Privacy-Preserving Hierarchical Clustering: Formal Security and Efficient Approximation

Xianrui Meng
Amazon Research
xianru@amazon.com

Dimitrios Papadopoulos
Hong Kong University of Science and Technology
dipapado@cse.ust.hk

Alina Oprea
Northeastern University
a.oprea@northeastern.edu

Nikos Triandopoulos
Stevens Institute of Technology
ntriando@stevens.edu

ABSTRACT

Machine Learning (ML) is widely used for predictive tasks in a number of critical applications. Recently, *collaborative or federated learning* is a new paradigm that enables multiple parties to jointly learn ML models on their combined datasets. Yet, in most application domains, such as healthcare and security analytics, privacy risks limit entities to *individually learning local models* over the sensitive datasets they own. In this work, we present the first formal study for *privacy-preserving collaborative hierarchical clustering*, overall featuring scalable cryptographic protocols that allow two parties to privately compute joint clusters on their combined sensitive datasets. First, we provide a formal definition that balances accuracy and privacy, and we present a provably secure protocol along with an optimized version for single linkage clustering. Second, we explore the integration of our protocol with existing approximation algorithms for hierarchical clustering, resulting in a protocol that can efficiently scale to very large datasets. Finally, we provide a prototype implementation and experimentally evaluate the feasibility and efficiency of our approach on synthetic and real datasets, with encouraging results. For example, for a dataset of one million records and 10 dimensions, our optimized privacy-preserving approximation protocol requires 35 seconds for end-to-end execution, just 896KB of communication, and achieves 97.09% accuracy.

1 INTRODUCTION

Big-data analytics is an ubiquitous practice already impacting our daily lives. Our digital interactions produce massive amounts of data which are processed with the goal of discovering unknown patterns or correlations that, in turn, can suggest useful conclusions and help making better decisions. At the core of this lies Machine Learning (ML) for devising complex *data models* and *predictive algorithms* that provide hidden insights and automated decisions while optimizing certain objectives. Example applications successfully employing ML frameworks include market-trend forecast, service personalization, speech/face recognition, autonomous driving, health diagnostics, and security analytics.

Data analysis is, of course, only as good as the analyzed data, but this goes beyond the need to properly inspect, cleanse or transform "high-fidelity" data prior to its modeling. In most learning domains, analyzing "big data" is of twofold semantics: *volume* and *variety*.

First, the larger the dataset available to an ML algorithm, the better the learning accuracy, as irregularities due to outliers fade faster away. Thus, *scalability* to large inputs is very important, especially so in *unsupervised learning*, where model inference uses unlabelled observations. Second, the more varied the data collected for analysis, the richer the inferred information, as degradation due to noise is reduced and domain coverage is increased. Indeed, for a given learning objective, say classification or anomaly detection, combining more datasets of similar type but different origin, allows for more elaborate analysis and discovery of complex hidden structures (e.g., correlation or causality). Thus, the predictive power of ML models naturally improves when they are built as *global models* over *multiple* datasets owned and contributed by *different* entities, in what is termed *collaborative learning*—and widely considered as the golden standard [70].

Privacy-preserving unsupervised learning. Several critical learning tasks, across a variety of different application domains (such as healthcare or security analytics), demand deriving accurate ML models over highly sensitive and/or proprietary data. By default, organizations in possession of such confidential datasets are left with no other option than simply running their own *local models*, severely impacting the efficacy of the learning task at hand. Thus, in the context of data analytics, *privacy risks are the main impediment against collaboratively learning over large volumes of individually contributed data.*

The security community has recently embraced the powerful concept of *privacy-preserving collaborative learning* (PCL), the premise being that effective analytics over sensitive data is feasible by building global models in ways that protect privacy. This is typically achieved by applying secure multiparty computation (MPC) or differential privacy (DP) to data analytics, so that "learning" is performed over encrypted or sanitized data. One notable example is the recently proposed privacy-preserving federated learning in which model parameters are aggregated and shared by multiple clients to generate a global model [9]. Existing work on PCL *almost exclusively address supervised rather than unsupervised learning tasks* (with a few exceptions such as k-means clustering). As unsupervised learning is a prevalent learning paradigm, the design of supporting ML protocols that promote collaboration, accuracy, and privacy in this setting is vital.

In this paper, we present the first formal study for *privacy-preserving hierarchical clustering*, featuring scalable cryptographic protocols that allow two parties to privately compute joint clusters on their combined sensitive input datasets. Previous work in this space introduces several cryptographic protocols (e.g., [17, 39, 40]), but without rigorous security definition and analysis.

Motivating applications. Hierarchical clustering is a class of unsupervised learning methods seeking to build a *hierarchy of clusters* over an input dataset (called *dendrogram*), typically using the *agglomerative* (bottom-up) approach. Clusters are initialized with single points and are iteratively merged using different cluster linkage (e.g., nearest neighbor and diameter for *single* and *complete* linkage, respectively). This is widely used in practice, often in application areas where the need for scalable PCL solutions is profound.

In healthcare, for instance, hierarchical clustering allows researchers, clinicians and policy makers to process medical data in order to discover useful correlations that can improve health practices—e.g., discover similar groups of genes [23], patient groups most in need of targeted intervention [55, 79], and changes in healthcare costs as a result of specific treatment [48]. To be of any value, such analyzed medical data contains sensitive information (e.g., patient records, gene information, or PII) that must be protected (also due to legislations such as HIPPA in US or GDPR in EU).

Also, in security analytics, hierarchical clustering allows enterprise security staff to process network and log data in order to discover suspicious activities or malicious events—e.g., detect botnets [34], malicious traffic [54], compromised accounts [12], and classify malware [8]. Again, security logs contain sensitive information (e.g., employee/customer data, enterprise security posture, etc.) that must be protected (due to industry regulations or for reduced liability).

As such, without privacy-protection provisions in place for collaborative learning, data owners are restricted to apply hierarchical clustering solely on their own private datasets, thus losing in accuracy and effectiveness. In contrast, our treatment of the problem as a PCL instance is a solid step towards employing the benefits of hierarchical clustering over any collection of available datasets. For instance, our techniques enable multiple hospitals to combine their patients' records and jointly cluster their medical data, thus allowing to provide their patients improved treatment options. They can also incentivize enterprises to combine their threat indicators and jointly cluster their security-related observations (e.g., determining whether threat indicators in two organizations are similar), thus allowing to provide more resilient "community-based" defenses against advanced threats. Importantly, in both cases, organizations remain compliant with privacy regulations on their private datasets.

Challenges and insights. Devising scalable and provably secure protocols for privacy-preserving collaborative hierarchical clustering entails a few technical challenges, as we briefly discuss.

The first challenge relates to the required formal treatment of the problem at the functionality level. MPC guarantees that no party learns anything about the other party's input, *other that what can be inferred directly from the joint output*. But what if the output is a superset of the input? Indeed, the dendrogram output by hierarchical clustering includes the input data and offers no privacy! To overcome this obstacle, we explore several refinements of hierarchical clustering that produce "redacted" outputs (i.e., dendrogram storing less information) and provide a definition that captures an ideal trade-off between accuracy and privacy. We are the first to present a formal security definition for the problem at hand in Section 3.

A number of challenges, then, relate to making the cryptographic protocols scalable to large datasets. Hierarchical clustering is already computationally heavy with running cost $O(n^3)$ (n being the size of training data). Standard methods for secure two-party computation (e.g., Yao's garbled circuits [81]) result in large communication, while use of fully homomorphic encryption [29] is still prohibitively costly, rendering the design of PCL protocols for hierarchical clustering a difficult task. Approximation algorithms for clustering such as CURE [35] is the *de facto* way to achieve scalability to massive amounts of data, however, incorporating both approximation and privacy protection in secure computation is far than obvious, and has not been considered previously in the literature.

We address these challenges through some unique insights that enable us to design practical PLC protocols for hierarchical clustering. First, in Section 4, we devise our main constructions as mixed protocols (e.g., [18, 36, 45]). We carefully decompose the computation needed in each iteration of hierarchical clustering into building blocks that can be efficiently implemented via either garbled circuits or additive homomorphic encryption [59]. We then explore performance trade-offs in our design space and select a combination of building blocks that achieves fast computation and low bandwidth usage, wherein we employ garbled circuits for cluster merging and distance updates, but homomorphic encryption for distance computation.

Moreover, in Section 5, we show how our design can be further optimized, by efficiently incorporating an existing *clustering approximation algorithm* [35] into our main framework, to achieve the best of both worlds: strong privacy guarantees and scalability. We explore several variants that exhibit different trade-offs between computational overhead and accuracy. We believe that this combination of cryptography with approximation holds great potential as it can eliminate the overheads associated with MPC at a small cost in the accuracy of the computed results.

In Section 6, we perform a comprehensive experimental evaluation of our protocols both on synthetic and real datasets, showing that they have modest overheads in practice. For example, our privacy-preserving approximate clustering protocol for single linkage, running on a dataset of one million records of 10 dimensions, requires 35 seconds of end-to-end time, 896KB of communication, at an accuracy of 97.09%.

Finally, we review related work in Section 7 and we conclude in Section 8. We present secure comparison building blocks, complete security proofs, and additional experiments in the Appendix.

Contributions. We summarize our contributions as:

- We provide a formal definition and design a provably secure mixed protocol for privacy-preserving hierarchical clustering supporting single and complete linkage.
- We present an optimized protocol for the case of single linkage that significantly improves computational overhead and communication cost.
- We propose a novel integration of widely used approximate clustering solutions into our protocols to produce versions that provide both scalability and strong privacy.

- We implement and experimentally evaluate our protocols both on synthetic and real datasets, validating their efficiency.

2 SYSTEM MODEL AND BACKGROUND

Here we provide descriptions of a class of hierarchical clustering algorithms supported by our framework, the adversarial model considered by our work, and the necessary cryptographic tools that we leverage.

Hierarchical clustering. Hierarchical agglomerative clustering is an unsupervised learning method used for creating a hierarchy of clusters in an iterative fashion. Let $\mathcal{D} = \{v_i\}_{i=1}^{n}$ be an unlabeled d-dimensional dataset, where $v_i \in \mathbb{X}^d$. $\mathbb{X}$ is the set of non-negative integers modulo 2^λ (e.g., $\lambda = 32$). We use the square Euclidian distance metric dist : $\mathbb{X}^d \times \mathbb{X}^d \rightarrow \mathbb{R}$, with $\text{dist}(\mathbf{x}, y) = \sum_{i=1}^{d}(\mathbf{x}_i - \mathbf{y}_i)^2$.

Hierarchical clustering (HC) is described in Figure 1 and graphically on the left of Figure 2. HC proceeds iteratively by creating first a cluster $C(v_i)$ for each datapoint $v_i \in \mathcal{D}$. In each iteration, the *closest clusters* according to a *cluster linkage distance* are merged and cluster linkages are updated. Common linkage distances which we support in this work are *single linkage* or *nearest neighbor* ($\delta(C_1, C_2) = \min_{\mathbf{x}\in C_1, \mathbf{y}\in C_2} \text{dist}(\mathbf{x}, \mathbf{y})$) and *complete linkage* or *diameter* ($\delta(C_1, C_2) = \max_{\mathbf{x}\in C_1, \mathbf{y}\in C_2} \text{dist}(\mathbf{x}, \mathbf{y})$). The clustering *dendrogram* T consists of the hierarchy of clusters created during this process (including the last level with the initial data points). The algorithm halts when exactly ℓ_t clusters are generated. Further, given T we can compute for each cluster $C \in T$, a representative $rep(C)$ and its size $size(C) = |C|$. Most commonly, a cluster's *representative* is the *centroid* defined as the average of all points in the cluster.

Problem definition and adversarial model. We define for the first time two-party privacy-preserving hierarchical clustering protocols. In our problem definition, two parties contribute independent datasets $P = (p_1, \ldots, p_{n_1})$ and $Q = (q_1, \ldots, q_{n_2})$, with $p_i, q_j \in \mathbb{X}^d$. They run a joint cryptographic protocol to generate a dendrogram T on the set $\mathcal{D} = P \cup Q$ of size $n = n_1 + n_2$. At the end of the protocol, the parties learn clusters' representatives and their sizes, while the input datasets are protected and not revealed during the cryptographic protocol. In this work we consider the setting of semi-honest adversaries, following the vast majority of works that target efficient secure computations (e.g., the privacy-preserving data mining works discussed in Section 7). In this setting, we assume that cheating parties follow the protocol, but try to infer additional information from the transcript of exchanged messages. We provide privacy definitions that minimize the amount of leakage of the input datasets, while including relevant information about the dendrogram computation.

Secure computation. We formulate the security of our private hierarchical clustering algorithm using the strong notion of *secure computation*, which allows two parties to evaluate a function on their joint inputs, while guaranteeing that each party learns nothing about the other's input other than what can be inferred from the output itself.

Hierarchical Clustering

(1) **[Level n] Initialize:** T has n leaf nodes $v_1, \ldots, v_n$. For each i, $C(v_i) \leftarrow \{v_i\}$.
(2) **[Level $n < i \leq \ell_t$]**
   (a) **Find closest clusters:** Find the two closest clusters $C(v_1)$ and $C(v_2)$ at level $i+1$ according to linkage $\delta(C(v_1), C(v_2)) = \min\{\delta(C(u), C(v)) : u, v \in \text{level-}(i+1)\}$..
   (b) **Cluster merging:** For each level-$(i+1)$ node $v \neq v_1, v_2$, level-i contains a node u with $C(u) \leftarrow C(v)$ and a directed edge is added from v to u. Finally, level-i forms a new cluster $C(v')$ with $C(v') \leftarrow C(v_1) \cup C(v_2)$ and two directed edges are added to T from v_1 and v_2 to v'.
   (c) **Update cluster linkage:** Compute linkage $\delta(C(v'), C(v))$, for all v at level i.
(3) **Representative computation:** Compute $rep(C)$ and $size(C)$ for each cluster $C \in T$.
(4) **Output generation:** Output dendrogram T with cluster representatives and sizes.

Figure 1: Hierarchical clustering algorithm.

We formalize our privacy requirements, using the standard two-party *ideal/real world* paradigm [33] which involves the following simulation experiment. Consider an *ideal* world, where the parties interact with a functionality f implemented by a *trusted third party* that privately interacts with both of them, collects their inputs, performs the computation, and returns its output. Each party learns the final output and, crucially, nothing else about the other party's input. In the *real* world, the trusted party is replaced by an interactive cryptographic protocol π executed jointly by the parties. Informally, π securely realizes f, if whatever can be learned by an adversarial party $\mathcal{A}$ while running this protocol, can be simulated by an algorithm, called the *simulator* Sim, that only interacts with the ideal functionality f, and does not know the other party's input.

Homomorphic encryption. *Homomorphic encryption* allows one to perform algebraic manipulations directly over ciphertexts, without decrypting them. Evaluating arbitrary functions over encrypted values is theoretically possible with Fully Homomorphic Encryption (FHE) [29], but not yet fully practical. However, the weaker notion of *partially* homomorphic encryption, where only specific arithmetic operations are supported, results in efficient implementations (e.g., [59, 62]). We use Paillier's additively homomorphic encryption (AHE) [59]. The plaintext space is $\mathbb{Z}_N$, with N public RSA modulus and pk, sk the encryption/decryption key pair. We denote the encryption of $m \in \mathbb{Z}_N$ by $[m]$ (omitting, for simplicity, the specific public key used). Paillier guarantees that decrypting $[m] \cdot [m'] \mod N^2$ results in $m + m' \mod N$. Moreover, decrypting $[m]^k$ modulo N^2 results in $km \mod N$. Finally, multiplying the ciphertext $[m]$ by $[0]$ results into a "freshly" randomized encryption of m.

Yao's Garbled Circuits. This celebrated result by Yao [82, 83] has become one of the most widely adopted methods for secure two-party computation. A Garbled Circuit (GC) protocol enables two parties P_1, P_2 to evaluate a boolean circuit C on joint inputs x_1, x_2 such that each party learns nothing about the other party's input,

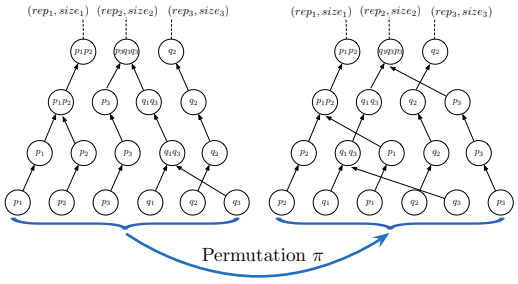


Figure 2: (Left) Dendrogram produced by hierarchical clustering over lists $P = (p_1, p_2, p_3)$ and $Q = (q_1, q_2, q_3)$ for 3 target clusters. (Right) The dendrogram after permuting the concatenated lists under permutation π.

other than what can be inferred by the output y. This is achieved by having one of them, called the *garbler*, generate an encrypted truth table for each gate in C. The other party, the *evaluator*, can then evaluate the circuit by sequentially decrypting these truth tables in a way that preserves input privacy.

3 SECURITY DEFINITION

In this work, we define privacy-preserving hierarchical clustering, according to standard definitions [33]. We need to define an ideal functionality that upon receiving the two parties' inputs responds with the hierarchical clustering output, without revealing the private inputs. As it is evident from the hierarchical clustering description in Section 2, its output is the entire dendrogram T, from which the inputs can be immediately inferred. Due to this, we need to re-define the problem of hierarchical clustering to reveal less (but still useful) information. Below we describe gradually our approach to get to the final definition.

Attempt 1: Removing the dendrogram. What if we simply remove the entire dendrogram T and just report the representatives $rep(v_1), \ldots, rep(v_{\ell_t})$ and sizes $|C(v_1)|, \ldots, |C(v_{\ell_t})|$ for each of the ℓ_t clusters in layer ℓ_t? Then the result is essentially similar as what can be achieved by simpler clustering techniques, e.g., k-means. However, the motivation for using hierarchical clustering in the first place, is that the rich dendrogram structure provides insights on how the clusters where formed and in which order. For instance, in healthcare the dendrogram provides useful information about relationships between features and factors that contribute to prevalence of a disease [23]. In biology, the dendrogram's hierarchical structure could reveal interesting relationships between plants and animals, their habitat and ecological subsystems [32]. Therefore, we need to keep the dendrogram structure, but reduce the amount of information it reveals about the exact inputs.

Attempt 2: Keep only the dendrogram structure. A second approach is to keep only the dendrogram structure without the intermediate clusters $C(v)$ at layers $\ell_t + 1, \ldots, n$. This has the advantage of protecting the inputs at the last layer. However, it will reveal the exact mapping between input points and the final clusters, which results in considerable leakage. For instance, a party can infer upper bounds on distances between its points and points owned by the other party, or distances between the other party's points since it knows the exact order in which clusters were merged.

Inputs Receive list $P \in [\mathbb{X}^d]^{n_1}$ from party 1, list $Q \in [\mathbb{X}^d]^{n_2}$ from party 2, and distance dist, linkage distance δ and number of target clusters ℓ_t.

Dendrogram Computation Choose a permutation $\pi : [n] \to [n]$, where $n = n_1 + n_2$. Let $L = P||Q$ be the concatenated input list. Compute list L' as $\pi(L)$ and compute the corresponding hierarchical clustering dendrogram T.

Information Redaction Initialize two empty lists REP, $SIZE$. For each $C(v) \in T$:

(1) If $C(v)$ is in layer ℓ_t, compute $rep(v)$, $|C(v)|$ and set $REP \leftarrow REP \cup rep(v)$ and $SIZE \leftarrow SIZE \cup |C(v)|$.
(2) Set $C(v) \leftarrow \emptyset$ in T.

Output Send REP, $SIZE$, T to both parties.

Figure 3: Ideal functionality for Hierarchical Clustering.

Attempt 3: Randomly permute nodes. We can further permute the nodes at each layer under a random permutation. This protects the mapping between inputs and clusters, but we "destroyed" too much information. In particular, the structure of permuted dendrogram only shows that two clusters were merged in each iteration, but this is something that the participants know already! We thus need to preserve the dendrogram structure with more utility.

Attempt 4: Randomly permuting once. Finally, we propose to keep the dendrogram structure, but permute nodes only once at the last layer. We believe that this represents the right balance between preserving the utility of the dendrogram, and reducing the amount of revealed information. The parties can still infer in which order the clusters where merged and the internal topology of the clusters, which has applications in medicine and biology [23, 32]. At the same time, this approach hides the mapping of points to clusters and the exact ordering of cluster merging, thus preventing simple attacks based on inferring distances between points.

Security definition. We are ready to provide the definition for privacy-preserving hierarchical clustering.

Definition 3.1. A protocol $\pi = \langle P_1, P_2 \rangle$ is a *secure privacy-preserving hierarchical clustering* in the presence of *static, semi-honest adversaries* if it realizes the ideal functionality f^*_{HC} (defined in Figure 3). Specifically, for $i = 1, 2$ and for all λ, there exists non-uniform probabilistic polynomial-time Sim_{P_i} such that $\mathsf{Sim}_{P_i}(1^\lambda, x_i, f^*_{HC}(x_1, x_2)) \cong \mathsf{view}^\pi_{\mathcal{A}_{P_i}}$, where x_1, x_2 are the respective inputs of P_1, P_2, and $\mathsf{view}^\pi_{\mathcal{A}_{P_i}}$ consists of the randomness tape of P_i and all incoming messages received while running protocol π.

4 MAIN CONSTRUCTION

We now present our main protocol for privacy-preserving hierarchical clustering between two parties P_1 and P_2. The protocol is split into two phases: a one-time setup phase called HClustering.Setup (described in Algorithm 1), and an iterative algorithm called HClustering that repeatedly agglomerates the two closest clusters until the desired number of clusters is reached (described in Algorithm 2). The general flow of our main construction is depicted in Figure 4. For clarity of presentation, we present the main construction for the case of single-dimensional data ($d = 1$).

Then, in Section 4.1 we discuss the necessary modifications to accommodate more dimensions. Finally, in Section 4.2 we present a highly optimized protocol tailored to single linkage.

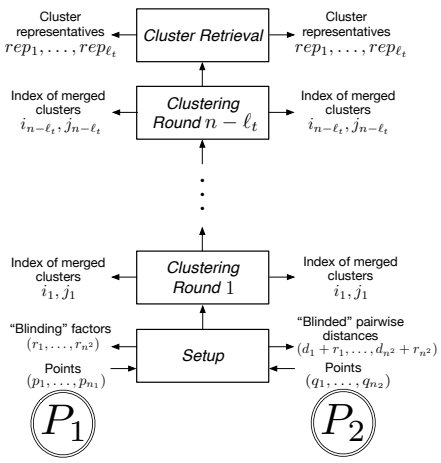


Figure 4: The interaction flow of our main hierarchical clustering construction between parties $\mathsf{P}_1, \mathsf{P}_2$ holding n_1, n_2 points ($n = n_1 + n_2$). The protocol terminates once there are ℓ_t clusters left and the parties learn for each cluster its size and representative. The parties also learn the *permuted* clustering tree.

Key insights. Designing efficient privacy-preserving protocols for a complex functionality like hierarchical clustering (with cubic cost in the input size) is technically challenging. We enumerate below the main challenges and our key insights to address them:

- *Computational complexity:* Applying MPC protocols like Yao's GC or FHE to the entire computation is infeasible. We resort to *mixed protocols*, a design paradigm that combines AHE with GC (following [10, 18, 43, 51, 53]). We decompose the computation into building blocks (e.g., comparison, minimum computation, cluster merging), design tailored efficient protocols for each of these, and finally combine the components.
- *Converting between representations:* In mixed protocol design, a substantial cost is involved in converting inputs from Yao's GC to AHE. To reduce this cost, we use square Euclidian distance metric, which has the nice property that a distance computation between two points can be computed solely using AHE.
- *Protecting input privacy:* To ensure privacy, we design a Setup protocol for randomly permuting and statistically blinding the inputs so that both parties are oblivious to the indices of their points in the dendrogram.
- *Iterative closest cluster computation:* A significant cost in HC is incurred by the iterative computation to identify closest clusters. For single linkage, we store the index of the closest cluster to each cluster, and simply update that when merging clusters. This results in an optimized protocol with quadratic cost in input size.

Secure comparison of values. Our construction uses several building blocks for secure comparison based on garbled circuits (see Appendix C for their implementation). First, ArgminSelect computes the index of the minimum value in a matrix of blinded

Algorithm 1: HClustering.Setup: Setup Encrypted Clusters

P_1**'s Input** : $\{p_1, \ldots, p_{n_1}\}$,(pk, sk), pk'
P_2**'s Input** : $\{q_1, \ldots, q_{n_2}\}$,(pk', sk'), pk

Phase I: Compute encrypted distance matrix

P_2:
Compute $n_2 \times n_2$ encrypted distance matrix $\mathbf{B}$ under pk', with $B_{ij} = [\text{dist}(q_i, q_j)]$ for $i, j = 1, \cdots, n_2$.
For each q_i compute $[q_i]$ and $Q_i = \{([q_i^2], [-2q_i])\}$, also under pk'.
Send $\mathbf{B}$, $\mathbf{Q} = \{[q_1], \cdots, [q_{n_2}]\}$, $\mathbf{Q}' = \{Q_1, \ldots, Q_{n_2}\}$ to P_1.

P_1:
Compute $n_1 \times n_1$ encrypted distance matrix $\mathbf{A}$ under pk', with $A_{ij} = [\text{dist}(p_i, p_j)]$ for $i, j = 1, \cdots, n_1$.
Construct an $n \times n$ encrypted symmetric matrix $\mathbf{M}$ where M_{ij} is computed as follows:
foreach $i, j = 1 \ldots n$ **do**
  if $1 \le i, j \le n_1$ **then**
    $M_{ij} = A_{ij}$
  else if $n_1 + 1 \le i, j \le n_1 + n_2$ **then**
    $M_{ij} = B_{(i-n_1)(j-n_1)}$
  else
    $M_{ij} = [p_i^2] \cdot [-2q_j]^{p_i} \cdot [q_j^2]$, where the first encryption is under pk' and the other two come from Q_j.
For each point p_i, compute $[p_i]$ under pk' and set $\mathbf{P} = \{[p_1], \ldots, [p_{n_1}]\}$. Set $\mathbf{L} = \{\mathbf{P}, \mathbf{Q}\}$ the encrypted list of all points.

Phase II: Shuffle M and L

P_1:
Create two $n \times n$ encrypted randomness matrices $\mathbf{R}$, $\mathbf{R}'$ as follows. Generate random values $r_{ij} \leftarrow \{0,1\}^\kappa$ for $i, j = 1, \ldots, n$ and set $\mathbf{R}_{ij} = [r_{ij}]$, where the encryption is under pk, and $\mathbf{R}'_{ij} = [r_{ij}]$ where the encryption is under pk'.
foreach $i, j = 1 \ldots n$ **do**
  Blind M_{ij} by setting $M_{ij} \leftarrow M_{ij} \cdot R'_{ij}$.
Create two encrypted lists S, S' as follows. Generate random values $\sigma_i \in \{0,1\}^\kappa$ for $i = 1, \ldots, n$. Set $\mathbf{S}_i = [\sigma_i]$, where encryption is under pk, and $\mathbf{S}'_i = [\sigma_i]$, where encryption is under pk'.
foreach $i = 1 \ldots n$ **do**
  Blind L_i by setting $L_i \leftarrow L_i \cdot S'_i$.
Generate a random permutation $\pi_1(n)$, use it to permute the rows and columns of $\mathbf{M}$, $\mathbf{R}$ and the lists $\mathbf{L}$, $\mathbf{S}$, and send them to P_2.

P_2 :
Create an empty $n \times n$ matrix $\mathbf{V}$.
foreach $i, j = 1 \ldots n$ **do**
  Generate random values $r'_{ij} \leftarrow \{0,1\}^\kappa$ for $i, j = 1, \ldots, n$.
  Decrypt M_{ij} using sk′ and store the result at V_{ij}.
  Set $V_{ij} \leftarrow V_{ij} + r'_{ij}$ and $R_{ij} \leftarrow R_{ij} \cdot [r'_{ij}]$ where encryption is under pk.
foreach $i = 1 \ldots n$ **do**
  Generate random values $s'_i \in \{0,1\}^\kappa$.
  Set $L_i \leftarrow L_i \cdot [s'_i]$ where encryption is under pk'.
  Set $S_i \leftarrow S_i \cdot [s'_i]$ where encryption is under pk.
Generate a random permutation $\pi_2(n)$, use it to permute the rows and columns of matrices $\mathbf{V}$, $\mathbf{R}$ and lists $\mathbf{L}$, $\mathbf{S}$, and send $\mathbf{R}$, $\mathbf{L}$, and $\mathbf{S}$ to P_1.

Phase III: Unblind list L

P_1:
Decrypt $\mathbf{S}$ using sk. Let $\sigma_1, \ldots, \sigma_n$ be the returned values. For $i = 1, \ldots, n$ set $L_i \leftarrow L_i \cdot [\sigma_i]^{-1}$ where the encryption is under pk'.
Decrypt $\mathbf{R}$ using sk and store the result at R_{ij} for $i, j = 1 \ldots, n$.

distances (e.g., [10, 46, 85]). P_2 holds a blinded encrypted matrix $V_{ij} = v_{ij} + r_{ij}$ for a large random value r_{ij} that has been chosen by P_1 and is unknown to P_2. On the other hand, P_1 holds the blinding factor r_{ij} for each V_{ij}. The garbled circuit for ArgminSelect takes all these values as inputs, removes the random factors, and compares all values. The final output sent to both parties is the indexes of the minimum element $\min_{i,j}\{v_{ij}\}$.

Second, we need protocols MinSelect and MaxSelect [6, 46] that operate on inputs u and v blinded with randomness values r_1 and r_2. The goal is to output either the minimum or maximum of the two values, blinded with a different factor. The function f that the two parties evaluate can be defined as $f(u, v, r_1, r_2, r') = \min\{u - r_1, v - r_2\} + r'$ (or maximum for MaxSelect). The first two input values are provided by P_2, who also receives the final output. The last three input values are provided by P_1 whose output is empty.

Setup phase. The setup phase is an interactive process described in Algorithm 1. Initially P_1 holds points $p_1, \ldots, p_{n_1}$ and P_2 holds points $q_1, \ldots, q_{n_2}$. Both parties have a Paillier encryption key pair, (pk, sk) and (pk', sk') respectively. The protocol is based exclusively on AHE operations, its main goal being of preparing the data needed for the joint clustering protocol. The output is an encrypted matrix of blinded pairwise distances between all $n = n_1 + n_2$ points and the list of encrypted points known to P_1, while P_2 holds the blinding randomness for the distance matrix.

The setup starts with **Phase I** (lines 1-16) for computing the encrypted pairwise distance matrix **M** and a list of encrypted points **L**. In lines 3-5, P_2 computes all pairwise Euclidean distances between his points q_i and encrypts them under pk', creating a $n_2 \times n_2$ matrix **B**. For each point q_i he also computes $[q_i^2], [-2q_i]$ and sends them to P_1. In lines 7-15, P_1 computes his own pairwise distance matrix **A**. He then builds the joint encrypted distance matrix **M** as follows. He fills entries M_{ij} with $1 \le i, j \le n_1$ (the "top-left" part of the matrix) with encryptions of the entries of **A**, i.e., distances between his own points. Likewise, he fills entries M_{ij} with $n_1 + 1 \le i, j \le n_1 + n_2$ (the "bottom-right" part of the matrix) with the entries of **B**, i.e., encryptions of distances between the points of P_2. Finally, he fills all other entries with the encrypted distances of pairs p_i, q_j i.e., pairs where one point belongs to each party. This is achieved by exploiting the homomorphic properties of Paillier encryption: $M_{ij} \leftarrow [p_i^2] \cdot [-2q_j]^{p_i} \cdot [q_j^2] = [p_i^2 - 2p_iq_j + q_j^2] = [(p_i - q_j)^2]$. P_1 also encrypts his own points p_i under pk' and stores a list **L** of encrypted points (line 16).

Phase II performs shuffling and blinding of **M** and **L**. To motivate this, assume that both parties know the indices of their points in **M**. In that case, P_1 could learn that his cluster is merged with one of P_2's points, implying that P_2's point is closer than any other point of P_1. In essence, this reveals significant information about the "internal topology" of the clusters and does not satisfy the security definition. To eliminate the above leakage, P_1 and P_2 permute the rows and columns of **M** and **L**, and blind them with random values. First P_1 homomorphically "blinds" entries of **M** by random $r_{ij} \in \{0,1\}^\kappa$, which he adds to M_{ij} (lines 19-21). κ is a statistical security parameter, used for the length of blinding factors. P_1 also blinds entries of **L** with random $\sigma_i, i \in [1, n]$ (lines 22-24). Then he chooses a random permutation $\pi_1(n)$ for the rows and columns of **M**, list **L**, as well as randomness matrix $\mathbf{R} = \{r_{ij}\}$ and list $\mathbf{S} = \{\sigma_i\}$ (line 25). P_1 sends both matrices and lists to P_2.

P_2 will now perform roughly the same process using a permutation π_2 of his choice (lines 27-36). The result will be that, since each party knows only one of the two permutations, the output of $\pi_2 \circ \pi_1$ looks random to both of them. P_2 decrypts the blinded distance matrix with his secret key sk', adds fresh randomness r'_{ij} to each cell, and permutes the resulting table with π_2. He also homomorphically blinds the randomness table **R** he received from P_1 using his freshly generated randomness and also permutes it. Finally, he repeats this process for **L** and the corresponding randomness list and sends the blinded encrypted matrix **V**, list **L**, and **R**, **S**, and **L** back to P_1.

Lastly, in **Phase III** P_1 "unblinds" the list **L**. P_1 decrypts the list **S** using sk and then homomorphically "unblinds" list **L**. This is done by taking the decrypted randomness, multiply by -1 (modulo P_2's choice of Paillier parameters), encrypt it under pk' and homomorphically add it to corresponding element in list **L**. Note that, if the protocol has been executed correctly, this value is the additive inverse of the sum of the random values used by the two parties to blind this entry, which cancels out. The final output for the two parties is as follows: P_2 holds a matrix **V** which contains the blinded encrypted pairwise distances, and P_1 holds a table with the corresponding randomness for each entry of V_{ij} *at the same position* and a list **L** of the points encrypted under pk', such that the points are ordered in the same way as the columns and rows of **V**. The two tables and the list are permuted such that neither party can associate any element with a specific party.

Iterative clustering. Next, we explain how the main clustering phase of the protocol runs (see Algorithm 2). The protocol follows the main steps of the hierarchical clustering algorithm from Figure 1. Initially, each row i of **V** corresponds to a single-point cluster C_i. The two parties run an iterative protocol where at each iteration (line 5-18) they identify the closest clusters according to the linkage metric, merge them into a new cluster, and update linkages to all other clusters. These steps are performed with sub-protocols instantiated with Yao's garbled circuits.

In more details, the two parties find in each round the location of the minimum distance between existing clusters with protocol ArgminSelect (lines 6-7). The output that both parties receive is the indexes i, j of the minimum value in the matrix (not the value itself). P_2 must at each iteration "merge" the two clusters C_i, C_j into a single cluster (lines 10-16), stored in row and column i, as follows. First, for each position $k \neq i, j$ he must set the distance of cluster k from this new cluster. To do so, the two parties run the comparison protocol for values V_{ik}, V_{jk}, i.e., the previous linkage between cluster C_k and clusters C_i, C_j. Depending on the linkage method (single or complete), the two parties run either MinSelect or MaxSelect. This protocol also receives as input from P_1 a fresh random value ρ and produces a one-sided output: P_2 gets the minimum (resp. maximum) of the two values, blinded with ρ (and P_1 gets nothing). P_2 then stores this result at V_{ik} and P_2 stores the randomness ρ at the same positions in his randomness table.

In order to recreate the clusters, P_1 maintains a data structure Λ storing indices of points from each cluster. This is initialized in line 3 and updated after cluster merging in line 8. Finally, P_2

Algorithm 2: HClustering: Privacy-preserving clustering

P_1**'s Input** : $\{p_1, \ldots, p_{n_1}\}$
P_2**'s Input** : $\{q_1, \ldots, q_{n_2}\}$, ℓ_t, type

1 P_1 generates (pk, sk) $\leftarrow \mathsf{Gen}(1^\lambda)$, P_2 generates (pk′, sk′) $\leftarrow \mathsf{Gen}(1^\lambda)$, and they send pk, pk′ to each other.
2 P_2 sets level $\ell = 0$ and runs HClustering.Setup with P_1.
3 P_2 gets matrix $\mathbf{V}$ and P_1 gets matrix $\mathbf{R}$ and list $\mathbf{L}$.
4 **Initialize:** P_1 sets $\Lambda[i] \leftarrow i$ for $i = 1, \ldots, n$.
5 **repeat**
6 **Find closest clusters:** Let $D = \{V_{ij}\}_{i \le j, i \ne j}$ be the blinded linkages between clusters C_i, C_j.
7 P_1 and P_2 execute ArgminSelect with P_2's input as D and P_1's input as $\{R_{ij}\}_{i \le j, i \ne j}$.
8 P_1, P_2 get index $\alpha = argmin\{D\}$, and compute i, j such that $in + j = \alpha$.
9 **Cluster merging:** P_1 sets $\Lambda[i] \leftarrow \Lambda[i]||\Lambda[j]$ and $\Lambda[j] \leftarrow$ null.
10 **foreach** $k = 1, \ldots, n, k \ne i, j$ **do**
11 **if** V_{ik} or $V_{jk} = \perp$ **then**
12 P_2 sets V_{ik} and V_{ki} to $\perp$ and P_1 sets R_{ik} and R_{ki} to $\perp$.
13 **else if** type = single **then**
14 **Update cluster linkage:** P_1 and P_2 execute MinSelect with P_2's input V_{ik}, V_{jk} and P_1's input R_{ik}, R_{jk}, ρ where $\rho \in \{0,1\}^\kappa$ is randomly chosen by P_1. P_2 gets value $v = min\{V_{ik}, V_{jk}\} + \rho$ and sets V_{ik} and V_{ki} to v and P_1 sets R_{ik} and R_{ki} to ρ.
15 **else**
16 **Update cluster linkage:** P_1 and P_2 execute MaxSelect with P_2's input V_{ik}, V_{jk} and P_1's input R_{ik}, R_{jk}, ρ where $\rho \in \{0,1\}^\kappa$ is randomly chosen by P_1. P_2 gets value $v = max\{V_{ik}, V_{jk}\} + \rho$ and sets V_{ik} and V_{ki} to v P_1 sets R_{ik} and R_{ki} to ρ.
17 P_2 updates $\mathbf{V}$ and P_1 updates $\mathbf{R}$ by setting the j-th row and column to $\perp$.
18 P_2 sets $\ell \leftarrow \ell + 1$.
19 **until** $\ell > \ell_t$;
20 **Representative computation:** P_1 performs the following:
21 **foreach** $i = 1, \ldots, n$ **do**
22 **if** $\Lambda[i] \ne$ null **then**
23 Let J_i be the set of indexes in $\Lambda[i]$. Compute $E_i = \prod_{j=1}^{J_i} L_j$.
24 P_1 sends to P_2 the ℓ_t pairs $E_i, |J_i|$.
25 P_2 decrypts each value E_i with sk' to get cluster representative rep_i, which he sends back to P_1.
26 **Output generation:** Both parties return as output the ℓ_t pairs $(rep_i/|J_i|, |J_i|)$.

sets all the values in the j-th row and column to $\perp$, signifying the non-existence of a cluster at this position.

The algorithm stops when ℓ_t clusters are created (after $n - \ell_t$ rounds). Clearly, when the protocol terminates there are ℓ_t non-null rows and columns, each corresponding to a cluster in the output. The last part of the protocol creates cluster representatives (lines 20-26). This is achieved by exploiting the homomorphic properties of AHE. Using Λ, P_1 identifies which points go into each cluster. Assuming that cluster i includes points with indices J_i, P_2 computes $\prod_{j=1,\ldots,J_i} L_j = \prod_{j=1,\ldots,J_i}[p_j]$ and sends the resulting ciphertext to P_2 together with the cardinality $|J_i|$ of the cluster. The latter decrypts it using sk' to get result $rep_i = \sum_{j=1,\ldots,J_i} p_j$ and sends it back to P_2. Both parties return as their final output the ℓ_t cluster representative/cardinality pairs $(rep_i/|J_i|, J_i)$.

We now state the theorem capturing our construction security (full proof in Appendix D).

Theorem 4.1. *Assuming Paillier's encryption scheme is semantically secure,* MinSelect, MaxSelect, *and* ArgminSelect *are instantiated with secure oblivious transfer and garbling scheme, the protocol π_{HC} securely evaluates function f^*_{HC} as per Definition 3.1.*

Asymptotic complexity. We assess the performance costs for the two parties during the execution of π_{HC}. We assume that each cryptographic operation takes constant time. During the setup phase, the dominant cost for both parties comes from performing $O(n^2)$ cryptographic operations (in order to populate tables $\mathbf{V}, \mathbf{R}$). The cost for garbling and evaluating a circuit C is $O(|C|)$ where $|C|$ is the number of wires in the circuit. During the ℓ round, ArgminSelect entails $n^2 - 2\ell$ comparisons of d^2-bit values (cluster distances) and subtractions of κ-bit values for a total size of $O(\kappa(n^2 - \ell))$. Likewise, the size of each circuit for MinSelect/MaxSelect is of size $O(\kappa)$ and $n^2 - 2\ell$ such circuits are evaluated at round ℓ. The final phase cost for both parties is $O(\ell_t)$. Thus, the total performance cost for both parties is $O(\kappa n^3)$, which is the same as the cost of unencrypted hierarchical clustering multiplied by a factor κ for the statistical hiding parameter.

Extensions. Our protocol can be extended to other distance metrics, including L_1, L_2 or Euclidian, and in general any L_p distance for $p \ge 1$. The only modification is computing the pairwise distance matrix during setup. We chose the squared Euclidian distance because it enables pairwise computation between the two parties' points solely using AHE and without interaction between the parties. If other distance metrics are used, then we need to design a mixed protocol for computing the pairwise distances initially.

Additional challenges emerge while trying to extend our protocols and maintain security in the malicious threat model, in which participants can mis-behave arbitrarily. Standard techniques such as augmenting the homomorphic ciphertexts with zero-knowledge proofs and cut-and-choose for garbled circuits can be applied, as well as, more recent developments in efficient secure computation [76].

We believe that our protocols could be extended to support multiple participants, as typically considered by federated learning. This would require drastically different techniques to achieve practical performance but recent works provide promising results [44, 77]. One possible direction is to replace the garbled circuit components with secret sharing solutions as in Araki et al. [4]. We plan to explore this in follow-up work.

4.1 Scaling to multiple dimensions

So far we considered single-dimensional data ($d = 1$). In practice, hierarchical clustering is mostly applied to higher-dimension data ($d > 1$) and our protocol can be easily adapted. The core part of our protocol deals with comparisons between squared Euclidean distances, therefore it remains entirely unaffected by the number of dimensions. The modifications thus have to do with the setup process and the representative computation.

Initially, P_2 represents each point not by three but by $3d$ encryptions, essentially running step 4 of Algorithm 1 independently for each dimension. Thus, list $\mathbf{Q}$ consists of dn_2 encryptions and $\mathbf{Q}'$ of $2dn_2$. Then, P_1 does the same for his points and computes the values M_{ij} (step 15) as the sum of the per dimension computation (which can be achieved with AHE). The shuffling process remains largely unaffected, other than the fact that lists $\mathbf{L}, \mathbf{S}, \mathbf{S}'$ consist of dn encryptions instead of n. Finally, the representative computation (Algorithm 2, step 23) needs to compute E_i as a vector of d values.

4.2 Optimization for single linkage

We discuss how our protocol can be optimized if we focus only on single linkage. In particular, we present a modified version of the protocol that runs in time $O(\kappa n^2)$, as opposed to $O(\kappa n^3)$.

Our main protocol performs a quadratic number of comparisons in every round of clustering to identify the closest clusters. This takes place with the sub-protocol ArgminSelect (steps 6-7 in Algorithm 2) which compares all the values in matrix $\mathbf{V}$. This "naive" approach works independently of the linkage function chosen. However, it turns out that for the case of single linkage we can use a much faster alternative that reduces the number of comparisons per round to $O(n)$ (as opposed to $O(n^2)$). This technique is well-known (e.g., see [52, Section 17.2.1]) and we explain it briefly.

After computing the encrypted distance matrix $\mathbf{V}$, the two parties compute the minimum distance per matrix row and store it in a separate array RowMin (RowMin$[i]$ stores the minimum value in the i-th row of $\mathbf{V}$). We note that this can be achieved by essentially running the ArgminSelect protocol separately per matrix row. Then, during every clustering round *instead* of running the ArgminSelect protocol (steps 6-7 in Algorithm 2) in order to locate the minimum value in $\mathbf{V}$, the two parties proceed as follows:

(1) P_1, P_2 run ArgminSelect over the values of RowMin and get the index i of the cluster that corresponds to the row of $\mathbf{V}$ that contains the minimum values.
(2) P_1, P_2 run ArgminSelect over the values from the i-th row of $\mathbf{V}$ and get the index j of the closest cluster to cluster i.

Moreover, after updating the inter-cluster distances in $\mathbf{V}$ (steps 10-17 in Algorithm 2) the two parties have to update array RowMin. This is done as follows:

(1) P_1, P_2 repeatedly run MinSelect over the i-th row of $\mathbf{V}$ and store the minimum distance in RowMin$[i]$.
(2) P_2 sets RowMin$[j]$ to $\perp$.
(3) For every $k \neq i$, P_1, P_2 run MinSelect comparing RowMin$[k]$ with V_{ki} and store the result in RowMin$[k]$.

Note that the above steps require at most $4(n-1)$ comparisons, as opposed to at most $n^2/2-1$ needed by our basic protocol to identify the minimum distance index. In practice, this results in significant improvements to the performance of our privacy-preserving hierarchical clustering solution, as demonstrated in Section 6, since all these comparisons are done using garbled circuits.

5 APPROXIMATE CLUSTERING

The cryptographic machinery required for our main privacy preserving hierarchical clustering protocol from Section 4 imposes overhead in practice. This is due to the fact that the privacy-preserving protocol performs the same number of operations as the standard (non-private) one, but every operation over plaintext values is replaced by a cryptographic operation. Independently of how well-optimized the cryptographic code is, cryptographic operations will ultimately be slower!

Therefore, to further scale to larger datasets, we explore how we can exploit efficient *approximations*. In particular, we adapt existing approximate clustering techniques to our private hierarchical clustering mechanism. We consider the CURE approximate clustering algorithm [35] and design for the first time a privacy-preserving approximate clustering variant of CURE. CURE is a classical algorithm known to be resilient to outliers and achieve high accuracy with a relatively small number of samples (less than 1% of original data). Due to these advantages, we select CURE in our design but we believe similar privacy-preserving techniques can be applied to other approximate clustering algorithms (e.g., Birch [84]).

Algorithm 3: The CURE approximate clustering algorithm

Input : n, s, p, q, t_1, t_2, R

1 [**Random sampling:**] Randomly sample s data records from the dataset.
2 [**Phase I Clustering:**]
3 [**Partitioning:**] Partition the sample of size s into p partitions, each of size s/p.
4 [**Clustering:**] Cluster each partition using hierarchical clustering algorithm until number of clusters is $s/(pq)$.
5 [**Outlier Elimination:**] Eliminate within-partition clusters with less than t_1 points.
6 [**Phase II Clustering:**]
7 [**Clustering:**] Cluster all remaining clusters from all partitions, using hierarchical clustering algorithm.
8 [**Outlier Elimination:**] Eliminate clusters with less than t_2 points.
9 [**Representative Selection:**] Sample R representatives from each remaining cluster.
10 [**Cluster Assignment:**] For each point, find the closest representative and assign it to the representative's cluster.

CURE (see Algorithm 3) starts by sampling s data points from the original dataset of size n. Then, **Phase I Clustering** (lines 2-5) partitions the sample into p equally-sized partitions. Standard hierarchical clustering algorithm is applied to each partition until the desired number of clusters $s/(pq)$ is achieved (q is a parameter related to the number of clusters per partition). Phase I ends with outlier elimination, in which small clusters with less than t_1 points are eliminated. In **Phase II Clustering** (lines 6-8), another round of clustering is performed on the output by Phase I. At the end of the second clustering, small clusters with less than t_2 points are eliminated. Finally, a number of R representatives are selected per cluster and each point is assigned to the cluster of the closest representative (lines 9-10). The parameters of the CURE clustering algorithm and the values we selected according to the original paper are given in Table 1.

Next, we design different privacy-preserving two-party approximate clustering algorithms based on CURE. The main design choice is to determine which steps of Algorithm 3 will be executed individually by each party, and at which phase the parties will start executing the protocol jointly. After deciding on this, we discuss

how the resulting scheme can be made privacy-preserving by using our constructions from Section 4 for hierarchical clustering. A naive implementation of two-party CURE algorithm will execute all phases of the protocol jointly, at the expense of higher computational complexity. At the other end of the spectrum, the parties will execute all phases disjointly and merge the resulting clusters at the very end. That will result in a much more efficient protocol (as the majority of the cost is born locally), at the expense of loss in accuracy. We thus study three variants of approximate CURE clustering (see Table 2). **Joint I-II Clustering** executes Phases I and II jointly, while the random sampling is done individually by parties. Second, **Joint II Clustering** executes only Phase II clustering jointly, but performs Phase I clustering individually at each party. Third, **Disjoint Clustering** performs all the steps of the protocol (lines 1-10) locally at each party.

5.1 Integrating approximation with privacy

Let us now see how our approximation variants of CURE can be executed using our privacy-preserving clustering protocol.

Disjoint Clustering. Clearly, for this case there is no need for modifications—all data is handled locally by parties without any risk of leakage.

Joint I-II Clustering. For this case there is one important challenge: how to make the transition from Phase I to Phase II. In particular, during Phase I the two parties can run p copies of our main construction in parallel, one for each partition, acquiring $s/(pq)$ cluster representatives and sizes. Phase II should consist of a single execution of our protocol over the returned clusters. On the positive side, the necessary outlier elimination is trivial, just by looking at the returned cluster sizes from Phase I. However, to proceed to Phase II the inter-cluster distances of all the remaining clusters need to be computed. Unfortunately, this is clearly impossible given only the cluster representatives.

There are various ways around this obstacle. For example, the parties can run a separate secure computation protocol, in order to compute inter-partition cluster distances and bootstrap Phase II. However, this introduces additional overhead. Alternatively, the parties can initiate Phase II by treating each cluster from Phase I as a singleton that consists only of its representative. However, this ignores the cluster size and may affect the clustering accuracy. Instead, we take a simpler approach by setting the number of partitions $p = 1$ for Phase I. In this case, all the intra-cluster distances are computed originally (since they all belong to the same "partition") and the transition between phases is seamless. Interestingly, this does not affect the algorithm's accuracy. In fact, multiple partitions were originally introduced for efficient parallel processing [35, Section 4.2]. Indeed, our experimental evaluation indicates that the accuracy is not impacted by the choice of p.

Joint II Clustering. In this case the parties perform Phase I separately (in a non-private way) and receive a number of clusters. Then, they run Phase II using our privacy-preserving protocol over these returned clusters. Semantically, this looks like computing the dendrogram of Figure 2 starting at an intermediate level where some clusters already have multiple points.

One part of our protocol needs to be modified. In order to fill table $\mathbf{V}_{ij}$, the two parties run a slightly modified version of Algorithm 1.

Parameter	Description	Value
n	Size of training set	≤ 1 million
s	Sample size	$100 - 1000$
p	Number of partitions	$p = 1, 3, 5$
q	Controls clusters per partition	$q = 3$
t_1	Phase I outlier threshold	$t_1 = 3$
t_2	Phase II outlier threshold	$t_2 = 5$
R	Representatives per cluster	$R = 1, 3, 5, 7, 10$

Table 1: CURE clustering parameters and values.

Algorithm	Local Computation	Joint Computation
Joint I-II clustering	Random sampling	Phase I Phase II
Joint II clustering	Random sampling Phase I	Phase II
Disjoint clustering	Random sampling Phase I Phase II	✗

Table 2: Two-party CURE clustering variants.

The algorithm computes inter-cluster distances in two ways. For clusters belonging to the same party, the distance is computed in the plain and then appropriately encrypted, similar to steps 2-7 in Algorithm 1, but this time using the specified linkage function. For each pair of clusters that belong to different owners, the parties run a sub-protocol that first privately computes squared Euclidean distances across all possible pairs of points in the clusters, and then evaluates the inter-cluster distance based on the linkage type (single or complete). This sub-protocol is just our Setup Algorithm 1 but executed only on the points in these two clusters each time (followed by a number of secure comparisons to compute the final distance). Not surprisingly, the cost for this step is $O(\kappa s^2)$ since we have at most s points across all clusters (s is the sample size) and Algorithm 1 is quadratic.

Representative selection and cluster assignment. One final modification to the CURE algorithm that is necessary for both **Joint I-II** and **Joint I-II Clustering** has to do with the way final representatives are selected and clusters assigned (steps 9-10 of Algorithm 3). Recall that our privacy-preserving hierarchical clustering protocol outputs a single representative for each cluster, its centroid. If we want to run CURE for $R > 1$, we run into a privacy issue: clusters in general contain points from both parties and it is not clear whether they are willing to explicitly reveal them to each other. One way to avoid this is to fix $R = 1$ and set this value to be the representative that was output by our privacy preserving protocol for each cluster. Since this is the average value in the cluster this will often be sufficiently accurate, especially for spherical clusters. In fact, parameter $R > 1$ was originally introduced to improve accuracy for non-spherical clusters (see [35, Section 4.3]).

If the parties have reasons to assume clusters of this form, they can follow a different approach, by setting a leakage "threshold", i.e., by agreeing to reveal R points from each cluster to each other (the ownership of these points will be mixed). We believe that this may be a realistic alternative for controlled disclosure, especially considering that R is quite small in practice (e.g., 3).

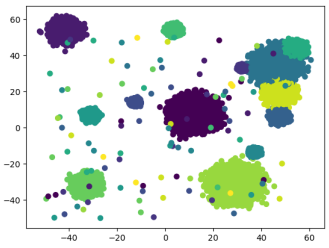


Figure 5: Synthetic data: 100,000 records and 0.1% outliers.

6 EXPERIMENTAL ANALYSIS

We implement the privacy preserving hierarchical clustering system based on our proposed protocol and show the experimental results in this section. The cryptographic protocol is implemented in C++, using the ABY framework [18] to realize the garbled circuits. We follow the standard parameter setting from ABY and set the symmetric security parameter to 128 (for AES encryption in garbled circuits), public-key security parameter to 1024 (for homomorphic encryption), and statistical security parameter κ to 40 (for blinding factors). We use the code from `libpaillier`[1] for our Paillier encryption. We run our experiments on two 24-core machines (each running one party), with Scientific Linux with 128GB memory and 2.9GHz Intel Xeon. The machines are on a university LAN with small RTT.

To evaluate our protocols we use a standard accuracy metric for clustering algorithms. We use datasets that include ground-truth labels (the correct class). After performing clustering, we assign the majority class label to all points of a cluster. Accuracy is defined as the fraction of points with correct labels relative to ground truth. In addition, we report running time and communication cost for our protocols. Our goal is to determine the tradeoffs between accuracy and performance in privacy-preserving hierarchical clustering.

6.1 Dataset description

Real-world datasets. To evaluate clustering accuracy, we need labeled, multi-class datasets. We picked 4 datasets from the UCI ML Repository[2], and restricted only to numeric attributes. (1) `Iris`: a dataset with various types of iris plants, with 150 instances and 4 attributes; (2) `Wine`: a dataset with results of chemical analysis of wines, with 178 records and 13 attributes; (3) `Heart`: a heart disease diagnosis dataset, with 303 records and 20 attributes; (4). `Breast`: a dataset that contains diagnostic Wisconsin Breast Cancer Database, with 569 records and 30 attributes.

Scalability experiments. We would like to test the scalability of our algorithms on much larger datasets (up to one million records). However, we need labeled multi-class datasets to evaluate the clustering accuracy, and we were not able to find labeled real-world datasets of this size. Fortunately, the performance of our protocol depends mainly on the dataset size and varies very little with the dataset dimension (as we will show experimentally). Moreover, the performance does not depend on the actual data values, as the type and sequence of cryptographic operations is data-independent. With these insights, we believe that it is sufficient to evaluate the scalability of our protocol using synthetic datasets, as the results will be very similar on real large-scale data.

We generated synthetic datasets of up to one million records and various dimensions $d \in [1, 20]$ following a Gaussian mixture distribution with the following parameters: (1) the number of clusters was chosen randomly between 8 and 15; (2) the cluster centers selected randomly in interval $[-50, 50]^d$, where we imposed a minimum separation between cluster centers (we stress that the choice of bounds for the values does not affect performance—computational costs are dominated by the κ additive parameter which is 40 bits); (3) standard deviation of the clusters randomly selected in $[0.5, 4]$. We generated different percentages of outliers to emulate different scenarios: low (0.1%), medium (1%), and high (5%). Outliers are generated uniformly at random in the same interval and assigned randomly to clusters. A visualization of one of the sample datasets generated by our method is given in Figure 5.

6.2 Performance of our basic protocol

We report the performance of our privacy-preserving protocol from Section 4, denoted by **Basic**. To evaluate the performance in our two-party setting, we randomly split the dataset and each party gets half the total number of records. We set the desired number of clusters $\ell_t = 5$. The cost of our protocols is linear in the number of iterations $(n - \ell_t)$, so the results we obtain for $\ell_t = 5$ are upper bounds on the running time of the protocol (as in general more than 5 clusters are desired).

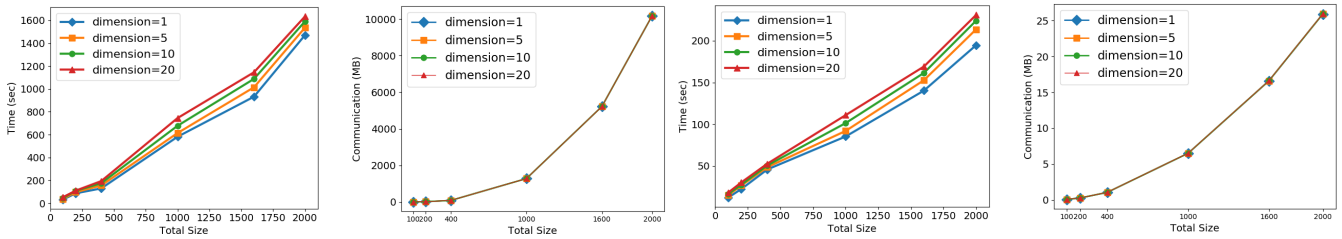
Figures 6a and 6b show the computational and communication cost for synthetic datasets of different sizes and dimensions. First of all, observe that dimension d has minimal impact on the former and no impact at all for the latter. This is due to the fact that most of the overhead is related to operating on distances between values/clusters, which is minimally affected by d (this is also consistent with our analysis in Section 4.1). In general, we observe that both costs increase steeply with the size n of the datasets. This is not surprising given the cubic asymptotic complexity of **Basic**. Still, for small real datasets such as the ones we experimented on (Figure 7a) the communication is $<$ 100MB and the computation time is under 4 minutes, which is reasonable in practice.

6.3 Performance of our optimized protocol

Next, we report the performance of our optimized protocol for the single linkage variant from Section 4.2, denoted by **Opt**. Figure 6c reports the computation time and Figure 6d the total communication size on synthetic data. The optimized version *significantly* improves both aspects, which is in line with our analysis from Section 4.2. For example, for a dataset with 2000 points with dimensionality 20, the running time is approximately 230 secs, which is around 8× faster than the **Basic** version. The performance improvement is even more pronounced for the case of communication size. For example, for the largest tested dataset, the communication cost decreases from 9GB to 26MB—almost a 400× improvement!

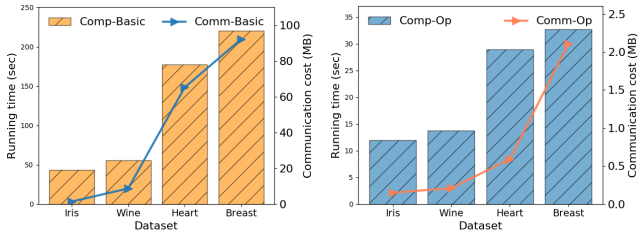
The difference in the magnitude of improvement between computation time and communication size is explained by the following observation. **Opt** does not improve the performance during the setup phase (indeed, it makes it slightly more costly), and setup time represents a large percentage of the total protocol time. In

[1] http://acsc.cs.utexas.edu/libpaillier
[2] http://archive.ics.uci.edu/ml/index.php



(a) Basic computational cost. **(b)** Basic communication size. **(c)** Opt computational cost. **(d)** Opt communication size.

Figure 6: Performance of our protocols Basic (6a & 6b) and Opt (6c & 6d) on synthetic datasets of size up to 2000 records.



(a) Performance of Basic. **(b)** Performance of Opt.

Figure 7: Performance of Basic and Opt on UCI data.

contrast, the amount of communication in setup is small relative to total protocol communication. We also evaluate the performance of **Opt** on several real datasets in Figure 7b. Again, we observe a significant improvement from **Basic**, e.g., for `Breast`, it takes under 35 seconds and less than 2.5 MB total communication.

6.4 Approximate clustering evaluation

We evaluate the three variants of privacy-preserving approximate clustering algorithms and compare their accuracy to the original CURE algorithm. Figure 8 shows the accuracy of these three variants and the CURE algorithm on synthetic datasets with one million records generated as explained above (for $p = 1$ partitions). Appendix A includes similar results for $p = 5$ partitions. We consider three settings for outliers: low (0.1%), medium (1%), and high (5%), and vary the sample size in CURE between 100 and 1000.

Our main observations are the following:

- The **Disjoint Clustering** algorithm in which CURE is run individually by each party and clusters are merged at the very end has poor accuracy. Indeed, this is not surprising, as each party generates clusters based on its own local samples and there is no interaction during the protocol. This variant will not incur the overhead of cryptographic operations, but the loss in accuracy (e.g., by 44.4% for one million records) is significant.
- For one partition ($p = 1$), **Joint I-II Clustering** and **Joint II Clustering** have similar accuracy to CURE for sufficiently large sample sizes. At 300 samples or higher, the accuracy of both variants is within 3% of the original CURE algorithm.
- We varied the partition size and observed that for higher values of p (e.g., $p = 5$) there is a difference between **Joint I-II Clustering** and **Joint II Clustering** for small sample sizes. For instance, at 200 sample size, the accuracy for **Joint II Clustering** is lower by 39.54% than that of **Joint I-II Clustering**. Still, when we increase the sample size above 500 records, the accuracies of the two variants are within 3.18%.
- We ran the approximate protocol with all combinations of $p = 1, 3, 5$ partitions and $R = 1, 3, 5, 7, 10$ representatives and observed that the accuracies of **Joint I-II Clustering** and **Joint II Clustering** are very close to CURE at $s = 1000$ samples. The largest difference we observed between **Joint II Clustering** and CURE is 3.57%, and between **Joint I-II Clustering** and CURE is 2.7%. For $p = 1$ and $R = 1$ the difference is less than 1% for 1000 samples. Therefore, our choice of $p = 1$ and $R = 1$ to protect data privacy, as argued in Section 5, does not impact the protocol's accuracy.

Based on these results, the **Joint II Clustering** variant is the preferred two-party approximate clustering method, for large enough sample sizes. It achieves accuracy close to the original CURE algorithm and lower overhead than other variants, while protecting data privacy for our parameter choice.

6.5 End-to-end approximation evaluation

We present the end-to-end approximation results to demonstrate that our protocol can be scaled to datasets up to one million records. We stress that, since the communication overhead is so small with our **Opt** protocol, network latency effect on end-to-end time is almost negligible; the vast majority of the time is spent on cryptographic operations. We select one million records, $d = 10$ dimensions with 1% outliers and choose the **Opt** version of our protocol. We set the partition to be $p = 1$ for the approximation and the number of clusters $\ell_t = 5$. We set $q = 3$ for **Joint II Clustering**.

Figure 9 shows the performance of the **Joint I-II Clustering** and **Joint II Clustering** for sample sizes s between 400 and 1000. The overall performance is practical, for example, with $s = 1000$ samples the cost of performing **Joint I-II Clustering** is 104 seconds, while the running time of **Joint II Clustering** is 35 seconds, which is faster by a factor of more than 3× compared to **Joint I-II Clustering** at a similar accuracy level (97.09%). The communication cost of **Joint I-II Clustering** is 6.5MB, while the cost of **Joint II Clustering** is 896KB, a factor of 7.25 smaller than that of **Joint I-II Clustering**.

7 RELATED WORK

The risks of information leakage in supervised learning has been demonstrated in practical attacks that either infer private information about the training data or the ML model and its hyperparameters [27, 38, 68]. To the best of our knowledge, no inference

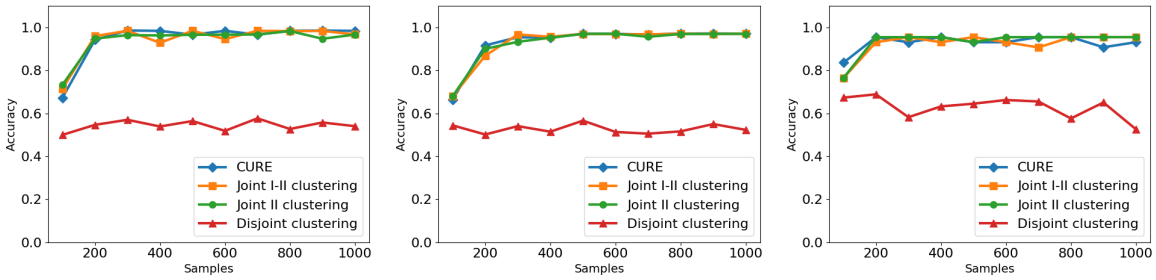
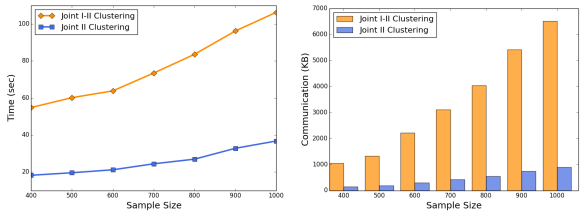


Figure 8: Approximate clustering accuracy: one million records, $p = 1$, outliers set at 0.1% (left), 1% (middle), and 5% (right).



(a) Computational cost. **(b) Communication cost.**

Figure 9: End-to-end performance evaluation on one million records.

attack against unsupervised learning models has been proposed. A large body of literature has been proposed to mitigate the attacks against supervised learning. The majority of these focus on classification models, including decision trees [50], SVM classification [75], linear regression [20, 21, 65], logistic regression [26], and neural networks [7, 58, 64]. Recently, a fast-growing number of works (e.g., [3, 10, 13, 14, 28, 30, 31, 37, 43, 47, 51, 53, 56, 61, 63]) achieve very strong security guarantees in this setting, by providing concrete security definitions and provably secure protocols that utilize multiple secure computation techniques [60]. Most of these works utilize highly efficient cryptographic back-ends to provide scalable solutions for supervised learning, often in the setting where an ML model is first trained over plaintext data and then is used for predictions at testing time in a privacy-preserving manner. Privacy-preserving federated learning is a recent design [9] in which multiple mobile users update a global model by sharing aggregated updates to model parameters.

The vast majority of works on privacy-preserving unsupervised learning study the problem of k-means clustering (e.g., see [11, 19, 24, 41, 42, 74]). Much less attention has been given to the problem of privacy-preserving hierarchical clustering, and existing works either lack formal security definitions and proofs [17, 39, 40], do not provide implementations [66], or both. One notable exception is [72] but that work provides a solution tailored only to the specific problem of document clustering.

An entirely different approach that applies to both types of learning attempts to preserve privacy through data perturbation (e.g., [1, 15, 16, 57, 67, 69]), i.e., by introducing statistical noise to hide the exact values, often by employing differential privacy [22]. These techniques are orthogonal to ours and can potentially be applied in unison towards a possibly more robust security treatment.

Finally, the interplay between cryptographic protocols and efficient approximation was first studied in [25]. Subsequent works have offered optimized protocols for various problems, e.g., pattern matching in genomic data [5, 78], k-means [71], and logistic regression [73, 80]. To the best of our knowledge, ours is the first work to compose secure cryptographic protocols with efficient approximation algorithms for hierarchical clustering.

8 CONCLUSION

We address the problem of privacy-preserving hierarchical clustering. We propose for the first time a formal security definition in the framework of secure computation. We design a secure clustering protocol that satisfies the definition for single and complete linkage, as well as an optimized version for the former. Finally, we combine our protocols with efficient approximate clustering in order to achieve the best of both worlds: strong security guarantees and scalability. Our experimental evaluation demonstrates that our protocol is efficient and scalable to one million records. We believe this work opens up new avenues of research in privacy-preserving unsupervised learning, e.g., design secure protocols for other linkage types (e.g., Ward) and other learning algorithms such as mixture models, association rules, and graph learning. Interesting avenues for future work include extending our protocols to the malicious threat model and supporting multiple participants.

REFERENCES

[1] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *ACM SIGSAC CCS 2016*. 308–318.

[2] Anonymous authors. [n. d.]. Privacy-Preserving Hierarchical Clustering: Formal Security and Efficient Approximation. Full version. Available at https://www.dropbox.com/s/cbi1vh9tf9szkbm/paper_full.pdf?dl=0.

[3] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. 2016. Scalable and Secure Logistic Regression via Homomorphic Encryption. In *ACM CODASPY 2016*. 142–144.

[4] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 805–817. https://doi.org/10.1145/2976749.2978331

[5] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. 2017. Privacy-Preserving Search of Similar Patients in Genomic Data. *IACR Cryptology ePrint Archive* 2017 (2017), 144. http://eprint.iacr.org/2017/144

[6] Foteini Baldimtsi, Dimitrios Papadopoulos, Stavros Papadopoulos, Alessandra Scafuro, and Nikos Triandopoulos. 2017. Server-Aided Secure Computation with Off-line Parties. In *ESORICS 2017*. 103–123.

[7] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. Privacy-Preserving ECG Classification With Branching Programs and Neural Networks. *IEEE Trans. Information Forensics and Security* 6, 2 (2011), 452–468. https://doi.org/10.1109/TIFS.2011.2108650

[8] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 2009. Scalable, Behavior-Based Malware Clustering.. In *Proceedings of the 16th Symposium on Network and Distributed System Security (NDSS)*.

[9] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning *(CCS '17)*. ACM, 1175–1191. https://doi.org/10.1145/3133956.3133982

[10] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data. In *NDSS 2015*.

[11] Paul Bunn and Rafail Ostrovsky. 2007. Secure two-party k-means clustering. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*. 486–497. https://doi.org/10.1145/1315245.1315306

[12] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. 2014. Uncovering Large Groups of Active Malicious Accounts in Online Social Networks. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*.

[13] Herv Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-Preserving Classification on Deep Neural Network. Cryptology ePrint Archive, Report 2017/035. https://eprint.iacr.org/2017/035.

[14] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. 2017. EzPC: Programmable, Efficient, and Scalable Secure Two-Party Computation. Cryptology ePrint Archive, Report 2017/1109. https://eprint.iacr.org/2017/1109.

[15] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, and Peter Rindal. 2017. Private Collaborative Neural Network Learning. *IACR Cryptology ePrint Archive* 2017 (2017), 762. http://eprint.iacr.org/2017/762

[16] Kamalika Chaudhuri and Claire Monteleoni. 2008. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems 21, 2008*. 289–296.

[17] Ipsa De and Animesh Tripathy. 2013. A Secure Two Party Hierarchical Clustering Approach for Vertically Partitioned Data Set with Accuracy Measure. In *ISI 2013*. 153–162.

[18] D. Demmler, T. Schneider, and M. Zohner. 2015. ABY - A framework for efficient mixed-protocol secure two-party computation. In *Proc. n 22nd Annual Network and Distributed System Security Symposium (NDSS)*.

[19] Mahir Can Doganay, Thomas Brochmann Pedersen, Yücel Saygin, Erkay Savas, and Albert Levi. 2008. Distributed privacy preserving k-means clustering with additive secret sharing. In *PAIS 2008*. 3–11.

[20] Wenliang Du and Mikhail J. Atallah. 2001. Privacy-Preserving Cooperative Scientific Computations. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*. 273–294.

[21] Wenliang Du, Yunghsiang S. Han, and Shigang Chen. 2004. Privacy-Preserving Multivariate Statistical Analysis: Linear Regression and Classification. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*. 222–233.

[22] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC 2006*. 265–284.

[23] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. 1998. Cluster analysis and display of genome-wide expression patterns. 95 (1998), 14863fi?!14868. Issue 25.

[24] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Reginald L. Lagendijk. 2013. Privacy-preserving distributed clustering. *EURASIP J. Information Security* 2013 (2013), 4. https://doi.org/10.1186/1687-417X-2013-4

[25] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. 2006. Secure multiparty computation of approximations. *ACM Trans. Algorithms* 2, 3 (2006), 435–472. https://doi.org/10.1145/1159892.1159900

[26] Stephen E. Fienberg, William J. Fulp, Aleksandra B. Slavkovic, and Tracey A. Wrobel. 2006. "Secure" Log-Linear and Logistic Regression Analysis of Distributed Databases. In *Privacy in Statistical Databases*. 277–290.

[27] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures *(CCS '15)*. ACM, New York, NY, USA, 1322–1333. https://doi.org/10.1145/2810103.2813677

[28] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2017. Privacy-Preserving Distributed Linear Regression on High-Dimensional Data. *PoPETs* 2017, 4 (2017), 345–364. https://doi.org/10.1515/popets-2017-0053

[29] Craig Gentry. 2009. *A Fully Homomorphic Encryption Scheme*. Ph.D. Dissertation. Stanford, CA, USA. Advisor(s) Boneh, Dan. AAI3382729.

[30] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proc. 33rd International Conference on Machine Learning (ICML)*.

[31] Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, Peter Rindal, and Mike Rosulek. 2016. Secure Data Exchange: A Marketplace in the Cloud. *IACR Cryptology ePrint Archive* 2016 (2016), 620. http://eprint.iacr.org/2016/620

[32] M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99, 12 (11 June 2002), 7821–7826. https://doi.org/10.1073/pnas.122653799

[33] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *ACM STOC 1987*. 218–229. https://doi.org/10.1145/28395.28420

[34] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol and Structure-independent Botnet Detection. In *Proceedings of the 17th USENIX Security Symposium*.

[35] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 2001. Cure: An Efficient Clustering Algorithm for Large Databases. *Inf. Syst.* 26, 1 (2001), 35–58. https://doi.org/10.1016/S0306-4379(01)00008-4

[36] W. Henecka, S. Kgl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. 1999. Tasty: Tool for automating secure two-party computations. In *Proc. ACM Conference on Computer and Communications Security (CCS)*. ACM.

[37] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep Neural Networks over Encrypted Data. *CoRR* abs/1711.05189 (2017). arXiv:1711.05189 http://arxiv.org/abs/1711.05189

[38] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. 2017. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In *ACM CCS 2017*. 603–618.

[39] Ali Inan, Selim Volkan Kaya, Yücel Saygin, Erkay Savas, Aycca Azgin Hintoglu, and Albert Levi. 2007. Privacy preserving clustering on horizontally partitioned data. *Data Knowl. Eng.* 63, 3 (2007), 646–666. https://doi.org/10.1016/j.datak.2007.03.015

[40] Geetha Jagannathan, Krishnan Pillaipakkamnatt, Rebecca N. Wright, and Daryl Umano. 2010. Communication-Efficient Privacy-Preserving Clustering. *Trans. Data Privacy* 3, 1 (2010), 1–25. http://www.tdp.cat/issues/abs.a028a09.php

[41] Geetha Jagannathan and Rebecca N. Wright. 2005. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *ACM SIGKDD 2005*. 593–599. https://doi.org/10.1145/1081870.1081942

[42] Somesh Jha, Luis Kruger, and Patrick McDaniel. 2005. Privacy Preserving Clustering. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS)*.

[43] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. Gazelle: A Low Latency Framework for Secure Neural Network Inference. *CoRR* abs/1801.05507 (2018). arXiv:1801.05507 https://arxiv.org/abs/1801.05507

[44] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: Making SPDZ Great Again. In *EUROCRYPT 2018*. 158–189. https://doi.org/10.1007/978-3-319-78372-7_6

[45] Florian Kerschbaum, Thomas Schneider, and Axel Schröpfer. 2014. Automatic Protocol Selection in Secure Two-Party Computations. In *ACNS 2014*. 566–584.

[46] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. 2009. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *CANS 2009*. 1–20.

[47] Yi Li, Yitao Duan, and Wei Xu. 2018. PrivPy: Enabling Scalable and General Privacy-Preserving Computation. *CoRR* abs/1801.10117 (2018). arXiv:1801.10117 http://arxiv.org/abs/1801.10117

[48] Minlei Liao, Yunfeng Li, Farid Kianifard, Engels Obi, and Stephen Arcona. 2016. Cluster analysis and its application to healthcare claims data: a study of end-stage renal disease patients who initiated hemodialysis. 17 (2016). Issue 25.

[49] Yehuda Lindell and Benny Pinkas. 2009. A Proof of Security of Yao's Protocol for Two-Party Computation. *J. Cryptology* 22, 2 (2009), 161–188. https://doi.org/10.1007/s00145-008-9036-8

[50] Y. Lindhell and B. Pinkas. 2000. Privacy Preserving Data Mining. In *Proc. Advances in Cryptology - CRYPTO*. Springer-Verlag.

[51] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *ACM SIGSAC CCS*. 619–631. https://doi.org/10.1145/3133956.3134056

[52] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press.

[53] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE Security and Privacy 2017*. 19–38. https://doi.org/10.1109/SP.2017.12

[54] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. 2013. ExecScent: Mining for New Domains in Live Networks with Adaptive Control Protocol Templates. In *Proceedings o the 22nd USENIX Security Symposium*.

[55] Sophia R. Newcomer, John F. Steiner, , and Elizabeth A. Bayliss. 2011. Identifying Subgroups of Complex Patients With Cluster Analysis. 17 (2011), 324fi?!332. Issue 8.

[56] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. 2013. Privacy-Preserving Ridge Regression on Hundreds of Millions of Records. In *Proc. IEEE Symposium on Security and Privacy (S & P)*. IEEE.

[57] Stanley R. M. Oliveira and Osmar R. Zaïane. 2003. Privacy Preserving Clustering by Data Transformation. In *XVIII Simpósio Brasileiro de Bancos de Dados, 6-8 de Outubro, Manaus, Amazonas, Brasil, Anais/Proceedings*. 304–318.

[58] Claudio Orlandi, Alessandro Piva, and Mauro Barni. 2007. Oblivious Neural Network Computing via Homomorphic Encryption. *EURASIP J. Information*

Security 2007 (2007). https://doi.org/10.1155/2007/37343
[59] P. Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. Advances in Cryptology - EUROCRYPT*. Springer-Verlag.
[60] Manoj Prabhakaran and Amit Sahai (Eds.). 2013. *Secure Multi-Party Computation*. Cryptology and Information Security Series, Vol. 10.
[61] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. *CoRR* abs/1801.03239 (2018). arXiv:1801.03239 http://arxiv.org/abs/1801.03239
[62] R L Rivest, L Adleman, and M L Dertouzos. 1978. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press* (1978), 169–179.
[63] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2017. DeepSecure: Scalable Provably-Secure Deep Learning. *CoRR* abs/1705.08963 (2017). arXiv:1705.08963 http://arxiv.org/abs/1705.08963
[64] Ahmad-Reza Sadeghi and Thomas Schneider. 2008. Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification. In *ICISC 2008*. 336–353. https://doi.org/10.1007/978-3-642-00730-9_21
[65] Ashish P. Sanil, Alan F. Karr, Xiaodong Lin, and Jerome P. Reiter. 2004. Privacy preserving regression modelling via distributed computation. In *ACM SIGKDD 2004*. 677–682.
[66] Mina Sheikhalishahi and Fabio Martinelli. 2017. Privacy preserving clustering over horizontal and vertical partitioned data. In *IEEE ISCC 2017*. 1237–1244. https://doi.org/10.1109/ISCC.2017.8024694
[67] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning *(CCS '15)*. ACM, New York, NY, USA, 1310–1321.
[68] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy*. 3–18.
[69] Shuang Song, Kamalika Chaudhuri, and Anand D. Sarwate. 2013. Stochastic gradient descent with differentially private updates. In *IEEE Global Conference on Signal and Information Processing 2013*. 245–248. https://doi.org/10.1109/GlobalSIP.2013.6736861
[70] Ion Stoica, Dawn Song, Raluca Ada Popa, David A. Patterson, Michael W. Mahoney, Randy H. Katz, Anthony D. Joseph, Michael I. Jordan, Joseph M. Hellerstein, Joseph E. Gonzalez, Ken Goldberg, Ali Ghodsi, David Culler, and Pieter Abbeel. 2017. A Berkeley View of Systems Challenges for AI. *CoRR* abs/1712.05855 (2017). arXiv:1712.05855 http://arxiv.org/abs/1712.05855
[71] Chunhua Su, Feng Bao, Jianying Zhou, Tsuyoshi Takagi, and Kouichi Sakurai. 2007. Privacy-Preserving Two-Party K-Means Clustering via Secure Approximation. In *AINA 2007*. 385–391.
[72] Chunhua Su, Jianying Zhou, Feng Bao, Tsuyoshi Takagi, and Kouichi Sakurai. 2014. Collaborative agglomerative document clustering with limited information disclosure. *Security and Communication Networks* 7, 6 (2014), 964–978. https://doi.org/10.1002/sec.811
[73] Toshiyuki Takada, Hiroyuki Hanada, Yoshiji Yamada, Jun Sakuma, and Ichiro Takeuchi. 2016. Secure Approximation Guarantee for Cryptographically Private Empirical Risk Minimization. In *ACML 2016*. 126–141. http://jmlr.org/proceedings/papers/v63/takada48.html
[74] Jaideep Vaidya and Chris Clifton. 2003. Privacy-preserving *k*-means clustering over vertically partitioned data. In *ACM SIGKDD 2003*. 206–215.
[75] Jaideep Vaidya, Hwanjo Yu, and Xiaoqian Jiang. 2008. Privacy-preserving SVM classification. *Knowl. Inf. Syst.* 14, 2 (2008), 161–178. https://doi.org/10.1007/s10115-007-0073-7
[76] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *ACM SIGSACCCS 2017*. 21–37. https://doi.org/10.1145/3133956.3134053
[77] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. In *ACM SIGSAC CCS 2017*. 39–56. https://doi.org/10.1145/3133956.3133979
[78] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. 2015. Efficient Genome-Wide, Privacy-Preserving Similar Patient Query Based on Private Edit Distance *(CCS '15)*. ACM, 492–503. https://doi.org/10.1145/2810103.2813725
[79] M.R. Weir, E.W. Maibach, G.L. Bakris, H.R. Black, P. Chawla, F.H. Messerli, J.M. Neutel, and M.A. Weber. 2000. Implications of a health lifestyle and medication analysis for improving hypertension control. 160 (2000), 481fi?!490. Issue 4.
[80] Wei Xie, Yang Wang, Steven M. Boker, and Donald E. Brown. 2016. PrivLogit: Efficient Privacy-preserving Logistic Regression by Tailoring Numerical Optimizers. *CoRR* abs/1611.01170 (2016). arXiv:1611.01170 http://arxiv.org/abs/1611.01170
[81] A. C. Yao. 1982. Protocols for secure computations. In *Proc. Symposium on Foundations of Computer Science (FOCS)*.
[82] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, 1982*. 160–164. https://doi.org/10.1109/SFCS.1982.38
[83] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science, 1986*. 162–167. https://doi.org/10.1109/SFCS.1986.25
[84] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *ACM SIGMOD 1996*. 103–114.
[85] Jan Henrik Ziegeldorf, Jens Hiller, Martin Henze, Hanno Wirtz, and Klaus Wehrle. 2015. Bandwidth-Optimized Secure Two-Party Computation of Minima. In *Cryptology and Network Security - CANS 2015*. 197–213.

A ADDITIONAL RESULTS

Figure 10 shows the accuracy of several variants of approximate clustering and the original CURE algorithm on synthetic datasets with one million records, for $p = 5$ partitions. We vary the number of outliers to low, medium, and high, as done in the experiments from Figure 8. We observe results that are similar to experiments described in Section 6.4. In particular, the accuracy loss of **Disjoint Clustering** variant is substantial, while the accuracy of the **Joint I-II Clustering** and **Joint II Clustering** variants is comparable to that of the original CURE algorithm for sample sizes larger than 500 points.

B GARBLED CIRCUITS

Here, we provide a more detailed description of garbled circuits and a running example in Figure 11. We defer readers to [49**?**] for a thorough formal treatment. Consider two parties, P_1 and P_2 that wish to evaluate a function f over their respective inputs x_1, x_2. One of the parties will play the role of the *garbler* and the other will play the role of the *evaluator*. Without loss of generality assume P_1 is the garbler and P_2 is the evaluator. Their interaction proceeds as follows.

First P_1 expresses f as a Boolean circuit, i.e., as a directed acyclic graph of Boolean AND and OR gates, and sends a "garbled" version of the circuit to P_2. We provide a concrete example in Figure 11 that depicts a Boolean circuit of two AND gates A, B and an OR gate C. The input x_1 is 11 whereas x_2 is 01 (both expressed in bits). The normal circuit should first compute the bitwise AND of the two inputs and forwards the results to the OR gate. In order to garble the circuit, P_1 will pick two random values from a large domain (e.g., 128-bits each) for the two possible bits of each wire of the circuit. We call these the *garbled values* for that wire. More concretely, if the first (second) bit of P_1's and P_2's inputs is to be inserted into gate A (resp. B), P_1 selects $w^0_{11}, w^1_{11}, w^0_{21}, w^1_{21}$ (resp. $w^0_{12}, w^1_{12}, w^0_{22}, w^1_{22}$). Mnemonically, the subscript of a w value for input wires corresponds to the party that provides it and the index of the bit on its input, whereas the superscript indicates the wire's plaintext bit (e.g., w^0_{21} is the value for P_2's first bit in case that is a zero). Proceeding further into the circuit, P_1 picks $w^0_A, w^1_A, w^0_B, w^1_B, w^0_C, w^1_C$ for the possible bit outputs of gates A, B, and C respectively. Without knowing how these random values where chosen, it is impossible to infer which corresponds to which bit. Note that w^0_C, w^1_C correspond to the final circuit output, i.e., to the value $f(x_1, x_2)$.

Next, P_1 creates a garbled truth table for every gate, which can be viewed as an encrypted version of the truth table of the Boolean gate. We explain this only for gate A and the other gates follow in a similar manner. The row $(1, 1) \rightarrow 1$ of the truth table of the AND gate A, should output 1 on input 1, 1. These inputs correspond to values w^1_{11}, w^1_{21}, respectively, whereas the output corresponds to w^1_A. Using w^1_{11}, w^1_{21} as encryption keys in a symmetric encryption

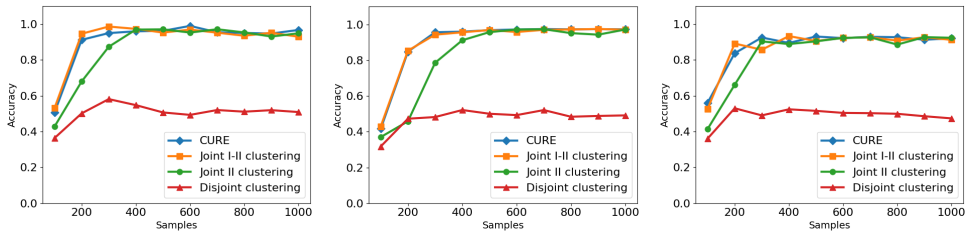


Figure 10: Approximate clustering accuracy for 1 million records with $p = 5$ partitions, and outliers set at 0.1% (left), 1% (middle), and 5% (right).

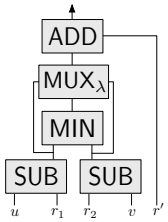


Figure 12: The circuit for MinSelect.

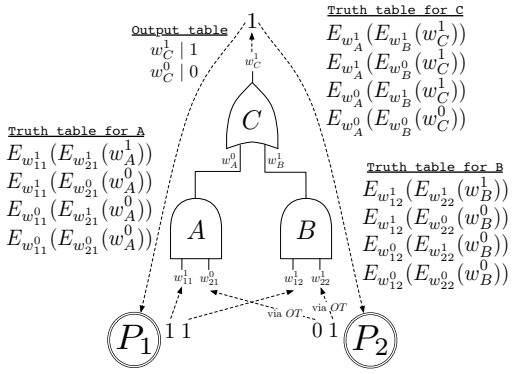


Figure 11: Garbled circuit for a function that takes two bits from each party, computes their pairwise AND and an OR over the result.

scheme (e.g., 128-bit AES) P_1 *double-encrypts* (i.e., encrypts twice in a layer manner) this possible output for A as $E_{w_{11}^1}(E_{w_{21}^1}(w_A^1))$. P_1 produces a similar encryption for every row of the truth table of gate A (and all other gates of the circuit) and sends them to P_2, *permuted* to hide the order of the rows. Observe that one can retrieve w_A^1 if and only if they possess *both* w_{11}^1, w_{21}^1. Conversely, if one possesses *only* w_{11}^1, w_{21}^1, all other entries of the garbled truth table (besides w_A^1) are indistinguishable from random, due to the semantic security of the encryption scheme. In order for P_2 to be able to retrieve the final output, P_1 also sends the output wire values w_C^0, w_C^1 *together* with their corresponding mapping to 0 and 1.

Observe that, given the garbled truth tables values, if P_2 knows the w value of each input wire of a gate, she can easily discover its output value. For example, if she has w_{11}^1, w_{21}^1, she can try to decrypt every value in the truth table until she finds the correct value w_A^1.

Note that, we need to assume that the encryption scheme allows detection of well-formed decryptions, i.e., it is possible to deduce whether the retrieved plaintext has a correct format. This can be easily achieved using a blockcipher and padding with a sufficient number of 0's, in which case well-formed decryptions will have a long suffix of 0's and decryptions under the wrong key will have a suffix of random bits. This property is referred to as *verifiable range* in [49]. P_2 sends the w values corresponding to his inputs (w_{11}^1, w_{12}^0) in the clear. Since these are random values, P_2 cannot map them to 0 or 1, thus P_1's input is protected. The last technical challenge is for P_2 to retrieve the w values corresponding to *his own* input (i.e., w_{21}^1, w_{22}^1), without telling P_1 which values he needs. (Recall that if P_1 just sends all of $w_{21}^0, w_{21}^1, w_{22}^1, w_{22}^0$, this allows more than one entries of the garbled truth table to be decrypted). This is achieved through a two-party secure computation protocol called (1-out-of-2) *oblivious transfer* (OT) [**?**]. At a very high level, and focusing on the first bit of S's input, P_2 can retrieve via OT from P_1 exactly one value from pair (w_{21}^0, w_{21}^1), without P_1 learning which of the two. To transfer all the necessary w values, the parties must execute (in parallel) an OT protocol for every bit of P_2's input. After retrieving these w values, P_2 evaluates the circuit on her own as described above, and sends the output bit (1 in our example) to P_1.

C SUB-PROTOCOLS FOR SECURE COMPARISON

Here we show circuits for ArgminSelect, MinSelect, and MaxSelect as defined in Section 4. These circuits are garbled and evaluated as described in Section 2 and called as sub-routines during our protocol from Section 4, We assume values of λ bits, and blinding randomness of κ bits, which makes the sum $\kappa + 1$ bits long. The final output of the circuit is denoted by an arrow in the figures.

The circuits use as building blocks gates for addition/subtraction ADD/SUB of $\kappa + 1$-bit integers and minimum/maximum comparison MIN/MAX of two λ-bit integers. By convention, the latter output a single bit that indicates which is the minimum/maximum value (e.g., MIN on input 3, 5 outputs 0 to indicate the first value is smaller). We also need a multiplexer gate MUX_i that upon input two inputs consisting of i bits each, and a selector bit s chooses the first or the second one, depending on the value of s. The circuit for ArgminSelect operates on n different inputs and returns the index of the minimum one. To facilitate the final return of the index (without having to explicitly provide it as input), we hard-code in the circuit "constant" gates CON_i that always output a fixed value

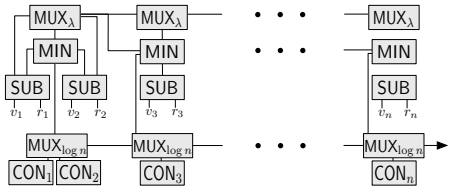


Figure 13: The circuit for ArgminSelect**.**

$1 \leq i \leq n$, e.g., CON_3 always outputs the binary representation of 3 using $\log n$ bits. Alternatively, we could provide the indexes as input and "carry" them throughout the circuit. Efficient implementations for the above types of gates can be found in existing literature (e.g., [46]).

Figure 12 shows the circuit for MinSelect. It takes as input five values u, v, r_1, r_2, r' where the first two are of $\kappa + 1$ bits and the last three are of κ bits. It first computes $u - r_1$ and $v - r_2$ using two SUB gates. Then it computes the minimum of the two values using a MIN gate the output of which is forwarded to a MUX_λ gate that takes as input the values $u - r_1$ and $v - r_2$. Finally, it blinds the output of the multiplexer again by adding r' with a ADD gate, before outputting it. The circuit for MaxSelect is the same but uses a MAX gate instead.

Figure 13 shows the circuit for ArgminSelect. It takes as input $2n$ values $v_1, \ldots, v_n$ and $r_1, \ldots, r_n$. The first n values are of $\kappa + 1$ bits whereas the rest are of κ bits. First, it uses n SUB gates to compute values $v_i - r_i$ for $i = 1, \ldots, n$. Then it uses $n-1$ MIN gates to compare the minimum as follows. The first gate compares v_1, v_2. It outputs a bit that is fed to a MUX gate as the selector. The input values for this multiplexer are provided by constant gates CON_1,CON_2. The output of MIN is also fed to another MUX gate that takes as input $v_1 - r_1$ and $v_2 - r_2$. The outputs of the two multiplexer gates correspond to the current minimum value and index. They are then forwarded to a new MIN gate for comparison with $v_3 - r_3$ and the process continues iteratively. For the i-th comparison, the index MUX gate will take the current minimum index and the output of CON_{i+1}. The final output after $n - 1$ comparisons is given by the last index MUX gate.

D PROOF OF THEOREM 4.1

We begin by recalling that, under the assumption that the oblivious transfer protocol used is secure, there exists simulator Sim_{OT} that can simulate the views of each of the parties $\mathsf{P}_1, \mathsf{P}_2$ during a single oblivious transfer execution when given as input the corresponding party's input (and output, in case it is non-empty) and randomness.

The core idea behind our proof is that, since all values seen by the two parties during the protocol execution (apart from the indexes of the merged clusters at each round) are "blinded" by large random factors. For example, assuming all values p_i, q_i are 32-bits and the chosen random values are 100-bits, it follows that the sum of the two is statistically indistinguishable from a 100-bit value chosen uniformly at random. In particular, this allows the simulator to effectively run the protocol with the adversary by simply choosing simulated values for the other party which he chooses himsellf at random (in the above example these would be random 32-bit values). We will handle the two case of corruption separately.

Corruption of P_2**.** The view of P_2 during the protocol execution consists of:

(1) Encrypted tables $\mathcal{M}, \mathcal{R}$ and encrypted lists L, S.
(2) For each round of clustering ℓ, messages received during the oblivious transfer execution for ArgminSelect, denoted by OT_ℓ and the min/max index α_ℓ.
(3) During each round of clustering ℓ, for each execution of MinSelect/MaxSelect for index k, messages received during the corresponding oblivious transfer execution, denoted by $OT_{\ell,k}$, corresponding garbled circuit $GC_{\ell,k}$, and output value $v_{\ell,k}$.
(4) Encrypted cluster representative values $E_1, \ldots, E_{\ell_t}$.

The simulator Sim_{P_2}, on input the random tape R_2, points $q_1, \ldots, q_{n_2}$, outputs $(rep_1/|J_1|, |J_1|, \ldots, rep_{\ell_t}/|J_{\ell_t}|, |J_{\ell_t}|)$, $\alpha_1, \ldots, \alpha_{\ell_t}$, computes the view of P_2 as follows.

- **(Ciphertext computation)** Using random tape R_2, the simulator runs the key generation algorithm for P_2 to receive sk', pk'. He then chooses values $p'_1, \ldots, p'_{n_1}$ uniformly at random from $\{0,1\}^d$. These will act as the "simulated" values for player P_1. He then runs π_{HC} honestly using the values p'_i as input for P_1 (and the actual values q_i of P_2), with the following modifications.
- **(Oblivious transfer simulation for** OT_ℓ**)** For $\ell = 1, \ldots, \ell_t$ let W_ℓ be the set of garbled input values computed by P_2 for the garbled circuit that evaluates MinSelect/MaxSelect at round ℓ. Since we are in the semi-honest setting, the corrupted P_2 computes these values uniformly at random. Therefore, the simulator can also compute them using R_2. Then, for $i = 1, \ldots, \ell$, the simulator includes in the view (instead of OT_ℓ) the output OT'_ℓ produced by simulator $\mathsf{Sim}^{(2)}_{OT}$ on input W_ℓ.[3] Note that P_2 does not receive any output from this oblivious transfer execution, thus $\mathsf{Sim}^{(2)}_{OT}$ only works given the input.
- **(Oblivious transfer simulation for** ArgminSelect**)** For each round ℓ, the simulator includes in the view, the index α_ℓ.
- **(Garbled circuit simulation for** $GC_{\ell,k}$**)** Next, the simulator needs to compute the garbled circuits $GC_{\ell,k}$. The simulator uses the corresponding values from $\mathcal{R}$ (as computed so far) and a "new" blinding factor $\rho\ell, k$ for P_1' inputs and computes a garbled circuit for evaluating ArgminSelect honestly. The simulator also includes in the view of P_2 the garbled inputs for the corresponding elements from $\mathcal{R}$.
- **(Oblivious transfer simulation for** $OT_{\ell,k}$**)** Let $y_{\ell,k}$ be the input of P_2 for the circuit $GC_{\ell,k}$ (i.e., the execution of ArgminSelect for index k during round ℓ). Since we are in the semi-honest case, the corrupted P_2 will provide as input the values that have been established from the interaction with P_1 (using the points p'_i) up to that point, therefore $y_{\ell,k}$ can be computed by the simulator. In order to compute the parts of the view that correspond to each of $OT_{\ell,k}$ the simulator includes in the view the output of Sim_{OT} on input $y_{\ell,k}$ and the corresponding choice from each pair of garbled inputs he chose in the previous step

[3] And corresponding randomness derived from R_2.

(as dictated by the bit representation of $y_{\ell,k}$), which we denote as $OT'_{\ell,k}$.

- **(Encrypted representatives computation)** For $\ell = 1, \ldots, \ell_t$, the simulator computes $rep_\ell = \lceil rep_\ell / |J_\ell| \cdot |J_i| \rceil$ and $E_\ell = [rep_\ell]$, where encryption is under (the previously computed) pk.

We now argue that the view produced by our simulator is indistinguishable from the view of P_2 when interacting with P_1 running π_{HC}. This is done via the following sequence of hybrids.

Hybrid 0. This is the view $\text{view}^{\pi_{HC}}_{\mathcal{A}_{P_2}}$, i.e., the view of P_2 when interacting with P_1 running π_{HC} for points p_i.

Hybrid 1. This is the same as Hybrid 0, but the output of GC_ℓ in $\text{view}^{\pi_{HC}}_{\mathcal{A}_{P_2}}$ is replaced by α_ℓ. This is indistinguishable from Hybrid 0 due to the correctness of the garbling scheme. Since we are in the semi-honest setting, both parties follow the protocol, therefore the outputs they evaluate are always α_ℓ.

Hybrid 2. This is the same as Hybrid 1, but values in $\mathcal{M}, L$ are computed using values p'_i. This is statistically indistinguishable from Hybrid 1 (i.e., even unbounded algorithms can only distinguish between the two with probability $O(2^\kappa)$ since in $\text{view}^{\pi_{HC}}_{\mathcal{A}_{P_2}}$ each of the values in $\mathcal{M}, L$ are computed as the sum of a random value from $\{0,1\}^\kappa$ and a distance between two clusters.

Hybrid 3. This is the same as Hybrid 2, but all values in $\mathcal{R}, S$ are replaced with encryptions of zero's. This is indistinguishable from Hybrid 2 due to the semantic security of Paillier's encryption scheme.

Hybrid 4. This is the same as Hybrid 3, but each of OT_ℓ is replaced by OT'_ℓ, computed as described above. This is indistinguishable from Hybrid 3 due to the security of the oblivious transfer protocol.

Hybrid 5. This is the same as Hybrid 4, but the garbled inputs given to P_2 for $GC_{\ell,k}$ are chosen based on the values that have been computed using values p'_i. Since garbled inputs are chosen uniformly at random (irrespectively of the actual input values), this follows the same distribution as Hybrid 3.

Hybrid 6. This is the same as Hybrid 5, but each of $OT_{\ell,k}$ is replaced by output of $OT_{\ell,k}$ computed as described above. This is indistinguishable from Hybrid 5 due to the security of the oblivious transfer protocol.

Hybrid 7. This is the same as Hybrid 6, but each value E_i sens to P_2 is computed as $[\lceil rep_i / |J_i| \cdot |J_i| \rceil]$ using public key pk'. This is indistinguishable from Hybrid 6 since we are in the semi-honest setting and both parties follow the protocol therefore the outputs they evaluate are always $rep_i / |J_i|$.

Note that Hybrid 7 corresponds to the view produced by our simulator and Hybrid 0 to the view that P_2 receives while interacting with P_1 during π_{HC} which concludes this part of the proof.

Corruption of P_1. The case where P_1 is corrupted is somewhat simpler as he does not receive any outputs from the circuits $GC_{\ell,k}$. The view of P_1 during the protocol execution consists of:

(1) Encrypted tables $\mathcal{B}, \mathcal{R}$ and encrypted lists Q, Q', L, S.
(2) For each round of clustering ℓ, a garbled circuit GC_ℓ for evaluating ArgminSelect, messages received during the corresponding oblivious transfer execution denoted by OT_ℓ.
(3) During each round of clustering ℓ, for each execution of MinSelect/MaxSelect for index k, messages received during the corresponding oblivious transfer execution denoted by $OT_{\ell,k}$.

The simulator Sim_{P_1}, on input the random tape R_1, points $p_1, \ldots, p_{n_1}$, outputs $(rep_1/|J_1|, |J_1|, \ldots, rep_{\ell_t}/|J_{\ell_t}|, |J_{\ell_t}|)$, $\alpha_1, \ldots, \alpha_{\ell_t}$, computes the view of P_1 as follows.

- **(Ciphertext computation)** Using random tape R_1, the simulator runs the key generation algorithm for P_1 to receive sk, pk and computes a pair sk', pk' for himself. He computes $\mathcal{B}, Q, Q', L$ consisting of encryptions of zeros under pk'. Moreover, he computes $\mathcal{R}$,S consisting of encryption of values chosen uniformly at random from $\{0,1\}^\kappa$ and encrypted under pk.
- **(Garbled circuit simulation for GC_ℓ)** Next the simulator needs to provide garbled circuits for the evaluation of ArgminSelect for each round of clustering ℓ. For this, the simulator creates a "rigged" garbled circuit GC'_ℓ that always outputs α_ℓ, irrespectively of the inputs. This is achieved by forcing all intermediate gates to always return the same garbled output and by setting the output translation temple to always to decode to the bit-representation of α_ℓ (this process is explained formally in [49]).
- **(Oblivious transfer simulation for** ArgminSelect**)** Let $W^{(1)}_\ell$, $W^{(2)}_\ell$ be the sets of pairs of input garbled values that the simulator choses while creating GC'_ℓ as described above (where the former corresponds to the input of P_1 and the latter to the input of P_2). The simulator includes in the view a random choice from each pair in $W^{(2)}$. Moreover, he replaces the messages in the view that correspond to the execution of $OT_{\ell,k}$, by the output of $\mathsf{Sim}^{(1)}_{OT}$ on input $(y_\ell, W^{(1)}_\ell)$, where y_ℓ is the bit description of the input of P_1 for GC_ℓ (which can be computed with the simulator since he has access to p_i, R_1).
- **(Oblivious transfer simulation for** MinSelect/MaxSelect**)** For each $GC_{\ell,k}$ let $W_{\ell,k}$ be the set of garbled input values computed by P_1 for the garbled circuit that evaluates MinSelect/MaxSelect at round ℓ and cluster k. Since we are in the semi-honest setting, the corrupted P_1 computes these values uniformly at random. Therefore, the simulator can also compute them using random tape R_1. Then, for each ℓ, k the simulator includes in the view (instead of $OT_{\ell,k}$) the output $OT'_{\ell,k}$ produced by simulator $\mathsf{Sim}^{(1)}_{OT}$ on input $W_{\ell,k}$ (and corresponding randomness derived from R_1). Note that P_1 does not receive any output from this oblivious transfer execution, thus $\mathsf{Sim}^{(1)}_{OT}$ only works given the input.

We now argue that the view produced by our simulator is indistinguishable from the view of P_1 when interacting with P_2 running π_{HC}. This is done via the following sequence of hybrids.

Hybrid 0. This is the view $\text{view}^{\pi_{HC}}_{\mathcal{A}_{P_1}}$, i.e., the view of P_1 when interacting with P_2 running π_{HC} for points q_i.

Hybrid 1. This is the same as Hybrid 0, but all values in $\mathcal{B}, Q, Q', L$ are replaced with encryptions of zero's. This is indistinguishable

from Hybrid 1 due to the semantic security of Paillier's encryption scheme.

Hybrid 2. This is the same as Hybrid 1, but values in $\mathcal{R}, S$ are computed as encryptions of values chosen uniformly at random from $\{0,1\}^\kappa$ under key pk. This is statistically indistinguishable from Hybrid 1 for the same reasons as for the case of P_2 above.

Hybrid 3. This is the same as Hybrid 2, but each of GC_ℓ is replaced by GC'_ℓ, computed as described above (including the values from $W_{(2)}$) This is indistinguishable from Hybrid 2 due to the security of encryption scheme used for the garbling scheme (this is formally described in [49]).

Hybrid 4. This is the same as Hybrid 3, but each of OT_ℓ is replaced by OT'_ℓ, computed as described above. This is indistinguishable from Hybrid 3 due to the security of the oblivious transfer protocol.

Hybrid 5. This is the same as Hybrid 4, but each of $OT_{\ell,k}$ is replaced by $OT'_{\ell,k}$ computed as described above. This is again indistinguishable from Hybrid 5 due to the security of the oblivious transfer protocol.

Note that Hybrid 5 corresponds to the view produced by our simulator and Hybrid 0 to the view that P_2 receives while interacting with P_1 during π_{HC} which concludes this part of the proof.