Privacy Preserving Collaborative IDS

Laylon Mokry, Cooper Guzzi, Samson Oni,
Adina Crainiceanu, Zhiyuan Chen, Karuna Joshi, Don Needham

Abstract

Intrusion Detection Systems (IDS) are commonly used by organizations to monitor network traffic and detect attacks or suspicious behaviours. However, many attacks occur across organizations and are often difficult to detect using any single IDS. Collaborative Intrusion Detection Systems (CIDS) could lead to more accurate prediction and detection of cyber threats, but most organizations are unwilling to disclose sensitive information about their internal network topology and traffic, lending these systems unusable. In this paper, we propose *SHP kPrototypes*, a distributed clustering algorithm for horizontally partitioned mixed data using additive secret sharing. This algorithm can be used to create a privacy preserving, collaborative intrusion detection system. Theoretical and experimental results show the performance of our algorithms and the cost (in terms of running time and bandwidth used) of ensuring privacy.

1 Introduction

Deployed IDSs monitor computer networks and provide the organizations with data concerning potential malicious activity occurring on these networks. Machine learning techniques can be employed on the data collected by these systems to help predict and prevent further malicious activity on these networks. However, in order for machine learning techniques to be successful in analyzing data and predicting how and when certain attacks are going to occur, a large data set is needed to train the machine learning model. Multiple organizations often experience the same types of attacks, sometimes occurring from the same malicious actors. A commonality of attacks and actors leads to the potential for multiple organizations to benefit from the training of a machine learning model with the data collected from intrusion detection systems deployed across multiple computer networks belonging to different organizations. However, with the increasing emphasis on data privacy and security, participating organizations are expected to be wary of training a collective machine learning model, unless their sensitive data is guaranteed to be kept private and secure from the other participating clients.

In this paper, we propose *SHP kPrototypes*, a privacy preserving, collaborative clustering algorithm for horizontally partitioned mixed numerical and categorical data and apply this algorithm to the analysis of the intrusion alert data collected by intrusion detection systems across multiple organizations.

2 Related Work

2.1 Federated Learning

As noted in [YLCT19], machine learning models require a large amount of data in order to be successful. However, real world situations are typically incapable of providing the amount or quality of data needed for success. Additionally, the increased emphasis on data privacy and data security has introduced more challenges as to how to best share data to a machine learning model to achieve the best results. In most industries, only a limited amount of data or poor quality data is available. Additionally, data typically exists in "isolated islands," where relevant data that would be needed to properly train a successful machine learning model is scattered across

different parties. In most cases, the barriers separating these data sources are very difficult, or impossible, to break. Furthermore, the action of sharing the data of these separate sources proves challenging when data security and data privacy must be conserved. [YLCT19] proposes the application of federated learning and transfer learning to solve the challenges faced by traditional machine learning techniques in order to apply machine learning to the majority of real world scenarios that face these data dilemmas. Specifically, the paper discusses the concept of Horizontal Federated Learning where the data set used to train the machine learning model is horizontally partitioned across the collaborative clients, meaning that the participating clients have their own data sets which share the same feature space but differ in samples. In the case of collaborative Intrusion Detection Systems, data could also be considered horizontally partitioned since each client shares a similar feature space (i.e. source and destination IP addresses and port numbers, timestamps, alert types, and protocols across network traffic), but different samples since the participating clients are collecting this similar data on their own networks.

2.2 Secure Multiparty Computation and Secret Sharing

[CDN15] discusses the theories and practices of Secure Multiparty Computation for scenarios in which separate parties wish to collectively train a machine learning model while maintaining data privacy and security between each individual participating client. It uses Statistical Euclidean Distance to define similarity between data inputs. It also discusses Secret Sharing techniques as a way to collaboratively train a machine learning model while preserving data privacy and security. In our work, we primarily use Statistical Euclidean Distance to solve for data similarity and Secret Sharing to create an implementation of a k-prototypes clustering algorithm on horizontally partitioned mixed data to create a collaborative, privacy preserving intrusion detection system.

2.3 Distributed Privacy Preserving K-Means Clustering with Additive Secret Sharing

[DPS+08] introduces a series of algorithms for K-means clustering with additive secret sharing for vertically partitioned data sets. This allows for the analysis of unions of data sets without revealing the data of a given party to any other party. In this method, parties distribute random shares to all other parties in the computation. for each of their entities, and requires four parties to take on special roles within the computation in order to calculate the closest cluster-mean for a given entity, without giving any party extra information than the result of the computation. Two of the special parties receive the random shares, and add them to determine the cluster mean closest to an entity. The owner of the entity is unknown to these parties, as the other two special parties determine a random permutation of each party's sum, obscuring the identity of the entity owner. Because the computation described in [DPS+08] is asymmetric, the four special parties store more data collectively than the other parties. This makes the algorithm less secure, as collaboration between any two of them would result in the visibility of a single party's entity to the collaborating actors.

2.4 Taxonomy and Survey of Collaborative Intrusion Detection

[VKMF15] describes the state of the art for Collaborative Intrusion Detection Systems (CIDS). CIDS share data from multiple organizations to more effectively detect attacks. This can be accomplished with both active measures such as the deployment of honey pots, and passive measures such as end point monitoring. In our work, we focus on sharing and correlating data collected by passive measures to detect intrusions. The architecture of a CIDS affects its performance when a high volume of data is being processed. Centralized systems which correlate and analyze data on a single machine, suffer bottleneck as the amount of computations required increase. Hierarchical systems take advantage of the pre-processing of data before forwarding it to the next level in the system ar-

chitecture, culminating at a central machine which manages the IDS, but still are suffer from bottleneck at the central monitor under high amounts of traffic. Furthermore both of these architectures suffer from single points of failure at the central machine of the system. Distributed architectures avoid bottleneck and single point of failure by sharing alert correlation responsibilities across all machines in the CIDS. Because CIDS are IT systems themselves, they are also vulnerable to compromise. With this in mind, resiliency must be a design consideration when building a CIDS. As noted in [VKMF15], some current systems use Probabilistic models to represent a trust ranking for each of the machines in a distributed architecture, providing resiliency in the event of a successful intrusion. A comparison between this ranking and a trust threshold determines whether or not a given machine is recognized as a malicious host.

2.5 Privacy Preserving Approach for Sharing and Processing Intrusion Alert Data

As acknowledged in [HW15], cooperation among different organizations is required to defend against multi-stage and large-scale cyber attacks. The collaboration of these multiple organizations is necessary to train a more powerful and successful machine learning model to discover patterns in intrusion detection systems' alert data. Additionally, the privacy concern of these different organizations' unwillingness to openly share data concerning their intranet topology, network services, and security infrastructure is noted. The overall idea of [HW15] is that participating organizations can encrypt their intrusion alert data and outsource their data to a shared server to reduce the cost of data storage and maintenance. It also discusses three different types of alert correlation algorithms: similarity-based methods, sequential-based methods, and case-based methods. It defines similarity-based methods as techniques that cluster and aggregate intrusion alerts based on the similarities of some selected features, such as source and destination IP addresses, source and destination port numbers, protocols, time-stamp information, and alert types. It defines sequential-based methods as techniques that correlate alerts by using causality relationships among the alerts. Finally, it defines case-based methods as techniques that rely on the existence of a knowledge-based system used to represent well-defined attack scenarios.

3 Preliminary Information

3.1 K-Prototypes Clustering for Mixed Data

Our approach for a privacy preserving, collaborative intrusion detection system uses a modified version of the k-means clustering algorithm to generate clusters of related security alerts sourced from the recorded traffic of multiple parties using the same intrusion detection system locally. Intrusion detection systems generate security alerts which are comprised of both values with a numerical representation in which the distance between values is meaningful, such as timestamps and IP addresses, and categorical data values, where their numerical representation is mostly arbitrary, such as the names of alert classifications. Since a collaborative intrusion detection system would be interested in both types of data, the challenge of running the k-means clustering algorithm on a mixed data set is introduced. In a regular k-means clustering algorithm, Euclidean distance is used to determine the distance from the data point and each cluster centroid. The data point is then assigned to the cluster it is closest to. After this has been done for all data points in the data set, the center of each cluster is recalculated by taking the mean of all of the data points assigned to that cluster. These two processes are repeated until the cluster centroids converge. This basic algorithm is modified to work on mixed data in [Hua98], where the Euclidean distance between two categorical values is not a meaningful metric. We define the general clustering algorithm for alert data in the centralized case as ckPrototypes, and it is defined in Algorithm 1. For our application, we refer to data points as entities, which, in our experiments, are defined as objects containing lists of the following attributes of a single alert: source IP,

destination IP, timestamp, rule ID, source port, destination port, and alert type. In order to assign entities to nearest clusters, a measure of similarity must be taken. Our approach requires multiple methods of calculating similarity: one for numerical attributes, one for qualitative attributes, and some way to combine them to calculate the similarity between two entities. In order to calculate the similarity between two entities for a given numerical attribute, the Euclidean distance is calculated. In order to calculate the similarity between two entities for a qualitative attribute, the corresponding values of the two entities are compared for an exact match, as in [Hua98]. These methods are used for each quality of the entity, and the outputs are summed to generate a total distance between the two entities, allowing each distance to be compared to find the nearest cluster center (prototype) to an entity. An important aspect of k-means clustering, which must be modified for use on mixed data, is the recalculation of cluster centers. In standard k-means clustering for numerical data, the mean value of an attribute of each entity in a cluster is calculated and set as the new value for its respective attribute in the cluster's centroid. For mixed data, this method works for quantitative attributes, but not qualitative ones. Therefore, the mode value for a qualitative attribute across the entities in the cluster is taken and assigned as the value for the centroid's attribute, as in [Hua98]. This approach, combined with the standard approach allows for clustering to occur over mixed data.

Algorithm 1 : `ckPrototypes`()

```
1: initialize cluster centroids to random values
2: finiashed = false
3: while not finished do
4:    for all entities e do
5:       assign e to nearest cluster
6:    end for
7:    for all clusters do
8:       set new centroid to average (for numerical attributes) and mode (for categorical attributes) of contained entities
9:    end for
10:   if centroids == new centroids then
11:      finished = true
12:   end if
13:   centroids = new centroids
14: end while
```

3.2 Additive Secret Sharing

In order to accomplish a privacy preserving implementation, additive secret sharing as discussed in [CDN15] is used to secure our ckPrototypes algorithm and calculate the cluster centroids without exposing each party's data to one another. In our application, the sensitive data that cannot be shared with each party are the values of source and destination IP addresses, timestamps, and the modes of each of the categorical data features within the data-set of a party. We implemented additive secret sharing by making shares of both the integer counts of the modes of the categorical values and double values of timestamps and IP addresses. The general method used to make n shares of an integer or a double secret value s and return those shares in an array is described in Algorithm 2. In additive secret sharing, to make n shares of an integer or double secret s, $n-1$ random shares are created by randomly choosing integers or doubles respectively between 0 and s exclusive, so that:

$$r_n = s - r_1 - r_2 - ...r_{n-1}$$

. The shares made are not modulo divided by a large prime, as they are in [CDN15], as it was determined that the summation and subsequent sharing of all the data within a party allowed for the sufficient obfuscation of one party's data to all others, and the modular arithmetic requires integer value, and we made use of doubles.

Algorithm 2 : `makeShares`(int/double s, int n)

```
1: int/double lastShare = s
2: int/double thisShare = 0
3: for all integers i in the range (0, n-2) do
4:    thisShare = random int/double in range (0, lastShare)
5:    lastShare = lastShare - thisShare
6: end for
7: shares[n-1] = lastShare
8: return shares
```

4 Privacy-Preserving K-Prototypes Clustering for Horizontally Partitioned Mixed Data

4.1 Horizontally Partitioned K-Prototypes Algorithm

We modify the ckPrototypes clustering algorithm to work in a decentralized, multi-party case, with horizontally partitioned data. Each party owns a subset of the full data-set, each subset having the same structure (attributes). The modified algorithm can be applied to clustering of IDS alerts across a number of multiple contributing parties with their own sets of alert data. This implementation assumes that each contributing party could shares its own data with all of the other contributing parties, but the computation is distributed. Algorithm 3 shows the pseudocode for *hkPrototypes*, the clustering algorithm implemented for this horizontally partitioned scenario. The approach for a multiparty case is similar with the one in *ckPrototypes* in terms of how entities are assigned to a cluster, but each party does its own assignment (see lines 4-8 in Algorithm 3), but computing a cluster's centroid requires cooperation between parties, as shown in lines 9-15 in Algorithm 3.

Algorithm 3 : `hkPrototypes`()

```
1: initialize cluster centroids to random values
2: finished = false
3: while not finished do
4:    for all parties do
5:       for all entities e do
6:          assign e to nearest cluster
7:       end for
8:    end for
9:    for all clusters c do
10:      c.entities = empty
11:      for all parties p do
12:         c.entities += p.getEntitiesAssignedtoCluster(c)
13:      end for
14:      c.centroid = average (for numerical attributes) and mode (for categorical attributes) of c.entities
15:   end for
16:   if centroids == new centroids then
17:      finished = true
18:   end if
19:   centroids = new centroids
20: end while
```

4.2 Privacy Preserving K-Prototypes Clustering Algorithm

In most cases, contributing parties would not be willing to share various aspects of their own alert data that might be considered sensitive information to their organization. In order to accomplish a privacy preserving version of hkPrototypes, additive secret sharing is employed. Now, instead of each party having full access to the alert data of the other parties, each party splits their data into random shares and send a share to each contributing party. Since the parties will only have a share of the other parties' data, it will not be able to infer what the actual data of that party is. This permits clustering on the alert data of the parties without ever exposing the data of a single party to the other contributing parties. The algorithm used to accomplish this is defined as skPrototypes and is explained in Algorithm 4. In order to make hkPrototypes privacy preserving, the

calculation of cluster centroids is done with additive secret sharing as defined as secureEntitySumProtocol and explained in Algorithm 5. In this secure sum protocol, each party computes the sum of the entities within a given cluster. Next, each party splits the secret sum into n shares, where n is equal to the number of parties. The shares are then distributed to each party, which computes the intermediate sum by adding the all of the shares it has been given by all of the other parties. Finally, the final sum, which is the centroid for the given cluster, is computed by adding up all of the party's calculated intermediate sums. The secureEntitySumProtocol is repeated for all clusters to calculate the centroids of each cluster.

Algorithm 4 : `skPrototypes`()

```
1: initialize cluster centroids to random values
2: while not finished do
3:    for all parties do
4:       for all entities e do
5:          assign e to nearest cluster
6:       end for
7:    end for
8:    for all clusters c do
9:       c.centroid = secureEntitySumProtocol(c)
10:   end for
11:   if centroids == new centroid then
12:      finished = true
13:   end if
14:   centroids = new centroids
15: end while
```

Algorithm 5 : `SecureEntitySumProtocol`(Cluster c)

```
1: for all Party p do
2:    Entity sum = empty Entity
3:    for all Entities e do
4:       if e.isAssignedTo(c) then
5:          sum += e
6:       end if
7:    end for
8:    Entity[] shares = makeShares(sum, numberOf-
      Parties)
9:    int i = 0
10:   for all Other Parties o do
11:      o.intermediateSum += shares[i]
12:      i++
13:   end for
14: end for
15: for all Parties p do
16:   for all other Parties o do
17:      o.finalSum += p.intermediateSum
18:   end for
19: end for
20: return finalSum of anyParty
```

5 Algorithm Cost Analysis

Standard K-Means clustering has a time complexity of O($inkt$), where n is the number of entities in the dataset, k is the number of clusters, t is the amount of time it takes to calculate the distance between two entities, and i is the number of iterations for which the algorithm is run. This can be done for a fixed number of iterations, or run until convergence. If run until convergence, i can vary greatly depending upon the random initialization of the cluster centroids. Our approach increases the time complexity with the inclusion of parties and secret sharing resulting in a modified time complexity of O(in^2ktp^2). The parties input size has a degree of 2, as each iteration requires each party to send shares of it's data to each of the other parties, increasing the complexity asymptotically by a factor of p^2. The n factor has a degree of 2 due to the complexity of developing additive shares of the maps used to calculate the mode of an attribute across all entities. Our implementation used hash-maps to store the values and counts for each attribute of the data in a party's data set. To compile this map, each entity in the data set must be accessed and added to the map. Then, to make shares of this map to send to the other parties, each entry must be accessed and split into p shares. This increases the time complexity significantly, as this process occurs p times per iteration of the algorithm. Each entry in the hash-map, must be accessed and used to generate additive shares in p new maps. This results

in an extra $O(n)$ computations each time shares are generated for a party's data-set.

6 Privacy Discussion

Assumption: Semi-honest model where each party follows the protocol but each party might want to infer more information.

What is protected: Values of numerical attributes, as well as the frequencies of occurrences of individual categorical features. The security of additive secret sharing of an integer value is explained in [CDN15]. The security does not change when the practice is extended to double data types. Since the process is proven secure in [CDN15] and the process is simply extended to multiple numerical values, our process assumes the same security definition.

What is learned: Whether or not a particular value of a categorical attribute appears in the data set of a party, but not the specific counts for that value. This is because the shares of the mode map include the unprotected values of the categorical data, as well as shares of the number of occurrences involving each value for each category. This allows a semi-honest party to discern from a share if a certain categorical value occurs within the data set of a party which sends a share. However, the exact prevalence of incidents is protected by the splitting and sharing of the value across the other parties. For example, in our application to IDS alert data, the presence of any activity on a specific port or alerts of a specific rule ID or type is learned whenever a mode map is shared and the values for those ports, alert types, or rule IDs are nonzero.

7 Experimental Evaluation

7.1 Set Up

Software/Hardware: We ran the experiments on a VMware Virtual Machine running Ubuntu 18.04.1 LTS(64bit), 2GB RAM, and 20GB hard disk the host machine, was running Windows 10, has 2-Core Processor, 8GB RAM, and 256GB hard disk. Since the following experiments were all run on a single machine, the results do not reflect any time delay that would be expected to be caused by network latency in a real-life application. Also, the results do not reflect the parallel computation that would occur simultaneously at each party.

Algorithms Implemented: After defining our ckPrototypes, hkPrototypes, and skPrototypes clustering algorithms, we modified the Java implementation of the k-means clustering algorithm found in [hmcjc20] to run our three algorithms on the alert data. Our code used for experimentation can be found at [htt20a].

Data: The following experiments were run using the full SNORT alert data from [htt20b]. Before experimentation could begin, the features used to determine likeness had to be parsed out of the data and normalized when appropriate. These features are source IP, destination IP, timestamp, rule ID, source port, destination port, and alert type. All source IPs, destination IPs, and timestamps were treated as doubles and subsequently normalized. For IP addresses, the dotted quad formatted strings were converted to doubles using a the "Power of 256" approach as described in [hciatdn20]. These values were then normalized by dividing them by the double equivalent of the universal broadcast IP address, 255.255.255.255. Each timestamp was converted to unix time using Java's toEpochSecond() function in the LocalDateTime library. These values were then normalized by dividing them by the max Epoch time value of December 31st, 2012 23:59:59.999 UTC, since all of our data was assumed to be collected during the 2012 calendar year. Rule ID, port numbers, and alert types were treated as categorical features. The distance between two entities' categorical features, as explained in section 3.1 of this paper, is measured by checking for an exact match. In the case of rule IDs and port numbers, the integer values of these features are simply compared for equality. Since alert type is simply a plain text, English string, Java's hash-map data structure is used to map each alert type to an integer. The corresponding integer values for the alert type strings are then compared for equality. To test the hkPrototypes and skPrototypes algorithms, three simulated collaborating parties were made by making

three random shares of the full SNORT alert data-set from [htt20b] using a "Round-Robin" approach.

Methodology: To conduct these experiments, each test was run ten times and the mean results were recorded.

Parameters Varied: Throughout experimentation, the number of clusters, number of entities, and number of parties were varied.

Performance Metrics: As performance metrics, we use megabytes of data shared during the clustering process and total run-time of the clustering program measured in milliseconds.

7.2 Experimental Results

7.2.1 Run-Time with Varied Cluster Counts

Figure 1 shows the relationship between overall run-time, in milliseconds, of our three implementations with varying numbers of clusters between three and eight. As shown in Figure 1, hkPrototypes is only slightly slower than ckPrototypes. Figure 1 also shows that the run-time of skPrototypes experiences exponential growth as the number of clusters increases.

7.2.2 Megabytes Shared with Varied Cluster Counts

Figure 2 shows the relationship between the total amount of megabytes shared during the clustering process versus the number of clusters. This experimentation yields no results for the centralized case, since no data needs to be shared when the calculations are being done centrally by one party. As the number of clusters is increased from three to eight, the number of megabytes shared throughout the clustering process experiences exponential growth. Additionally, skPrototypes requires significantly more megabytes of data shared than hkPrototypes, which is expected since multiple shares of the data itself must now be shared amongst the collaborating parties, rather than just the sums themselves.

7.2.3 Megabytes Shared with Varied Entity Counts

Figure 3 shows the relationship between the megabytes shared during both skPrototypes and hkProtoypes with varying entity counts. This shows that the megabytes of data shared during the clustering process experiences polynomial growth as the entity count, or number of alerts that are being clustered, increases. This is due to the increased number of operations that must occur to generate and make shares of the mode of a categorical value.

7.2.4 Megabytes Shared with Varied Cluster Counts and Party Counts

Figure 4 shows the relationship between the megabytes shared during the clustering process and varying cluster and party counts. The experiment was run with cluster sizes varying between four and eight and parties varying between three and five. This shows that the megabytes of data shared during the clustering process experiences exponential growth in skPrototypes as the cluster count increases. It also shows the increase in megabytes of data shared during the clustering processes of each algorithm.

8 Conclusion and Future Work

8.1 Conclusion

In conclusion, we propose skPrototypes as a clustering algorithm that utilizes additive secret sharing to securely cluster data points, or entities, of alert data containing mixed data types. Queries on the clustered data can help intrusion detection systems identify and prevent network attacks on the computer networks of multiple different institutions. The benefit of the application of a privacy preserving collaborative intrusion detection system includes a more accurately trained machine learning model for attack detection and prevention, since the model can be trained by the alert data of multiple collaborative parties. Additionally, our implementation can do so while maintaining the data privacy and security of each participating party. What our imple-

mentation sacrifices for data privacy and security is the run-time of and the amount of data required to be sent during the clustering process. Both the run-time of and amount of data sent during the clustering process grow exponentially as the cluster and entity counts increase. Furthermore, these metrics increase significantly as the number of collaborating parties increases. Other possible implementations that could be looked in to in order to reduce these metrics are mentioned below.

8.2 Future Work

Our experiments simulated multiparty computation by representing each party as an object and conducting their respective calculations iteratively. This fails to account for other factors that would affect the run time in a real-world application. These include network delays between parties, and the parallel computation of the cluster means. Additionally, our results indicate the increased time complexity of implementing secret sharing to cluster data that is partitioned horizontally. Further development is to be made to reduce this time complexity, allowing for faster computation across multiple parties, especially when calculating the mode of a cluster. Also, there is potential for the implementation of other privacy preserving techniques such as homomorphic encryption, which may have a lower time complexity than secret sharing. Furthermore, other machine-learning techniques, such as a rules-based approach, can be implemented with additive secret sharing to compare with mixed data clustering.

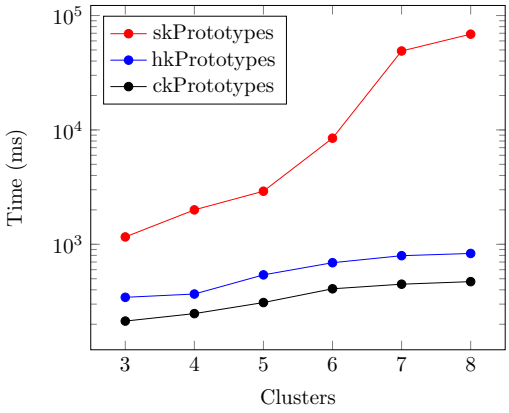

Figure 1: Time vs. Cluster Count

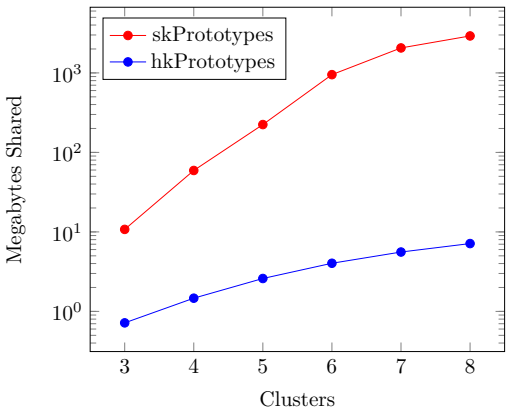

Figure 2: Megabytes Shared vs. Cluster Count

References

[CDN15] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[DPS+08] Mahir Can Doganay, Thomas B. Pedersen, Yücel Saygin, Erkay Savaundefined, and Albert Levi. Distributed privacy preserving k-means clustering with additive secret sharing. In *Proceedings of the 2008 International Workshop on Privacy and Anonymity in Information Society*, PAIS '08, page 3–11, New York, NY, USA, 2008. Association for Computing Machinery.

[hciatdn20] https://mkyong.com/java/java-convert-ip-address-to-decimal number/. Java – convert ip address to decimal number, Last accessed April 2020.

[hmcjc20] https://www.dataonfocus.com/k-means-clustering-java code/. K means

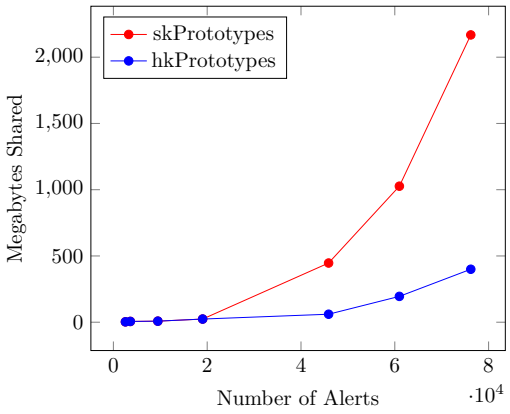

Figure 3: Megabytes Shared vs. Number of Entities

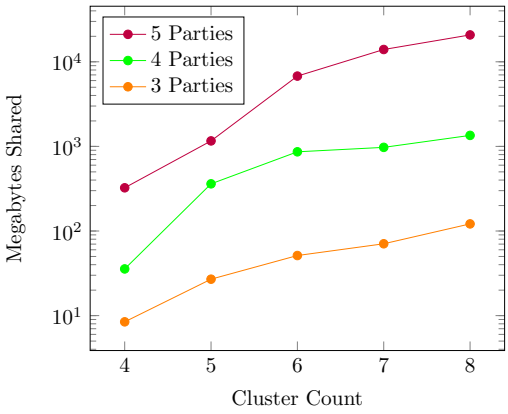

Figure 4: Megabytes Shared vs. Number of Clusters for Varying Number of Parties for skPrototypes

clustering java code, Last accessed April 2020.

[htt20a] https://github.com/CooperGuzzi/CollaborativeIDS. Github repo for collaborative ids, Last accessed April 2020.

[htt20b] https://www.secrepo.com/maccdc2012/. Maccdc 2012 snort alert data-set, Last accessed April 2020.

[Hua98] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304, September 1998.

[HW15] Hoang Giang Do and Wee Keong Ng. Privacy-preserving approach for sharing and processing intrusion alert data. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, April 2015.

[VKMF15] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. Taxonomy and survey of collaborative intrusion detection. *ACM Comput. Surv.*, 47(4), May 2015.

[YLCT19] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2):12:1–12:19, January 2019.