Private Hierarchical Clustering and Efficient Approximation

Xianrui Meng
xianru@amazon.com
Amazon Web Services

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

Alina Oprea
a.oprea@northeastern.edu
Northeastern University

Nikos Triandopoulos
ntriando@stevens.eduu
Stevens Institute of Technology

ABSTRACT

In *collaborative learning*, multiple parties contribute their datasets to jointly deduce global machine learning models for numerous predictive tasks. Despite its efficacy, this learning paradigm fails to encompass critical application domains that involve highly sensitive data, such as healthcare and security analytics, where privacy risks limit entities to individually train models using only their own datasets. In this work, we target *privacy-preserving collaborative hierarchical clustering*. We introduce a formal security definition that aims to achieve balance between utility and privacy and present a two-party protocol that provably satisfies it. We then extend our protocol with: (i) an optimized version for single-linkage clustering, and (ii) scalable approximation variants. We implement all our schemes and experimentally evaluate their performance and accuracy on synthetic and real datasets, obtaining very encouraging results. For example, end-to-end execution of our secure approximate protocol for over 1M 10-dimensional data samples requires 35sec of computation and achieves 97.09% accuracy.

CCS CONCEPTS

• **Security and privacy → Cryptography**; • **Computing methodologies → Unsupervised learning**.

KEYWORDS

secure computation; private hierarchical clustering; secure approximation

ACM Reference Format:
Xianrui Meng, Dimitrios Papadopoulos, Alina Oprea, and Nikos Triandopoulos. 2021. Private Hierarchical Clustering and Efficient Approximation. In *Proceedings of the 2021 Cloud Computing Security Workshop (CCSW'21), November 15, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3474123.3486760

1 INTRODUCTION

Big-data analytics is an ubiquitous practice with a noticeable impact on our lives. Our digital interactions produce massive amounts of data that are analyzed in order to discover unknown patterns or correlations, which help us draw safer conclusions or make informed decisions. At the core of this lies Machine Learning (ML) for devising complex data models and predictive algorithms that provide hidden insights or automated actions, while optimizing certain objectives. Example applications that successfully employ ML are market forecast, service personalization, speech/face recognition, autonomous driving, health diagnostics and security analytics.

Of course, data analysis is 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 its learning accuracy, as irregularities due to outliers fade away faster. Indeed, scalability to large dataset sizes is very important, especially so in unsupervised learning, where model inference uses unlabelled observations (evading points of saturation, encountered in supervised learning, where new training sets improve accuracy only marginally). Also, the more varied the collected data, the more elaborate its analysis, as degradation due to noise reduces and domain coverage increases. Indeed, for a given learning objective, say classification or anomaly detection, combining more datasets of similar type but different origin enables discovery of more complex, interesting, hidden structures and of richer association rules (correlation or causality) among attributes. So, ML models improve their predictive power when they are built over multiple datasets owned and contributed by different entities, in what is termed *collaborative learning*—and widely considered as the golden standard [100].

Privacy-preserving hierarchical clustering. Several learning tasks of interest, across a variety of application domains, such as healthcare or security analytics, demand deriving accurate ML models over highly sensitive data—e.g., personal, proprietary, customer, or other types of data that induce liability risks. By default, since collaborative learning inherently implies some form of data sharing, entities 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, *privacy risks are the main impediment to collaboratively learning* richer models over large volumes of varied, individually contributed, data.

The security and ML community has embraced the 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 closely related to (privacy-preserving) ML-as-a-Service [42, 52, 53, 104] that utilizes cloud providers for ML tasks, without parties revealing their sensitive raw data (e.g., using encrypted or sanitized data. Existing work on PCL mostly focuses on supervised rather than unsupervised

CCSW '21, November 15, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8653-1/21/11…$15.00
https://doi.org/10.1145/3474123.3486760

learning tasks (with a few exceptions such as k-means clustering). As unsupervised learning is a prevalent paradigm, the design of ML protocols that promote collaboration and privacy is vital.

In this paper, we study the problem of *privacy-preserving hierarchical clustering*. This unsupervised learning method groups data points into similarity clusters, using some well-defined distance metric. The "hierarchic" part is because each data point starts as a separate "singleton" cluster and clusters are iteratively merged building increasingly larger clusters. This process forms a natural hierarchy of clusters that is part of the output, showing how the final clustering was produced. We present scalable cryptographic protocols that allow two parties to privately learn a model for the joint clusters of their combined datasets. Importantly, we propose a formal security definition for this task in the MPC framework and prove our protocols satisfy it. In contrast, prior works for privacy-preserving hierarchical clustering have proposed crypto-assisted protocols but without offering rigorous security definitions or analysis (e.g., [27, 55, 57]; see detailed discussion in Section 8).

Motivating applications. Hierarchical clustering is a class of unsupervised learning methods that build a *hierarchy of clusters* over an input dataset, typically in bottom-up fashion. Clusters are initialized to each contain a single input point and are iteratively merged in pairs, according to a *linkage* metric that measures clusters' closeness based on their contained points. Here, unlike other clustering methods (k-means or spectral clustering), different distance metrics can define cluster linkage (e.g., nearest neighbor and diameter for *single* and *complete* linkage, respectively) and flexible conditions on these metrics can determine when merging ends. The final output is a *dendrogram* with all formed clusters and their merging history. This richer clustering type is widely used in practice, often in 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 and discover useful correlations to improve health practices—e.g., discover similar genes types [34], patient profiles most in need of targeted intervention [80, 110] or changes in healthcare costs for specific treatments [68]. To be of any predictive value, such 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 personnel to process log data on network/users activity to discover suspicious or malicious events—e.g., detect botnets [46], malicious traffic [79], compromised accounts [19], or malware [13]. Again, such data contains sensitive information (e.g., employee/customer data, enterprise security posture, defense practices, etc.) that must be protected, also due to industrial regulations or for reduced liability. As such, without privacy provisions for joint cluster analysis, entities are restricted to learn only local clusters, thus confined in accuracy and effectiveness. E.g., a clinical-trial analysis over patients of one hospital may introduce bias on geographic population, or network inspection of one enterprise may miss crucial insight from attacks against others.

In contrast, our treatment of clustering as a PCL instance is a solid step towards richer classification. Our protocols for private hierarchical clustering incentivize entities to contribute their private datasets for joint cluster analysis over larger and more varied data collections, to get in return more refined results. For instance, hospitals can jointly cluster medical data extracted from their combined patient records, to provide better treatment, and enterprises can jointly cluster threat indicators collected from their combined SIEM tools, to present timely and stronger defenses against attacks.[1] At all times, data owners protect the confidentiality of their private data and remain compliant with current regulations.

Challenges and insights. A first challenge we faced is how to rigorously specify the secure functionality that such protocols must achieve. A secure protocol guarantees that no party learns anything about the input of the other party, *except what can be inferred after parties learn the output*. But since the output dendrogram of hierarchical clustering already includes the (now partitioned) input, this problem cannot directly benefit from MPC. This issue is partially the reason why previous approaches for hierarchical clustering (see discussion in Section 8 and an excellent survey of related work by Hegde et al. [49]) lack formal security analysis or have significant information leakage. To overcome this, our approach is to modify and refine what private hierarchical clustering should produce, redacting the joint output—sufficiently enough, to allow the needed input privacy protection, but minimally so, to preserve the learning utility. We introduce a security notion that is based on *point-agnostic dendrograms*, which explicitly capture only the merging history of formed joint clusters and useful statistics thereof, to balance the intended accuracy against the achieved privacy. To the best of our knowledge, our formal security definition (Section 3) is the first such attempt for the case of hierarchical clustering.

The next challenge is to securely realize this functionality efficiently. Standard tools for secure two-party computation, e.g., garbled circuits [113, 114], result in large communication, while fully homomorphic encryption [41] is still rather impractical, so designing scalable hierarchical clustering PCL protocols is challenging. Moreover, hierarchical clustering of n points is already computation-heavy—of $O(n^3)$ cost. As such, approximation algorithms, e.g., CURE [47], are the *de facto* means to scale to massive datasets, but incorporating approximation to private computation is not trivial—as complications often arise in defining security [37].

In Section 4, we follow a modular design approach and use cryptography judiciously by devising our main construction as a *mixed protocol* (e.g., [28, 50, 63]). We decompose our refined hierarchical clustering into building blocks and then we select a combination of tools that achieves fast computation and low bandwidth usage. In particular, we conveniently use garbled circuits for cluster merging, but additive homomorphic encryption [86] for cluster encoding, while securely "connecting" the two steps' outputs.

In Section 5, we evaluate the performance and security of our main protocol and present an optimized variant of $O(n^2)$ cost for single linkage. In Section 6, we integrate the CURE method [47] for *approximate clustering* into our design, to get the best-of-two-worlds quality of high scalability and privacy. We study different secure approximate variants that exhibit trade-offs between efficiency and accuracy without extra leakage due to approximation. In Section 7, we report results from the experimental evaluation of

[1] In line with current trends toward collaborative learning in healthcare/security analytics; e.g., AI-based clinical-trial predictions [1], threat-intelligence sharing [7, 8, 29, 36].

our protocols on synthetic and real data that confirm their practicality. For example, end-to-end execution of our private approximate single-linkage protocol for 1M $10-d$ records, achieves 97.09% accuracy at very with only 35sec of computation time.

Summary of contributions. Overall, in this work our results can be summarized as follows:

- We provide a formal definition and secure two-party protocols for private hierarchical clustering for single or complete linkage.
- We present an optimized protocol for single linkage that significantly improves the computational and communication costs.
- We combine approximate clustering methods with our protocols to get variants that achieve both scalability and strong privacy.
- We experimentally evaluate the performance of our protocols via a prototype implementation over synthetic and real datasets.

2 PRELIMINARIES

Hierarchical clustering (HC). For fixed positive integers d, l, let $\mathcal{D} = \{v_i | v_i \in \mathbb{Z}_{2^l}^d\}_{i=1}^n$ be an *unlabeled* indexed dataset of n d-dimensional points, where w.l.o.g, we set the domain to $\{0, \ldots, 2^l - 1\}$. Over pairs $x, y \in \mathcal{D}$ of points, *point distance* is measured using the standard *square Euclidean distance* metric $\mathsf{dist}(x, y) = \sum_{j=1}^d (x_j - y_j)^2$. Over pairs $X, Y \subseteq \mathcal{D}$ of sets of points, *set closeness* is measured using a *linkage distance* metric $\delta(X, Y)$, as a function of the cross-set distances of points contained in X, Y. The most commonly used linkage distances are the *single linkage* (or nearest neighbor) defined as $\delta(X, Y) = \min_{x \in X, y \in Y} \mathsf{dist}(x, y)$, and the *complete linkage* (or diameter) defined as $\delta(X, Y) = \max_{x \in X, y \in Y} \mathsf{dist}(x, y)$.

Standard agglomerative HC methods use set closeness to form clusters in a bottom-up fashion, as described in algorithm HCAlg (Figure 1). It receives an n-point dataset $\mathcal{D}$ and groups its points into a total of $\ell_t \leq n$ *target* clusters, by iteratively merging pairs of closest clusters into their union. The merging history is stored (redundantly) in a *dendrogram* T, that is, a forest of clusters of $n - \ell_t + 1$ levels, where siblings correspond to merged clusters and levels to dataset partitions, build level-by-level as follows:

- Initially, each input point $v_i \in \mathcal{D}$ forms a singleton cluster $\{v_i\}$ as a leaf in T (at its lowest level n).
- Iteratively, in $n - \ell_t$ *clustering rounds*, the i root clusters (at top level i) form $i - 1$ new root clusters in T (at higher level $i - 1$), with the closest two merged into a union cluster as their parent, and each other cluster copied to level $i - 1$ as its parent.

When a new level of ℓ_t target clusters is reached, HCAlg halts and outputs T. The exact value of $\ell_t \in [1 : n]$ is determined during execution via a predefined condition End checked over the current state T and a termination parameter t provided as additional input. This allows for flexible termination conditions—e.g., stopping when inter-cluster distance drops below an threshold specified by t, or simply when exactly $\ell_t = t$ target clusters are formed.

Typically, the dendrogram T is augmented to store some associated cluster metadata, by keeping, after any union/copy cluster is formed, some useful statistics over its contained points. Common such statistics for cluster C is its *size* $size(C) = |C|$ and *representative* value $rep(C)$, usually defined as its *centroid* (i.e., a certain type of average) point. Overall, for a set M of cluster statistics of interest and specified linkage distance and termination condition, HCAlg is viewed to operate on indexed dataset $\mathcal{D} = \{v_i\}_{i=1}^n$ and return an

Hierarchical Clustering Algorithm HCAlg

Input: Indexed set $\mathcal{D} = \{\mathbf{v}_i\}_{i=1}^n$, termination parameter t

Output: Dendrogram T, clusters $C(T)$, metadata $M(T)$

Parameters: Linkage distance $\delta(\cdot, \cdot)$, termination condition $\mathsf{End}(\cdot, \cdot)$, cluster statistics set $M \supseteq \{rep(\cdot), size(\cdot)\}$

[Initially, at level n]

1. **Initialize dendrogram** T**:** For each $i = 1, \ldots, n$:
– Create node u_i as the ith left-most leaf in T.
– Set $C(u_i) = \{\mathbf{v}_i\}$ as the singleton cluster of u_i.
– Compute $M(u_i) = \{m(\mathbf{v}_i) | m \in M\}$ as statistics of u_i.
2. **Set up linkages:** Compute linkages of all pairs of singleton clusters as a dictionary D, where $\{C(u_i), C(u_j)\}$ is keyed under $\delta(C(u_i), C(u_j))$, $1 \leq i < j \leq n$.

[Iteratively, at level $i = n, \ldots, \ell_t + 1$]

1. **Update** T**:** If N_i is the set of nodes in T at level i:
– Find in D the min-linkage pair (u, u') of nodes in N_i, breaking ties using a fixed rule over leaf-node indices.
– Create node $w \in N_{i-1}$ as parent of u and u'; set $C(w) = C(u) \cup C(u')$; for each node $\bar{u} \in N_i - \{u, u'\}$, create node $\bar{w} \in N_{i-1}$ as parent of $\bar{u}$; set $C(\bar{w}) = C(\bar{u})$.
– For each node $\hat{w} \in N_{i-1}$, compute $M(\hat{w})$.
2. **Check termination:** If $\mathsf{End}(T, t) == 1$, terminate.
3. **Update linkages:** Compute linkage $\delta(C(w), C(\bar{w}))$, for all $\bar{w} \in N_{i-1} - \{w\}$, and consistently update dictionary D.

Figure 1: Agglomerative hierarchical clustering.

M-augmented dendrogram T, comprised of: (1) the *forest structure of dendrogram* T, specifying the full merging history of input points into formed clusters (from n singletons to ℓ_t target ones); (2) the *cluster set* $C(T)$; and (3) the *metadata set* $M(T)$ associated with (clusters in) T. Assuming that HCAlg employs a fixed tie-breaking method in merging clusters, its execution is *deterministic*.

Secure computation and threat model. We consider the standard setting for private two-party computation, where two parties wishing to evaluate function $f(\cdot, \cdot)$ on their individual, private inputs x_1, x_2, engage in an interactive cryptographic protocol that upon termination returns to them the common output $y = f(x_1, x_2)$. Protocol security has this semantics: Subject to certain computational assumptions and misbehavior types during protocol execution, no party learns anything about the input of the other party, other than what can be inferred by its own input x_i and the learned result y. In this context, we study privacy-preserving hierarchical clustering in the *semi-honest* adversarial model which assumes that parties are honest, but curious: They will follow the prescribed protocol but also seek to infer information about the input of the other party, by examining the transcript of exchanged messages—the latter, assumed to be transferred over a reliable channel.

Although, in practice, parties may choose to be malicious, deviating from the prescribed protocol if they can benefit from this and can avoid detection, the semi-honest adversarial model still has its merits, especially in the studied PCL setting. Namely, it provides essential privacy protection for any privacy-aware party to enter the joint computation to benefit from collaborative learning. We note that, by trading off efficiency, security can be hardened via known generic techniques for compiling protocols secure in this model into counterparts secure against malicious parties.

Garbled circuits. One of the most widely used tools for two-party secure computation, *Garbled Circuits* (GC) [113, 114] allow two parties to evaluate a boolean circuit on their joint data without revealing their respective inputs. This is done by generating an encrypted truth table for each gate while evaluating the circuit by decrypting these tables in a way that preserves input privacy. In Appendix A, we provide more details about the GC framework.

Homomorphic encryption. This technique allows carrying out operations over encrypted data. Fully Homomorphic Encryption (FHE) [41] can evaluate arbitrary functions over ciphertexts, but remains rather impractical. Partially homomorphic encryption supports only specific arithmetic operations over ciphertexts, but allows for very efficient implementations [86, 91]. We use Paillier's scheme for Additively Homomorphic Encryption (AHE) [86], summarized as follows. For security parameter λ, keys generated by running $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and a public RSA modulus N, the scheme encrypts (with public key pk) any message m in the plaintext space $\mathbb{Z}_N$ into a ciphertext $[m]$, ensuring that decryption (with secret key *sk*) of any ciphertext product $[m] \cdot [m']$ mod N^2 (computable without sk) results in the plaintext sum $m + m'$ mod N. Thus, decrypting $[m]^k$ mod N^2 results in km mod N, and the ciphertext product $[m] \cdot [0]$ results in a fresh encryption of m.

3 FORMAL PROBLEM SPECIFICATION

We introduce a model for studying private hierarchical clustering, the first to provide formal specifications for secure two-party protocols for this central PCL problem. Importantly, we define security for a refined learning task that achieves a meaningful balance between the intended accuracy and privacy—a necessary compromise for the problem at hand to even be defined as a PCL instance!

We first formulate two-party privacy-preserving hierarchical clustering as a secure computation. Parties P_1, P_2 hold independently owned datasets P, Q of points in $\mathbb{Z}_{2^l}^d$, and wish to perform a collaborative hierarchical clustering over the combined set $\mathcal{D} = P \cup Q$. They agree on the *exact specification* f_{HC} of this learning task, as a function of their individually contributed datasets that encompasses all other parameters (e.g., for termination).

Let Π be a two-party protocol that *correctly* realizes $f_{HC}(\cdot,\cdot)$: Run jointly on inputs x_1, x_2, Π returns the common output $f_{HC}(x_1, x_2)$. Thus, parties P_1, P_2 can learn cluster model $f_{HC}(P, Q)$ by running protocol Π on their inputs P, Q. As discussed, Π is considered to be secure if its execution prevents an honest-but-curious party from learning anything about the other party's input that is not implied by the learned output. We formalize this intuitive privacy requirement via the standard two-party *ideal/real world* paradigm [45].

Ideal functionality. First, we define what one can best hope for. Cluster analysis with *perfect* privacy is trivial in an *ideal* world, where P_1, P_2 instantly hand-in their inputs x_1, x_2 to a *trusted* third party, called the *ideal functionality* f_{HC}, that computes and announces $f_{HC}(x_1, x_2)$ (and explodes). Here, the use of terms "perfect" and "ideal" is fully justified for no information about any private input is leaked *during* the computation. Some information about x_1 or x_2 may be inferred *after* the output is announced, by combining the known x_2 or x_1 with the learned $f_{HC}(x_1, x_2)$: It is the *inherent price for collaboratively learning a non-trivial function.*

Ideal Functionality $f^*_{HC}(\cdot,\cdot)$
Input: Sets $P = \{p_i\}_1^{n_1}$, $Q = \{q_j\}_1^{n_2}$
Output: Dendrogram T^*, metadata $M^* \supseteq \{rep(\cdot), size(\cdot)\}$
Parameters: Linkage distance $\delta(\cdot,\cdot)$, termination condition $\mathsf{End}(\cdot, t)$, cluster statistics set M, selection function $\mathsf{S}(\cdot)$
[Pre-process] Form input of size $n = n_1 + n_2$ for HCAlg:
1. Set $\mathcal{D} = \{d_k\}_1^n$ s.t. $d_k = p_k$, if $k \le n_1$, or else $d_k = q_{k-n_1}$.
2. Pick random permutation $\pi : [n] \to [n]$; set $\mathcal{D}^* = \pi(\mathcal{D})$.
[HC-process] Run $\mathsf{HCAlg}(\mathcal{D}^*, t)$ w/ parameters δ, M, End.
[Post-process] Redact output T^*, $C(T^*)$, $M(T^*)$ of HCAlg:
1. Set $M^* = \emptyset$; $\forall v \in T^*$: if $\mathsf{S}(v) == 1$, $M^* \leftarrow M^* \cup \{M(v)\}$.
2. Return T^*, M^*.

Figure 2: Ideal functionality f^*_{HC} for hierarchical clustering.

In the *real* world, P_1, P_2 learn $f_{HC}(x_1, x_2)$ by interacting in the joint execution of a protocol Π. We measure the privacy quality of Π against the ideal-world perfect privacy, dictating that running Π is effectively *equivalent* to calling the ideal functionality f_{HC}. Informally, Π *securely realizes* f_{HC}, if anything computable by an efficient semi-honest party P_i in the real world, can be simulated by an efficient algorithm (called the simulator Sim), acting as P_i in the ideal world; i.e., Π leaks no information about a private input during execution, subject to the price for learning $f_{HC}(x_1, x_2)$.

Next comes the question of which ideal functionality f_{HC} should Π securely realize for private joint hierarchical clustering? Though tempting, equating f_{HC} with the legacy algorithm HCAlg (Figure 1), thus learning a full-form augmented dendrogram, slides us into a degeneracy. Assume f_{HC} merely runs HCAlg on the combined indexed set $\mathcal{D} = P \cup Q = \{d_k\}_{k=1}^n$, $n = |P| + |Q|$.[2] The learned model is the dendrogram T along with its associated clusters $C(T)$ and metadata $M(T)$. But set $C(T)$ itself reveals the input $\mathcal{D}$; in this case, the price for collaborative learning is full disclosure of sensitive data and nothing is to be protected! This raises the question of limiting exactly what information about P, Q should be revealed by f_{HC} which is the focus of the remainder of this section.

Refined cluster analysis. In the PCL setting, we need a new definition of hierarchical clustering that distills the full augmented dendrogram $\{T, C(T), M(T)\}$ into a redacted, but still useful, learned model, balancing between accuracy (to benefit from clustering) and privacy (to allow collaboration). If allowing the ideal functionality f_{HC} to return $C(T)$ is one extreme that diminishes privacy, removing the dendrogram T from the output—to learn only about its associated information $C(T)$, $M(T)$—is another that diminishes accuracy. Indeed, if T, which captures the full merging history in its structure, is excluded from the output of f_{HC}, a core feature in HC is lost: the ability to gain insights on *how target clusters were formed*, under what hierarchies and in which order. This renders the HC analysis only as good as much simpler techniques (e.g., k-means) that merely discover pure similarity statistics of target clusters. As the motivation for studying collaborative HC as a prominent and widely used unsupervised learning task, in the first place, lies

[2] If P, Q are indexed, then $\mathcal{D} = Q \| P$, or else a *fixed* ordering is used.

exactly on its ability to discover such rich inter-cluster relations, we must keep the forest structure of T in f_{HC}'s output.[3]

Avoiding the above two degenerate extremes suggests that the learned model $f_{HC}(P, Q)$ should necessarily include the cluster hierarchy T but not the clusters $C(T)$ themselves. Yet, the obvious *middle-point* approach of learning model $f^m_{HC}(P, Q) = \{T, M(T)\}$ remains suboptimal in terms of privacy protections, as the learned output can still be strongly correlated to exact input points. Indeed, given T and a party's own input, inferring points of the other party's input simply amounts to identifying singleton clusters, which is generally possible by inspecting and correlating the (hard-coded in HCAlg) indices in $\mathcal{D}$ with the metadata associated to singletons (or their close neighbors). For instance, if w is the parent of singleton u and cluster u' in T, then P_1 can infer input point $C(u)$ of P_2, either directly from output $M(u)$, if u is known to store none of its input points, or indirectly from $M(u')$, $M(w)$, if these output values imply a value of $M(u)$ that is consistent with none of its own inputs.

Also, even without singleton clusters in the output, there is still leakage from the positioning of the points at the leaf level of T. E.g., assuming P, Q are ordered from left to right, a merging of two points at the right half of the tree during the first merge reveals to P_1 that P_2 has a pair of points with smaller distance than the minimum distance observed among points in P. Hence, it is crucial to eliminate information about the positioning of clusters in T.

Point-agnostic dendrogram. Such considerations naturally lead to a new goal: We seek to refine further, but minimally so, the middle-point model $f^m_{HC}(P, Q) = \{T, M(T)\}$ into an optimized model $f^*_{HC}(P, Q) = \{T^*, M^*(T)\}$, whereby *no private input points directly leak to any of the parties*, after the output is announced. This quality is well-defined, intuitive and useful: Unless the intended joint hierarchical clustering explicitly copies some of input points to the output, the learned model $f^*_{HC}(P, Q)$ should allow no party to explicitly learn, that is, to deterministically deduce with certainty, any of the unknown input points of the other party.

We accordingly set our ideal functionality f^*_{HC} for hierarchical clustering to outputs a *point-agnostic augmented dendrogram*, defined by merely running algorithm HCAlg, subject to a twofold *correction* of its input P, Q and returned dendrogram (Figure 2):

- **Pre-process input:** Run HCAlg on indexed set $\mathcal{D}^*$ that is a *random permutation* over the combined set $\mathcal{D} = P \cup Q = \{d_k\}_{k=1}^n$.
- **Post-process output:** Return the output T^*, $C(T^*)$, $M(T^*)$ of HCAlg *redacted* as $T^*, M^*(T^*) \subset M(T^*)$, including metadata of only a few *safe* clusters in T^*.

Our ideal functionality f^*_{HC} refines the ordinary dendrogram T, $C(T)$, $M(T)$: Running HCAlg on the randomly permuted input $\mathcal{D}^*$ (instead of $\mathcal{D}$) results in a new *randomized* forest structure T^* (instead of T) and, although its associated sets of formed clusters $C(T^*)$ and metadata $M(T^*)$ remain the same, the learned model includes no elements from $C(T^*)$, but only specific elements from $M(T^*)$, determined by a *selection function* $\mathsf{S}(\cdot)$ (as a parameter agreed upon among the parties and hard-coded in f^*_{HC}). Such metadata is safe to learn, in the sense that it does not directly leak any input points.

[3]Cluster hierarchy is vital in HC learning, e.g., in healthcare, revealing useful causal factors that contribute to prevalence of diseases [34] and in biology, revealing useful relationships among plants, animals and their habitat ecological subsystems [44].

We propose the following two orthogonal strategies for safe metadata selection for point-agnostic dendrograms:

- s-**Merging selection:** $M(w) \in M(T^*)$ if w is the parent of u, u' in T^* and $|C(u)|, |C(u')| > s$: any non-singleton cluster formed by merging two clusters of size above threshold $s > 0$, is safe;
- **Target selection:** $M(w) \in M(T^*)$ if w is root in T^*: any target cluster at level ℓ_t in T^* is safe.

Above, the first strategy ensures that no direct leakage of private input points occurs by correlating statistics of thin neighboring clusters; in particular, no cluster statistics are learned for singletons or their parents ($s = 1$), thus eliminating the type of leakage allowed by model $f^m_{HC}(P, Q)$. The second strategy ensures that only statistics of target clusters are learned, that is, input points may be directly learned only explicitly as part of the intended cluster analysis.

Overall, the resulting dendrogram is point-agnostic in the sense that neither the forest structure of T^* nor the metadata $M(T^*)$ reveal which singletons a party's points are mapped to. As points are randomly mapped to singletons, ties in cluster merging are randomly broken, and no statistics are learned for singleton (or thin) clusters, no party can deduce with certainty any of the other party's input points. For instance, the applied permutation eliminates leakage from the positioning of the singleton cluster at the leaves that, in our previous example, allowed one to infer whether the other party owned points with smaller distance than its own pairs, from the first-round clustering result. More generally, anything inferable about a party's private input relates to a meta-analysis that must necessarily encompass the (unknown) input distribution and the random permutation used by f^*_{HC}. This can be viewed as an *inherent price of collaborative hierarchical clustering*. The following defines the security of privacy-preserving hierarchical clustering.

Definition 3.1. A two-party protocol Π, jointly run by P_1, P_2 on respective inputs x_1, x_2 using individual random tapes r_1, r_2 that result in incoming-message transcripts t_1, t_2, is said to be *secure* for *collaborative privacy-preserving hierarchical clustering* in the presence of *static, semi-honest adversaries*, if it securely realizes the ideal functionality f^*_{HC} defined in Figure 2, by satisfying the following: For $i = 1, 2$ and for any security parameter λ, there exists a non-uniform probabilistic polynomial-time simulator $\mathsf{Sim}_{\mathsf{P}_i}$ so that $\mathsf{Sim}_{\mathsf{P}_i}(1^\lambda, x_i, f^*_{HC}(x_1, x_2)) \cong \mathsf{view}^{\Pi}_{\mathcal{A}_{\mathsf{P}_i}} \triangleq \{r_i, t_i\}$.

4 MAIN CONSTRUCTION

We now present our main construction, protocol PHC for **P**rivate **H**iearchical **C**lustering that securely realizes the ideal functionality f^*_{HC} (of Figure 2) when jointly run by parties $\mathsf{P}_1, \mathsf{P}_2$.

General approach. As discussed earlier, for efficiency reasons, we seek to avoid carrying out hierarchical clustering—a complex and inherently iterative process of cubic costs—in its entirety by computing over ciphertext (e.g., via GC or FHE). Instead, we adopt a *mixed-protocols* design, decomposing hierarchical clustering into more elementary tasks. We then use tailored secure and efficient protocols for each task, and combine these components into a final protocol, in ways that minimize the cost in converting data encoding between individual sub-protocols. Hence, our solution is a secure mixed-protocol specifically tailored for hierarchical clustering.

It is worth noting that generic solutions from 2-party computation (2PC) (e.g., [28]), would solve the problem but would not

Algorithm 1: PHC: Private Cluster Analysis

P_1**'s Input:** $P = \{p_1, \ldots, p_{n_1}\}$, security parameter λ
P_2**'s Input:** $Q = \{q_1, \ldots, q_{n_2}\}$, security parameter λ
Output: Merging history, $\{rep(\cdot), size(\cdot)\}$ of t target nodes
Parameters: Default configurations

1 P_1: Generate $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{Gen}(1^\lambda)$; send pk to P_2
2 P_2: Generate $(\mathrm{pk}', \mathrm{sk}') \leftarrow \mathrm{Gen}(1^\lambda)$; send pk′ to P_1
3 P_1, P_2: Jointly run PHC.Setup, PHC.Cluster, PHC.Output

easily scale to large datasets. During hierarchical clustering, we need to maintain a distance matrix between two parties with space complexity $O(n^2)$. If one relies solely on a single generic approach such as GC or secret sharing, the communication bandwidth would become the bottleneck. Hence, using additively homomorphic encryption during our protocol's setup phase in order to produce a "shared permuted" distance matrix allows us not only to hide the correspondence between euclidean distances and original points, but also to be more communication efficient eventually. Another advantage compared to other 2PC techniques is that our approach can achieve better precision as we explain in more detail in Section 7.

Our protocol securely implements f^*_{HC} for the configuration that the parties specify: linkage $\delta(\cdot,\cdot)$, termination condition $\mathsf{End}(\cdot, t)$, cluster statistics set M, selection function $\mathsf{S}(\cdot)$. Yet for simplicity, hereby, we use the following *default configurations*, where:

(1) *complete linkage* over *one-dimensional* data is used;
(2) the termination condition results in t *target nodes*;
(3) *target selection* is used for safe metadata selection; and
(4) *only* representative values and size statistics are learned.

That is, by (2) - (4) in what follows (and in our experiments in Section 7), the set of redacted statistics M^* consists of the representatives $rep_1, \ldots, rep_{\ell_t}$ and sizes $size_1, \ldots, size_{\ell_t}$ of $\ell_t = t$ target clusters (recall that representatives are a predefined type of centroid of the cluster, e.g., average or median), where t is fixed in advance. Configuration 1) is used only for clarity; we discuss optimizations for single linkage and extensions to higher dimensions in Section 5 (and we report on the evaluation of such extensions in Section 7).

Protocol overview. After choosing configurations, P_1, P_2 run protocol PHC (Algorithm 1), with inputs their datasets P, Q of n_1, n_2 points, $n = n_1 + n_2$, security parameter λ, and statistical parameter κ. Each party establishes its individual Paillier key-pair, and then parties exchange their corresponding public keys.

Then, parties run sub-protocols PHC.Setup, PHC.Cluster and PHC.Output, which comprise the three main phases in our protocol, in direct analogy to the three components of f^*_{HC}. The general flow of our protocol is described below, in reference to also Figure 3.

In a setup phase, sub-protocol PHC.Setup processes the n input points, viewed as an input array I, and all pairwise distances among these points, viewed as a $n \times n$ cluster distance matrix Δ. Here, I, Δ are only virtual, corresponding to an early joint state of P_1, P_2 that is actually *secret-shared* between them. Specifically, P_1 holds an array L with exactly I's elements but each *AHE-encrypted* under P_2's secret key, and a $n \times n$ matrix R with *random blinding terms*, whereas P_2 holds the matrix $B = \Delta + R$ with *blinded pairwise cluster distances*. Importantly, as f^*_{HC} specifies, the joint state $\{I, \Delta\}$ is split only after I's elements and Δ's rows and columns are *randomly shuffled*, with P_1, P_2 not knowing the exact shuffling used.

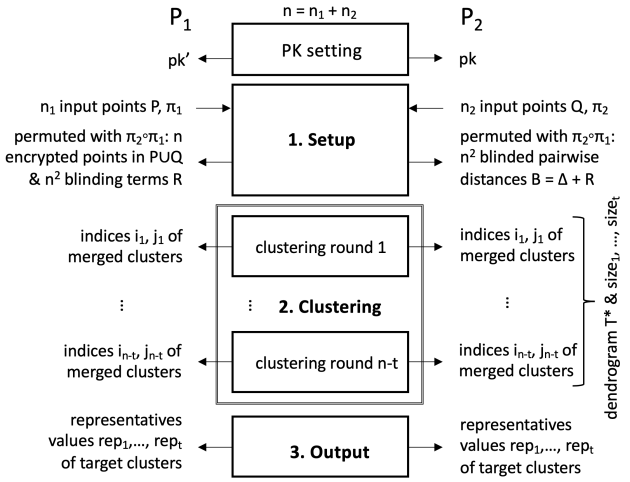


Figure 3: Overall workflow of our protocol PHC.

In a clustering phase, sub-protocol PHC.Cluster virtually runs the ordinary hierarchical clustering algorithm HCAlg on matrix Δ: P_1, P_2 process their individual states R, B to iteratively merge singletons into target clusters, based on inter-cluster distances in Δ. Each iteration merges two clusters into a new one via three tasks:

- **Find pair:** First, P_1, P_2 find the closest-cluster pair $(i, j) = \mathrm{argMin}(\Delta)$, $i < j$, to merge, i.e., the indices in Δ of the minimum inter-cluster distance D_{ij}.
- **Update linkages:** Then, P_1, P_2 update Δ to $\Delta' = B' - R'$ with the new cluster distances after pair (i, j) is merged into cluster $C = C_i \cup C_j$. This entails computing (and splitting via a fresh blinding term) distance $\delta(C, C')$ between C and each not-merged cluster C', which equals to the largest.smallest) of $\delta(C_i, C')$ and $\delta(C_j, C')$ (by associativity of the max/min operator).
- **Record merging:** Finally, P_1, P_2 record in Δ' that the new cluster C is formed by merging C_i and C_j.

In an output phase, sub-protocol PHC.Output processes the final state $\{I, \Delta\}$ to compute the merging history and metadata for all safe (target) clusters. As Figure 3 indicates, conceptually the output can be considered to be computed in two phases: During clustering, the indices of merged clusters learned after each clustering round collectively encode information about the dendrogram T^* and the sizes of the t target clusters. The output phase solely computes the representative values of these clusters. This view is accurate enough to ease presentation but, as we discuss later, the exact details involve processing of carefully recorded data, after each one of the $n - t$ cluster-merging rounds executed during the clustering phase.

A main consideration when devising our protocol was to improve efficiency via a modular design, where separate parts can be securely achieved via different techniques. By securely splitting the joint state $\{I, \Delta\}$ into $\{L, R\}$, B, we can implement all protocol components that involve (distance or metadata) computations over points using Paillier-based AHE, except when computing max (or min), for which we rely on GC. Conveniently, all protocol components required by the setup phase to form the joint state $\{I, \Delta\}$, namely to construct, shuffle and split $\{I, \Delta\}$ into $\{L, R\}$, B, can be securely implemented by relying on homomorphic encryption.

We next provide more details on how each component is implemented. We assume points are unambiguously mapped into

λ, κ	security and statistical parameters
t, ℓ_t	termination parameter, # of target clusters
$(\mathsf{pk}, \mathsf{sk})$, $(\mathsf{pk}', \mathsf{sk}')$	public and secret keys of parties $\mathsf{P}_1, \mathsf{P}_2$
$[d]$, $[\![d]\!]$	AHE-encrypted plaintext d under $\mathsf{pk}, \mathsf{pk}'$
$\langle c \rangle$	AHE-decrypted ciphertext c
Σ	$n \times n$ matrices R, B stored by $\mathsf{P}_1, \mathsf{P}_2$
$(O_1; O_2) \leftarrow \mathsf{GC}(I_1; I_2)$	$\mathsf{P}_1, \mathsf{P}_2$ run GC on I_1, I_2 to get O_1, O_2
π_1, π_2	permutations contributed by $\mathsf{P}_1, \mathsf{P}_2$
$\mathsf{dist}(p, q)$	square Euclidean distance of p, q
$rep(\cdot), size(\cdot)$	representatives and sizes of clusters

Figure 4: Basic notation used in our protocol PHC.

integers in $\mathbb{Z}_N$ and all homomorphic (resp. plaintext) operations are reduced modulo N^2 (resp. N). We consistently denote the AHE-encrypted, under pk (resp. pk′), plaintext d by $[d]$ (resp. $[\![d]\!]$) and the AHE-decrypted, under any key, ciphertext c by $\langle c \rangle$. Whenever the context is clear, we denote each of the two $n \times n$ matrices R, B (maintained by $\mathsf{P}_1, \mathsf{P}_2$) by Σ. Finally, we denote the joint execution by $\mathsf{P}_1, \mathsf{P}_2$ of a GC-based protocol GC, on private inputs I_1, I_2 to get private outputs O_1, O_2, by $(O_1; O_2) \leftarrow \mathsf{GC}(I_1; I_2)$. Figure 4 summarizes the used notation by our detailed protocol descriptions.

Setup phase. $\mathsf{P}_1, \mathsf{P}_2$ set up their states in three rounds of interactions, as shown in Algorithm 2, using only homomorphic operations over AHE-encrypted data and contributing equally to the randomized state permutation and splitting. Initially, P_2 prepares, encrypts under its own key and sends to P_1, information related to its input set Q, which includes its encrypted points among other helper information H, and their encrypted pairwise distances D (lines 1-4).

Then, P_1 is tasked to initialize the states L, R and B. First, the list L of all encrypted (under pk′) points in $P \cup Q$ is created (by arranging the sets in some fixed ordering and then concatenating Q after P), and all points are further blinded by random additive terms in S (lines 5-9). Similarly, the matrix B of encrypted (also under pk′) pairwise distances is computed (using the ordering induced by L to arrange the points), and all distances are blinded by random additive terms in S (lines 10-14). The computation of square Euclidean distances across sets P, Q (line 12, using also elements in H) and the blinding of L and B (lines 8, 13) are all performed in the ciphertext domain via the homomorphic property of AHE encryption. All blinding terms in S and R are then encrypted (each under pk, lines 9, 14) and S, L, R and B are sent to P_2, after their elements are shuffled using a random permutation π_1 (line 15).

Finally, P_2 roughly mirrors this by further blinding the encrypted points in L and P_1's encrypted terms in S by random additive terms in S' (both in the ciphertext domain, lines 16-19) and also blinding the encrypted distances in B and P_1's encrypted terms in R by random additive terms in R' (the former in the plaintext domain and the latter in the ciphertext domain, lines 20-22). The freshly blinded S, L, R are sent to P_1, after their elements are shuffled using a random permutation π_2 (line 23). Finally, P_1 decrypts the mutually-contributed blinding terms in S and R, and uses the recovered values in S to completely remove the terms from L (in the ciphertext domain, by the properties of AHE encryption, lines 24-27). Due to this, permutation $\pi_2 \circ \pi_1$ looks completely random to both parties, while they have securely split joint state $\{I, \Delta\}$ into $\{L, R\}$, B.

Clustering phase. Once $\mathsf{P}_1, \mathsf{P}_2$ have set up their states, they run the hierarchical clustering iterative process (Algorithm 3) operating solely on their individual matrices R, B via two special-purpose GC-based protocols for secure comparison. Importantly, each party

Algorithm 2: PHC.Setup: Setup Phase

1 P_2: %Create & send helper info
2 Compute matrix $\mathbf{H}$: $H_{1,i} = [\![q_i]\!]$, $H_{2,i} = [\![-2q_i]\!]$, $H_{3,i} = [\![q_i^2]\!]$ $i \in [1:n_2]$
3 Compute matrix $\mathbf{D}$: $D_{i,j} = [\![\mathsf{dist}(q_i, q_j)]\!]$ $i, j \in [1:n_2]$
4 Send $\{\mathbf{H}, \mathbf{D}\}$ to P_1
5 P_1: %Blind points, linkages
6 Compute array $\mathbf{S}$: $S_i = s_i$, $s_i \xleftarrow{\$} \{0,1\}^\kappa$ $i \in [1:n]$
7 Compute array $\mathbf{L}$: $L_i = [\![p_i]\!]$, if $i \le n_1$; else $L_i = H_{1,i-n_1}$
8 Blind $\mathbf{L}$ as: $L_i := L_i \cdot [\![S_i]\!]$
9 Encrypt $\mathbf{S}$ as: $S_i := [S_i]$
10 Compute matrix $\mathbf{R}$: $R_{i,j} = r_{i,j}$, $r_{i,j} \xleftarrow{\$} \{0,1\}^\kappa$ $i, j \in [1:n]$
11 Compute matrix $\mathbf{B}$: $B_{i,j} = [\![\mathsf{dist}(p_i, p_j)]\!]$, if $i, j \le n_1$;
12 $B_{i,j} = D_{i-n_1,j-n_1}$, if $n_1 < i, j$; else for $i < j$, $B_{i,j} = [\![p_i^2]\!] \cdot H_{2,j}^{p_i} \cdot H_{3,j}$
13 Blind $\mathbf{B}$ as: $B_{ij} := B_{ij} \cdot [\![R_{i,j}]\!]$
14 Encrypt $\mathbf{R}$ as: $R_{i,j} := [R_{i,j}]$
15 Permute $\mathbf{S}, \mathbf{L}, \mathbf{R}$ and $\mathbf{B}$ via a random permutation $\pi_1(n)$
16 Send $\{\mathbf{S}, \mathbf{L}, \mathbf{R}, \mathbf{B}\}$ to P_2 %Send permuted blinded data
17 P_2: %Blind received data
18 Compute array $\mathbf{S}'$: $S'_i = s'_i$, $s'_i \xleftarrow{\$} \{0,1\}^\kappa$ $i \in [1:n]$
19 Blind $\mathbf{L}$ as: $L_i := L_i \cdot [\![S'_i]\!]$
20 Blind $\mathbf{S}$ as: $S_i := S_i \cdot [S'_i]$
21 Compute matrix $\mathbf{R}'$: $R'_{i,j} = r'_{i,j}$, $r'_{i,j} \xleftarrow{\$} \{0,1\}^\kappa$ $i, j \in [1:n]$
22 Decrypt and re-blind matrix $\mathbf{B}$: $B_{i,j} = \langle B_{i,j} \rangle + R'_{i,j}$
23 Blind $\mathbf{R}$ as: $R_{i,j} := R_{i,j} \cdot [R'_{i,j}]$
24 Permute $\mathbf{S}, \mathbf{L}$ and $\mathbf{R}$ via a random permutation $\pi_2(n)$
25 Send $\{\mathbf{S}, \mathbf{L}, \mathbf{R}\}$ to P_1 %Send permuted points and blinding terms
26 P_1: %Store permuted points & linkages' blinding terms
27 Decrypt $\mathbf{S}$ as: $S_i := \langle S_i \rangle$ $i \in [1:n]$
28 Unblind $\mathbf{L}$ as: $L_i := L_i \cdot [\![S_i]\!]^{-1}$
29 Decrypt $\mathbf{R}$ as: $R_{i,j} := \langle R_{i,j} \rangle$ $i, j \in [1:n]$

Algorithm 3: PHC.Cluster: Clustering Phase

1 $\mathsf{P}_1, \mathsf{P}_2$:
2 Initialize merging history: $\Sigma_{i,i} = (i, \perp)$ $i \in [1:n]$
3 Initialize: $\ell = 1$, $\ell_t = t$
4 **repeat**
5 Jointly run $(i, j; i, j) \leftarrow \mathsf{ArgMin}(\mathbf{R}; \mathbf{B})$, $i < j$ %Find pair
6 **foreach** $k = 1, \ldots, n, k \neq i, j$ **do**
7 **if** $\Sigma_{i,k} \neq \perp$ and $\Sigma_{j,k} \neq \perp$ **then**
8 P_1: $X \xleftarrow{\$} \{0,1\}^\kappa$ %Pick new blinding term
9 $\mathsf{P}_1, \mathsf{P}_2$: Jointly run %Find re-blinded max linkage
$(\perp; Y) \leftarrow \mathsf{MaxDist}(R_{i,k}, R_{j,k}, X; B_{i,k}, B_{j,k})$
10 P_1: Set: $R_{ik} = X$, $R_{ki} = X$ %Update linkages
11 P_2: Set: $B_{ik} = Y$, $B_{ki} = Y$
12 Set: $\Sigma_{j,j} := ((\Sigma_{j,j}, \ell), i)$, $\Sigma_{i,i} := ((\Sigma_{i,i}, \Sigma_{j,j}, \ell), \perp)$
13 Set: $\Sigma_{k,j} = \perp, \Sigma_{j,k} = \perp$ $k \in [1:n] \setminus \{j\}$
14 Set: $\ell := \ell + 1$ %Record merging
15 **until** $\ell > n - \ell_t$;

encodes cluster information in the diagonal of its matrix state Σ; initially, the i-th entry stores $(i, \perp)$, denoting the (never-merged but already permuted) singleton of rank i. Hierarchical clustering runs in exactly $n - \ell_t = n - t$ iterations, or clustering rounds.

First, at the start of each iteration, $\mathsf{P}_1, \mathsf{P}_2$ find which pair of clusters must be merged by jointly running the GC-protocol $(i, j; i, j) \leftarrow \mathsf{ArgMin}(R; B)$ (line 5): The parties contribute their individual matrices R, B of blinding terms and blinded linkages, to learn the indices (i, j) of the minimum value $B_{i,j} - R_{i,j}$, with $i < j$ by convention (since R, B are symmetric matrices). The garbled circuit for ArgMin first removes the blinding terms by computing $D = B - R$, compares all values in D to find the minimum element $D_{i,j} = \min_{x,y} B_{x,y}$, and returns to both parties the indices i, j. Next,

Algorithm 4: PHC.Output: Output Phase

1 P_1: %Compute encrypted point averages
2 Initialize arrays $\mathbf{E}, \mathbf{J}$: $E_i = J_i = \perp$ $i \in [1:n]$
3 **foreach** $i = 1, \ldots, n$ **do**
4 **if** $R_{i,i}$ encodes a target cluster C_i **then**
5 Find the index set I_i of points in cluster C_i
6 Set $E_i = \prod_{j \in I_i} L_j$, $J_i = |I_i|$
7 Send $\{\mathbf{E}, \mathbf{J}\}$ to P_2
8 P_2: %Compute point averages
9 Decrypt $\mathbf{E}$ as: $E_i := \langle E_i \rangle$ $i \in [1:n]$
10 Send $\mathbf{E}$ to P_1
11 $\mathsf{P}_1, \mathsf{P}_2$: %Return output
12 Output $\{\Sigma_{i,i}, E_i/|J_i|, |J_i|\}$ $i \in [1:n]$

once pair (i, j) is known to $\mathsf{P}_1, \mathsf{P}_2$, they proceed to jointly update the linkages (lines 7-12). For each cluster k in Σ, they change its linkage to the newly merged cluster as the maximum between its linkages to clusters i, j, by jointly running the GC-protocol $(\perp; Y) \leftarrow \mathsf{MaxDist}(R_{i,k}, R_{j,k}, X; B_{i,k}, B_{j,k})$ (line 10): The parties contribute the two entries from their individual matrices R, B that are needed for comparing the linkages $B_{i,k} - R_{i,k}$, $B_{j,k} - R_{j,k}$ between cluster k and clusters i, j, and P_2 learns the maximum value of the two but blinded by the random blinding term X inputted by P_1. The garbled circuit for MaxDist simply returns to (only) P_2 the value $\max\{B_{i,k} - R_{i,k}, B_{j,k} - R_{j,k}\} + X$. Finally, at the end of iteration ℓ, $\mathsf{P}_1, \mathsf{P}_2$ record information about the merging of clusters i, j, $i < j$ (lines 14-15): By convention, the new cluster is stored at location i, by adding the rank ℓ and the information stored at location j (updated with a pointer to i), and by deleting all distances related to cluster j. Overall, the full merging history is recorded.

In Appendix B, we provide details on our implementation of GC-protocols ArgMin, MaxDist, also used in [11, 16, 65, 120].

Output phase. Once clustering is over, P_1, P_2 compute in two rounds of interaction (Algorithm 4) the common output, consisting of the merging history and the representatives and sizes of the target clusters using homomorphic operations over encrypted data.

First, P_1 computes encrypted point averages in all target clusters, by exploiting the homomorphic properties of AHE (lines 1-7): Using the diagonal in matrix R, P_1 first identifies each (of t total) target cluster C_i and then finds the index set I_i (over permuted input points $\pi_2 \circ \pi_1(P \cup Q)$) of the points contained in C_i, to finally compute $\prod_{j \in I_i} L_j = \prod_{j \in I_i} [\![p_j]\!]$. The resulted t encrypted point averages and cluster sizes are sent to P_2, who returns to P_1 the t plaintext point averages, i.e., $\sum_{j \in I_i} p_j = \sum_{p_j \in C_i} p_j$ for each C_i (lines 8-10). At this point, both parties can form the common output (line 12).

5 PROTOCOL ANALYSIS

Efficiency. Asymptotically, our protocol achieves optimal performance, as it incurs no extra overheads to the performance costs associated with running HC (ignoring the dependency on the security parameter λ). The asymptotic overheads incurred on $\mathsf{P}_1, \mathsf{P}_2$, during execution of each phase of PHC, are as follows: In setup phase, the cost overhead for each party is $O(n^2)$, primarily related to the cryptographic operations needed to populate its individual state Σ. In clustering phase, each of the $n - \ell_t = O(n)$ total iterations incurs costs proportional the complexity of running GC-based protocols ArgMin, MaxDist, where the cost of garbling and evaluating a circuit C, with a total number of wires $|C|$, is $O(|C|)$. Thus, during the ℓ-th iteration: Evaluating circuit ArgMin entails $n^2 - 2\ell$ comparisons of l^2-bit values (of cluster distances) and subtractions of κ-bit values (of blinding terms), for a total size of $O(\kappa(n^2 - \ell))$; likewise, evaluating circuit MaxDist entails a constant number of comparisons of κ-bit values and $O(n)$ such circuits are evaluated at iteration ℓ; thus, the total cost during this phase is $O(\kappa n^3)$ for each party. In output phase, the cost is $O(\ell_t) = O(n)$ for each party. Thus, the total running time for both parties is $O(\kappa n^3)$. Communication consists of $O(n^2)$ ciphertexts during setup (encrypted distances), $O(\kappa n^2)$ during each clustering round (for the garbled circuits' truth tables) and $O(n^2)$ ciphertexts during the output phase.

Optimized single-linkage protocol OPT**.** As described, our protocol exploits the associativity of operator max to update the complete linkage between newly formed clusters C and other clusters C', as the max of the linkages between C's constituent clusters and C', securely realized via GC-protocol $(\cdot;\cdot) \leftarrow \mathsf{MaxDist}(\cdot;\cdot)$. Single linkages can be supported readily by updating inter-cluster distances between C and C' as the min of the distances between C's constituent clusters and C': Line 10 in Algorithm 3 now has $\mathsf{P}_1, \mathsf{P}_2$ jointly run GC-protocol $(\perp; Y) \leftarrow \mathsf{MinDist}(R_{i,k}, R_{j,k}, X; B_{i,k}, B_{j,k})$ (see Appendix B) to split the new distance $\Delta_{i,k} = \min\{B_{i,k} - R_{i,k}, B_{j,k} - R_{j,k}\}$ into X, $Y = \Delta_{i,k} + X$, without asymptotic efficiency changes.

More generally, the skeleton of protocol PHC allows for extensions that support a wider class of linkage functions, such as average or centroid linkage, by appropriately refining GC-protocols ArgMin, MinDist—but still, at quadratic cost per merged cluster and cubic total cost. Yet, our single-linkage protocol can be optimized to process each new cluster in only $O(\kappa n)$ time, for a reduced $O(\kappa n^2)$ total running time, with GC-protocol $(j; j) \leftarrow \mathsf{ArgMin}(X; Y)$ now refined, on input arrays X, Y, to return as common output the minimum-value index j of $Y - X$, excluding any non-linkage values.

The main idea is to exploit the associativity of operator min and that single-linkage clustering only relates to minimum inter-cluster distances, to find the closest pair (i, j) in linear time, by looking up an array $\bar{\Delta} = \bar{B} - \bar{R}$ storing the minimum row-wise distances in $\Delta = B - R$ (a known technique in information retrieval [73, Section 17.2.1]). Our modified protocol takes only $O(\kappa n)$ comparisons per clustering, as opposed to $O(\kappa n^2)$ of our main protocol. As shown in Section 7, this results in significant performance improvement.

Specifically, at the end of the setup phase, $\mathsf{P}_1, \mathsf{P}_2$ now also jointly run $(j_i; j_i) \leftarrow \mathsf{ArgMin}(R_i; B_i)$, $i \in [1:n]$, to learn the minimum-linkage index j_i of the ith row $B_i - R_i$ of Δ (excluding its ith location, as $B_{i,i}$, $R_{i,i}$ store cluster i), and they both initialize array $\bar{J}$ as $\bar{J}_i = j_i$, whereas P_1 initializes array $\bar{R}$ as $\bar{R}_i = R_{i,j_i}$ and P_2 array $\bar{B}$ as $\bar{B}_i = B_{i,j_i}$. Then, at the start of each iteration in the clustering phase (line 5 in Algorithm 3) and assuming that $\perp = +\infty$, $\mathsf{P}_1, \mathsf{P}_2$ now jointly run $(i; i) \leftarrow \mathsf{ArgMin}(\bar{R}; \bar{B})$ to find the closest-cluster pair (i, j), $j = \bar{J}_i$, in only $O(\kappa n)$ time. Conveniently, as soon as they update linkages $\Delta_{i,k} = \Delta_{k,i}$, for some $k \neq i, j$ (lines 9-12, as $Y - X$ with $(\perp; Y) \leftarrow \mathsf{MinDist}(R_{i,k}, R_{j,k}, X; B_{i,k}, B_{j,k})$), P_1, P_2 also update the joint state $\{\bar{B} - \bar{R}, \bar{J}\}$ for updated row $m \in \{i, k\}$: First, by jointly running $(z; z) \leftarrow \mathsf{ArgMin}(\hat{R}; \hat{B})$ for arrays $\hat{R} = [X, \bar{R}_m]$, $\hat{B} = [Y, \bar{B}_m]$ of size 2, and then, if $z = 1$, by setting $\bar{R}_m = \hat{R}_z$, $\bar{B}_m = \hat{B}_z$ and $\bar{J}_m = \{i, k\} \setminus m$. At the end of each iteration (lines 14-16), they also set $\bar{R}_j = \bar{B}_j = \bar{J}_j = \perp$, as needed for consistency.

Protocol extensions. Our protocol can be easily adapted to handle higher dimensions ($d > 1$). Its sub-protocol (PHC.Cluster) compares squared Euclidean distances thus it is almost unaffected by the number of dimensions; only the setup and output phases need to be modified, as follows. P_2 computes helper information H, representing each point not by 3 but by $3d$ encryptions (i.e., line 4 of Algorithm 2 runs independently for each dimension). Analogously, P_1, P_2 compute square Euclidean distances (lines 3 and 11-12) as the sum of squared per-dimension differences across all dimensions (over AHE). Shuffling remains largely unaffected, besides lists L, S, S' consisting of dn encryptions each. Finally, representatives (line 6 in Algorithm 4) are now computed over vectors of d values. Our protocol can also extended to other distance metrics, e.g., L_1, L_2 or Euclidean, and any L_p distance for $p \geq 1$, with modifications for computing the distance matrix during setup. With squared Euclidean the distances are securely computed with AHE; for other metrics, more elaborate sub-protocol may be required.

Security. In Appendix C, we prove the following result:

Theorem 5.1. *Assuming Paillier's encryption scheme is semantically secure and that* ArgMin *and* MaxDist *are securely realized by GC-based protocols, protocol* PHC *securely realizes functionality* f^*_{HC}.

6 SCALABILITY VIA APPROXIMATION

The cryptographic machinery of our protocol imposes a noticeable overhead in practice. Although it is asymptotically similar to plaintext HC, standard operations are now replaced by cryptographic ones—no matter how well-optimized the code, such crypto-hardened operations will ultimately be slower. Hence, to scale to larger datasets, we seek to exploit *approximate* schemes for hierarchical clustering. In our case, approximation refers to performing clustering over a high-volume dataset by applying the HCAlg algorithm only on a small subset of the dataset. The effect of this is twofold: Cluster analysis is much faster but using fewer points lowers accuracy and increases sensitivity to outliers.

In what follows, we adapt the CURE approximate clustering algorithm [47] and seamlessly integrate it to our main protocol PHC, within a flexible design framework that offers a variety of configurations for balancing tradeoffs between performance and accuracy, to overall get the first variants of CURE for private collaborative hierarchical clustering. Although, in principle, our framework can be applied to any approximate clustering scheme (e.g., BIRCH [117]), we choose CURE for its strong resilience to outliers and high accuracy (even on samples less than 1% of original data)—features that place it among the best options for scalable hierarchical clustering.

Described in Figure 5, on input the original dataset $\mathcal{D}$ of size n and a number of approximation parameters, CURE first randomly samples s data points from $\mathcal{D}$ to form sample set $\mathcal{S}$. During A-clustering, $\mathcal{S}$ is partitioned into p equally-sized parts $\mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_p$, and the ordinary algorithm HCAlg runs p times to form a set C_A of A-clusters: Its ith execution is on input $\mathcal{P}_i$, $i \in [1, p]$, until exactly $s/(pq)$ clusters are formed, of which only those of size at least t_1 are included in C_A and the rest are eliminated as outliers. During B-clustering, HCAlg runs once again, this time over set C_A, to form a set C_B of B-clusters, from which clusters of size less than $t_2 > t_1$ are eventually eliminated as outliers. Finally, for each B-cluster in C_B a number of R random representatives are selected, and each

The CURE **approximate clustering algorithm**

Input: $\mathcal{D}, n, s, p, q, t_1, t_2, R$ **Output:** Clusters C over $\mathcal{D}$

[Sampling] Randomly pick s points in $\mathcal{D}$ to form sample $\mathcal{S}$.

[Clustering A]

1. Partition $\mathcal{S}$ into p partitions $\mathcal{P}_i$s, each of size s/p.
2. Run HCAlg to cluster each $\mathcal{P}_i$ into $s/(pq)$ target clusters.
3. Eliminate within each $\mathcal{P}_i$ clusters of size less than t_1.

[Clustering B]

1. Run HCAlg to cluster all remaining A-clusters C_A in $\mathcal{S}$.
2. Eliminate clusters of size less than t_2 to get B-clusters C_B.
3. Set R random points in each B-cluster as its representatives.

[Classification]

1. Assign singletons in $\mathcal{D}$ to B-cluster of closest representative.

Figure 5: The CURE **approximate clustering algorithm.**

singleton point in $\mathcal{D}$ is included to the B-cluster containing its closest representative. Table 1 summarizes suggested values for each parameter as per CURE's original description [47].

Private CURE**-approximate clustering.** We adapt the CURE algorithm to design private protocols for approximate clustering in our model for two-party joint hierarchical clustering. In applying our security formulation (Section 3) and our private protocols (Sections 4, 5) to this problem instance, the following facts are vital:

1. CURE involves three main tasks: input *sampling*, *clustering* of sample, and unlabeled-points *classification*.
2. Clustering involves $p + 1$ invocations of HCAlg$'$, which extends ordinary algorithm HCAlg to receive *clusters as input* and compute its *output over an input subset*.
3. If $p = 1$ and O_A, O_B are the A- and B-outliers, then:
   i. HCAlg$'$ first runs on $\mathcal{S}$ to form C_A over $\mathcal{S}_A \triangleq \mathcal{S} \setminus O_A$; C_A is exactly the output of HCAlg run on $\mathcal{S}_A$; and next
   ii. HCAlg$'$ runs on C_A to form C_B over $\mathcal{S}_B \triangleq \mathcal{S} \setminus \{O_A \cup O_B\}$; C_B is exactly the output of HCAlg run on $\mathcal{S}_B$.

Fact 1 refines our protocol-design space to only securely realizing the clustering task, where sampling and classification are viewed as input pre-processing and output post-processing of clustering. Specifically, P_1, P_2: (1) individually form random input samples $\mathcal{S}_P, \mathcal{S}_Q$ of their own datasets P, Q; (2) compute B-clusters and their representatives (as specified by CURE); and (3) use these B-cluster representatives to individually classify their own unlabeled points.

As such, the default private realization of CURE would entail having the parties perform *clustering A and B jointly*. Yet, since our design space is already restricted to provide approximate solutions, we also consider two protocol variants, where parties trade even more accuracy for efficiency, by performing: (1) *clustering A locally and only B jointly*; and, in the extreme case (2) *clustering A and B locally*. We denote these protocols by PCure2, PCure1 and PCure0.

In PCure0, P_1, P_2 non-collaboratively compute B-clusters of their samples and announce the representatives selected. Though a degenerate solution, as it involves no interaction, this consideration is still useful: First, to serve as a baseline for evaluating the other variants, but mostly to further refine our design space. PCure0 (trivially) preserves privacy during B-cluster computation, but violates the privacy guarantees offered by our point-agnostic dendrograms, by revealing a subset of a party's input points to the other party. To rectify this, present also in PCure2 and PCure1, we fix $R = 1$

Parameters	Description	Value
n, s	Sizes of dataset and its sample	$\leq$ 1M, $[10^2 : 10^3]$
p, q	# parts, cluster/part control	$p = 1, 3, 5, q = 3$
t_1, t_2	A-, B-cluster outlier thresholds	$3 = t_1 < t_2 = 5$
R	Representatives per B-cluster	$R = 1, 3, 5, 7, 10$

Table 1: CURE clustering parameters and values.

and have each B-cluster be represented by its centroid. Using average values is expected to have no impact on accuracy, at least for spherical clusters (in [47], $R > 1$ is only used to improve accuracy of non-spherical clusters). Fact 2 then ensures that B-clusters (and their centroids) can be computed by essentially running algorithm HCAlg, possibly with slight modifications (discussed below).

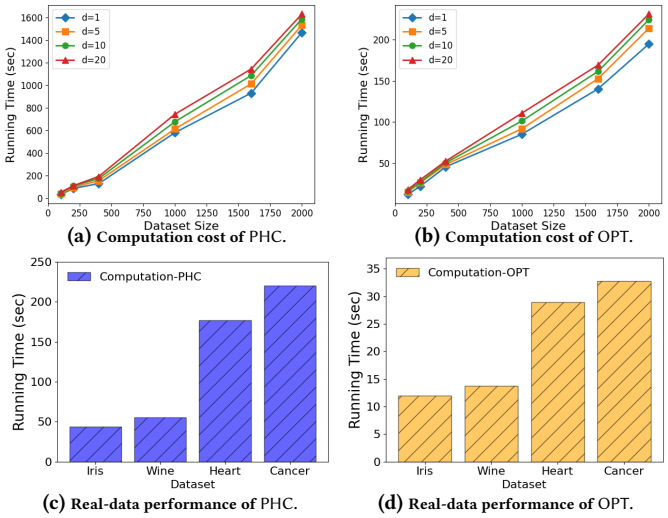
In PCure1, P_1, P_2 non-collaboratively compute A-clusters of their samples and then jointly merge them to B-clusters. Semantically, they run HCAlg, not starting at level n (singletons) but at an intermediate level i, where each input A-cluster contains at least t_1 points. Our PHC can be employed, with one modification: At setup, the parties' joint state encodes their individual A-clusters and their pairwise linkages. Accordingly, sub-protocol PHC.Setup is modified: (1) Lines 3 and 11 now compute inter-cluster distances (of same-party pairs), and (2) lines 2 and 12 are used as a subroutine to compute all point distances across a given A-cluster pair, over which inter-cluster linkages (of cross-party pairs) are evaluated with ArgMin. The running time of modified PHC.Setup is $O(\lambda s^2)$, as $O(s^2)$ distances are computed across $O(s)$ A-cluster points.

In PCure2, P_1, P_2 jointly compute A- and B-clusters. This introduces the challenge of how to transition from A to B. Simply running p copies of HCAlg in parallel for A-clusters does not provide the cluster linkages that are necessary for HCAlg to compute B-clusters. Possible solutions are either to treat A-clusters as singletons, which can drastically impair accuracy, or running an intermediate MPC protocol to bootstrap HCAlg with cluster linkages, which can impair performance. Instead, we simply fix $p = 1$, seamlessly using the final joint state of clustering A as initial state for clustering B. Missing speedups by parallelism is compensated by avoiding a costly bootstrap-protocol, at no accuracy loss, as our experiments confirm ($p > 1$, is only suggested for parallelism in [47]).

Finally, the security of protocols PCure1 and PCure2 can be reduced to that of PHC. Our modular design and facts 2 and 3, ensure that security in our private CURE-approximate clustering is captured by our ideal functionality f^*_{HC} of Section 3: The intended two-party computation merely involves computing B-cluster representatives, which f^*_{HC} provides, and any input/output modification in HCAlg causes a trivial change to the pre-/post-processing component of f^*_{HC}, consistent to our point-agnostic dendrograms.

7 EXPERIMENTAL EVALUATION

Our main goal is to evaluate the computational cost of our protocols and to determine the improvement of the optimized and approximate variants. We use four datasets from the UCI ML Repository [2], restricted to numeric attributes: (1) `Iris` for iris plants classification (150 records, 4 attributes); (2) `Wine` for chemical analysis of wines (178 records, 13 attributes); (3) `Heart` for heart disease diagnosis (303 records, 20 attributes); and (4) `Cancer` for breast cancer diagnostics (569 records, 30 attributes). As these are relatively small, we also generate our own synthetic datasets, scaling the size to millions of samples. Note that our protocol's performance depends mainly



(a) Computation cost of PHC. **(b) Computation cost of OPT.**

(c) Real-data performance of PHC. **(d) Real-data performance of OPT.**

Figure 6: Performance of PHC (left) Vs. OPT (right).

on the dataset size, is invariant to actual data values, and varies very little with data dimensionality, as our experiments confirm.

We introduced several variants of approximate clustering based on CURE and want to evaluate their accuracy and determine possible between performance-accuracy tradeoffs. Traditionally, hierarchical clustering is an unsupervised learning task, for which accuracy metrics are not well defined. However, it is common to evaluate the accuracy of clustering via ground truth datasets including class labels on samples. A good clustering algorithm will generate "pure clusters" and separate data according to the ground truth. Each cluster will be labeled with the majority class of its samples, and the accuracy of the protocol is defined as the fraction of input points that are clustered into their correct class relative to the ground truth. We employ this measure of accuracy to evaluate approximate clustering variants (PCure0, PCure1, and PCure2). Our standard privacy-preserving clustering protocol PHC and the optimized version OPT maintain the same accuracy as the original non-private protocol, hence we do not report accuracy for them.

We generate synthetic d-dimensional datasets of sizes up to 1M records and $d \in [1, 20]$, using a Gaussian mixture distribution, as follows: (1) The number of clusters is randomly chosen in $[8 : 15]$; (2) Each cluster center is randomly chosen in $[-50, 50]^d$ (performance is dominated by κ but not exact data values), subject to a minimum-separation distance between pairs; (3) Cluster standard deviation is randomly chosen in $[0.5, 4]$; and (4) Outliers are selected uniformly at random in the same interval and assigned randomly to clusters to emulate 3 noise percentage scenarios: low 0.1%, medium 1%, and high 5%. We randomly split each dataset into two halves which form the private inputs of the parties. We set the number of target clusters to $\ell_t = 5$; as our protocol incurs costs linear in the number of iterations ($n - \ell_t$), this choice comprises a worst-case setting, as in practice more than 5 target clusters are desired.

We adapted our protocols to support floating point numbers. Here, due to the simplicity of the involved operations, we can rely on fixed-precision floating point numbers and it suffices to multiply floating point values by a constant K (e.g. $K = 2^{20}$ for IEEE

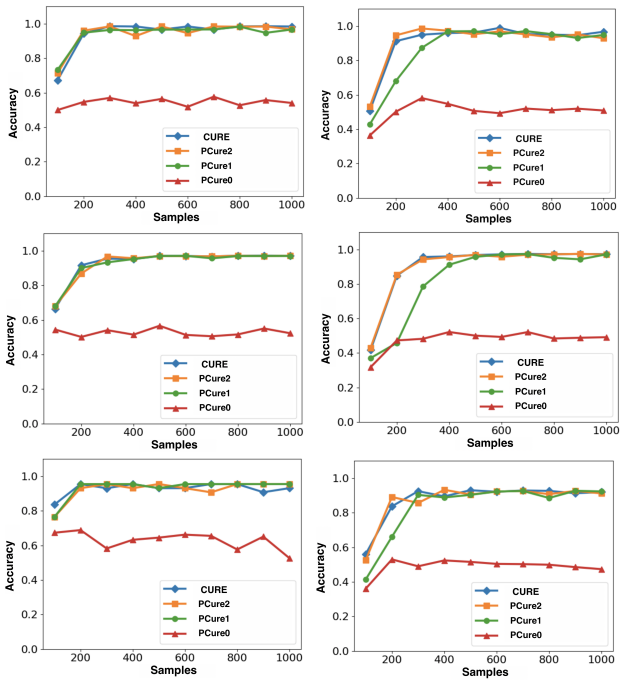


Figure 7: Accuracy of CURE, PCure∗**:** $p = 1$ **(left),** $p = 5$ **(right), #outliers** = 0.1% **(top),** 1% **(middle),** 5% **(bottom).**

754 doubles). During PHC.Setup, we can achieve higher precision. After each party decrypts the blinded values (line 22 and line 29), they can re-scale by dividing the constant K without affecting precision. During Cluster, as we only merge the points based on the comparisons between the distances, multiplying by a constant does not affect the results.

Finally, we use the ABY C++ framework [28], 128-bit AES for GC, 1024-bits Pailler, and set $\kappa = 40$. We use `libpaillier` [3] for Paillier encryption. We run our experiments on a 24-core machine, running Scientific Linux with 128GB memory on 2.9GHz Intel Xeon.

Protocol PHC**.** We first report results on the performance of our PHC protocol from Section 4. Figure 6a shows the computational cost for synthetic datasets of various sizes and dimensions, averaged over single and complete linkages. First, consistently with our analysis in Section 5, dimensions have minimal impact, since PHC's performance relates primarily to computing inter-cluster distances that is minimally affected by d. As expected its cubic asymptotic complexity, the overhead increases steeply with dataset size n.

Protocol OPT**.** Figure 6b shows the computational costs on synthetic datasets for our optimized single-linkage variant OPT (with configurations identical to those for PHC). In line with our analysis in Section 5, OPT significantly improves performance, reducing running time by an order of magnitude. E.g., for datasets of 2000 20-dimensional points, the running time is approximately 230 secs, an 8× speedup compared to PHC. The difference in our above example,is explained by the following observations: (1) although OPT improves performance during clustering by a linear factor, it adds costs during setup; and (2) the involved constants of the quadratic costs are higher for running time in setup phase, and vice versa in clustering phase. As shown in Figure 6d, OPT significantly improves performance over PHC, also when tested over our real datasets.

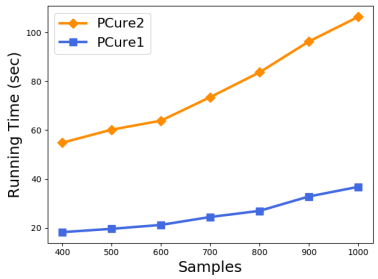


Figure 8: End-to-end computation of PCure1, PCure2**.**

Protocols PCure∗**.** Figure 7 shows the accuracy of our CURE-variant protocols PCure0, PCure1, PCure2 from Section 6, and the non-private CURE algorithm on synthetic datasets of 1M records for 10^2–10^3 samples, partition parameters $p = 1$ and $p = 5$, and for low (0.1%), medium (1%), and high (5%) outliers-to-data percentages. Clearly, PCure0, where parties run CURE on their own samples, without any interaction besides announcing representatives for individually computed clusters, exhibits very poor accuracy. e.g., 44.4% loss for 1M records. For $p = 1$, PCure1 and PCure2 achieve similar accuracy, which approaches that of CURE for large enough samples: At 300 samples or higher, the gap is within 3%. For higher values of p, e.g., $p = 5$, PCure1 and PCure2 exhibit a difference in accuracy: E.g., at 200 samples the accuracy for PCure1 is lower by 39.54% than PCure2; but at 500 samples or more, they are within 3.18%.

Moreover, experimenting with all combinations of $p = 1, 3, 5$ partitions and $R = 1, 3, 5, 7, 10$ representatives shows that the accuracies of PCure2 and PCure1 are very close to CURE at $s = 1000$ samples (or more). The largest observed difference between PCure1 and CURE is 3.57%, and between PCure2 and CURE is 2.7%. For $p = 1$ and $R = 1$ either difference is less than 1% at 1000 samples (or more). Thus, our choice of $p = 1$ and $R = 1$ to protect data privacy, as argued in Section 6, does not impact the protocol's accuracy.

We also compare end-to-end computation for PCure1 and PCure2 (with OPT), $n = 10^6$, $d = 10$, 1% outliers, no sample partitioning ($p = 1$), $\ell_t = 5$ target clusters, and $q = 3$ for A-Clustering. Figure 8 shows their good performance for sample sizes $s \in [400 : 1000]$. For 10^3 samples, PCure2 runs in 104sec, while PCure1 runs in 35sec – 3× faster, but with similar accuracy 97.09%.

Network Latency Impact. Although our experiments show the efficiency of our schemes, if executed over WAN this would be affected by network latency and data transmission. To estimate this impact, we considered two AWS machines in us-east and us-west and measured their latency to 50 – 60ms. Taking PCure2 with 400 samples and $\ell_t = 5$ as our use case, a single clustering round with four roundtrips (assuming distance update is done with a single garbled circuit) would take approximately 200-240ms. Regarding data transmission of the two garbled circuits for finding the minimum distance and updating the cluster distances, using the circuits for addition/subtraction, comparison, and min-index-selection from [65] for 100, 64-bit values, we estimate their size as roughly 10MB (not including the OT data which is dominated by the circuits). Under the mild assumption of a 100Mbps connection, transmission would take ∼100ms for a total of $<$ 350ms. In subsequent rounds, the circuits become progressively smaller but the number of roundtrips remains the same; conservatively multiplying by 395 rounds, we

have approximately 128sec of total communication time. For comparison, in Figure 8, for the same setting computation takes ~55sec.

Hence, communication indeed becomes a bottleneck for our schemes when run over WAN, but not to the point where they are entirely impractical. Furthermore, our goal when implementing our schemes was not to minimize end-to-end latency but computation, so there is plenty of room for optimizations. E.g., our protocols can be run in "round batches" merging k clusters with one interaction (by larger circuits) which would decrease RTTs by a factor of k. Finally, dedicated cloud technologies, such as AWS VPC [4], can offer private connections drastically reducing communication time.

8 RELATED WORK

Secure machine learning. There exists a rapidly growing line of works that propose secure protocols for a variety of ML tasks. This includes constructions for private classification models in the supervised learning setting (such as decision trees [70], SVM classification [108], linear regression [31, 32, 94], logistic regression [38] and neural networks [12, 85, 93]), as well as federated learning tasks [15]. Another focus has been on proposing MPC-based protocols that are provably secure under a well-defined real/ideal definition, similar to ours (e.g., [9, 16, 20, 22, 40, 42, 43, 51, 61, 67, 72, 77, 81, 90, 92]), for numerous tasks with a focus on neural networks and deep learning.

The above works can be split into two categories: those that focus on private model training and those that focus on private inference/classification. In our unsupervised setting, our protocol protects the privacy of the parties' data during the clustering phase.

Deployed techniques. In terms of techniques, most works use (some variant of) homomorphic encryption (e.g., [41, 86]). More advanced ML tasks often require hybrid techniques, e.g., combining the above with garbled circuits (e.g., [92]) or other MPC techniques [75, 90]. Our construction adopts such "mixed" techniques for the problem of hierarchical clustering. More recently, solutions have been proposed based on trusted hardware (such as Intel SGX), e.g., [21, 82, 105]. This avoids the need for "heavy" cryptography, however, it weakens the threat model as it requires trusting the hardware vendor. Finally, a different approach seeks privacy via data perturbation [5, 23, 24, 83, 97, 99], by adding statistical noise to hide data values, e.g., differential privacy [33]. Such techniques are orthogonal to the cryptographic methods that we apply here but they can potentially be combined (e.g., as in [87]). Using noise to hide whether a specific point has been included in a given cluster would be complement very nicely our cluster-information-reduction approach, potentially leading to more robust security treatment.

Privacy-preserving clustering. Many previous works proposed private solutions for different clustering tasks with the majority focusing on the popular, but conceptually simpler, k-means problem (e.g., [18, 30, 35, 58–60, 64, 76, 89, 107]) and other partitioning-based clustering methods (e.g., [62, 116]). Fewer other works consider private density-based [17, 25, 115] or distribution-based [48] clustering. An in-depth literature survey and comparative analysis of private clustering schemes can be found in the recent work of [49].

Focusing on private hierarchical clustering, no previous work offers a formal security definition, relying instead on ad-hoc analysis [27, 55–57, 96]. Moreover some schemes leak information to the participants that can clearly be harfmul—and is much more than what our protocol reveals—e.g., [95, 106] reveal all distances across parties' records. One notable exception is the scheme of Su et al. [102] which, however, is designed specifically for the case of document clustering. Here, we proposed a security formulation within the widely studied read/ideal paradigm of MPC that characterizes precisely what information is revealed to the collaborating parties. Besides making it easier to compare our solution with potential future ones that follow our formulation, this is, to the best of our knowledge the only private hierarchical clustering scheme with formal proofs of security. Finally, it is an interesting problem to combine optimizations for "plaintext" clustering (e.g.,[26, 78, 84]) with privacy-preserving techniques to improve efficiency.

Secure approximate computation. The interplay between cryptography and efficient approximation [37] has already been studied for pattern matching in genomic data [10, 109], k-means [101], and logistic regression [103, 111]. To the best of our knowledge, ours is the first work to compose secure cryptographic protocols with efficient approximation algorithms for hierarchical clustering.

Leakage in machine learning. The significant impact of information leakage in collaborative, distributed, or federated learning has been the topic of a long line of research (e.g., see [6, 66, 71, 97]). Various practical attacks have been demonstrated that infer information about the training data or the ML model and its hyper-parameters, (e.g., [39, 54, 98]). This is even more important in collaborative learning where parties could otherwise benefit from sharing data but such leakage may stop them (e.g., [74, 112, 118, 119]). Hence, it is crucial for our protocol to formally characterize what is the shared information for the two parties.

9 CONCLUSION

We propose the first formal security definition for private hierarchical clustering and design a protocol for single and complete linkage, as well as an optimized version. We also combine this with approximate clustering to increase scalability. We hope this work motivates further research in privacy-preserving unsupervised learning, including secure protocols for other linkage types (e.g., Ward), alternative approximation frameworks (e.g., BIRCH [117]), different tasks (e.g., mixture models, association rules or graph learning), or schemes for more than two parties to benefit from larger-scale collaborations. Specific to our definition of privacy, we believe it would be helpful to experimentally and empirically evaluate the impact (even our significantly redacted) dendrogram leakage can have, e.g., by demonstrating possible leakage-abuse attacks.

ACKNOWLEDGMENTS

The authors would like to thank the members of the AWS Crypto team for their useful comments and inputs, the anonymous reviewers for their valuable feedback, and Anrin Chakraborti for shepherding this paper. Dimitrios Papadopoulos was supported by the Hong Kong Research Grants Council (RGC) under grant ECS-26208318. Alina Oprea and Nikos Triandopoulos were supported by the National Science Foundation (NSF) under grants CNS-171763 and CNS-1718782.

REFERENCES

[1] 2017. The Intelligent Trial: AI Comes To Clinical Trials. Clinical Informatics News. http://www.clinicalinformaticsnews.com/2017/09/29/the-intelligent-trial-ai-comes-to-clinical-trials.aspx.

[2] 2019. The UCI Machine Learning Data Repository. http://archive.ics.uci.edu/ml/index.php.

[3] 2019. UTexas Paillier Library. http://acsc.cs.utexas.edu/libpaillier.

[4] 2021. AWS VPC. https://aws.amazon.com/vpc.

[5] 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.

[6] Mohammad Al-Rubaie and J. Morris Chang. 2019. Privacy-Preserving Machine Learning: Threats and Solutions. *IEEE Secur. Priv.* 17, 2 (2019), 49–58. https://doi.org/10.1109/MSEC.2018.2888775

[7] AlienVault. 2020. Open Threat Exchange. Available at https://otx.alienvault.com/.

[8] Cyber Threat Alliance. 2020. Available at http://cyberthreatalliance.org/.

[9] 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.

[10] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. 2018. Privacy-Preserving Search of Similar Patients in Genomic Data. *PoPETs* 2018, 4 (2018), 104–124. https://doi.org/10.1515/popets-2018-0034

[11] 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.

[12] 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

[13] 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)*.

[14] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *ACM CCS 2012*. 784–796. https://doi.org/10.1145/2382196.2382279

[15] 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

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

[17] Beyza Bozdemir, Sébastien Canard, Orhan Ermis, Helen Möllering, Melek Önen, and Thomas Schneider. 2021. Privacy-preserving Density-based Clustering. In *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*. ACM, 658–671. https://doi.org/10.1145/3433210.3453104

[18] 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

[19] 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)*.

[20] Hervé 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.

[21] Javad Ghareh Chamani and Dimitrios Papadopoulos. 2020. Mitigating Leakage in Federated Learning with Trusted Hardware. *CoRR* abs/2011.04948 (2020). arXiv:2011.04948 https://arxiv.org/abs/2011.04948

[22] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. [n.d.]. EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019*. 496–511. https://doi.org/10.1109/EuroSP.2019.00043

[23] 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

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

[25] Jung Hee Cheon, Duhyeong Kim, and Jai Hyun Park. 2019. Towards a Practical Cluster Analysis over Encrypted Data. In *Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11959)*. Springer, 227–249. https://doi.org/10.1007/978-3-030-38471-5_10

[26] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. [n.d.]. Hierarchical Clustering: Objective Functions and Algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, Artur Czumaj (Ed.). 378–397. https://doi.org/10.1137/1.9781611975031.26

[27] Ipsa De and Animesh Tripathy. 2014. A Secure Two Party Hierarchical Clustering Approach for Vertically Partitioned Data Set with Accuracy Measure. In *Recent Advances in Intelligent Informatics*. Springer International Publishing, 153–162.

[28] 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)*.

[29] Ben Dickson. 2016. How threat intelligence sharing can help deal with cybersecurity challenges. Available at https://techcrunch.com/2016/05/15/how-threat-intelligence-sharing-can-help-deal-with-cybersecurity-challenges/.

[30] 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.

[31] Wenliang Du and Mikhail J. Atallah. 2001. Privacy-Preserving Cooperative Scientific Computations. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001)*. 273–294.

[32] 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*. 222–233.

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

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

[35] 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

[36] Facebook. 2018. Threat Exchange. Available at https://developers.facebook.com/products/threat-exchange.

[37] 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

[38] 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.

[39] 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

[40] 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

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

[42] 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)*.

[43] Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, Peter Rindal, and Mike Rosulek. [n.d.]. Secure Data Exchange: A Marketplace in the Cloud. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019*. 117–128. https://doi.org/10.1145/3338466.3358924

[44] 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

[45] 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

[46] 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*.

[47] 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

[48] Mona Hamidi, Mina Sheikhalishahi, and Fabio Martinelli. 2018. Privacy Preserving Expectation Maximization (EM) Clustering Construction. In *DCAI 2018 (Advances in Intelligent Systems and Computing, Vol. 800)*. Springer, 255–263. https://doi.org/10.1007/978-3-319-94649-8_31

[49] Aditya Hegde, Helen Möllering, Thomas Schneider, and Hossein Yalame. 2021. SoK: Efficient Privacy-preserving Clustering. *Proc. Priv. Enhancing Technol.* 2021, 4 (2021), 225–248. https://doi.org/10.2478/popets-2021-0068

[50] W. Henecka, S. Kögl, 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)*.

[51] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. [n.d.]. Deep Neural Networks Classification over Encrypted Data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019*. 97–108. https://doi.org/10.1145/3292006.3300044

[52] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Catherine Jones. 2017. Privacy-preserving Machine Learning in Cloud. In *Proceedings of the 9th Cloud Computing Security Workshop, CCSW@CCS 2017, Dallas, TX, USA, November 3, 2017*, Bhavani M. Thuraisingham, Ghassan Karame, and Angelos Stavrou (Eds.). ACM, 39–43.

[53] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. 2018. Privacy-preserving Machine Learning as a Service. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 123–142.

[54] 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.

[55] 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

[56] Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N. Wright. 2006. A New Privacy-Preserving Distributed k-Clustering Algorithm. In *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*. SIAM, 494–498. https://doi.org/10.1137/1.9781611972764.47

[57] 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

[58] 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

[59] Angela Jäschke and Frederik Armknecht. 2018. Unsupervised Machine Learning on Encrypted Data. In *Selected Areas in Cryptography - SAC 2018m Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11349)*. Springer, 453–478. https://doi.org/10.1007/978-3-030-10970-7_21

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

[61] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. [n.d.]. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium, USENIX Security 2018*. 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[62] Hannah Keller, Helen Möllering, Thomas Schneider, and Hossein Yalame. 2021. Balancing Quality and Efficiency in Private Clustering with Affinity Propagation. In *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*. SCITEPRESS, 173–184. https://doi.org/10.5220/0010547801730184

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

[64] Hyeong-Jin Kim and Jae-Woo Chang. 2018. A Privacy-Preserving k-Means Clustering Algorithm Using Secure Comparison Protocol and Density-Based Center Point Selection. In *11th IEEE International Conference on Cloud Computing, CLOUD 2018*. IEEE Computer Society, 928–931. https://doi.org/10.1109/CLOUD.2018.00138

[65] 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.

[66] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* 37, 3 (2020), 50–60. https://doi.org/10.1109/MSP.2020.2975749

[67] 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

[68] 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. *BMC Nephrology* 17 (2016). Issue 25.

[69] 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

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

[71] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. 2021. When Machine Learning Meets Privacy: A Survey and Outlook. *ACM Comput. Surv.* 54, 2, Article 31 (March 2021), 36 pages. https://doi.org/10.1145/3436755

[72] 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

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

[74] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. [n.d.]. Exploiting Unintended Feature Leakage in Collaborative Learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019*. 691–706. https://doi.org/10.1109/SP.2019.00029

[75] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. [n.d.]. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium, USENIX Security 2020*. 2505–2522. https://www.usenix.org/conference/usenixsecurity20/presentation/mishra

[76] Payman Mohassel, Mike Rosulek, and Ni Trieu. 2020. Practical Privacy-Preserving K-means Clustering. *Proc. Priv. Enhancing Technol.* 2020, 4 (2020), 414–433. https://doi.org/10.2478/popets-2020-0080

[77] 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

[78] Fionn Murtagh and Pedro Contreras. 2017. Algorithms for hierarchical clustering: an overview, II. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 7, 6 (2017). https://doi.org/10.1002/widm.1219

[79] 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*.

[80] Sophia R. Newcomer, John F. Steiner, , and Elizabeth A. Bayliss. 2011. Identifying Subgroups of Complex Patients With Cluster Analysis. *American Journal of Managed Care* 17 (2011), 324–332. Issue 8.

[81] 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.

[82] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. [n.d.]. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium, USENIX Security 16*. 619–636.

[83] Stanley R. M. Oliveira and Osmar R. Zaïane. 2003. Privacy Preserving Clustering by Data Transformation. In *XVIII Simpósio Brasileiro de Bancos de Dados, Anais/Proceedings*. 304–318.

[84] Clark F. Olson. 1995. Parallel Algorithms for Hierarchical Clustering. *Parallel Comput.* 21, 8 (1995), 1313–1325. https://doi.org/10.1016/0167-8191(95)00017-I

[85] 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

[86] P. Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. Advances in Cryptology - EUROCRYPT*. Springer-Verlag.

[87] Martin Pettai and Peeter Laud. 2015. Combining Differential Privacy and Secure Multiparty Computation. In *Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA, December 7-11, 2015*. ACM, 421–430.

[88] Michael O. Rabin. 1981. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University.

[89] Fang-Yu Rao, Bharath K. Samanthula, Elisa Bertino, Xun Yi, and Dongxi Liu. 2015. Privacy-Preserving and Outsourced Multi-user K-Means Clustering. In *IEEE Conference on Collaboration and Internet Computing, CIC 2015, Hangzhou, China, October 27-30, 2015*. IEEE Computer Society, 80–89. https://doi.org/10.1109/CIC.2015.20

[90] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. [n.d.]. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *AsiaCCS 2018*. 707–721. https://doi.org/10.1145/3196494.3196522

[91] 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.

[92] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: scalable provably-secure deep learning. In *DAC 2018*. ACM, 2:1–2:6. https://doi.org/10.1145/3195970.3196023

[93] 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

[94] 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.

[95] Mina Sheikhalishahi, Mona Hamidi, and Fabio Martinelli. [n.d.]. Privacy Preserving Collaborative Agglomerative Hierarchical Clustering Construction. In *Information Systems Security and Privacy - 4th International Conference, ICISSP 2018*, Vol. 977. 261–280. https://doi.org/10.1007/978-3-030-25109-3_14

[96] 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

[97] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In *ACM SIGSAC CCS 2015*. 1310–1321.
[98] 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.
[99] 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
[100] 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
[101] 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.
[102] 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
[103] 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
[104] Harry Chandra Tanuwidjaja, Rakyong Choi, Seunggeun Baek, and Kwangjo Kim. 2020. Privacy-Preserving Deep Learning on Machine Learning as a Service - a Comprehensive Survey. *IEEE Access* 8 (2020), 167425–167447.
[105] Florian Tramèr and Dan Boneh. [n.d.]. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *7th International Conference on Learning Representations, ICLR 2019*. https://openreview.net/forum?id=rJVorjCcKQ
[106] A. Tripathy and I. De. 2013. Privacy Preserving Two-Party Hierarchical Clustering Over Vertically Partitioned Dataset. *Journal of Software Engineering and Applications* 06 (2013), 26–31.
[107] Jaideep Vaidya and Chris Clifton. 2003. Privacy-preserving *k*-means clustering over vertically partitioned data. In *ACM SIGKDD 2003*. 206–215.
[108] 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
[109] 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
[110] 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. *Archives of Internal Medicine* 160 (2000), 481–490. Issue 4.
[111] 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
[112] Hongyang Yan, Li Hu, Xiaoyu Xiang, Zheli Liu, and Xu Yuan. 2021. PPCL: Privacy-preserving collaborative learning for mitigating indirect information leakage. *Inf. Sci.* 548 (2021), 423–437. https://doi.org/10.1016/j.ins.2020.09.064
[113] 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
[114] 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
[115] Samee Zahur and David Evans. 2013. Circuit Structures for Improving Efficiency of Security and Privacy Tools. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 493–507. https://doi.org/10.1109/SP.2013.40
[116] Qingchen Zhang, Laurence T. Yang, Zhikui Chen, and Peng Li. 2017. PPHOPCM: Privacy-preserving High-order Possibilistic c-Means Algorithm for Big Data Clustering with Cloud Computing. *IEEE Transactions on Big Data* (2017), 1–1. https://doi.org/10.1109/TBDATA.2017.2701816
[117] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *ACM SIGMOD 1996*. 103–114.
[118] Lingchen Zhao, Qian Wang, Qin Zou, Yan Zhang, and Yanjiao Chen. 2020. Privacy-Preserving Collaborative Deep Learning With Unreliable Participants. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 1486–1500. https://doi.org/10.1109/TIFS.2019.2939713
[119] Qi Zhao, Chuan Zhao, Shujie Cui, Shan Jing, and Zhenxiang Chen. 2020. PrivateDL PrivateDL : Privacy-preserving collaborative deep learning against leakage from gradient sharing. *Int. J. Intell. Syst.* 35, 8 (2020), 1262–1279. https://doi.org/10.1002/int.22241
[120] Jan Henrik Ziegeldorf, Jens Hiller, Martin Henze, Hanno Wirtz, and Klaus Wehrle. [n.d.]. Bandwidth-Optimized Secure Two-Party Computation of Minima. In *CANS 2015*. 197–213.

A GARBLED CIRCUITS

Garbled circuits (GC) [113, 114] provide a general framework for securely realizing two-party computation of any functionality. The framework has been thoroughly studied in the literature (e.g., see formal treatments of the topic [14, 69]) and we here overview the specific procedures involved in it.

In our running example, parties P_1 and P_2 wish to evaluate a specific function f over their respective inputs x_1, x_2 and engage in an interactive 2-phase protocol, where one party plays the role of the *garbler* and the other the role of the *evaluator*. Without loss of generality, P_1 is the garbler and P_2 is the evaluator, and their interaction proceeds as follows.

In phase I, P_1 expresses f as a Boolean circuit C_f, i.e., as a directed acyclic graph of Boolean AND and OR gates, and then sends a "garbled," i.e., encrypted, version of C_f to P_2.

In our example, C_f corresponds to a circuit of two AND gates A, B and an OR gate C, shown in Figure 9: Inputs x_1, x_2 are 11 and 01, and output $f(x_1, x_2)$ is 1, computed by feeding to the OR gate the two bitwise ANDs of the inputs.

To garble C_f, P_1 first maps (the two possible bits 0, 1 of) each wire X in C_f to two random values w_X^0, w_X^1 (from a large domain, e.g., $\{0,1\}^{128}$), called the *garbled values* of X.

Specifically, P_1 maps the output wires of gates A, B, and C to random garbled values $\{w_A^0, w_A^1\}, \{w_B^0, w_B^1\}$ and respectively $\{w_C^0, w_C^1\}$, and also maps the two input wires of gate A (respectively, gate B) to random garbled values $\{w_{11}^0, w_{11}^1\}, \{w_{21}^0, w_{21}^1\}$ (respectively, $\{w_{12}^0, w_{12}^1\}, \{w_{22}^0, w_{22}^1\}$), where mnemonically the i-th input bit of party P_j corresponds to the ij-wire, $i, j \in \{1, 2\}$.

Next, P_1 sends to P_2 the *garbled truth table* of every Boolean gate in C_f, which is the *permuted* encrypted truth table of the gate, where row in the truth table is appropriately encrypted using the garbled values of its three associated wires. We only specify the garbled truth table of the AND gate A, as other gates can be handled similarly. The row $(1, 1) \to 1$ in the truth table of A dictates that the

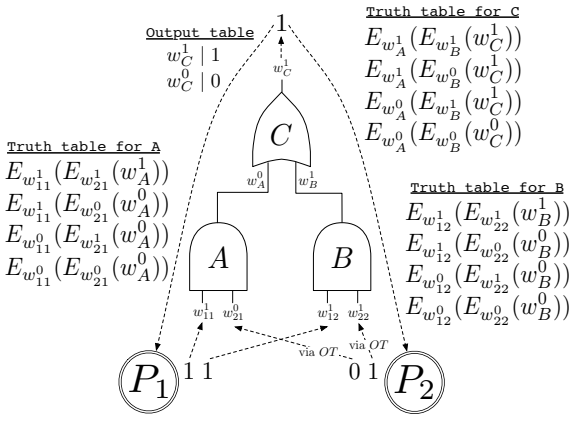


Figure 9: Garbled circuit C_f of a specific function f that computes the OR over the pairwise ANDs of the 2-bit inputs.

output is 1 when input is 1, 1 or, using garbled values, that the output is w_A^1 when input is w_{11}^1, w_{21}^1. Accordingly, using a semantically-secure symmetric encryption scheme $E_k(\cdot)$ (e.g., 128-bit AES), P_1 can express this condition as ciphertext $E_{w_{11}^1}(E_{w_{21}^1}(w_A^1))$, where the output w_A^1 is *successively* encrypted using the inputs w_{11}^1, w_{21}^1 as encryption keys. P_1 produces a similar ciphetext for each other row in the truth table of A 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, and that if one possesses *only* w_{11}^1, w_{21}^1, all other entries of the garbled truth table of A (besides w_A^1) are indistinguishable from random, due to the semantic security of the encryption scheme $E_k(\cdot)$.

Finally, to allow P_2 to retrieve the final output $f(x_1, x_2)$, P_1 also sends the garbled values w_C^0, w_C^1 of the output wire *together* with their corresponding mapping to 0 and 1. Note that P_2 is no privy to any other mappings between wires' garbled values and their possible bit values.

In phase II, P_2 evaluates the entire circuit C_f over the received garbled truth tables of the gates in it, by evaluating gates one by one in the ordering hierarchy induced by (the DAG structure of) C_f. Indeed, if P_2 knows the w value of each input wire of a gate and its garbled truth table, P_2 can easily discover its output value, by attempting to decrypt all rows in the table and accepting only the one that returns a correct output value. For example, if P_2 has w_{11}^1, w_{21}^0, P_2 can try to decrypt every value in the garbled truth table of A, until P_2 finds the correct value w_A^1.[4]

To initiate this circuit-evaluation process, P_2 needs to learn the garbled values of each of the input wires in C_f, which is achieved as follows: (1) P_1 sends to P_2 the w values w_{11}^1, w_{12}^1 corresponding to the input wires of P_1 in the clear (note that since these are random values, P_2 cannot map them to 0 or 1, thus P_1's input is protected); (2) P_2 privately query from P_1 the w values w_{21}^0, w_{22}^1 corresponding to the input wires of P_2, that is, without P_1 learning which garbled values were queried, via a two-party secure computation protocol called 1-out-of-2 *oblivious transfer* (OT) [88]. At a very high level, and focusing on a single input bit, OT allows P_2 to retrieve from P_1 exactly one value in pair (w_{21}^0, w_{21}^1) without P_1 learning which value was retrieved. After running the OT protocol for every input bit, P_2 can evaluate C_f, as above, to finally compute and send back to P_1 the correct output $f(x_1, x_2) = 1$, deduced by the final garbled value w_C^1 of the output wire.

B SECURE min-SELECTION PROTOCOLS

Here, we overview the design of GC-based protocols ArgMin and MaxDist/MinDist for secure selection of min/max values, or their index/location, over secret-shared data. These protocols have been defined in Sections 4 and 5 and comprise integral components of our solutions. We provide the exact two circuits over which we can directly apply the garbled-circuits framework (see Appendix A) to get GC-protocols ArgMin and MinDist, noting that the circuit in support of MaxDist is similar to the case of MinDist.

[4]For this, 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 [69].

Recall that data consists of λ-bit values and is secret shared among the two parties as κ-bit random blinding terms, $\kappa > \lambda$, and $\kappa+1$-bit blinded values, each resulted by adding a random blinding term to an ordinary data value.

Our circuits use as building blocks the following gates, efficient implementations of which are well studied [65]:

- ADD/SUB that adds/subtracts $\kappa+1$-bit integers;
- MIN/MAX that selects the min/max of two λ-bit integers, using a one-bit output to encode which input value is the min/max value (e.g., on input 3, 5 MIN outputs 0 to indicate the first value is smaller);
- a multiplexer gate MUX_i that on input two i-bit inputs and a selector bit s, outputs the first or the second one, depending on the value of s; and finally
- hard-coded in the circuit constant gates CON_i, $1 \le i \le n$, that always output the $(\log n)$-bit fixed value i (e.g., CON_3 outputs the binary representation of 3).

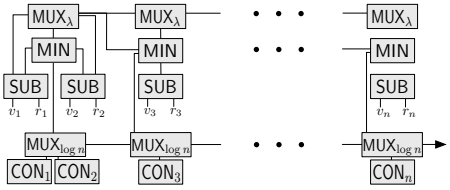


Figure 10: The circuit for protocol ArgMin**.**

Figure 10 shows the circuit of protocol ArgMin for selecting the index of the minimum value in an array of n different values. On input n $\kappa+1$-bit values $v_1, \ldots, v_n$ and n κ-bit blinding terms $r_1, \ldots, r_n$, the circuit first uses n SUB gates to compute (the secret) values $v_i - r_i$, $i = 1, \ldots, n$, and then selects the index of the minimum such value in $n-1$ successive comparisons as follows. In the ith comparison, a MIN gate compares the currently minimum value m_i of index loc_i (initially, $m_1 = v_1 - r_1$, $loc_1 = 1$) to value $v_{i+1} - r_{i+1}$ of index $i+1$, and its output bit is fed, as the selector bit, to two multiplexer gates MUX_μ:

- $\mu = \log n$: once for selecting among two $(\log n)$-bit indices loc_i and $i+1$, the latter conveniently encoded as the output of constant gate CON_{i+1} (such hard-coded indices significantly facilitate their propagation in the circuit, compared to the alternative of handling indexes as input and carrying them over throughout the circuit); and
- $\mu = \lambda$: once for selecting among two λ-bit values m_i and $v_{i+1} - r_{i+1}$,

overall propagating the updated minimum value $m_{i+1} = \min\{a, b\}$ and its index loc_{i+1} to the next $(i+1)$th comparison. The final output (see arrow wire) corresponds to the output of the $(n-1)$th index-selection multiplexer gate.

Figure 11 shows the circuit for protocol MinDist for selecting and re-blinding the minimum value among two secret-shared values. On input two $\kappa+1$-bit blinded values u, v and three κ-bit blinding terms r_1, r_2, r', the circuit first computes $u - r_1$, $v - r_2$ using two SUB gates, then computes the minimum of these two values using a MIN gate, and its output bit is fed, as the selector bit, to a multiplexer gate MUX_λ for selecting the minimum among two λ-bit values $u - r_1$ and $v - r_2$, which is becomes the final output (see arrow wire)

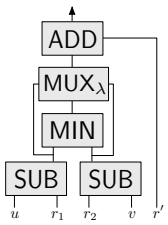


Figure 11: The circuit for protocol MinDist**.**

after it is blinded by adding the input blinding term r' through a ADD gate. (The circuit for protocol MaxDist is the same with a MAX gate replacing the MIN gate.)

C PROOF OF THEOREM 5.1

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 clustering round) are "blinded" by large random factors, these values can be perfectly simulated, as needed in our proof, by randomly selected values. 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 handle the two cases of party corruption separately.

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

(1) Encrypted matrices B, R and encrypted arrays L, S.
(2) For each clustering round ℓ, messages received during the oblivious transfer execution for ArgMin, denoted by OT_ℓ and the min/max index α_ℓ.
(3) During each clustering round ℓ, for each execution of MinDist/MaxDist 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 protocol PHC 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 MinDist/MaxDist at clustering 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.[5] 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** Argmin**)** For each clustering 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 ArgMin 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 ArgMin for index k during clustering 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 (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 PHC. This is done via the following sequence of hybrids.

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

Hybrid 1. This is the same as Hybrid 0, but the output of GC_ℓ in $\mathsf{view}^{\mathsf{PHC}}_{\mathcal{A}_{P_2}}$ 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 B, 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 $\mathsf{view}^{\mathsf{PHC}}_{\mathcal{A}_{P_2}}$, each of the values in B, L are computed as the sum of a random value from $\{0,1\}^\kappa$ and a distance between two clusters.

[5] And corresponding randomness derived from R_2.

Hybrid 3. This is the same as Hybrid 2, but all values in 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 D, R and encrypted arrays H, L, S.
(2) For each clustering round ℓ, a garbled circuit GC_ℓ for evaluating ArgMin, messages received during the corresponding oblivious transfer execution denoted by OT_ℓ.
(3) During each clustering round ℓ, for each execution of MinDist/MaxDist 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 D, H, L consisting of encryptions of zeros under pk'. Moreover, he computes 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 ArgMin for each clustering round 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 [69]).
- **(Oblivious transfer simulation for** ArgMin**)** Let $W_\ell^{(1)}$, $W_\ell^{(2)}$ 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 $\mathsf{Sim}_{OT}^{(1)}$ on input $(y_\ell, W_\ell^{(1)})$, 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** MinDist/MaxDist**)** 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 MinDist/MaxDist at clustering 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 $\mathsf{Sim}_{OT}^{(1)}$ 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 $\mathsf{Sim}_{OT}^{(1)}$ 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 PHC. This is done via the following sequence of hybrids.
Hybrid 0. This is the view $\mathsf{view}_{\mathcal{A}_{P_1}^{\mathsf{PHC}}}$, 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 D, H', 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 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 [69]).
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 PHC which concludes this part of the proof.