Private Hierarchical Clustering and Efficient Approximation

Xianrui Meng
xianru@amazon.com
Amazon Web Services

Alina Oprea
a.oprea@northeastern.edu
Northeastern University

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

Nikos Triandopoulos
ntriando@stevens.edu
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 35 sec 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

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 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 "hierarchical" 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
proven 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. 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, redirecting 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 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 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.
Hierarchical clustering (HC). For fixed positive integers \(d, l\), let \(D = \{v_i | v_i \in \mathbb{R}^d\}_{i=1}^n\) 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 \(x, y \in D\) of points, point distance is measured using the standard square Euclidean distance metric \(\text{dist}(x, y) = \sum_{j=1}^d(x_j - y_j)^2\). Over pairs \(X, Y \subseteq D\) 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 \(\delta(X, Y) = \min_{x \in X, y \in Y} \text{dist}(x, y)\), and the complete linkage (or diameter) defined as \(\delta(X, Y) = \max_{x \in X, y \in Y} \text{dist}(x, y)\).

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 \(D\) and groups its points into a total of \(\ell_1 \leq n\) 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 \(n - \ell_1 + 1\) levels, where siblings correspond to merged clusters and levels to dataset partitions, build level-by-level as follows:

- Initially, each input point \(v_i \in D\) forms a singleton cluster \(\{v_i\}\) as a leaf in \(T\) (at its lowest level \(n\)).
- Iteratively, in \(n - \ell_1\) clustering rounds, the \(i\) root clusters (at top level \(l\)) form \(i - 1\) new root clusters in \(T\) (at higher level \(l - 1\)), with the closest two merged into a union cluster as their parent, and each other cluster copied to level \(l - 1\) as its parent.

When a new level of \(\ell_1\) target clusters is reached, HCAlg halts and outputs \(T\). The exact value of \(\ell_1 \in [1 : n]\) is determined during execution via a predefined condition \(\text{End}\) 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 a threshold specified by \(t\) or simply when exactly \(\ell_1 = t\) 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 \(\text{size}(C) = |C|\) and representative value \(\text{rep}(C)\), 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 \(D = \{v_i\}_{i=1}^n\) and return an \(M\)-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 \(\ell_1\) 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 \(\lambda\), keys generated by running (\(pk, sk\)) \(\leftarrow\) Gen\((1^\lambda)\) and a public RSA modulus \(N\), the
We introduce a model for studying private hierarchical clustering. Thus, parties decrypting $\{m\}^k \mod N^2$ results in $km \mod N$, and the ciphertext product $[m] \cdot [m'] \mod N^2$ 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 $\mathbb{Z}_N^d$, and wish to perform a collaborative hierarchical clustering over the combined set $D = P \cup Q$. 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 $\Pi$ be a two-party protocol that correctly realizes $f_{HC}(\cdot, \cdot)$: Run jointly on inputs $x_1, x_2$. $\Pi$ 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 $\Pi$ on their inputs $P, Q$. As discussed, $\Pi$ 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 $f_{HC}^\star$, that computes and announces $f_{HC}^\star(x_1, x_2)$ (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 $f_{HC}^\star(x_1, x_2)$: It is the inherent price for collaboratively learning a non-trivial function.

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

Next comes the question of which ideal functionality $f_{HC}^\star$ should $\Pi$ securely realize for private joint hierarchical clustering? Though tempting, equating $f_{HC}^\star$ with the legacy algorithm HCAlg (Figure 1), thus learning a full-form augmented dendrogram, slides us into a degeneracy. Assume $HCAlg$ merely runs HCAlg on the combined indexed set $D = P \cup Q = \{d_k\}_{k=1}^n, n = |P| + |Q|$. 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 $D$; 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 $f_{HC}^\star$ 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 reduced, but still useful, learned model, balancing between accuracy (to benefit from clustering) and privacy (to allow collaboration). If allowing the ideal functionality $f_{HC}^\star$ 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 $f_{HC}^\star$, 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 exactly on its ability to discover such rich inter-cluster relations, we must keep the forest structure of $T$ in $f_{HC}^\star$’s output.

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 $f_{HC}(P, Q) = \{T, M(T)\}$ 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 $D$ 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.

---

1If $P, Q$ are indexed, then $D = Q \parallel P$, or else a fixed ordering is used.

3Cluster 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 [41].

---

Figure 2: Ideal functionality $f_{HC}^\star$ for hierarchical clustering.
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 \(f^{HC}_{\text{middle}}(P, Q) = \{T, M(T)\}\) into an optimized model \(f^{HC}_{\text{opt}}(P, Q) = \{T^*, M^*(T)\}\), 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 \(f^{HC}_{\text{opt}}(P, Q)\) 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 \(f^{HC}_{\text{opt}}\), 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 \(D^+\) that is a random permutation over the combined set \(D = P \cup Q = \{d_k\}_{k=1}^n\).
- **Post-process output:** Return the output \(T^*, C(T^*), M(T^*)\) of HCAlg redacted as \(T^*, M^*(T^*) \subset T^*\), including metadata of only a few safe clusters in \(T^*\).

Our ideal functionality \(f^{HC}_{\text{opt}}\) refines the ordinary dendrogram \(T, C(T), M(T)\): Running HCAlg on the randomly permuted input \(D^+\) (instead of \(D\)) 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 \(S(\cdot)\) (as a parameter agreed upon among the parties and hard-coded in \(f^{HC}_{\text{opt}}\)). Such metadata is safe to learn, in the sense that it does not directly leak any input points.

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 \(t_i\) 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 \(f^{HC}_{\text{middle}}(P, Q)\). 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 \(f^{HC}_{\text{opt}}\). 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 \(\Pi\), 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 \(f^{HC}_{\text{opt}}\), defined in Figure 2, by satisfying the following: For \(i = 1, 2\) and for any security parameter \(\lambda\), there exists a non-uniform probabilistic polynomial-time simulator \(\Pi_\lambda\) so that \(\Pi_\lambda((1^\lambda, x_i, f^{HC}_{\text{opt}}(x_i, x_{1-i}))) \equiv \text{view}_{\Pi_\lambda} \triangleq \{r_i, t_i\}\).

4 MAIN CONSTRUCTION

We now present our main construction, protocol PHC for Private Hierarchical Clustering that securely realizes the ideal functionality \(f^{HC}_{\text{opt}}\) (of Figure 2) when jointly run by parties \(P_1, P_2\).

**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 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.

1. Our protocol securely implements \(f^{HC}_{\text{opt}}\) for the configuration that the parties specify: linkage \(\delta(\cdot, \cdot)\), termination condition \(\text{End}(\cdot, t)\), cluster statistics set \(M\), selection function \(S(\cdot)\). 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.
Algorithm 1: PHC: Private Cluster Analysis

| 1  | P₁: Generate (pk, sk) ← Gen(1^λ); send pk to P₂ |
| 2  | P₂: Generate (pk', sk') ← Gen(1^λ); send pk' to P₁ |

P₁, P₂ jointly run PHC.Setup, PHC.Cluster, PHC.Output

### Parameters
- Default configurations
- Security and statistical parameters
- Termination parameter, \( n \) of target clusters
- Public and secret keys of parties P₁, P₂
- AHE-encrypted plaintext \( d \) under pk, pk'
- AHE-decrypted ciphertext \( c \) n x n matrices R, B stored by P₁, P₂
- P₁, P₂ run GC on \( I_1, I_2 \) to get \( O_1, O_2 \)
- Permutations contributed by P₁, P₂
- Square Euclidean distances of \( p, q \)
- Representative states of \( C_i \)

### Notation
- \( n \), \( n_1 \), \( n_2 \): Number of points, \( n_1 \), \( n_2 \)
- \( \lambda \), \( \kappa \): Security and statistical parameters
- \( t, t_1 \): Termination parameter, \( n \) of target clusters
- \( [d], [d] \): AHE-encrypted plaintext \( d \) under pk, pk'
- \( \langle \cdot \rangle \): AHE-decrypted ciphertext \( c \)
- \( \delta(C, C') \): Distance between clusters \( C \) and \( C' \)
- \( \delta(C_i, C_j) \): Distance between clusters \( C_i \) and \( C_j \)
- \( \text{Dist}(p, q) \): Euclidean distance of \( p, q \)
- \( \text{size}(\cdot) \): Size of clusters

### Output
- \( \text{PK setting} \)
- \( n = n_1 + n_2 \)
- \( \text{Output} \)
- \( \text{Cluster} \)
- \( \text{Representatives} \)
- \( \text{Sizes} \)

#### Output of sub-protocol PHC
- P₁, P₂ process their individual states \( R, B \) to iteratively merge singletons into target clusters, based on inter-cluster distances in \( \Delta \).
- Each iteration merges two clusters into a new one via three tasks:
  - **Find pair**: First, P₁, P₂ find the closest-cluster pair \( (i, j) = \arg\min(\Delta) \), \( i < j \), to merge, i.e., the indices in \( \Delta \) of the minimum inter-cluster distance \( D_{ij} \).
  - **Update linkages**: Then, P₁, P₂ update \( \Delta \) to \( \Delta' = \Delta - R' \) with the new cluster distances after pair \( (i, j) \) is merged into cluster \( C = C_i \cup C_j \).
  - **Record merging**: Finally, P₁, P₂ record \( \Delta' \) 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 \( (\Delta, \Delta') \) 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 \( T^{\Delta} \) 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 \( (\Delta, \Delta') \) 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 \( (\Delta, \Delta') \), namely to construct, shuffle and split \( (\Delta, \Delta') \) 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 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 P₁, P₂) by \( \Sigma \). Finally, we denote the joint execution by P₁, P₂ of a GC-based protocol GC, on private inputs \( I_1, I_2 \) to get private outputs \( O_1, O_2 \), by \( \Sigma \langle GC(I_1; I_2) \rangle \).

**Setup phase**: P₁, P₂ 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 $p_k$) 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 $p_k$) 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 $p_k$, lines 9, 14) and $S$, $L$, $R$ and $B$ are sent to $P_2$, after their elements are shuffled using a random permutation $\pi_1$ (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 $\pi_2$ (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 $\{ L, R \}$ into $\{ L, R \}$, $B$.

### Clustering phase

Once $P_1, P_2$ 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 encodes cluster information in the diagonal of its matrix state $\Sigma$; initially, the $i$-th entry stores $(i, L)$, denoting the (never-merged but already permuted) singleton of rank $i$. Hierarchical clustering runs in exactly $n - t = n - t$ iterations, or clustering rounds.

First, the start of each iteration, $P_1, P_2$ find which pair of clusters must be merged by jointly running the GC-protocol $(i, j; i, j) \leftarrow \text{ArgMin}(R; B)$ (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 $B_{ij} - R_{ij}$, 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 $D_{ij} = \min_{x,y} R_{xy}$, and returns to both parties the indices $i, j$. Next, once pair $(i, j)$ is known to $P_1, P_2$, they proceed to jointly update the linkages (lines 7-12). For each cluster $k$ in $X$, 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 $(i, j; X) \leftarrow \text{MaxDist}(R_{ik}, R_{jk}; X; B_{ik}, B_{jk})$ (line 10): The parties contribute the two entries from their individual matrices $R, B$ that are needed for comparing the linkages $B_{ik} - R_{ik}, B_{jk} - R_{jk}$ between cluster $k$ and clusters $i, j$, and $P_2$ learns the maximum value of the two but blindered by the random blinding term $X$ inputted by $P_1$. The garbled circuit for MaxDist simply returns to ($P_2$) the value $\max(B_{ik} - R_{ik}, B_{jk} - R_{jk}) + X$. Finally, at the end of iteration $t$, $P_1, P_2$ 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 $t$ 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].
 tween tocol exploits the associativity of operator
Optimized single-linkage protocol

between the output phase, the cost is

likewise, evaluating circuit

MaxDist

\( \nu \)

\( \ell \)

iteration

\( C \)

\( \nu n \)

Asymptotically, our protocol achieves optimal perfor-

5 PROTOCOL ANALYSIS

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 the 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 \( I_{ij} \) (over permuted input points
\( \pi_2 = \pi_1(P \cup Q) \)) of the points contained in \( C_i, \) to finally compute

\( \prod_{j \in I_{ij}} L_j = |I_{ij}| \). The resulted encrypted point averages and

cluster sizes are sent to \( P_2, \) who returns to \( P_1 \) the \( t \) plaintext point

averages, i.e., \( \sum_{j \in I_{ij}} P_j = \sum_{j \in I_{ij}} E_{i, j} \) 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 perform-
ance, as it incurs no extra overheads to the performance costs
associated with running HC (ignoring the dependency on the se-
curity parameter \( \lambda \)). The asymptotic overheads incurred on \( P_1, P_2, \)
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 \( X. \) In clustering phase, each of the \( n - \ell_t = O(n) \) total iter-
ations incurs costs proportional the complexity of running GC-based
protocols ArgMin, MaxDist, where the cost of garbling and evalu-
ating a circuit \( C, \) with a total number of wires \(|C|\), is \( O(|C|) \). Thus,

during the \( t \)-th iteration: Evaluating circuit ArgMin entails \( n^2 - 2t \)
comparisons of \( 2^t \)-bit values of (cluster distances) and subtractions
of \( k \)-bit values (of blinding terms), for a total size of \( O(\kappa n^2 - t) \); likewise, evaluating circuit MaxDist entails a constant number of
comparisons of \( k \)-bit values and \( O(n) \) such circuits are evaluated at
iteration \( t; \) thus, the total cost during this phase is \( O(\kappa n^2) \) 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 \( O(n^3) \). Communication
consists of \( O(n^2) \) ciphertexts during setup (encrypted distances),
\( O(n^2) \) 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 pro-
tocol exploits the associativity of operator over max to update the com-
plete linkage between newly formed clusters \( C \) and other clusters \( C_i \),
as the max of the linkages between \( C \)’s constituent clusters and \( C_i \),
securely realized via GC-protocol \((\cdot, \cdot) \leftarrow \text{MaxDist}(\cdot, \cdot)\). Single
linkages can be supported readily by updating inter-cluster distances
between \( C \) and \( C_i \) as the min of the distances between \( C \)'s constituent
clusters and \( C_i \). Line 10 in Algorithm 3 now has \( P_1, P_2 \) jointly run
GC-protocol \((\cdot, Y) \leftarrow \text{MinDist}(R_{i, k}, R_{j, k}, X; B_{i, j}, B_{j, k})\) (see Appen-
dix B) to split the new distance \( \Delta_{i,k} = \min(|B_{j,k} - R_{i,k}, B_{j,k} - R_{j,k}) \)
to \( X, Y = \Delta_{i,k} + X, \) without asymptotic efficiency changes.

More generally, the skeleton of protocol PHC allows for exten-
sions 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 run-
ing time, with GC-protocol \((j, j) \leftarrow \text{ArgMin}(X; Y)\) 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 \( \Delta = \hat{B} - \hat{R} \) storing the minimum row-wise distances
in \( \Delta = \hat{B} - \hat{R} \) (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, \( P_1, P_2 \) now also jointly
run \((j; j) \leftarrow \text{ArgMin}(R_{i}; B_{j}), i \in [1 : n], \) to learn the minimum-
linkage index \( j_i \) of the row \( R_i \) while \( \Delta = \hat{B} - \hat{R} \) (excluding its \( i \)th location,
as \( B_{i, i} \), \( R_{i, i} \) store cluster \( i \)’s), and they both initialize array \( J \) such
that \( P_1 \) initializes array \( R = R_{i} = R_{i, j}, \) and \( P_2 \) array \( B = B_{j} = B_{j, j} \).
Then, at the start of each iteration in the clustering phase (line 5 in Algorithm 3) and assuming that \( \Delta = \infty, P_1, P_2 \) now jointly run \((i, i) \leftarrow \text{ArgMin}(R, \hat{B}) \) to find the closest-cluster
pair \((i, j), \) \( j = J_i, \) in only \( O(\kappa n) \) time. Conveniently, as soon as
they update linkages \( \Delta_{i,k} = \Delta_{j,k}, \) for some \( k \neq i, j \) (lines 9-12), as
\( Y - X \) with \((\Delta, Y) \leftarrow \text{MinDist}(R_{i,j,k}, R_{j,k}, X; B_{i,j,k})\), \( P_1, P_2 \) also
update the joint state \((B, J) \) for updated row \( m \in [i, k] \); First,
by jointly running \((z, z) \leftarrow \text{ArgMin}(R, \hat{B}) \) for arrays \( \hat{B} = [X, \hat{B}_m], \)
\( \hat{B}_{m} = [Y, \hat{B}_m] \) of size \( 2, \) and then, if \( z = 1, \) by setting \( \hat{B}_m = \hat{B}_m, \)
\( \hat{B}_m = \hat{B}_2 \) and \( J_m = \{i, k\} \setminus m \). At the end of each iteration (lines
14-16), they also set \( R_j = B_j = J_j = J = \infty, \) as needed for consistency.

Protocol extensions. Our protocol can be easily adapted to handle
higher dimensions \((d > 1)\). Its sub-protocol (PHC.Cluster) com-
pares 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 dimen-
sions (over AHE). Shuffling remains largely unaffected, besides lists
\( L, S, S^c \) consisting of \( d \) points 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 Eu-
clidean distances, the distances can be securely computed with AHE; for other
metrics, more elaborate sub-protocol may be required.

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

The CURE approximate clustering algorithm

Input: \( D, n, s, p, q, t_1, t_2, R \)  
Output: Clusters \( C \) over \( D \)

**[Sampling]** Randomly pick \( s \) points in \( D \) to form sample \( S \).

**[Clustering A]**
1. Partition \( S \) into \( p \) partitions \( P_i \), each of size \( s/p \).
2. Run HCAlg to cluster each \( P_i \) into \( s/(pq) \) target clusters.
3. Eliminate within each \( P_i \) clusters of size less than \( t_1 \).

**[Clustering B]**
1. Run HCAlg to cluster all remaining A-clusters \( C_A \) in \( S \).
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 to B-cluster of closest representative.

![Figure 5: The CURE approximate clustering algorithm.](image)

**Theorem 5.1.** Assuming Pailler’s encryption scheme is semantically secure and that ArgMin and MaxDist are securely realized by GC-based protocols, protocol PHC securely realizes functionality \( f_{HC}^* \).

### Table 1: CURE clustering parameters and values.

| Parameters | Description | Value |
|------------|-------------|-------|
| \( n, s \) | Sizes of dataset and its sample | \( \leq 1M, \[10^6 : 10^7\] \) |
| \( p, q \) | # 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 | \( R = 1, 3, 5, 7, 10 \) |

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 \( S \) to form \( C_A \) over \( S_A \triangleq S \setminus O_A; C_A \) is exactly the output of HCAlg run on \( S_A \); and next
   - ii. HCAlg runs on \( C_A \) to form \( C_B \) over \( S_B \triangleq S \setminus \{ O_A \cup O_B \} \); \( C_B \) is exactly the output of HCAlg run on \( S_B \).

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 \( S_P, S_Q \) 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 PCure1, PCure0 and PCure0.

In PCure1, \( P_1, P_2 \) 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 \( R = 1 \) 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 \(O(\lambda^2)\), 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 \(\mathcal{F}_{HC}\) of Section 3. The intended two-party computation merely involves computing \(B\)-cluster representatives, which \(\mathcal{F}_{HC}\) provides, and any input/output modification in HCAlg causes a trivial change to the pre-/post-processing component of \(\mathcal{F}_{HC}\), consistent to our point-agnostic dendograms.

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 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.

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

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 \(\kappa\) 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 \(t = 5\); as our protocol incurs costs linear in the number of iterations \(n - t\), 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. \(K = 2^{60}\) for IEEE 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 libbaillier [3] for Pailler 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 \(n\).
At 300 samples or higher, the gap is within 3%. For higher values of

\[ p \]

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 (\( p = 1 \)), \( q = 5 \) 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 – 3X 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 \( q = 5 \) 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 \(~800\)ms for a total of \(~1sec\). In subsequent rounds, the circuits become progressively smaller but the number of roundtrips remains the same; even conservatively multiplying by 395 rounds, we have approximately 400sec 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 [79], 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 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 harmful—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 Anrini Chakraborti for shepherding this paper. Dimitrios Papadopoulo 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/2017/09/20/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.ca.utexas.edu/libpaillier
[4] 2021. AWS VPC. https://aws.amazon.com/vpc
[5] Martin 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-Ruabe 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 Linhui 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. PETS 2018, 4 (2018), 104–124. https://doi.org/10.1515/popets-2018-0034
[11] Fotini Baldimtsi, Dimitrios Papadopoulos, Stavros Papadopoulos, Alessandra Scafuro, and Nikos Triandopoulos. 2017. Server-Aided Secure Computation with Off-line Parties. In ESORICS 2017: 103–123.
A GARBED 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 $A$ and $B$ 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 in $C_f$ to two random values $w_0, w_1$ (from a large domain, e.g., $\{0, 1\}^{112}$), called the garbled values of $X$.

Specifically, $P_1$ maps the output wires of gates $A$, $B$, and $C$ to random garbled values $\{w_0^A, w_1^A\}$, $\{w_0^B, w_1^B\}$ and respectively $\{w_0^C, w_1^C\}$, and also maps the two input wires of gate $A$ (respectively, gate $B$) to random garbled values $\{w_0^1, w_1^1\}$, $\{w_0^2, w_1^2\}$ (respectively, $\{w_0^2, w_1^2\}$, $\{w_0^2, w_1^2\}$), where menonically the i-th input bit of party $P_1$ corresponds to the $i$-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) $\rightarrow$ 1 in the truth table of $A$ dictates that the output is 1 when input is 1, 1 or, using garbled values, that the output is $w_1^A$ when input is $w_1^1, w_1^2$. Accordingly, using a semantically-secure symmetric encryption scheme $E_k(\cdot)$ (e.g., 128-bit AES), $P_1$ can express this condition as ciphertext $E_{w_1^A}(E_{w_1^1}(E_{w_1^2}(1)))$, where the output $w_1^A$ is successively encrypted using the inputs $w_1^1, w_1^2$ as encryption keys. $P_1$ produces a similar ciphertext for each other row in the truth table of $A$ and sends to $P_2$, permuted to hide the order of the rows. Observe that one can retrieve $w_1^A$ if and only if they possess both $w_1^1, w_1^2$ and that if one possesses only $w_1^1, w_1^2$, all other entries of the garbled truth table of $A$ (besides $w_1^A$) 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 $w^0, w^1$ 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.

Figure 9: Garbled circuit $C_f$ of a specific function $f$ that computes the OR over the corresponding ANDs of the 2-bit inputs.
The circuit for protocol ArgMin.

Figure 10: The circuit for protocol ArgMin.

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 $w_{11}, w_{21}$, $P_2$ can try to decrypt every value in the garbled truth table of $A$, until $P_2$ finds the correct value $v_A$.

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 $w_{11}, w_{12}$ 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 $w_{21}, w_{22}$ 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 $(w_{21}, w_{22})$ 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 $v_{C_f}$ of the output wire.

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 $n \times 1$-bit values $v_1, \ldots, v_n$ and $n \times k$-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 $i$th 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 $MUX_{i'}$:• $\mu = \log n$: once for selecting among two $(\log n)$-bit indices $loc_i$ and $i + 1$, the latter conveniently encoded as the output of constant gate $CON_{21}$ (such hard-coded indices significantly facilitate their propagation in the circuit, compared to the alternative of handling indexes as input and carrying them throughout the circuit); and • $\mu = \lambda$: once for selecting among two $\lambda$-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.

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.

Recall that data consists of $\lambda$-bit values and is secret shared among the two parties as $\kappa$-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 $\lambda$-bit integers, using a one-bit output to encode which input value is 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 $CON_i$, $1 \leq i \leq n$, that always output the $(\log n)$-bit fixed value $i$ (e.g., $CON_1$ outputs the binary representation of 3).

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 $\kappa$-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 $MUX_3$ for selecting the minimum among two $\lambda$-bit values $u - r_1$ and $v - r_2$, which becomes the final output (see arrow wire) 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 $P_1, P_2$ 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 \( p_1, q_1 \) 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 himself 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 \( t \), messages received during the oblivious transfer execution for \( \text{ArgMin} \), denoted by \( OT_t \) and the min/max index \( a_t \).
3. During each clustering round \( t \), for each execution of \( \text{MinDist}/\text{MaxDist} \) for index \( k \), messages received during the corresponding oblivious transfer execution, denoted by \( OT_{t,k} \), corresponding garbled circuit \( GC_{t,k} \), and output value \( q_{t,k} \).
4. Encrypted cluster representative values \( E_1, \ldots, E_t \).

The simulator \( \text{Sim}_{P_2} \), on input the random tape \( R_2 \), points \( q_1, \ldots, q_n \), outputs \( \text{rep}_t = \{ rep_1/|J_1|, \ldots, rep_n/|J_n|, |J_1|, \ldots, |J_n| \} \), \( \alpha_1, \ldots, \alpha_t \), 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 \( p_1', \ldots, p_n' \) 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 \( p_1' \) as input for \( P_1 \) (and the actual values \( q_t \) of \( P_2 \)), with the following modifications.

- **(Oblivious transfer simulation for \( OT_t \))** For \( t = 1, \ldots, t \) let \( W_t \) be the set of garbled input values computed by \( P_2 \) for the garbled circuit that evaluates \( \text{MinDist}/\text{MaxDist} \) at clustering round \( t \). 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 \( i = 1, \ldots, t \), the simulator includes in the view (instead of \( OT_t \)) the output \( OT_t' \) produced by simulator \( \text{Sim}_{OT}^{(2)} \) on input \( W_t \).\(^5\) Note that \( P_2 \) does not receive any output from this oblivious transfer execution, thus \( \text{Sim}_{OT}^{(2)} \) only works given the input.

- **(Oblivious transfer simulation for \( \text{ArgMin} \))** For each clustering round \( t \), the simulator includes in the view, the index \( a_t \).

- **(Garbled circuit simulation for \( GC_{t,k} \))** Next, the simulator needs to compute the garbled circuits \( GC_{t,k} \). The simulator uses the corresponding values from \( R \) (as computed so far) and a “new” blinding factor \( p_{t,k} \) for \( P_1' \) inputs and computes a garbled circuit for evaluating \( \text{ArgMin} \) honestly.

The simulator also includes in the view of \( P_2 \) the garbled inputs for the corresponding elements from \( R \).

- **(Oblivious transfer simulation for \( OT_{t,k} \))** Let \( y_{t,k} \) be the input of \( P_2 \) for the circuit \( GC_{t,k} \) (i.e., the execution of \( \text{ArgMin} \) for index \( k \) during clustering round \( t \)). 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 \( p_1' \)) up to that point, therefore \( y_{t,k} \) can be computed by the simulator. In order to compute the parts of the view that correspond to each of \( OT_{t,k} \) the simulator includes in the view the output of \( \text{Sim}_{OT} \) on input \( y_{t,k} \) and the corresponding choice from each pair of garbled inputs he chose in the previous step (as dictated by the bit representation of \( y_{t,k} \)), which we denote as \( OT_{t,k}' \).

- **(Encrypted representatives computation)** For \( \ell = 1, \ldots, \ell_t \), the simulator computes \( r_{\ell, t} = \{ \text{rep}_1/|J_1| \cdot |J_1| \} \) and \( E_{\ell} = \{ \text{rep}_1 \} \), 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 view \( \mathcal{V}_{\text{PHC}} \), i.e., the view of \( P_2 \) when interacting with \( P_1 \) running PHC for points \( p_k \).

**Hybrid 1.** This is the same as Hybrid 0, but the output of \( GC_{t} \) in view \( \mathcal{V}_{\text{PHC}} \) is replaced by \( \alpha_t \). This is indistinguishable from Hybrid 1 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 \( \alpha_t \).

**Hybrid 2.** This is the same as Hybrid 1, but values in \( B, L \) are computed using values \( p_1' \). This is statistically indistinguishable from Hybrid 1 (i.e., even unbounded algorithms can only distinguish between the two with probability \( O(2^n) \) since in view \( \mathcal{V}_{\text{PHC}} \), each of the values in \( B, L \) are computed as the sum of a random value from \( \{0, 1\}^d \) and a distance between two clusters.

**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 \( OT_t \) is replaced by \( OT_t' \), 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 \( GC_{t,k} \) are chosen based on the values that have been computed using values \( p_1' \). Since garbled inputs are chosen uniformly at random (irrespective 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 \( OT_{t,k} \) is replaced by output of \( OT_{t,k} \) 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 \( E_t \) sent to \( P_2 \) is computed as \( \{ \text{rep}_1/|J_1| \cdot |J_1| \} \) 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 \( \text{rep}_1/|J_1| \).

\(^5\)And corresponding randomness derived from \( R_2 \).
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 \( \pi_{HC} \) 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 \( \ell \), a garbled circuit \( GC_{\ell} \) for evaluating \( \text{ArgMin} \), messages received during the corresponding oblivious transfer execution denoted by \( OT_{\ell} \).
3. During each clustering round \( \ell \), for each execution of \( \text{MinDist}/\text{MaxDist} \) 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 \( p_1, \ldots, p_n \), outputs \((\text{rep}_1//|j_1|, |j_1|, \ldots, \text{rep}_r//|j_r|, |j_r|), \alpha_1, \ldots, \alpha_r \), 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 \( s_k, p_k \) and computes a pair \( s k^\ell, p k^\ell \) for himself. He computes \( D, H, L \) consisting of encryptions of zeros under \( p k^\ell \). Moreover, he computes \( R, S \) consisting of encryption of values chosen uniformly at random from \( \{0,1\}^e \) and encrypted under \( p k \).
- **(Garbled circuit simulation for** \( GC_{\ell} \)**)** Next the simulator needs to provide garbled circuits for the evaluation of \( \text{ArgMin} \) for each clustering round \( \ell \). For this, the simulator creates a "rigged" garbled circuit \( GC_{\ell}^r \) that always outputs \( \alpha_\ell \), 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 template to always to decode to the bit-representation of \( \alpha_\ell \) (this process is explained formally in [69]).
- **(Oblivious transfer simulation for** \( \text{ArgMin} \)**)** Let \( W_\ell^{(1)}, W_\ell^{(2)} \) be the sets of pairs of input garbled values that the simulator choses while creating \( GC_{\ell}^r \) 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 \( Sim_{OT}^{(1)} \) on input \((y_\ell, W_\ell^{(1)}) \), where \( y_\ell \) is the bit description of the input of \( P_1 \) for \( GC_{\ell} \) (which can be computed with the simulator since he has access to \( p_1, R_1 \)).
- **(Oblivious transfer simulation for** \( \text{MinDist}/\text{MaxDist} \)**)** 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 \( \text{MinDist}/\text{MaxDist} \) at clustering round \( \ell \) 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 \( \ell, k \) the simulator includes in the view (instead of \( OT_{\ell,k} \)) the output \( OT_{\ell,k}^{(1)} \) produced by simulator \( Sim_{OT}^{(1)} \) 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 \( Sim_{OT}^{(1)} \) 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 view \( \pi_{PHC}^{P_1} \), i.e., the view of \( P_1 \) when interacting with \( P_2 \) running \( \pi_{HC} \) 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 \( \{0,1\}^e \) under key \( p_k \). 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 \( GC_{\ell} \) is replaced by \( GC_{\ell}^r \) computed as described above (including the values from \( W_{\ell,1} \)). 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 \( OT_{\ell} \) is replaced by \( OT_{\ell}^r \), 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 \( OT_{\ell,k}^r \) 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.