Efficient and Privacy-Preserving Collaborative Intrusion Detection Using Additive Secret Sharing and Differential Privacy

Laylon Mokry†, Paul Slife†, Patrick Bishop†, Jose Quiroz†, Cooper Guzzi†,
Zhiyuan Chen*, Adina Crainiceanu† and Don Needham†
†United States Naval Academy, Annapolis, MD
Email: {laylonmokry, pattybish50, jequiroz99,cooperguzzi,}gmail.com
paul.slife@protonmail.com, {adina, needham}@usna.edu
* University of Maryland Baltimore County, Baltimore, MD
Email: zhchen@umbc.edu

2021 IEEE International Conference on Big Data (Big Data) | 978-1-6654-3902-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BigData52589.2021.9671428

***Abstract*—Intrusion Detection Systems 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 could lead to more accurate prediction and detection of cyber threats as well as a reduction of security administrators' workload as similar threats from different places can be merged. However, most organizations are unwilling to disclose sensitive information about their internal network topology and traffic, lending these systems unusable. Existing solutions using homomorphic encryption and secure multi-party computation are often expensive. In this paper, we propose efficient and privacy preserving techniques to correlate alerts generated at different organizations. We propose $skPrototypes$, 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. We also propose $dpkPrototypes$ which uses differential privacy on categorical attributes and is more efficient than $skPrototypes$ for categorical attributes with many distinct values. Theoretical and experimental results validate the effectiveness of our algorithms.**

I. Introduction

Deployed Intrusion Detection Systems (IDS) monitor computer networks and provide organizations with data concerning potential malicious activity occurring on their 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.

Collaborative instruction detection systems (CIDS) [1] can conduct intrusion detection in a distributed manner. More specifically, each local intrusion detection system monitors its local network and generates alerts or other security data that can be correlated, aggregated, and analyzed by a collaborative analysis unit. 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.

Existing solutions for privacy-preserving collaborative intrusion detection use homomorphic encryption [2], [3] or secure multi-party computation [4], [5] techniques. The idea is to use these techniques to correlate or aggregate alerts generated from local IDS. However such techniques are expensive [6] and do not scale to accommodate big data.

In this paper we focus on a common task in CIDS: the aggregation and correlation of similar alerts generated by local IDS using privacy-preserving clustering. Although privacy-preserving clustering has been studied [7], [8], most solutions apply to either numerical data or categorical data, whereas alerts often have both categorical and numerical data. In this paper, we make the following contributions:

1) We propose $skPrototypes$, a privacy preserving, collaborative clustering algorithm based on additive secret sharing for horizontally partitioned mixed numerical and categorical data. Additive secret sharing has been shown to be more scalable than homomorphic encryption techniques [6]. We apply this algorithm to the analysis of the intrusion alert data collected by intrusion detection systems across multiple organizations.
2) We propose $dpkPrototypes$ which applies a differential privacy technique on categorical attributes. Using differential privacy reduces communication overhead of secret

This work was partially supported by Office of Naval Research grant# N00014-18-1-2452.

addition on categorical attributes without compromising privacy.

3) We performed preliminary experiments comparing $skPrototypes$ and $dpkPrototypes$ with a non privacy-preserving baseline. The results show reasonable overhead of our proposed approaches. Interestingly, although $dpkPrototypes$ reduces communication cost in each iteration of clustering, it does not lead to lower overall communication costs compared to $skPrototypes$ because the noise introduced by differential privacy leads to slower conversion of clusters. This is a trade off that is worth further investigation.

The rest of the paper is organized as follows. Section II gives a survey of related work. Section III reviews necessary background regarding clustering for mixed data. Section IV describes the two algorithms we propose. Section V analyzes the costs of our proposed algorithms. Section VI examines privacy concerns regarding what is protected and what is revealed by our approach. Section VII reviews our experimental results. Section VIII gives our conclusions and considers future work.

II. Related Work

A. Federated Learning

Federated learning [9] builds a machine learning model from data distributed at multiple sites without sharing the data directly. Instead, the machine learning step is distributed to local sites and intermediate results are shared. A recent survey of this approach is found in [10]. Our efforts fit the Horizontal Federated Learning task described in [10] 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.

B. Secure Multi-Party Computation and Secret Sharing

Secure multi-party computation uses cryptography protocols to compute a function over data distributed at multiple parties without revealing data at each party. Cramer et al. [11] surveyed the theories and practices of Secure Multiparty Computation (SMC). Two commonly used SMC techniques are homomorphic encryption and secret sharing. In our work, we use Secret Sharing to implement a k-prototypes clustering algorithm on horizontally partitioned mixed data as part of a collaborative, privacy preserving intrusion detection system.

C. Distributed Privacy Preserving Clustering

K-means clustering is a popular clustering method for numerical data. A survey of privacy preserving k-means clustering can be found in [7]. For data with categorical attributes, Dash et al. proposed a priavcy preserving k-medoid algorithm [8]. Doganay et al. [12] proposed several algorithms for K-means clustering with additive secret sharing for vertically partitioned data sets. In these methods, parties distribute random shares of its own data to all other parties in the computation. Our approach proposes privacy-preserving clustering techniques for horizontally partitioned data with mixed types.

D. Collaborative Intrusion Detection

The state of the art for CIDS is described in [1]. 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.

There are three types of architecture for a CIDS: 1) centralized systems which correlate and analyze data on a single machine, 2) hierarchical systems that aggregate data before forwarding it to the next level in the system architecture, and 3) distributed architectures that share alert correlation responsibilities across all machines in the CIDS. In the centralized and hierarchical architecture, there is a single point of failure (the machine that does the final analysis) as well as a bottleneck point when large amount of data needs to be correlated. In our work, we assume the distributed architecture.

Do and Ng [13] proposed a privacy-preserving collaborative intrusion detection system where 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 to represent well-defined attack scenarios. The solution we propose in this paper falls into the category of similarity-based correlation methods.

E. Differential Privacy

Differential privacy [14] is a strong privacy model that provides worst case privacy guarantees. Essentially, adversaries cannot distinguish the results generated by two data sets that differ by a single record. More specifically, let D_1 and D_2 be two data sets that differ by just one record. Let X be a randomization method that outputs a result r given a data set. Let $P_X(D_1, r)$ be the probability of X generating r given D_1, and $P_X(D_2, r)$ be the probability of X generating r given D_2. The randomization mechanism is α differential private ($0 \leq \alpha \leq 1$) if $\frac{P_X(D_1,r)}{P_X(D_2,r)}$ is in the range of $[\alpha, 1/\alpha]$. Note that

this definition is a simplified version of the standard definition where a privacy budget ϵ is used and here $\alpha = e^{-\epsilon}$.

The Laplace mechanism [14] is a commonly used method to implement differential privacy. A drawback of the Laplace mechanism is that it only works for numerical data and the noise is unbounded. Nonetheless, [15] provides a two-sided geometric mechanism which instead outputs discrete noise and is more suitable for categorical attributes. α is a parameter that controls the degree of privacy protection (also called privacy budget). The geometric mechanism generates noise z by: $Pr[Z = z] = \frac{1-\alpha}{1+\alpha}\alpha^{|z|}$, for every integer z. The perturbed data is "α- differential private."

III. Background

A. 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 multiple local intrusion detection systems. Security alerts contain both numerical attributes such as timestamps and IP addresses, as well as categorical data values such as the names of alert classifications.

The k-means clustering algorithm computes Euclidean distance from a 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.

We used k-prototype clustering [16] which is a modified version of k-means that works for mixed types of data. The clustering algorithm for alert data in the centralized case,*ckPrototypes*, is provided in Algorithm 1.

For our application, we refer to data points as entities, which, in our experiments, are defined as security alerts. The alert data used in this paper is generated by SNORT [17] and contains the following attributes: source IP, destination IP, timestamp, rule ID, source port, destination port, and alert type.

We compute the distance between two entities as follows. Let x_{ij} be the j-th attribute of entity x_i, N be the set of numerical attributes, and C the set of categorical attributes. We use Euclidean distance to calculate the distance between two entities for numerical attributes. For a categorical attribute, the distance equals zero if the two entities have the same value for the given categorical attribute (an exact match), and the distance equals to one otherwise, as in [16]. More specifically, the distance between two entities x_i and x_l is

$$\sqrt{\sum_{j \in N}(x_{ij} - x_{lj})^2 + \gamma \sum_{u \in C} \delta(x_{iu}, x_{lu})} \qquad (1)$$

Here, $\delta(x_{iu}, x_{lu}) = 0$ if $x_{iu} = x_{lu}$ and $\delta(x_{iu}, x_{lu}) = 1$ if $x_{iu} \neq x_{lu}$. γ is a weight parameter.

Another difference between k-prototype and k-means is 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. This method works for numerical attributes, but not categorical ones. Instead, k-prototype computes the mode value for a qualitative attribute across the entities as the value for the centroid's attribute, as in [16]. Note that to compute mode we need to compute the frequency of each distinct value of a categorical attribute in a cluster.

Algorithm 1 : `ckPrototypes()`

```
initialize cluster centroids to random values
finished = false
while not finished do
    for all entities e do
        assign e to nearest cluster
    end for
    for all clusters do
        set new centroid to average (for numerical attributes)
        and mode (for categorical attributes) of contained
        entities
    end for
    if distance(old centroids, new centroids) < t then
        finished = true
    end if
    centroids = new centroids
end while
```

B. Additive Secret Sharing

Our solution uses additive secret sharing [11]. We selected additive secret sharing instead of other secret sharing methods such as Shamir's secret sharing [18] because of its low computation overhead. 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 in the local data-set of a party. We implemented additive secret sharing by making shares of both the frequencies of distinct values of the categorical attributes (necessary to compute the mode) and values of timestamps and IP addresses (numerical attributes).

The general method used to make n shares of an integer 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 secret s, $n-1$ random shares are created by randomly choosing integers or doubles respectively between 0 and s exclusive, so that:

$$r_n = (s - r_1 - r_2 - ...r_{n-1}) \mod Q$$

The last share is modulo divided by a large prime Q. To reconstruct s, one just needs to sum up all shares and then modulo Q. If a numerical attribute has a floating point number, we can convert it to integer by multiplying a large integer and then rounding it to the closest integer.

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

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

IV. Privacy-Preserving K-Prototypes Clustering for Horizontally Partitioned Mixed Data

A. Horizontally Partitioned K-Prototypes Algorithm

We propose a modified $ckPrototypes$ clustering algorithm to work in a decentralized, multi-party case, with horizontally partitioned data. Each party owns a subset of the full dataset, with each subset having the same structure (attributes). The modified algorithm can be applied to the clustering of IDS alerts across a number of multiple contributing parties each with their own sets of alert data. Our 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 pseudo-code 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), and 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 party p_i do
5:       for all entities e at party p_i do
6:          assign e to nearest cluster
7:       end for
8:    end for
9:    for all clusters c do
10:      for all parties p_i do
11:         compute local sum of numerical attributes in cluster c and frequencies of distinct values for categorical attributes
12:         send these local sums and frequencies to a randomly chosen coordinator
13:      end for
14:      The coordinator computes new centroid as average (for numerical attributes) and mode (for categorical attributes)
15:   end for
16:   if distance(old centroids, new centroids) < t then
17:      finished = true
18:   end if
19:   centroids = new centroids
20: end while
```

B. Privacy Preserving K-Prototypes Clustering Algorithm

We propose $skPrototypes$, a privacy-preserving version of $hkPrototypes$. $skPrototypes$ uses additive secret sharing to protect sensitive values. We used secret sharing because it is more efficient than other cryptographic methods.

In our algorithm, shown in Algorithm 4, the assignment of entries to the nearest cluster is not affected because each party only needs to use its local data. The step to recompute centroids does require information from all parties. In order to make this step privacy preserving, the calculation of cluster centroids is done with additive secret sharing as defined as secureEntitySumProtocol in Algorithm 5.

In this secure sum protocol, each party computes the local sum of each of the features of the local entities within a given cluster (Lines 1 to 7). For numerical attributes, the local sum is simply the sum of each attribute's values. For a categorical attribute, the local sum is the frequency of each distinct value of that attribute.

Next, each party splits each entry in the local sum into n random shares, where p is equal to the number of parties (line 8). $p-1$ shares are then distributed to other parties. Each party then computes the intermediate sum by adding up all of the shares it has received from all of the other parties as well as its local share. Finally, all parties, other than a randomly chosen coordinator, send their intermediate sum to the coordinator. The coordinator computes the final sum by adding up all of the party's calculated intermediate sums. The secureEntitySumProtocol is repeated for all clusters to calculate the final sums of each cluster. For numerical attributes, the final sums are then divided by the total number of entities within each respective cluster to calculate the resulting centroids. For categorical attributes, the algorithm selects the distinct value with highest frequency (i.e., the mode) for the new centroid.

Figure 1 shows an example of $skPrototypes$. There are three parties and two attributes are shown: timestamp and rule ID. We only show the values for the ruleID that appear in the dataset. The example shows one cluster, where rule ID has two distinct values (Snort rule 1003 and 957) in the cluster. Each party has a local sum computed for the numerical attribute (timestamp), as well as a count for each distinct value for categorical attributes. The random shares are also shown below the local sum. Party 1 will send the second random share (the second row in its random share table) to party 2 and send the third random share to party 3. Party 2 and 3 will do the same. Each party then computes intermediate sum based on its own random share (e.g., row 1 for party 1) and random shares received from other parties. Finally these intermediate sums will be sent to a randomly chosen coordinator to compute the final sum.

Algorithm 4 : `skPrototypes()`

initialize cluster centroids to random values
while not finished **do**
  for all party p_i **do**
    for all entities e at party p_i **do**
      assign e to nearest cluster
    end for
  end for
  for all clusters c **do**
    c.centroid = secureEntitySumProtocol(c) / (number of Entities assigned to c)
  end for
  if distance(old centroids, new centroids) $< t$ **then**
    finished = true
  end if
  centroids = new centroids
end while

Algorithm 5 : `secureEntitySumProtocol`(Cluster c)

for all Party p_i **do**
  Initialize localSum as an array of zeros, with one entry for each numerical attribute and one entry for each distinct value of a categorical attribute
  for all Entities e on p_i **do**
    if e.isAssignedTo(c) **then**
      for numerical attributes, add e's value to the corresponding entry in localSum; for categorical attributes, add one to the entry corresponding to the value of e.
    end if
  end for
  Entity[] shares = makeShares(localSum, numberOfParties)
  for all Other Parties p_j **do**
    party p_i sends shares[j] to p_j
    p_j.intermediateSum += shares from party p_i
  end for
end for
for all Parties p_i other than a coordinator o **do**
  send p_i.intermediateSum to coordinator o
  coordinator o computes finalSum += p.intermediateSum
end for
return finalSum from o

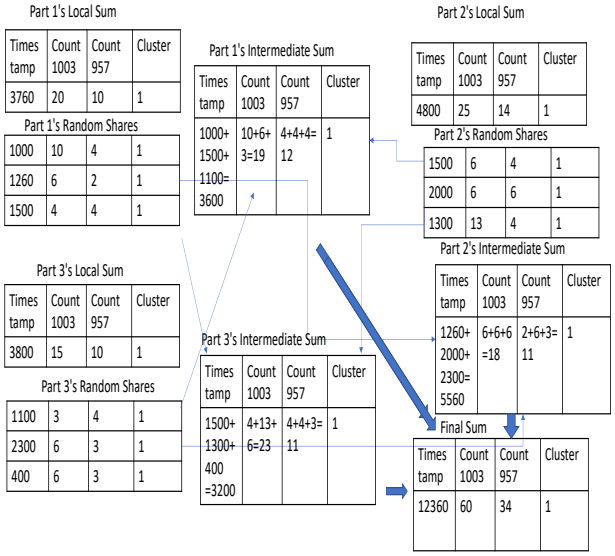

Fig. 1. Example for $skPrototypes$

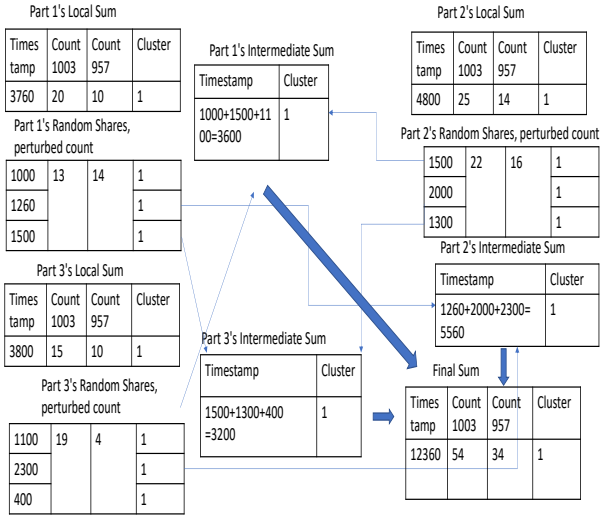

Fig. 2. Example for $dpkPrototypes$

C. Using Differential Privacy for Categorical Data

One problem of $skPrototypes$ is the communication overhead for computing secret shares of categorical attributes. For a numerical attribute, only p shares need to be sent by each party. For a categorical attribute with d distinct values, each party needs to send dp shares where p is the number of parties. The overall communication overhead for a categorical attribute for one iteration of clustering is $O(dp^2)$ because there are p parties. The communication overhead can be high for categorical attributes with many distinct values. Security alerts typically contain many categorical attributes and some of them (e.g., port number) have many distinct values.

To address the communication overhead issue, we propose another algorithm, $dpkPrototypes$ which uses differential privacy for the categorical attributes. This algorithm works exactly as $skPrototypes$ for the numerical attributes, but uses an approximate frequency for each distinct categorical value and sends this approximate frequency directly to all other parties, so there is no need to send random shares for the categorical attributes. As a result, the communication overhead

is reduced from $O(p^2d)$ to $O(pd)$ for each categorical attribute.

The pseudo-code for the $dpkPrototypes$ algorithm is the same as for $skPrototypes$ shown in Algorithm 4, but instead of using $secureEntitySumProtocol$ in line 9, it uses the $differentiallySecureEntitySumProtocol$ shown in Algorithm 6. Algorithm 6 describes our method of computing a new centroid using both secure sharing and differential privacy. For numerical attributes, secure sharing is used. For categorical attributes, noise is added to frequency of a distinct value v, based on privacy parameter α.

We use a noise following truncated two-sided geometric distribution [15]. The two-sided geometric distribution is defined as below:

$$Pr[Z = z] = \frac{1-\alpha}{1+\alpha}\alpha^{|z|} \quad (2)$$

Algorithm 7 shows the details of how the noise is added. First, a random number x in the range of 0 to 1 is generated (line 1). If x is less than $\frac{1-\alpha}{1+\alpha}$, the algorithm returns the original frequency (line 4) because this is the probability when noise $z = 0$ in Equation 2. Otherwise, it will compute the new frequency. In line 7 another random number u is generated and line 8 computes $x = Ceiling(log(u)/log(1-p))$, which follows standard geometric distribution based on inverse transform sampling [19], a classical method to generate random numbers following a distribution. Lines 9 to 12 flip x's sign to negative with 0.5 probability so as to make x follow a two sided geometric distribution.

The remaining issue is that noise needs to be truncated such that the perturbed frequency must be in the range of zero to maximal possible frequency (i.e., number of entities in that cluster). This is resolved in lines 13 to 20 where the perturbed frequency is truncated to zero or the maximal frequency if it is less than zero or greater than maximal frequency, thereby enforcing α differential privacy according to [15].

Figure 2 shows an example of $dpkPrototypes$. The setup is the same as Figure 1. The difference is that now for categorical attributes (rule ID here), only perturbed count is computed for each distinct value of categorical attributes. For instance, at Party 1 the perturbed count for rule 1003 is now 13 instead of the original count 20. Only random shares of numerical attributes need to be sent to other parties. For categorical attributes, the perturbed count will be directly sent to the chosen coordinator to compute final sum. In this example, although the final count of rule 1003 and 957 are different from the correct count shown in Figure 1, the relative order between these two counts is still preserved so rule 1003 will be still the mode for the new cluster centroid.

Algorithm 6 : `differentiallySecureEntitySumProtocol`(Cluster c)

1: **for all** Party p_i **do**
2: Initialize localSum as an array of zeros, with one entry for each numerical attribute and one entry for each distinct value of a categorical attribute
3: **for all** Entities e on p_i **do**
4: **if** e.isAssignedTo(c) **then**
5: for numerical attributes, add e's value to the corresponding entry in localSum; for categorical attributes, add one to the entry corresponding to the value of e.
6: **end if**
7: **end for**
8: **for all** categorical attribute A **do**
9: **for all** distinct value v of A at party p_i **do**
10: set frequency of v = perturb(frequency of v from localSum)
11: **end for**
12: **end for**
13: Entity[] shares = makeSharesNumericalOnly(localSum, numberOfParties)
14: **for all** Other Parties p_j **do**
15: party p_i sends shares[j] for numerical attributes to p_j
16: p_j.intermediateSum += shares from party p_i
17: **end for**
18: **end for**
19: **for all** Parties p_i other than a coordinator o **do**
20: for numerical attributes, p_i sends p_i.intermediateSum to coordinator o
21: coordinator o computes finalSum += p_i.intermediateSum
22: for categorical attributes, p_i sends perturbed frequency for each seen value v directly to o
23: o computes sum of approximate frequencies for each v
24: **end for**
25: **return** finalSum and sum of approximate frequencies from o

V. Algorithm Cost Analysis

In this section we consider the costs of our algorithms. Let n be the number of entities, k be number of clusters, p be number of parties, m_n be the number of numerical attributes and m_c be the number of categorical attributes. At each iteration, the $hkPrototypes$ method needs to compute distances between every data entity to cluster centroids. For numerical attributes, the cost is $O(m_n)$ for each distance computation. For categorical attributes, the cost is $O(m_c)$. So the overall cost of distance computation is $O(nk(m_n + m_c))$ in each iteration. Computing the local sum takes $O(nm_n)$ on numerical attributes and $O(nm_c)$ for categorical attributes if a hash table is used to keep every distinct value of a categorical attribute (so it costs constant time to increment frequency for each value of a categorical attribute). For each numerical attribute at each party, the local sum of k clusters needs to be sent to the coordinator, so the communication cost is $O(km_np)$. For each categorical attribute, the frequency of every distinct value in each cluster needs to be sent to the coordinator. As a result, the communication cost for categorical attributes is $O(km_cdp)$. This process repeats for i iterations. Therefore, the communication overhead is $O(k(m_n+m_cd)pi)$. The coordinator needs to sum up the received local sums and frequencies of each distinct value from each party, which costs

Algorithm 7 : `perturb`(frequency c)

```
x = RandomDouble()
check = (1 - alpha) / (1 + alpha)
if x < check then
    return c
else
    p = 1-alpha
    u = RandomDouble()
    x = Ceiling(log(u)/log(1-p))
    posOrNeg = RandomDouble()
    if posOrNeg <= .5 then
        x = x*(-1)
    end if
    if x + c < 0 then
        newFrequency = 0
    else if x + c > maxFrequency then
        newFrequency = maxFrequency
    else
        newFrequency = c + x
    end if
end if
return newFrequency
```

$O(k(m_n + m_c d)p)$. So the overall computation complexity is $O(kn(m_n + m_c)i + k(m_n + m_c d)pi)$.

$skPrototypes$ differs from $hkPrototypes$ in computing the new centroids. It needs to split every local sum for numerical attributes in a cluster or frequency for a distinct value of a categorical attribute in a cluster into p shares, and send $p-1$ shares to different parties. So the computation overhead is $O(kn(m_n + m_c)i + k(m_n + m_c d)p^2 i)$. The communication overhead is $O(k(m_n + m_c d)p^2 i)$. The p^2 term in both formulas exists because every party needs to send $p-1$ shares. So if d and p are large (i.e., categorical attributes have many distinct values and there are many parties), $skPrototypes$ will become quite expensive.

$dpkPrototypes$ differs from $skPrototypes$ on how it processes categorical attributes. Instead of using additive secret sharing, it adds noise to the frequency of each distinct value. So its computation complexity is $O(kn(m_n + m_c)i + km_n p^2 i + km_c dpi)$. This is lower than the computation complexity of $skPrototypes$ as the p^2 term for categorical attributes becomes p. Its communication complexity is $O(km_n p^2 i + km_c dpi)$, which is also lower than that of $skPrototypes$.

VI. Privacy Discussion

We assume a semi-honest model where each party follows the protocol but each party might want to infer more information. This is a common assumption for most secure multi-party computation methods.

What is protected: Our methods protect values of numerical attributes, as well as the frequencies of occurrences of individual categorical features through secret sharing or differential privacy. The security of additive secret sharing of an integer value is explained in [11] and any adversaries with fewer than $p-1$ shares cannot reconstruct the original values. In $dpkPrototypes$, differential privacy is used to protect the frequency of distinct values of categorical attributes.

What is revealed: Our methods reveal 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 random shares (in the case of secret sharing) or the approximated (via differential privacy) frequency of each value for each category attribute. This allows a semi-honest party to discern from a share whether a certain categorical value occurs within the data set of a party which sends a share. However, the exact frequency of a categorical value 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.

VII. Experimental Evaluation

A. Set Up

Software/Hardware: We ran our experiments on a set of distributed machines connected via a central server on a local area network. The machines and server were each running Ubuntu 18.04.5 LTS(64bit), with 16GB RAM, and about 100GB hard disk. The following experiments were run on a distributed system, so the results do reflect the time delay expected by network latency in a real-life application.

Algorithms Implemented: We implemented $skPrototypes$, $hkPrototypes$, and $dpkPrototypes$ clustering algorithms by modifying the Java implementation of the centralized k-means clustering algorithm found in [20].

Data: The following experiments were run using the full SNORT alert data from [21], which is generated from MACCDC 2012 data. We preprocessed the data to extract features including source IP, destination IP, timestamp, rule ID, source port, destination port, and alert type. All source IPs, destination IPs, and timestamps were converted to numbers. For IP addresses, the dotted quad formatted strings were converted to integers using a the "Power of 256" approach as described in [22]. 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. The numerical attributes were later converted to integers in secret sharing by multiplying by a large integer. Rule ID, port numbers, and alert types were treated as categorical features. In the case of rule IDs and port numbers, the integer values of these features are simply compared for equality. Since alert type is a plain text, Java's hash-map data structure was 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 algorithms, three separate machines, each representing a party, executed the various CIDS algorithms by making three random shares of the full SNORT alert data-set from [21].

Methodology: To conduct these experiments, each of the algorithms was run 10 times for each configuration and we took the average execution time.

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

Performance Metrics: We use bytes of data shared during the clustering process and average execution time of the clustering program measured in milliseconds.

B. Experimental Results

Execution time with Varied number of clusters: Figure 3 shows execution time, in milliseconds, of our three algorithms with varying numbers of clusters ranging between three and eight, and with five parties involved. As shown in Figure 3, $hkPrototypes$ is faster than $skPrototypes$ and $dpkPrototypes$, which is expected as the former does not protect privacy. Surprisingly, $dpkPrototypes$ is slightly slower than $skPrototypes$. We dug deeper and found that although $dpKprototypes$ is faster than $skPrototypes$ in each iteration of the clustering algorithm, the noise added to make the result differential private also leads to slower convergence and more iterations are needed for $dpkPrototypes$. All three algorithms' execution time also increases with number of clusters. The increase seems to be super linear, possibly because when number of clusters increases, the clustering algorithms also converge slower.

Bytes transferred with varied Cluster Counts: Figure 4 shows the relationship between the total number of bytes transferred during the clustering process versus the number of clusters. Five parties are involved. skPrototypes and dpkPrototypes transfer significantly more bytes 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.

Again it is surprising that dpkPrototypes transfers more Bytes than skPrototypes. dpkPrototypes is expected to transfer fewer bytes because it uses differential privacy on categorical attributes and thus does not need to transfer multiple shares for such attributes. We found that the noise added to enforce differential privacy led to slower convergence and more iterations. Figure 5 shows the number of bytes transferred per iteration and it is clear that $dpkPrototypes$ transferred slightly less data than $skPrototypes$.

Varying privacy parameter ϵ: In Algorithm 7, the α is actually determined from the function $e^{-\epsilon}$ where ϵ is the privacy budget set by the user. The greater the ϵ, the smaller the α and the smaller the noise added to the original frequency. Additionally, the smaller the α, the more quickly the perturbed data will converge in the clustering algorithm, thereby minimizing the number of iterations. This relationship can be seen in Figure 6. Given a fixed number of clusters (in this case, seven), differing values for ϵ resulted in variable iteration counts for $dpkPrototypes$. hkPrototypes and skPrototypes do not require any ϵ value as no noise is being added so we just show the number of iterations for $skPrototypes$ as a reference. As shown, the greater the ϵ value, the less the data will be perturbed meaning it will converge faster. The results also show that $dpkPrototypes$ lead to more iteractions than $skPrototypes$ for most ϵ values except when $\epsilon = 1$.

Bytes transferred with varied number of clusters and number of parties Figure 7 shows the Bytes transferred during the clustering process when varying the number of clusters and parties. The experiment was run with the number of clusters varying between three and eight, and number of parties varying between three and five. In addition, the algorithm tested on was skPrototypes. This shows the number of bytes transferred increases with number of clusters and number of parties, which is expected. The increase is linear with number of clusters, which is expected based on the analysis in Section V.

Quality of generated clusters: The optimal number of clusters is seven as the sum squared error does not decrease much using more clusters. We manually inspected the cluster centroid of each cluster generated by the different algorithms and these centroids are quite similar across the different algorithms, meaning protecting privacy does not change much of the results. These centroids also represent similar alerts (of similar IP addresses, alert types, etc.). So the results confirm that using collaborative intrusion detection can significantly reduce the number of threats that need further investigation as similar attacks across organizations have been merged.

VIII. Conclusion and Future Work

We implemented two privacy-preserving distributed clustering methods $skPrototypes$ and $dpkPrototypes$, both of which work on mixed types of data and can be used by a CIDS to merge alerts generated from different organizations. Our experiments show that the quality of clusters generated by our methods is close to the clusters generated by methods without privacy protection. Privacy protection does incur extra execution time and communication cost, but the increase is not dramatic.

Our experiments implemented a distributed system by having each party member communicate via a central server. While the computations are distributed, the communications can be bottle necked in the server, and they rely on trusting the server. Further work should limit server usage to establishing network connection to other hosts as well as securing the data connections to ensure privacy. We will also look into how to improve $dpkPrototypes$ to reduce the amount of noise such that the clustering process can converge faster.

References

[1] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM Comput. Surv.*, vol. 47, no. 4, May 2015. [Online]. Available: https://doi.org/10.1145/2716260

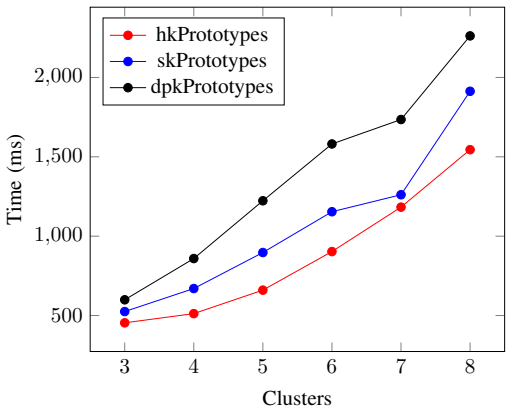

Fig. 3. Execution Time varying number of clusters

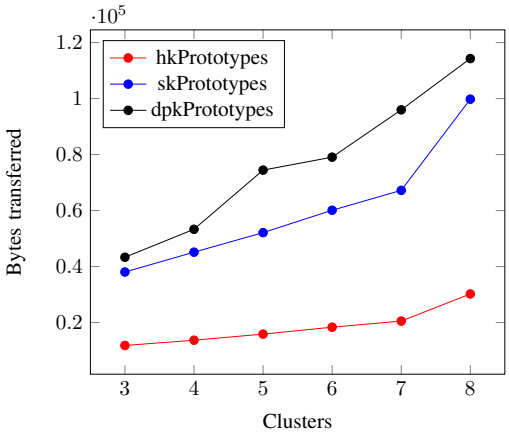

Fig. 4. Bytes transferred vs. Number of clusters

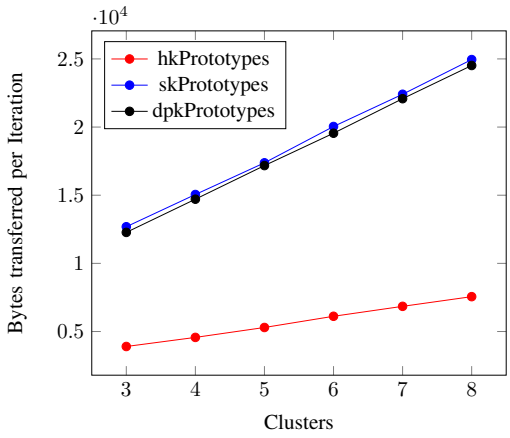

Fig. 5. Bytes transferred per Iteration vs. Cluster Count

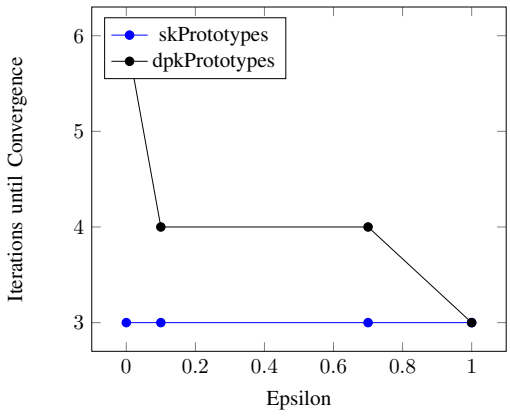

Fig. 6. Selected Epsilon vs. Iterations until Convergence

[2] G. Spathoulas, G. Theodoridis, and G.-P. Damiris, "Using homomorphic encryption for privacy-preserving clustering of intrusion detection alerts," *International Journal of Information Security*, vol. 20, no. 3, pp. 347–370, 2021.

[3] L. Sgaglione, L. Coppolino, S. D'Antonio, G. Mazzeo, L. Romano, D. Cotroneo, and A. Scognamiglio, "Privacy preserving intrusion detection via homomorphic encryption," in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, 2019, pp. 321–326.

[4] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," *Network*, vol. 1, no. 101101, pp. 15–32, 2010.

[5] M. Burkhart and X. Dimitropoulos, "Privacy-preserving distributed network troubleshooting—bridging the gap between theory and practice," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 4, pp. 1–30, 2008.

[6] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: secure multi-party computation on big data," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–18.

[7] F. Meskine and S. N. Bahloul, "Privacy preserving k-means clustering: a survey research." *Int. Arab J. Inf. Technol.*, vol. 9, no. 2, pp. 194–200, 2012.

[8] S. K. Dash, D. P. Mishra, R. Mishra, and S. Dash, "Privacy preserving k-medoids clustering: an approach towards securing data in mobile cloud architecture," in *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*, 2012, pp. 439–443.

[9] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[10] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 12:1–12:19, Jan. 2019. [Online]. Available: http://doi.acm.org/10.1145/3298981

[11] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[12] M. C. Doganay, T. B. Pedersen, Y. Saygin, E. Savaundefined, and A. 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*, ser. PAIS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 3–11. [Online]. Available: https://doi.org/10.1145/1379287.1379291

[13] 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)*, April 2015, pp. 1–6.

[14] C. Dwork, "Differential privacy," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2006, pp. 1–12.

[15] A. Ghosh, T. Roughgarden, and M. Sundararajan, *Universally Utility-Maximizing Privacy Mechanisms*. New York, NY, USA: Association for Computing Machinery, 2009, p. 351–360. [Online]. Available: https://doi.org/10.1145/1536414.1536464

[16] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 283–304, Sep. 1998. [Online]. Available: https://doi.org/10.1023/A:1009769707641

[17] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[18] A. Shamir, "How to share a secret," *Communications of the ACM*,

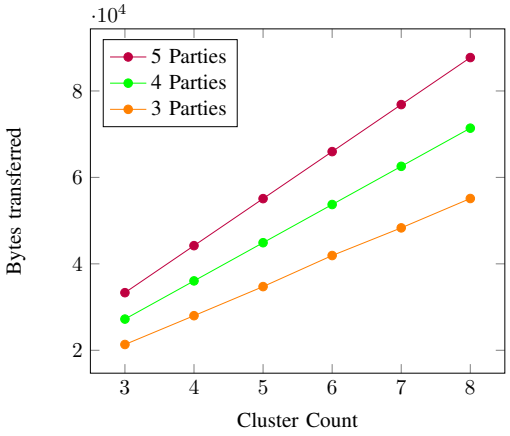

Fig. 7. Bytes transferred vs. Number of Clusters for Varying Number of Parties for skPrototypes

vol. 22, no. 11, pp. 612–613, 1979.

[19] L. Devroye, "Sample-based non-uniform random variate generation," in *Proceedings of the 18th conference on Winter simulation*, 1986, pp. 260–265.

[20] https://www.dataonfocus.com/k-means-clustering-java code/, "K means clustering java code," Last accessed April 2020. [Online]. Available: https://www.dataonfocus.com/k-means-clustering-java-code/

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

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