcal{G}_1$ will now have 2^{d+1} clients. Finally at the root level all clients will have their sets updated with removed duplicates such that they have sets $S'_1, ..., S'_m$, where $\sum_{i=1}^{n} S'_i = \bigcup_{i=1}^{m} S_i$.

To the best of our knowledge existing OPRFs depend on public key hard problem assumptions and so does this variant of P-MPD. Note that the P-MPD of Section 4.2.2 is based on symmetric primitive such as PRP. It is to be noted that in terms of cryptographic agility symmetric primitives based P-MPD is easier to get a post-quantum variant rather than primitives based on hard problem assumptions. However, this variant of P-MPD has the advantage that $\mathcal{TEE}$ needs to involve only during the setup phase. So we expect this P-MPD to be efficient in terms of time performance.

4.3 Naive Approaches

4.3.1 Running a Two-Party PSI Multiple Times

One might be tempted to allow each party to invoke an existing two-party PSI with every other party. However, this approach in total requires $\sum_{i=1}^{m-1}(m-i)$ invocations of the two-party PSI. Whereas DE requires only $\sum_{i=1}^{\log_2(m)}(\frac{m}{2^i}) = m-1$ invocations of our efficient tailor-made PSI, EG-PSI. To provide concrete values, when $m = 256$, the naive scheme requires 32640 invocations of a standard two-party PSI, whereas our scheme requires only 255 invocations of EG-PSI.

4.3.2 Running Multi-Party PSI Once

One might be tempted to execute an existing multi-party PSI protocol only once on all parties' sets, hoping to identify and subsequently remove duplicated elements. Nevertheless, this approach falls short of identifying all duplications. The limitation arises from the fact that a multi-party PSI can uncover elements common to *all* parties, making it incapable of detecting elements shared among only a subset of parties.

4.3.3 Deduplicating all Sets at Once

There is a simplified version of DE. In this variant, each party reaches an agreement with the others on a secret key. Subsequently, each party encrypts its set elements using every key agreed upon with the other parties. Finally, all parties transmit their encrypted set elements to $\mathcal{TEE}$.

Upon receiving the encrypted sets from all parties, $\mathcal{TEE}$ identifies all duplicated elements. It then transmits to each party: (1) the duplicated elements found in the original elements that the respective party sent to $\mathcal{TEE}$, and (2) the index of the party with which it shares the same element. Based on $\mathcal{TEE}$'s response, the parties with smaller indices update their local sets accordingly.

However, this approach requires that $\mathcal{TEE}$ maintains a local storage or memory space equal to the size of the concatenation of all encrypted sets. In contrast, DE allows $\mathcal{TEE}$ to have a substantially smaller storage or memory space. This is achieved because, before $\mathcal{TEE}$ receives all parties' encrypted sets at level $d = log_2(m)$, parties apply updates to their sets multiple times. This iterative process progressively reduces the size of the parties' sets, and the reduction can be particularly significant when the number of duplications is substantial.

4.4 Security Analysis

Generally, in any secure (multi-party) deduplication protocol, a participating party will, by definition, learn the exact redundant elements within its set upon the protocol's completion. However, in DE, a party gains slightly more information. To be specific, upon the completion of DE, each party acquires the knowledge of the index (or identity) of the party that shares the redundant element. This additional information gained by a party during the execution of DE is formally defined as the output of a leakage function $\mathcal{W}$ which we define below.

$\mathcal{W}$ takes as input (i) sets $S_1, ..., S_m$ where each set S_i belongs to party $\mathcal{C}_i$, and (ii) sets $R_1, ..., R_m$, where each R_i contains the redundancy (or intersection) that a set S_i has with every other set. $\mathcal{W}$ outputs to i-th party a set Q_i of triples, where each triple is of the form (r_i, i, l), $r_i \in R_i$ is a redundancy between sets S_i and S_l, i is the index of S_i, and l is the index of S_l.

Definition 6. Let $S = \{S_1, ..., S_m\}$ be a set of sets, where each set S_i belongs to party $\mathcal{C}_i$. Also, let $R = \{R_1, ..., R_m\}$ be a set of sets, where each set R_i comprises the redundancy that S_i has with every other j-th set, for all j, $1 \leq j \leq m$ and

$i \neq j$. Then, leakage function $\mathcal{W}$ is defined as follows: $\mathcal{W}(S, R) \rightarrow (Q_1, ..., Q_m)$, where each Q_i is a set of triples of the form (r_i, i, l), $r_i \in R_i$ is a redundancy between sets S_i and S_l, i is the index of S_i, and l is the index of S_l.

Theorem 3. *Let $f_{\text{P-MPD}}$ be the functionality defined in Relation 6 and $\mathcal{W}$ be the leakage function presented in Definition 6. If EG-PSI is secure (w.r.t. Definition 2), then DE (presented in Figure 5) securely realises $f_{\text{P-MPD}}$, w.r.t. Definition 5.*

- *Parties.* A set of clients $\{\mathcal{C}_1, ..., \mathcal{C}_m\}$.
- *Inputs.* Sets $S_1, ..., S_m$, where each S_i belongs to client $\mathcal{C}_i$, $1 \leq i \leq m$, and m is a power of two.
- *Outputs.* Updated sets $S'_1, ..., S'_m$, where each S'_i belongs to client $\mathcal{C}_i$, and $\sum_{i=1}^{n} S'_i = \bigcup_{i=1}^{n} S_i = S_{\cup}$.

The parties take the following steps. $\forall d$, $1 \leq d \leq \log_2(m)$:

1. *Forming Clusters.* Every distinct 2^d clients enter the same cluster. Thus, for every d, there will be $\frac{m}{2^d}$ cluster(s). For instance, when $d = 1$, then there are the following clusters: $(\mathcal{C}_1, \mathcal{C}_2), ..., (\mathcal{C}_{m-1}, \mathcal{C}_m)$.
2. *Forming Group.* In each cluster, the first $\frac{2^d}{2}$ clients enter group 0, $\mathcal{G}_0$, and the rest of the clients enter group 1, $\mathcal{G}_1$.
3. *Finding Intersection.* In each cluster, the clients of $\mathcal{G}_0$ and $\mathcal{G}_1$ together engage an instance of EG-PSI. Each client $\mathcal{C}_j$ receives a vector $\vec{v}_j$ from EG-PSI, where $\vec{v}_j$ specifies the elements that $\mathcal{C}_j$ has in common with each client of the other group in the same cluster. It would be sufficient if only clients of $\mathcal{G}_0$ in each cluster were aware of the intersection result.
4. *Removing Duplication.* For every element $e \in \vec{v}_j$, each $\mathcal{C}_j$ in $\mathcal{G}_0$ calls $\texttt{Update}(S_j, \{e\}) \rightarrow S_j$.

Figure 5: Duplication Eliminator (DE): the construction of P-MPD.

5 Experiments

5.1 Implementation Details

OPRF algorithm [3]. Curve25519 and Ristretto group PRP algorithm: AES 128

Server: Holds the initial model θ

Users: Each user $U_i \in \mathcal{U}$ has the dataset $\mathcal{D}_i$.

Local deduplication

Each user runs a deduplication algorithm on their local dataset. At the end user U_i receives an updated dataset $\mathcal{D}'_i$

Global deduplication

1. All the users participate in a DE.
2. Each users $U_i \in \mathcal{U}$ gets updated sets $\mathcal{D}''_i$, such that $\sum_{U_i \in \mathcal{U}} \mathcal{D}''_i = \bigcup_{U_i \in \mathcal{U}} = \mathcal{D}'_i$.

Federated learning

1. Servers initiates the learning by sharing the initial model $\boldsymbol{\theta}$ to each user.
2. Each user $U_i \in \mathcal{U}$ trains on their local dataset $\mathcal{D}''_i$ and shares the local model with the server.

Figure 6: Description of the efficient deduplicated federated learning protocol

6 Related Works

A detailed deduplication method has been proposed in [11] that used existing techniques of finding duplicates such as suffix array [12] and MinHash [2]. However, there is no privacy requirements in this method as the data data owner is responsible for the deduplication.

In [8], the problem of privacy preserving deduplication was discussed in a centralised settings – there is a server that stores users' files after removing duplicated ones. Their method involves another server which helps in setting up keys for each user using Pseudorandom Function (OPRF). Our work is different from [8] in many ways. First, we apply OPRF for encrypting the dataset elements that allows efficient deduplication. Second, our deduplication requirement is more granular, we want entries of users data sets to be deduplicated. If the two data sets have the same elements irrespective of their order, that will automatically remove one of the data sets. However, since [8] considers file as a whole, so two files M_1 and M_2 having the same elements but in a different order will not remove one of them. Third, our deduplication is federated as opposed to [8] which is centralized (all data is stored in a centralized server). Fourth, [8] is a two-server setting, whereas we require one third party only.

7 Conclusions

References

[1] Bender, E.M., Gebru, T., McMillan-Major, A., Shmitchell, S.: On the dangers of stochastic parrots: Can language models be too big? In: Elish, M.C., Isaac, W., Zemel, R.S. (eds.) FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021. pp. 610–623. ACM (2021). https://doi.org/10.1145/3442188.3445922, https://doi.org/10.1145/3442188.3445922

[2] Broder, A.Z.: On the resemblance and containment of documents. In: Carpentieri, B., Santis, A.D., Vaccaro, U., Storer, J.A. (eds.) Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings. pp. 21–29. IEEE (1997). https://doi.org/10.1109/SEQUEN.1997.666900, https://doi.org/10.1109/SEQUEN.1997.666900

[3] Burns, J., Moore, D., Ray, K., Speers, R., Vohaska, B.: Ec-oprf: Oblivious pseudorandom functions using elliptic curves. Cryptology ePrint Archive, Paper 2017/111 (2017), https://eprint.iacr.org/2017/111, https://eprint.iacr.org/2017/111

[4] Casacuberta, S., Hesse, J., Lehmann, A.: Sok: Oblivious pseudorandom functions. In: IEEE EuroS&P. IEEE (2022)

[5] Dong, B., Liu, R., Wang, W.H.: Prada: Privacy-preserving data-deduplication-as-a-service. In: Li, J., Wang, X.S., Garofalakis, M.N., Soboroff, I., Suel, T., Wang, M. (eds.) Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014 (2014)

[6] Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)

[7] Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC Press (2007)

[8] Keelveedhi, S., Bellare, M., Ristenpart, T.: Dupless: Server-aided encryption for deduplicated storage. In: King, S.T. (ed.) Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013. pp. 179–194. USENIX Association (2013), https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bellare

[9] Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. Cryptology ePrint Archive, Paper 2017/799 (2017), https://eprint.iacr.org/2017/799, https://eprint.iacr.org/2017/799

[10] Koo, D., Hur, J.: Privacy-preserving deduplication of encrypted data with dynamic ownership management in fog computing. Future Gener. Comput. Syst. (2018)

[11] Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., Carlini, N.: Deduplicating training data makes language models better. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 8424–8445. Association for Computational Linguistics, Dublin, Ireland (May 2022). https://doi.org/10.18653/v1/2022.acl-long.577, https://aclanthology.org/2022.acl-long.577

[12] Manber, U., Myers, E.W.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**, 935–948 (1993), https://api.semanticscholar.org/CorpusID:5074629

[13] McMahan, H.B., Moore, E., Ramage, D., y Arcas, B.A.: Federated learning of deep networks using model averaging. CoRR **abs/1602.05629** (2016), http://arxiv.org/abs/1602.05629

[14] Pass, R., Shi, E., Tramèr, F.: In: EUROCRYPT (2017)

[15] Patterson, D.A., Gonzalez, J., Le, Q.V., Liang, C., Munguia, L., Rothchild, D., So, D.R., Texier, M., Dean, J.: Carbon emissions and large neural network training. CoRR **abs/2104.10350** (2021), https://arxiv.org/abs/2104.10350

[16] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. **21**, 140:1–140:67 (2020), http://jmlr.org/papers/v21/20-074.html

[17] Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: Korhonen, A., Traum, D.R., Màrquez, L. (eds.) Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers. pp. 3645–3650. Association for Computational Linguistics (2019). https://doi.org/10.18653/v1/p19-1355, https://doi.org/10.18653/v1/p19-1355

[18] Tramèr, F., Zhang, F., Lin, H., Hubaux, J., Juels, A., Shi, E.: Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In: EuroS&P (2017)

[19] Zhang, C., Miao, Y., Xie, Q., Guo, Y., Du, H., Jia, X.: Privacy-preserving deduplication of sensor compressed data in distributed fog computing. IEEE Trans. Parallel Distributed Syst. (2022)

[20] Zhang, F., Zhang, H.: Sok: A study of using hardware-assisted isolated execution environments for security. In: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016 (2016)

A Security Proof of EG-PSI$^{(\mathbf{I})}$

In this section, we prove the security of EG-PSI$^{(\mathrm{I})}$, i.e., Theorem 1.

Proof. We consider a set of cases where in each case a party is corrupt.

A.1 Corrupt Client

In this case, we focus on the view of a client $\mathcal{C}_{1,i}$ in $\mathcal{G}_1$. Because each client of $\mathcal{G}_0$ receives a subset of information received by a client in $\mathcal{G}_1$' specifically, a client of $\mathcal{G}_0$ does not receive a key from a client of $\mathcal{G}_1$. Thus, the proof asserting the protocol is secure regarding a corrupt client in $\mathcal{G}_1$ will imply the security of the same protocol concerning a client in $\mathcal{G}_0$. In the real execution of the protocol, the view of a client, $\mathcal{C}_{1,i}$, is defined as follows:

$$\{\mathsf{View}^{\text{EG-PSI}^{(\mathrm{I})}}_{\mathcal{C}_{1,i}}(S,\emptyset)\}_S = \{S_{1,i}, r_{1,i}, \vec{k}, R_{1,i}, \vec{v}_{1,i}\},$$

where S is the set containing all clients' sets, $r_{1,i}$ is the outcome of internal random coins of $\mathcal{C}_{1,i}$, $\vec{k} = [k_{i,1}, ..., k_{i,m}]$ is a vector of m keys received from the clients of $\mathcal{G}_0$, $R_{1,i}$ is the encrypted intersection $\mathcal{C}_{1,i}$ receives from $\mathcal{TEE}$, and $\vec{v}_{j,i}$ equals $[\vec{v}_{j,i,1}, ..., \vec{v}_{j,i,m}]$, each l-th vector in $\vec{v}_{j,i}$ contains the elements that are common between $\mathcal{C}_{1,i}$'s set and the set that belongs to the l-th client in the other

group. We proceed to construct a simulator, $\mathsf{Sim}_{\mathcal{C}_{1,i}}$, in the ideal model. The simulator, receiving the party's input $S_{1,i}$ and output $\vec{v}_{1,i}$, operates as follows.

1. constructs an empty view. It appends to it $S_{1,i}$ and uniformly random coins $r'_{1,i}$.

2. constructs an empty vector $\vec{k}'$ and then chooses m keys for PRP. It appends the keys to $\vec{k}'$ and adds $\vec{k}'$ to the view.

3. constructs an empty set R'. It encrypts the elements of every l-th vector $\vec{v}_{1,i,l} \in \vec{v}_{1,i}$ using l-th key k_l in $\vec{k}'$ and PRP. It inserts the ciphertexts into R'.

4. appends R' and $\vec{v}_{1,i}$ to the view. It outputs the view.

Now, we are prepared to demonstrate that the two views are computationally indistinguishable. The input set $S_{1,i}$ has identical distribution in both models, as their elements are identical. Furthermore, since we are in the semi-honest model, in the real execution of the protocol, the client picks its random coins $r_{1,i}$ according to the protocol description. In the ideal model, coins $r'_{1,i}$ have been selected uniformly at random. Therefore $r_{1,i}$ and $r'_{1,i}$ have identical distributions. Vectors $\vec{k}$ and $\vec{k}'$ have identical distributions too because their elements have been chosen uniformly at random.

In the real model, $R_{1,i}$ contains the encrypted elements of $\vec{v}_{1,i}$, where the keys in $\vec{k}$ are used for the encryption. Similarly, in the ideal model, each element of R' is the encryption of an element in $\vec{v}_{1,i}$ where a key in $\vec{k}'$ is used for the encryption. Thus, R and R' have identical distributions. Furthermore, elements of $\vec{v}_{1,i}$ are identical in both models; therefore, they have identical distributions in the real and ideal models.

We conclude that the views in the real and ideal models are indistinguishable.

A.2 Corrupt $\mathcal{TEE}$

In the real model, the view of $\mathcal{TEE}$ is as follows:

$$\{\mathsf{View}^{\mathcal{A},\Gamma}_{\mathcal{TP}}(S,\emptyset)\}_S = \{(S'_{0,1},...,S'_{0,m}),(S'_{1,1},...,S'_{1,m})\}$$

where each $S'_{j,i}$ contains the encrypted set elements of client $\mathcal{C}_{j,i}$. Next, we construct a simulator $\mathsf{Sim}^{\mathcal{L}}_{\mathcal{TEE}}$ that receives the output $\vec{s}$ of leakage function $\mathcal{L}$ (as defined in Definition 4).

1. constructs an empty view.

2. constructs m empty sets for each $\mathcal{G}_0$ and $\mathcal{G}_1$. Specifically, it constructs empty sets $(\underbrace{S''_{0,1},...,S''_{0,m}}_{\mathcal{G}_0}),(\underbrace{S''_{1,1},...,S''_{1,m}}_{\mathcal{G}_1})$.

3. fills each set $S''_{j,i}$ as follows, given that $\vec{s} = [\vec{s}_{0,1}, ..., \vec{s}_{0,m}, \vec{s}_{1,1}, ..., \vec{s}_{1,m}]$, where $\vec{s}_{j,i} = \Big[\big|[S_{j,i} \cap S_{1-j,1}]\big|, ..., \big|[S_{j,i} \cap S_{1-j,m}]\big|\Big]$, as defined in Definition 3.
   (a) for every y-th element of each $\vec{s}_{0,i}$ (where $1 \leq i, y \leq m$) picks $\vec{s}_{0,i}[y]$ random values (from the range of PRP). It appends these random values to both sets $S''_{0,i}$ and $S''_{1,y}$.
   (b) pads every set $S''_{j,i}$ with some random values (from the range of PRP) such that the size of each $S''_{j,i}$ after padding equals the size of $S'_{j,i}$.
4. appends sets $(S''_{0,1}, ..., S''_{0,m}), (S''_{1,1}, ..., S''_{1,m})$ to the view and outputs the view.

We proceed to demonstrate that the views in the real and ideal models are computationally indistinguishable. In the real model, each elements of a set $S'_{j,i}$ is an output of PRP. However, in the ideal model, each element of a set $S''_{j,i}$ has been picked uniformly at random (from the range of PRP). By the security of PRP (i.e., an output of PRP is computationally indistinguishable from a random value), sets' elements in the real and ideal model are computationally indistinguishable.

Moreover, the size of the intersection between each set $S'_{0,i}$ and each set $S'_{1,l}$ (in the real model) equals the size of the intersection between each set $S''_{0,i}$ and each set $S''_{1,l}$ (in the ideal model). Also, the size of each $S'_{j,i}$ in the real model equals the size of the corresponding set $S''_{j,i}$ in the ideal model. Hence, sets $S'_{0,1}, ..., S'_{0,m}, S'_{1,1}, ..., S'_{1,m}$ in the real model are computationally indistinguishable from sets $S''_{0,1}, ..., S''_{0,m}, S''_{1,1}, ..., S''_{1,m}$ in the ideal model.

Thus, the views in the real and ideal models are computationally indistinguishable.

□

B Security Proof of DE

C Symmetric Key Based EG-PSI without Third Party (Type III)

- *Parties.* Clients in group $\mathcal{G}_0$: $\{\mathcal{C}_{0,1}, ..., \mathcal{C}_{0,m}\}$, and clients in group $\mathcal{G}_1$: $\{\mathcal{C}_{1,1}, ..., \mathcal{C}_{1,m}\}$.
- *Inputs.* Sets $S_{0,1}, ..., S_{0,m}, S_{1,1}, ..., S_{1,m}$, where each $S_{j,i}$ belongs to client $\mathcal{C}_{j,i}$, $0 \leq j \leq 1$ and $1 \leq i \leq m$.
- *Outputs.* $\vec{v}_{0,i}$ to $\mathcal{C}_{0,i}$, where $\vec{v}_{0,i} = \Big[[S_{0,i} \cap S_{1,1}], ..., [S_{0,i} \cap S_{1,m}]\Big]$.

---

1. *Setup.*
   (a) each $\mathcal{C}_{1,i}$ takes the following steps:
      i. for every element $e \in S_{1,i}$ decides a key $k_e = Hash(e)$ and encrypts that e as $\texttt{PRP}(k_e, e)$. Let $S'_{1,i}$ contain the encrypted set elements of $\mathcal{C}_{1,i}$.
   (b) $\mathcal{C}_{1,i}$ sends their respective sets $S'_{1,i}$ to all the clients $\mathcal{C}_{0,i}$.
2. *Finding the Intersection.* Each $\mathcal{C}_{0,i}$ takes the following steps.
   (a) constructs a vector $\vec{v}_{0,i} = [\vec{v}_{0,i,1}, ..., \vec{v}_{0,i,m}]$, where each vector in $\vec{v}_{0,i}$ is initially empty.
   (b) for every element $e \in S_{0,i}$ determines $k_e = Hash(e)$ and $\texttt{PRP}(k_e, e)$,
      - if $\texttt{PRP}(k_e, e) \in S'_{1,\ell}$, appends e to $\vec{v}_{0,i,\ell}$
   (c) considers $\vec{v}_{0,i}$ as the result.

Figure 7: Variant of Efficient Group PSI (Type III).