International Journal of Information Security (2021) 20:347–370
https://doi.org/10.1007/s10207-020-00506-7

REGULAR CONTRIBUTION

Using homomorphic encryption for privacy-preserving clustering of intrusion detection alerts

Georgios Spathoulas[1] · Georgios Theodoridis[1] · Georgios-Paraskevas Damiris[1]

Published online: 13 June 2020

Abstract
Cyber-security attacks are becoming more frequent and more severe day by day. To detect the execution of such attacks, organizations install intrusion detection systems. It would be beneficial for such organizations to collaborate, to better assess the severity and the importance of each detected attack. On the other hand, it is very difficult for them to exchange data, such as network traffic or intrusion detection alerts, due to privacy reasons. A privacy-preserving collaboration system for attack detection is proposed in this paper. Specifically, homomorphic encryption is used to perform alerts clustering at an inter-organizational level, with the use of an honest but curious trusted third party. Results have shown that privacy-preserving clustering of intrusion detection alerts is feasible, with a tolerable performance overhead.

Keywords Intrusion detection · Clustering · Homomorphic encryption · Privacy

1 Introduction

Security has become one of the most important aspects of digital life. As installed systems and services become larger and more complex, their attack surface becomes more difficult to be controlled and protected. Organizations try to protect their digital infrastructure through the use of various countermeasures, and one of those is intrusion detection systems [3, 21]. The main concept in such systems is the analysis of data relevant to the activity of hosts or networks, to detect cyber-security events that occur in the protected system.

Through the last two decades, the security research community has been very active in the intrusion detection domain. Multiple systems, employing various approaches, have been proposed, to increase the detection performance against various attacks. The main taxonomies of intrusion detection systems are network-based systems, which analyze the traffic in a protected network, and host-based systems, which analyze the logs, the integrity checking results and the traffic from a specific host. Additionally, such systems are categorized according to the detection approach they use, with the main options being misuse detection and anomaly detection. Misuse detection-based systems detect predefined patterns of malicious activity and then produce informative alerts. On the other hand, anomaly-based systems model the normal activity and then detect significant deviations from that. The alerts produced by such systems are less informative about the actual attack going on. [18]

While intrusion detection systems detect a significant portion of the attacks executed against an organization, their performance has been proven insufficient [2, 13]. They fail to detect an important percentage of committed attacks, especially when it comes to recent attacks that have not been modeled by misuse based systems [22, 31]. Moreover, they produce a high rate of false-positive results, which are alerts that do not correspond to real attacks [26, 32]. This hinders such systems from providing high-quality representation of the committed attacks, even in the cases that those are successfully detected.

An interesting approach that enables intrusion detection systems to produce better results is collaborative intrusion detection [34, 41], where multiple intrusion detection systems, installed in different organizations, collaborate, to produce more accurate results. The data collected at each

✉ Georgios Spathoulas
gspathoulas@uth.gr

Georgios Theodoridis
gtheodoridis@uth.gr

Georgios-Paraskevas Damiris
gdamiris@uth.gr

[1] Department of Computer Science and Biomedical Informatics, University of Thessaly, Lamia, Greece

system may not be sufficient to detect an attack, while the aggregation of such data can be used to achieve better detection accuracy.

While collaborative intrusion detection seems like a promising approach, its use raises a significant issue related to privacy violations due to the exchange of information between organizations. Independently of the exact information exchanged (this can be raw traffic data or intrusion detection alerts), a significant privacy violation risk is present, which has recently been recognized by the research community [4, 6, 38]. The standard way to process intrusion detection alerts produced by different organizations is to gather such alerts on a single node and commit the required processing there. Irrespective of whether this node belongs to one of the organizations or not, its owner has access to all alerts' fields for all organizations. Those fields contain private data related to network connections. IPs of communications that happen in organizations' networks (e.g., an organization's employee accesses an external server or an organization's client accesses a service offered by the organization) are revealed along with the services (ports) those communications refer to. Additionally, intrusion detection alerts reveal vulnerabilities that exist on those networks or attacks that have already happened against them. Such information is sensitive for both organizations and users interacting with their networks. Gathering all such data on a single node creates considerable privacy risks as the node controller cannot be fully trusted, while at the same time such a node can be a tempting target for malicious attackers, due to the significance of the stored data.

In this paper, a collaborative system for intrusion detection alerts clustering is proposed. It enables the clustering of alerts produced by different organizations, while at the same time it protects the privacy of each organization's data. Inter-organizational clustering can produce a high-level representation of large-scale attacks that may not be detected at the organization level or may reveal multiple instances of the same attack being executed against different organizations. The system consists of a trusted third party that adheres to the honest but curious model and one node per each organization that is locally installed and coupled with a local intrusion detection system. The proposed scheme is based on the use of homomorphic encryption, and the role of the trusted third party is to commit the required processing while having access to encrypted data only. To preserve the privacy of the participating organizations, the homomorphic encryption algorithm Paillier [25], which enables the processing of encrypted data, has been employed. Specifically, all the participating nodes encrypt their alerts and submit those to the trusted third-party server. The latter conducts the clustering, without decrypting the data, and returns the resulting clusters' information to the nodes, in encrypted form. Each node's data are not exposed to the trusted third party or any other node, and all nodes get to know the clustering results. There is an initial step, during which the third party calculates a distance metric for all possible alerts' pairs. Consequently, it commits an iterative procedure for forming the clusters of alerts. The result of this procedure is clusters of similar alerts at an inter-organizational level. At the end of the procedure, the private information of organizations is not revealed nor to other organizations or the trusted third party. The trusted third party gets access only to encrypted data, while the organization's nodes are not able to infer private data from other organizations.

The rest of the paper is structured as follows : Sect. 2 discusses the related work, Sect. 3 describes the main concept or the proposed approach, Sect. 4 presents prerequisite methods, Sect. 5 presents the architecture of the proposed system, Sect. 6 analyzes its implementation, Sect. 7 presents the experiments conducted along with the corresponding results and finally Sect. 8 discusses the main conclusions of the paper.

2 Related work

There has been a lot of work on collaborative intrusion detection systems. Previous research efforts have shown that collaborative intrusion detection systems among multiple partners can be more effective than a single such system installed on the premises of a single organization. Recent research efforts have either highlighted the requirement for collaborative systems or have even proposed such architectures. We present a literature review of such efforts in this Section. Section 2.1 presents the applications of collaborative intrusion detection on different domains, Sect. 2.2 discusses why privacy protection is significant in such schemes and Sect. 2.3 enumerates the recent approaches in privacy-preserving collaborative intrusion detection. Finally, Sect. 2.4 discusses the proposed system regarding related work.

2.1 Collaborative intrusion detection

The requirement for moving on from single installations of intrusion detection agents to networks of collaborating agents installed in different networks has emerged in different domains such as cloud computing, multiple devices networks or detection of large-scale attacks.

Tan and Nagar discussed intrusion detection in cloud computing context [33] and concluded that an enhancement to the security of cloud systems through collaborative intrusion detection is necessary. They propose a cooperative intrusion detection system characterized by fast detection, minimal false positive rates, scalability, self-adaption to changes in the cloud computing environment and resistance

to compromise. The primary components of the proposed architecture are cooperative agents and a central coordinator. While the system they propose seems interesting, they do not thoroughly analyze it, so their contribution is limited. Liang and Ge moved further by presenting a collaborative multilevel intrusion detection system for cloud computing, to achieve more accurate and effective protection in cloud environments [20]. They deploy the intrusion detection system as a lightweight service through a noncentralized architecture between different cloud providers. Specifically, detectors are offered as a service, and machine learning methods are used to create detection rules. Authors also designed a mechanism to exchange alerts between cloud systems and to share knowledge about attacks. Their experimental results have shown that their system enhances security when network attacks occur. Dermott, Shi and Kifayat developed a cloud intrusion detection method for collaborative intrusion detection to be used in a federated cloud environment [8]. They base the proposed approach upon the distribution of responsibilities to set up a more resilient system. The main entities of the system are the cloud broker which offers security services to the rest, the monitoring nodes that act locally in each installation, the local coordinator that manages all local monitoring nodes and the global coordinator which all local coordinators refer to. They used the Dempster–Shafer theory to capture the findings of all monitoring nodes and to make the final decision about an attack. The aim of the system is to enable collaboration among cloud service providers, through a security as a service paradigm, in order for them to be more secure against different cloud threats.

Andreolini, Colajanni and Marchetti introduced a new category of attacks where an attacker breaks down the malicious load in such a way that the most capable intrusion detection systems can detect no part of it [1]. Sending different parts of the payload from different networks enables the attackers to avoid being detected. They then proposed an original detection solution and implemented it as an extension of the Snort system that enables mobile network operators to collaboratively detect such attacks. The proposed scheme allows sharing of internal state information among multiple NIDSs deployed in different networks or network segments. They base the implementation on a lightweight agent and a set of plug-ins handling different protocols; thus, it is characterized by great flexibility in terms of deployment. Their experimental results confirmed the effectiveness of the proposed solution for various protocols at a negligible cost in terms of performance. Morais and Cavalli proposed a distributed intrusion detection systems architecture for wireless mesh networks [23]. In such self-organized and self-configured networks, the nodes need to trust each other since a node depends on intermediate nodes to reach other nodes. Authors detect real-time attacks by analyzing traffic and creating corresponding communication flows. A distributed intrusion detection engine (DIDE) applies restrictions to these flows, while a cooperative consensus mechanism (CCM) performs bad behavior measurements to identify the source of the intrusions. The system was implemented on a virtual mesh network platform, and experimental results have shown that it detects message-based attacks with high accuracy and low resources requirements. Hong and Liu studied the use of collaborative intrusion detection on a network of smart electronic devices [14]. They have had a significant contribution regarding the actual integration of intrusion detection systems with electronic devices, as they have designed and implemented the integrated devices. These devices can monitor and detect abnormal network behavior. They also can work with other neighboring devices to make accurate decisions and detect the origins of attacks. Because of having the intrusion detection system implemented on the hardware layer of the devices, their approach can provide reliable, fault-free and very efficient intrusion detection functionality. In their experiments, a common embedded system was used to measure the proposed system's performance for a power supply network. The results showed that the network of electronic devices worked accurately and efficiently.

Large-scale attacks detection has also been proposed as a domain for collaborative intrusion detection application by Zhou [39]. They propose a scalable decentralized framework which provides a platform for sharing suspicious evidence between participants to detect large-scale attacks at an early stage. Each local node periodically sends evidence collected from its own sub-network to the large-scale intrusion detection service, and it is notified if the evidence has been confirmed as a potential attack. All suspicious evidence is exchanged anonymously. Authors state that if there is sufficient geographical diversity among the participants, then their system can detect stealthy port-scans or worm outbreaks at an early stage. The same authors [40] proposed a multidimensional alert clustering algorithm for extracting important patterns from alerts. They used a two-phase correlation algorithm that first clusters the alerts locally into each IDS and then reports significant alert patterns to a common IDS correlation network. Through a probabilistic approach, they decide when a pattern at the local level is important enough, to use it at the network level. Their experiments have shown that this approach can achieve a significant reduction in the number of false alerts. On the same concept, Francois and others have presented a collaborative system that detects flooding DDoS attacks at the Internet service provider (ISP) level before those reach the victim host [11]. The authors propose a distributed architecture composed of multiple ISPs who collaborate by computing and exchanging belief scores on potential attacks. The calculation of the threat score is based on the overall traffic bandwidth directed to the customer compared to the maximum bandwidth it

supports. The results obtained through experiments show that the proposed system is much more capable of single installations of intrusion detection systems in different ISPs regarding detecting flooding attacks.

2.2 Requirement for privacy

Jin et al. [16] studied the required compromise between privacy and utility, when using a collaborative intrusion detection system, through a game theoretic approach. They proposed a two-level game with one leader and multiple followers. According to their theoretical analysis, it is possible to model the expected behavior of the attacker and the intrusion detection system, and to produce a utility–privacy curve. In addition, Nash equilibrium was proven and an asynchronous dynamic algorithm was proposed to calculate the best collaborative strategies for intrusion detection systems. Finally, through a simulation, they tried to validate their analysis.

The team of Li and Meng [19] created a trust model for intrusion detection systems' networks. They used machine learning techniques to automate assessing trust. Specifically, they tried to enable intrusion detection systems to automatically decide if they should trust other systems or not. For evaluation, they compared the performance of three different supervised classifiers, while they also tested their trust model under different attack scenarios. Their experimental results showed that it is very important to have increased trust between nodes in such networks and a misbehaving node may create large issues for other participants. The proposed trust model can enhance the accuracy of detection of malicious nodes.

2.3 Privacy-preserving approaches

While it is commonly accepted that global collaborative intrusion detection systems can enhance results obtained by local systems, it is not possible to overlook the privacy implications of such schemes. There are references in the literature which are focused on providing privacy-preserving methodologies for collaborative intrusion detection but most of it is work in progress or methods that solve too specific problems.

Multiple approaches are still in the early stages or lack robust technical implementation. Dara and Muralidhara [6] presented the landscape of privacy-preserving collaborative intrusion detection in a position paper and discussed potential architectures. Zhang et al. [38] proposed a secure multiparty computation method to conduct PCA upon data collected from different organizations. Finally, Benali and others presented ideas on privacy-preserving methods to enable the network manager to collect information on the state of the network from different nodes and react to abnormal situations [4]. Authors in [9] presented an approach that is based on storing homomorphically encrypted alerts of different intrusion detection systems on a common infrastructure and then provided the means for checking the similarity between a pair of alerts. While there is no concrete technical presentation of the solution, the method used is interesting and could be used as the basis for more mature solutions.

In other cases, more advanced research efforts focus on more specific problems of the domain. Vasilomanolakis and Krugl analyzed the need to move from the traditionally isolated intrusion detection systems to a large and distributed IDS (CIDS) [35]. They presented a new CIDS approach, which is able to share alarms only on tracking sensors that may communicate with each other. In addition, when data are being distributed, they argue that the system ensures that the data are protected. Authors in [36, 37] proposed a fog-based privacy-preserving approach for distributed signature-based intrusion detection, where they focus on offloading the procedure of conducting signature matching calculations to cloud-based infrastructure. To protect data privacy, they have used Rabin fingerprint algorithm to conduct the required calculations and to prevent the cloud provider from getting access to sensitive data of the installation.

2.4 Our contribution

The review of the relevant literature clarifies that using collaborative intrusion detection structures can bring a significant efficiency improvement regarding traditional systems. Despite the plethora of published work in the domain, few researchers are dealing with privacy leakage in such collaborative systems. Data collected by intrusion detection systems contain sensitive personal information of both the protected organization and the individuals using its services. Exchanging traffic data or the outcome of any kind of processing of such data violates privacy of anyone related to the corresponding traffic flows.

In the present paper, we present one of the first research efforts to develop a complete system, which will enable multiple collaborating intrusion detection systems to unify their results in a privacy-preserving way. The presented workflow allows for a privacy preserving clustering procedure that will end up with clusters of similar alerts between different organizations, without though leaking any private data of their users.

3 Concept

In this and in the subsequent sections, the proposed system is thoroughly described. In the current section, we give the general concept of the system, we present specific examples for its use and we briefly describe its functionality. The main

underlying mathematical concepts are discussed in Sect. 4, and the detailed description of the system's architecture and workflow are presented in Sect. 5.

Current cyber-security landscape is characterized by large-scale attacks during which similar events take place at the same time in different networks or systems. These may either relate to different instances of the same attack executed by the same or different attackers against multiple targets or be part of a single distributed attack (e.g., distributed denial of service attack).

For such cyber-security attacks, the combination of detected events on different networks or organizations could improve the detection accuracy or the assessment of the attacks' severity. An example could be a distributed attack such as the distributed denial of service attempts. Recently, Mirai [17] botnet attacks have created significant problems globally, as attackers took control of many IoT devices and used those to execute a large-scale denial of service attacks. The combination of the IDSs findings for each one of the networks to which the infected devices belonged would enable the timely detection of the attack and would allow to perform more efficient mitigation measures. Apart from that, there are attacks that, because of a recently disclosed vulnerability or the release of new tooling, tend to happen concurrently against different targets across over one organization. Such an example is recent ransomware attacks [10], where the release of such tools triggered a series of similar attack attempts against different organizations. In such cases, a collaborative clustering approach would enable the formation of clusters, representative of the magnitude of the problem. This would reveal the volume of attack attempts and would help to better protect uninfected installations.

The main concept in this paper is the combination of the results got locally at different organizations through their network IDS, which can produce useful information about such attacks and enable the faster and more effective application of mitigation measures. By clustering of alerts produced by different organizations, the produced result will be clusters that comprise similar alerts produced by resembling activity in different networks. A distributed attack or multiple executions of the same attack against different targets will produce a single cluster that will be informative about the ongoing activity.

To commit intrusion detection alerts clustering, the most significant alerts' features that could reflect the similarity between alerts of different organizations have been selected. Features of alerts that are not significant or related to the local installation (e.g., network interface) have not been included as they are irrelevant to the global clustering procedure. Specifically, the features that have been used are:

- *External IP address* Each IDS alert holds the source and destination IP address of the IP packet that has triggered the alert. One of the two that is external to the network of the organization (IP of the host potentially committing the attack) is used as the first alert feature.
- *Alert signature* In signature-based IDSs, each alert is characterized by a signature id which corresponds to the specific signature (rule), upon which the alert has been triggered. Signatures are attack specific, and each one corresponds to a single attack.
- *Alert class* Each alert also carries a class feature which corresponds to the generic attack type the specific alert belongs to. Such classes correspond to broad categories of attacks such as attempted administrator privilege gain, attempted denial of service, detection of a network scan or access to a potentially vulnerable web application.
- *Time stamp* Each alert holds a time stamp which corresponds to the exact time of the packet that has triggered the alert. This is very useful for correlating events that happen concurrently in different networks, as different parts of a large scale attack.

We use the aforementioned features, to compute a distance metric between pairs of alerts that is directly related to their similarity. Upon this distance metric, the proposed method creates clusters of similar alerts that can reveal information about attacks that affect over one organization at the same time. While bringing together on a single host, the features for each alert of each organization would enable the execution of any clustering algorithm and it would also raise significant privacy concerns. The alerts are related to the network traffic of the organization and could reveal sensitive information about its users.

To overcome such concerns, an alternative approach, based on homomorphic encryption, has been chosen. Homomorphic encryption enables the processing of encrypted data and the production of encrypted results. Fully homomorphic encryption techniques offer more flexibility regarding the processing allowed, but are not yet efficient enough, to be used in large-scale computations. Partially homomorphic encryption algorithms impose restrictions on the calculations that can be carried on encrypted data, but are more efficient. Paillier encryption algorithm has been chosen as it offers a good balance between calculations feasibility and processing efficiency.

There is a trusted third-party server that functions as the main point for the procedure, while there is a client at each organization, which is coupled with a local IDS. Each client of the system (installed at each organization) gathers alerts produced locally for a specific time window. Clients share a common pair of Paillier encryption keys. They encrypt all the data they send to the server through the use of this key pair. Only encrypted data are sent to the server and any processing that happens there is on encrypted data.

Consequently, the trusted third-party server commits the clustering procedure for the global set of encrypted alerts.

The process comprises an initial phase which is the calculation of the encrypted distance between every pair of alerts and an iterative execution of the *K*-medoids clustering algorithm. *K*-medoids is a variation of *K*-means algorithm, where the center of each cluster is always one of the existing data points. This variation minimizes the distance calculation requirements, as all distances that may be required during the process have been already calculated in the initial phase of our protocol.

Because of the limitations for the processing that can be carried out on encrypted data, as those will be analyzed in Sect. 4.2, there are specific points in the workflow at which the server cannot conduct the required calculations. In such cases, the server randomly picks a client, sends the input data (in encrypted form) to it and asks for the result. The client decrypts the data, commits the required processing, encrypts the result and sends that back to the server.

At the end of the procedure, encrypted information about the formed clusters is sent to the clients from the server. The clients can decrypt this information by using the shared Paillier keys' pair. This procedure is then repeated for the alerts gathered during the subsequent time window.

The following sections present the proposed method, and specifically, Sect. 4 discusses both *K*-medoids algorithm and Paillier encryption algorithm. Section 5 presents the high-level design behind the proposed approach, and Sect. 6 contains all the implementation details.

4 Prerequisites

4.1 *K*-medoids

The *K*-medoids algorithm is a clustering algorithm that combines both *K*-means and medoidshift algorithms [30]. *K*-means and *K*-medoids algorithms are very similar as they both break down the set of data points into groups and then try to minimize the distance between the points belonging to each group and its center, by shifting points among groups. The main difference between the two algorithms is that *K*-medoids selects existing data points (medoids) as clusters' centers, while k-means calculates an optimal center point for each cluster. Additionally, *K*-medoids typically uses Manhattan distance, while k-means uses Euclidean distance.

K-medoids uses only distances between existing points, without generating new center points at each iteration. This characteristic makes *K*-medoids an appropriate choice for the proposed scheme, as it enables the calculation of distances for all pairs of points in advance. Then, during the iterative *K*-medoids rounds, the algorithm uses these precalculated distances and skips distance calculations.

The main steps of the algorithm are:

- Arbitrary selection of k points as the initial clusters' centers (medoids) of the individual clusters.
- The procedure, with the following individual steps, is then repeated:
  - Each one of the points is associated with the cluster represented by the nearest medoid
  - For each medoid o_j and for each object o_{random} belonging to this:
    - Calculate the total cost S of the o_j exchange with o_{random}
    - If $S < 0$ replace o_j with o_{random}
- The iterative process is completed when there is no change regarding the previous rounds.

4.2 Paillier

Paillier cryptosystem [25] is a probabilistic asymmetric public key cryptographic algorithm. It is characterized by an additive homomorphic property, so given the public key and two encrypted numbers **e1** and **e2**, one can calculate the encrypted result of their addition **e1 + e2**, with no decryption taking place. Key generation for Paillier algorithm is based on randomly choosing two large prime numbers p and q.

4.2.1 Encryption: decryption

Encryption procedure is characterized by a probabilistic property, as in order to encrypt a plain message m the formula is :

$$c = g^m \cdot r^n \mod n^2 \tag{1}$$

where g is part of the public key and r is randomly selected, $0 \leq r < n$

The decryption of ciphertext c is conducted by the formula :

$$e = L(c^\lambda \bmod n^2) \cdot \mu \bmod n \tag{2}$$

where λ and μ are parts of the private key and $L(x) = \frac{x-1}{n}$.

4.2.2 Homomorphic properties

The most notable features of the Paillier cryptosystem is the homomorphic properties it is characterized by and its nondeterministic encryption function. The encryption function is additionally homomorphic, and the following three properties enable partial processing of encrypted data :

– *Homomorphic addition of encrypted values*
  It is possible to add two encrypted numbers, by calculating the product of the two corresponding ciphertexts. The result can then be decrypted into the sum of the respective plaintexts.

$$D(E(e_1, r_1) \cdot E(e_2, r_2) \bmod n^2) = e_1 + e_2 \bmod n \tag{3}$$

– *Homomorphic addition of an encrypted value and a non encrypted value*
  It is possible to add an encrypted number with a nonencrypted one. The product of a ciphertext and the g-based exponential of the nonencrypted number can then be decrypted into the sum of the plaintext and the nonencrypted number.

$$D(E(e_1, r_1) \cdot g^{e_2} \bmod n^2) = e_1 + e_2 \bmod n \tag{4}$$

– *Homomorphic multiplication of an encrypted value and a nonencrypted value*
  It is possible to multiply an encrypted number by a nonencrypted one. If ciphertext is raised in the power of a nonencrypted number, then the result will be decrypted in the product of the plaintext and the nonencrypted number.

$$D(E(e_1, r_1)^{e_2} \bmod n^2) = e_1 e_2 \bmod n \tag{5}$$

– *Homomorphic opposite of an encrypted value* Additionally, it is possible to calculate the encrypted opposite of an encrypted number. Because of the structure of Paillier algorithm, this can happen by calculating the multiplicative inverse of the encrypted number with respect to n^2. This property can be used to conduct subtraction between two encrypted numbers.

$$D(E(e)^{-1} \bmod n^2) = -e \bmod n \tag{6}$$

On the other hand, Paillier algorithm is partially homomorphic as given two ciphertexts, there is no way to calculate the encryption of the product of the corresponding plaintexts, without knowing the private key. Additionally, there is no way to make a comparison of two encrypted values and decide which one of the two is larger than the other. These are the main limitations of the algorithm with respect to the proposed method requirements. As we will analyze it in subsequent sections, it has been chosen to outsource infeasible calculations to clients, while the privacy of the values is still protected by obfuscation. The mechanism to achieve this is described in Sect. 5.5.

Finally, another interesting feature of the algorithm is the nondeterministic encryption. The integer r used during encryption is randomly selected, and that results into different ciphertexts produced by successive encryptions of the same plaintext. This property is important as it prevents anyone who has access to the ciphertexts of two identical plaintexts, to perceive this equality.

We have chosen the Paillier encryption algorithm for the system proposed in the present paper, as it enables for calculations required for alerts clustering to be done while the alerts' data remain in encrypted form. To give an example for that, we assume that the data fields of two distinct alerts that belong to two different parties (organizations) have been encrypted with the same Paillier public key. As it will be further analyzed, in Sect. 5.2, the distance of the two alerts (inversely proportional to the their similarity) is calculated as a function of the values of the alert's data fields.

Calculating the aforementioned distance would require one of the two parties to send the fields of the alert to the other which could then decrypt the received data and carry on the required calculations. That would directly violate the data privacy for the first party. An alternative option because of the homomorphic properties of Paillier algorithm would be to have a third party that does not possess the corresponding Paillier private key but only the public one. In that case, and given that the two parties have share the encrypted alerts' data with the third party, the latter can proceed with the calculation of the distance of the alerts by using the properties of Eqs. 3, 4, 5 and 6. The feasible calculations are not unlimited, but by structuring the distance calculation formula appropriately it is feasible to conduct the required calculations with minimum data privacy loss. Thus, Paillier algorithm can be one of the main building blocks of a privacy-preserving alert clustering algorithm.

5 System design

5.1 Architecture

The proposed system enables multiple organizations to collaborate on processing the alerts produced by their intrusion detection systems, to make more meaningful conclusions out of those alerts. Specifically, the organizations can perform alerts' clustering at an inter-organization level and produce global clusters that correspond to events that affect over one of them at the same time. The system comprises two subsystems, the client subsystem that is installed at every participating organization and the server subsystem that is installed at a trusted third party. These subsystems collaborate, to commit the inter-organization clustering of the alerts.

The clients and the server commit the clustering for the alerts produced by the local IDSs for a given time window. When this procedure ends, it is repeated for the alerts of the next time window. The approach periodically produces the clusters of alerts of all organizations for the current time window. The size of the time window t_w is configurable and

is related to the volume of alerts (which is on average proportional to the volume of traffic) of the organizations.

Each one of the client subsystems is tightly coupled with a locally installed IDS. At the end of time windows, a client subsystem initially reads the alerts produced by the IDS and extracts the four main features for each alert, as described in Sect. 3, External IP address, Signature, Class and Time stamp. It then encrypts the values of these features by using the Paillier algorithm, and the resulting encrypted data are fed to the server, as a vector of four elements, to be used as input for the global clustering procedure. Throughout this procedure, the clients support the server, as some of the required operations are not feasible on the encrypted alert data. Finally, encrypted information about the produced clusters is returned to the clients, where it can be decrypted, in order for each participating organization to get access to it.

The main processing, regarding alerts' clustering, is executed on the server subsystem, which is installed at a trusted third party. It receives the four encrypted features for each alert from the clients, and it executes the *K*-medoids clustering algorithm on them. Most of the required operations can be executed on the encrypted data. Whenever an operation is not feasible, because of the encryption, the server requests support from a randomly selected client, to conduct it. The result of the global procedure is encrypted information about the formed alerts' clusters, which is then returned to the clients.

The general architecture of the system is depicted in Fig. 1, while we give a more detailed description of the procedure through the rest of Sect. 5.

5.2 Distance between alerts

The main metric used to conduct the clustering is the distance between alerts. The more similar two alerts are, the less their distance is. So similar alerts eventually end up in the same clusters. The distance between two alerts is calculated upon four distance coefficients, based on the four features presented in Sect. 3.

Specifically, the IP addresses distance coefficient provides the information on the similarity of two IP addresses. The least significant byte of the IP addresses is more important concerning the IPs comparison, and then the other bytes from last to first follow. Thus, a formula that takes into account the bytes in reverse order is required. The signature and class fields are categorical fields, and only an equality (or nonequality) relationship can be justified for their values. Thus, a simple comparison of values for these two features suffices for the distance calculation. The timestamp field is an integer field, and the distance between two values can be calculated by subtracting those. The subtraction and the calculation of the absolute value of the result would well indicate the similarity of two timestamp values. The calculation for each one of the four distance coefficients is presented in the next Subsections.

5.2.1 External IP distance coefficient

The first distance coefficient d^{IP} is decided upon the external IP of each alert. Each client extracts from the IP fields of the alert the IP address that does not belong to the network of the protected organization, denoted as IP_{ext}. This practically is the IP address of the external malicious attacker and may appear in either source or destination IP fields of the alert.

The coefficient is calculated according to the four bytes of the IP address. If the external IP addresses of alert a^i are $\text{IP}^i_{\text{ext}} : x^i_4.x^i_3.x^i_2.x^i_1$, then the distance of IP addresses is

$$d^{\text{ip}}(a_i, a_j) = 2^3 * |x^i_4 - x^j_4| + 2^2 * |x^i_3 - x^j_3| + 2^1 * |x^i_2 - x^j_2| + 2^0 * |x^i_1 - x^j_1| \quad (7)$$

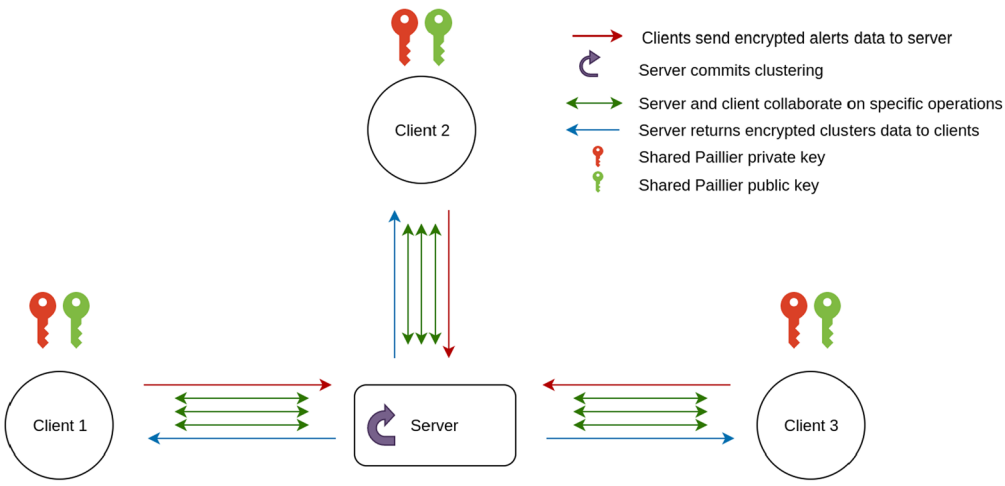

Fig. 1 General architecture of the system

It is obvious that the four bytes of the IP addresses are taken into account with different weights. The IP distance coefficient is an integer in the range $[0, d^{\text{ip}}_{\max}]$, and according to Eq. 7, $d^{\text{ip}}_{\max} = 3850$.

5.2.2 Alert signature distance coefficient

The second coefficient is calculated upon the alert signature (or alert id). This is an integer number that corresponds to the attack detected and is denoted as $\text{alert}_{\text{sig}}$. The signature distance coefficient d^{sig} between two alerts is a binary value that is equal to one if the two alerts share the same signature, while it is zero otherwise.

$$d^{\text{sig}}(a_i, a_j) = \begin{cases} 1 & \text{if } a^i_{\text{sig}} = a^j_{\text{sig}} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Signature distance coefficient is an integer in the range [0, 1].

5.2.3 Alert class distance coefficient

Alerts, besides their signature, are also characterized by a more general class attribute, which denotes a more general attack type. Alerts with different signatures can belong to the same attack type. The class distance coefficient d^{class} between two alerts is a binary value equal to one if the two alerts share the same class, or zero otherwise.

$$d^{\text{class}}(a_i, a_j) = \begin{cases} 1 & \text{if } a^i_{\text{class}} = a^j_{\text{class}} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Alert class distance coefficient is an integer in the range [0, 1].

5.2.4 Alert time distance coefficient

The last distance coefficient is related to the time stamp of alerts, and more specifically to the difference of time stamps of the alerts to be compared. The time difference metric is upper bounded by the time window width t_{w}.

$$d^{\text{time}}(a_i, a_j) = |a^i_{\text{time}} - a^j_{\text{time}}| \tag{10}$$

Time distance coefficient is an integer in the range $[0, t_{\text{w}}]$.

5.2.5 Distance calculation

The distance between two alerts a_i, a_j is denoted as $d(a_i, a_j)$ and is calculated as a combination of the four distance coefficients. It has to be noted that all coefficients are represented by integer values and thus it is feasible to use those within Paillier algorithm calculations. In order for all coefficients to be equally taken into account when calculating the final distance, a normalization step is required. The range for coefficients' values differs significantly as it has been analyzed in the previous subsections. Normalization factors are used to make all coefficients equivalent.

The normalization factors are dependent to the maximum value for each coefficient. Specifically, the normalization factor for signature coefficient is the product of the maximum values of all other three coefficients, as depicted in Eqs. 11 and 12. Similarly, a normalization factor is predefined for each coefficient and shared among clients:

$$d^{\text{sig}}_n = d^{\text{sig}} * nf^{\text{sig}} \tag{11}$$

$$nf^{\text{sig}} = d^{\text{ip}}_m * d^{\text{class}}_m * d^{\text{time}}_m \tag{12}$$

Finally, after the four coefficients have been normalized, the final distance can be calculated according to Eq. 13:

$$\begin{aligned} d(a_i, a_j) = {} & d^{\text{sig}}_n(a_i, a_j) + d^{\text{ip}}_n(a_i, a_j) \\ & + d^{\text{class}}_n(a_i, a_j) + d^{\text{time}}_n(a_i, a_j) \end{aligned} \tag{13}$$

5.3 Clients

All the client subsystems share a common pair of Paillier keys. The private key of this pair shall not be available to the server entity. Clients gather alerts produced by the local IDSs for a time interval, which depends on the volume of traffic and is predecided among all partners. As soon as these alerts are produced for a specific time window, the client subsystems commit two tasks:

- They encrypt the four alerts fields with the common Paillier public key
- They calculate the distances between the locally produced alerts and encrypt the distances with the common Paillier public key. The formula for calculating distance between two alerts is a function of the four main features of the two alerts, and it is analyzed in Sect. 5.2.

Clients send to the server the list of encrypted features of alerts, produced locally, along with the two-dimensional matrix (upper triangle) that contains the encrypted distances between these alerts.

As soon as all clients have sent their input, they become available to support the server. Server carries on with the clustering procedure on the encrypted data. This procedure is further analyzed in Sect. 6. Whenever the server needs to commit an unfeasible operation on a set of encrypted values, it randomly selects a client to send the values to, in order for the operation to be committed. The selected client decrypts

the values with the common Paillier private key, commits the operation, encrypts the result with the common Paillier public key and returns it to the server. Specific measures are taken to protect data privacy, as it is analyzed in Sect. 5.5.

Finally, when the clustering is finalized, clients receive the resulting clusters' information from the server and decrypt it by using the common Paillier private key.

5.4 Server

At the end of each time interval, the server waits for all clients to send their input, four encrypted features of their alerts, along with the encrypted distances of all pairs formed between their alerts. As soon as the server receives all inputs, it calculates the distances between alerts of different clients. When these are calculated, the server builds a matrix with the distances of all the alerts' pairs irrespective of their origin and starts executing the *K*-medoids algorithm.

The server makes use of the Paillier algorithm's homomorphic properties, to commit additions between encrypted values and additions and multiplications between an encrypted value and a non-encrypted one. Whenever the server needs to commit any other operation, it requests support from a randomly selected client, as it has been stated in the previous section. This approach practically overcomes the limitations of partially homomorphic encryption, but has an efficiency cost and may cause a minimal loss of privacy. We further discuss the implications of this approach in Sect. 7.

5.5 Auxiliary client calculations

As it has been stated, clients proceed with the calculations that the server is not able to commit, due to data being encrypted on its side. As it is described in Sect. 4.2, Paillier is a partially homomorphic encryption algorithm, so specific operations are feasible on encrypted data. All such operations happen on the server side. There are some cases, though, when the calculations that have to be conducted are not feasible while the data are encrypted. For such cases, clients offer auxiliary functionality to the server. The server obfuscates the input values, in a way that this obfuscation is reversible on the server side, and sends the data to a random client. The client decrypts the input data, commits the required calculations, encrypts the result and sends it back to the server.

The different auxiliary calculations offered from the clients to the server are:

5.5.1 Multiplication

Multiplication of encrypted values is impossible under the homomorphic properties of Paillier algorithm, so when such a calculation is required, the server uses a client.

Let us assume that the server needs to multiply two values x and y. Before sending those to the client, it adds to both of them random values r_x, r_y where $r_x, r_y > 0$. This is feasible, as server needs to add an encrypted with a nonencrypted value, which can happen as described in Eq. 4 in Sect. 4.2.

The client then decrypts the values, calculates the product $(x + r_x) * (y + r_y)$, encrypts the result and returns it to the server, without being able to know the actual values of x and y. Finally, the server uses the returned value, which is the encrypted $(x + r_x) * (y + r_y)$ product, to calculate the encrypted $x * y$ product, as shown in Eq. 14.

$$x * y = (x + r_x)(y + r_y) - (r_x * y + r_y * x + r_x * r_y) \quad (14)$$

The server has received $(x + r_x)(y + r_y)$ from the client in encrypted form, while it can calculate $(r_x * y + r_y * x + r_x * r_y)$ according to Paillier homomorphic properties.

It is possible to calculate both the product and the sum of a nonencrypted value with an encrypted one, as it is analyzed in Eqs. 4, 5 in Sect. 4.2. Server holds x and y in encrypted form, while it also holds r_x and r_y in nonencrypted form. According to the third Paillier homomorphic property, the server can calculate the encrypted values of $r_x * y$ and $r_y * y$, while it can straightly multiply r_x and r_y to calculate $r_x * r_y$ in nonencrypted form. Finally, the encrypted version of $(r_x * y + r_y * x + r_x * r_y)$ element can be calculated according to the first and the second homomorphic properties of the Paillier algorithm (Eqs. 3, 4).

The server holds an encrypted version of $(x + r_x) * (y + r_y)$ received by the client and an encrypted version of $(r_x * y + r_y * x + r_x * r_y)$ element that has been locally calculated. Finally, the encrypted product $x * y$ is calculated by conducting the subtraction on the right side of Eq. 14. This operation is feasible even if both elements are encrypted, because of the first and the fourth properties of Paillier algorithm (Eqs. 3, 6).

5.5.2 Comparison

Clients are also required to conduct some processing, when server needs to compare values. As it will be analyzed in Sect. 6, the server needs to find the minimum value out of a set of values, while zero values may be required to be excluded. Let us suppose that the server needs to find the minimum element out of a set $[x_1, x_2, \ldots x_n]$ of encrypted values. The client multiplies the elements with a random highly composite value $r, r > 0$ [24] and randomly scrambles the order of the elements to obfuscate the data sent to the client. The multiplication is feasible as the server can multiply encrypted values $[x_1, x_2, \ldots x_n]$ by a nonencrypted one (value of r), according to Eq. 5. The server also keeps in memory the changes in the order of the elements, to revert

the obfuscation at the end of the procedure. The server then sends the obfuscated set to the client.

The client decrypts all values of the set and finds the position of the minimum element (potentially without taking into account the zero valued elements). The multiplication by the r value does not alter the ordering of the elements, but just obfuscates their real values. Zero-valued elements retain the same value even after the multiplication by r. The client sets the value in the selected position of the set equal to one, while it sets the value in all other positions equal to zero. The client encrypts the elements of the produced set and returns it to the server.

The server conducts the reverse of the initial scrambling of order and ends up with a set of encrypted values, all of which are zero, except the one that corresponds to the position of the minimum element of the initial set. It has to be noted that due to the probabilistic property of the Paillier algorithm, the encrypted versions of all zero-valued elements have different values, so it is impossible for the server to learn anything about the result.

5.5.3 Absolute value

If the server needs to calculate the absolute value $|z|$ of an encrypted integer number z, which is bounded as $z \in [-z_m, z_m]$, it also has to communicate with a randomly selected client. The server initially calculates both $(z + a) * b$ and $(-z + a) * b$, where a is a random integer, with $a > z_m$ and b is a random highly composite integer. This is feasible as the server can add or multiply an encrypted number to a nonencrypted one, according to Eqs. 4, 5 of Sect. 4.2. It then sends both $(z + a) * b$ and $(-z + a) * b$, to the client, in random order and in encrypted form

The client decrypts the two received values and answers which one of the two is the largest one. If $(z + a) * b$ is largest then $|z| = z$, otherwise $|z| = -z$. The client does not know the values of the random variables a, b, nor the ordering, so it cannot make any guess about the number, the absolute value of which needs to be calculated.

5.6 Workflow

The workflow for a specific time window is described in this section. K-medoids algorithm operates with a predefined clusters' number, so the server initially defines this number as k (number of clusters). This parameter is decided by the server and is related to both the required granularity of results and the available processing resources. A higher value for the number of clusters would produce more detailed information about ongoing attacks, while it would require more calculations.

5.6.1 Clients send data to server

In the first step of the procedure, clients send initial data to the server. Each client shares the four features of the alerts the local IDS has observed during the previous time window. These data are encrypted using Paillier algorithm before being sent to the server. The server is required to calculate the distances for all pairs of alerts sent from clients.

Distance calculation is a demanding operation in terms of resources. To make the procedure more efficient, each client calculates the distances for the pairs of alerts that have been produced locally, as required features are available in non-encrypted form. The calculated distances are also encrypted and sent to the server. This will save the server from a significant amount of operations. Specifically for n clients, which individually hold approximately m_c alerts, the total number of distances to be calculated is calc_{tot}:

$$\text{calc}_{\text{tot}} = \frac{n * m_c(n * m_c - 1)}{2} \tag{15}$$

The number of calculations that can be carried out locally at the clients is calc_{loc} :

$$\text{calc}_{\text{loc}} = n * \frac{m_c(m_c - 1)}{2} \tag{16}$$

The ratio of these estimated calculations is :

$$\text{ratio} = \frac{\text{calc}_{\text{loc}}}{\text{calc}_{\text{tot}}} = \frac{n * \frac{m_c(m_c-1)}{2}}{\frac{n*m_c(n*m_c-1)}{2}} = \frac{m_c - 1}{n * m_c - 1} \tag{17}$$

Thus, for a relatively small number of clients n and numerous alerts per client m_c (which is the most common case), this ratio tends to be equal to $\frac{1}{n}$, which is a significant percentage of the calculations.

After the first step is finalized, the server holds the alerts of all clients in encrypted form. Additionally, it holds all the distances between alerts that have been produced in the same client, also in encrypted form.

5.6.2 Server calculates the rest of the distances

The second step of the workflow is the calculation of all distances between pairs of alerts that have been produced at different clients. The calculation of the distances must be carried out in the server, while the alerts' features remain encrypted. As it has been noted in Sect. 4.2, Paillier algorithm is characterized by specific homomorphic properties that enable committing additions between two encrypted values and both multiplications and additions between one encrypted and one non encrypted value. Distance calculation demands multiplications between encrypted values at

some point, and these calculations are conducted by returning the pair of encrypted values to a random client. The random client decrypts the values, commits the multiplication, encrypts the result and returns that to the server. The server then carries on with the calculation of the distance between the alerts. At the end of this step, the server holds the distances between all pairs of alerts in encrypted form.

5.6.3 Server commits clustering

As soon as all distances are available, the clustering algorithm can then be executed. As it is described in Sect. 4.1, *K*-medoids algorithm creates clusters, the centers of which are being selected from the points (alerts) to be clustered. The distance of a point (alert) from the center of a cluster is one of the already calculated distances between points (alerts) of the initial dataset. The previous step has produced all the required distances, so the repetitive procedure of formulating the clusters does not require any additional distances to be calculated.

These distances have to be compared to conduct the actual clustering. Each point (alert) is assigned to the cluster, the center of which is the nearest to it. This means that the server has to conduct comparisons between encrypted distance values. Actually, the server has to pick the minimum from a set of encrypted values, so it requests intervention from a random client. The server sends a set of encrypted elements to the client which then picks the minimum one out of these values. In practice, the client returns an encrypted version of a set of elements (equal in size with the submitted set). All values of the returned set are equal to zero, except the one that corresponds to the same index as the minimum value of the set submitted by the server which is set equal to one.

The final step of each round ends up with the allocation of points (alerts) to clusters. This information is being returned to the server by the clients in nonencrypted format. This happens because the server has to check whether there has been any change in this allocation between any two consecutive rounds, to decide that the execution of the *K*-medoids clustering algorithm has reached to an end.

Through this procedure, the server can run all the required rounds for the *K*-medoids algorithm and eventually produce a final allocation of the alerts to the *k* different clusters.

5.6.4 Server returns the results

After the execution of the *K*-medoids algorithm has been terminated, the server can return to the clients information about the security events that have occurred throughout the network in the last time slot. According to the setup chosen, different amounts of information may be revealed. For example, server may reveal to a client:

- only the volume of alerts in a cluster, an alert of the client belongs to.
- the IPs of the alerts in a cluster, an alert of the client belongs to.
- all data about alerts in a cluster, an alert of the client belongs to.
- all data about alerts in all clusters.

According to the information in the cluster results returned, there may be a limited data privacy leak between participating organizations. The amount of information being shared between organizations ranges according to the predefined scheme they have agreed to collaborate on.

For example, it can be the case that when a relatively large cluster involving multiple organizations comes up, then all organizations are notified about the size of the cluster and the set of external IPs that are involved in it. All organizations can better defend their networks without getting to know what has exactly happened to other organizations or specific details about their networks. This is a great improvement in terms of privacy compared to the case of trying to achieve the same result by having a node gathering all alert data from all organizations and conducting the clustering locally.

Another case would be to have a persistent cluster of alerts between two organizations, which could mean that these two may be victims of the same attack because they share the same vulnerability in their networks. In that case, more information could be revealed to the two organizations (e.g., the set of alert's signatures or the set of ports), in order for them to understand what is the exact issue and probably collaborate offline to find a solution. Again this would end up revealing bits of information of the two organizations to each other, but the level of data privacy loss would be minimal compared to the data privacy loss when clustering the data on a single node.

While the scheme under which organizations share data at the end of the procedure is out of the context of this paper, it can be set in a way that the privacy loss risks are minimized.

6 Implementation

In this section, we present a more detailed description of all the operations of the proposed scheme.

6.1 Main parameters

To analyze the implementation of the proposed method, we present the main parameters used in advance. We assume that we have *m* alerts and that the *K*-medoids algorithm will create *k* clusters.

The *distances matrix DIST (mxm)* is a square matrix that contains the distances between all m alerts. Both rows and columns of this matrix correspond to alerts. DIST matrix is symmetric as the distance between two alerts is the same irrespective of the order in which these two alerts are taken into consideration in the calculation.

The *clusters' centers matrix CC (mxk)* is a matrix that holds the information regarding which alert is the center for each cluster. Each row of the matrix corresponds to an alert, and each column of the matrix corresponds to a cluster. Each column contains zeros and only one element, the one in the row that corresponds to the alert being the center of the cluster is equal to one.

The *distance from clusters matrix DFC (mxk)* is a matrix that stores the distance of each alert from each cluster. Each row corresponds to each alert and each column to each cluster. The elements in the row are equal to the distances of the corresponding alert from all clusters (according to the column of the element).

The *belong to cluster matrix BTC (mxk)* is a matrix that stores information about to which cluster each alert belongs to. Each row corresponds to each one alert. Each row holds one element equal to one, which corresponds to the cluster the alert belongs to, while all other elements of the row are equal to zero.

The *clusters groups CG matrix (mxm)* is a square matrix that denotes which alerts are in the same cluster. Each row corresponds to each alert, and each column also corresponds to each alert. An element in this matrix is equal to one, if the alert that corresponds to the row and the alert that corresponds to the column of the element belong to the same cluster. Otherwise, the element's value is zero.

The *distance in groups DG matrix (mxm)* is a square matrix that holds the distance of an alert from all other alerts that belong to the same cluster. Each row corresponds to each alert, and each column also corresponds to each alert. An element in this matrix is equal to the distance of the alerts that correspond to the row and the column of the element if these two alerts belong to the same cluster. Otherwise, the elements value is zero.

The *center metrics vector CM (mx1)* is a one-dimensional vector. Each element belongs to each alert and is equal to the sum of all distances of the specific alert from the alerts with which it belong to the same cluster. In practice, it is a metric used for updating the clusters' centers.

$$cm_i = \sum \text{dist}(a_i, a_j), \forall j : a_j \in c^i \tag{18}$$

where c^i is the cluster to which a_i belongs to.

The *cluster center metric CCM (mxk)* is a matrix, each column of which corresponds to each one of the clusters. Each element stores the calculated center metric for the alert that corresponds to the row of the element, regarding the specific cluster. The values of elements that correspond to alerts that do not belong to the specific column's cluster are equal to zero.

6.2 Calculations

Each client holds a shared pair of private and public Paillier keys. The keys pair is common among all clients in order for the procedure to be feasible, while the server does not have access to this information.

6.2.1 Vectors alerts$_i$ and matrices dist$_i$

Initially, the clients have to submit to the server all the data regarding their alerts along with the distances between the latter. Each client c_i, holding m_i alerts for the specific time window, submits to the server a vector of length m_i with alerts' data:

$$\text{alerts}_i = [ad_1, \ldots ad_{m_i}] \tag{19}$$

where each ad_j is a tuple of data relevant to the specific alert.

$$ad_j = (a^i_{\text{sig}}, a^i_{\text{class}}, a^i_{\text{time}}, x^j_4, x^j_3, x^j_2, x^j_1,) \tag{20}$$

In Eq. 20, a^i_{sig} stands for the signature of the alert. The signature of the alert is represented by a vector equal in length with the possible signature values. It holds zeros, and only the element that corresponds to the signature of the alert is equal to one. Similarly, a^i_{class} stands for the class of the alert and is represented by a vector equal in length with the possible class values. All values are zero except the one that corresponds to the class of the alert, which is equal to one. Regarding time stamp, a^i_{time} is an integer value that corresponds to the time that the alert is produced with regards to the start of the examined time window. The granularity of this parameter (seconds/milliseconds) can be commonly adjusted by the clients, according to their needs. Finally, the $x^j_4, x^j_3, x^j_2, x^j_1$ variables stand for the four coefficients of the external IP address of the alert.

Each client constructs the vector of Eq. 19 for all the alerts monitored by the corresponding IDS sensor in the previous time window. Additionally, the client calculates the distances between the m_i alerts and builds a square matrix of size m_ixm_i denoted as dist$_i$ that contains the results.

$$\text{dist}_i = \begin{bmatrix} d_{11} & d_{12} & \ldots & d_{1m_i} \\ d_{21} & d_{22} & \ldots & d_{2m_i} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m_i1} & d_{m_i2} & \ldots & d_{m_im_i} \end{bmatrix} \tag{21}$$

In Eq. 21, each element d_{ij} corresponds to the distance for alerts a_i, a_j, that has been calculated according to the procedure defined in Sect. 5.2.

Clients encrypt all tuple elements in the vector of alerts and all distances one by one with the common Paillier public key and construct the encrypted versions of vector alerts$_i$ and matrix dist$_i$ denoted as e(alert$_i$) and e(dist$_i$). These encrypted data structures are consequently sent to the server.

6.2.2 Constructing DIST matrix

The server receives a pair of e(alert$_i$) and e(dist$_i$) from each client. When all clients have sent their input, the server can construct the global distances matrix DIST, which holds the distances between all alerts of all clients. The rows and columns of DIST matrix correspond to alerts in the global set. The ordering of these alerts is restricted as they have to be grouped by the client that has produced them. The alerts of a specific client have to appear in a continuous range regarding the indexing of DIST array's rows and columns.

Partial submatrices of DIST matrix have already been calculated by clients in the form of dist$_i$ matrices as presented in Eq. 21. These submatrices are aligned along the DIST matrix diagonal and practically correspond to the distances between alerts submitted from the same client. The rest of the elements that correspond to pairs of alerts produced by different clients have to be calculated by the server. These calculations are conducted by using the encrypted values in the e(alert$_i$) vectors submitted by clients (Eq. 19).

An example is given in Eq. 22, where it is assumed that two clients have submitted three alerts each. Here, the bold elements belong to the two distance matrices the clients have sent, while the rest of the elements have to be calculated by the server.

$$DIST = \begin{bmatrix} \boldsymbol{d_{11}} & \boldsymbol{d_{12}} & \boldsymbol{d_{13}} & d_{14} & d_{15} & d_{16} \\ \boldsymbol{d_{21}} & \boldsymbol{d_{22}} & \boldsymbol{d_{23}} & d_{24} & d_{25} & d_{26} \\ \boldsymbol{d_{31}} & \boldsymbol{d_{32}} & \boldsymbol{d_{33}} & d_{34} & d_{35} & d_{36} \\ d_{41} & d_{42} & d_{43} & \boldsymbol{d_{44}} & \boldsymbol{d_{45}} & \boldsymbol{d_{46}} \\ d_{51} & d_{52} & d_{53} & \boldsymbol{d_{54}} & \boldsymbol{d_{55}} & \boldsymbol{d_{56}} \\ d_{61} & d_{62} & d_{63} & \boldsymbol{d_{64}} & \boldsymbol{d_{65}} & \boldsymbol{d_{66}} \end{bmatrix} \tag{22}$$

The server will use the already available distances between pairs of alerts of the same client to fill in the bold elements. Then, the server will calculate the distances between all other pairs of alerts one by one and fill in the nonbold elements. At the end of the procedure, the server will have the DIST matrix, as described in Sect. 6.1, in encrypted form.

Distance calculation follows what we have described in Sect. 5.2 and is straightforward when it is conducted by the client for pairs of local alerts, given that the required field is available in nonencrypted form. In the server's case, the required data are encrypted, and the server needs to conduct calculations by using Paillier homomorphic properties, while at some points, it is required to use the auxiliary processing offered by the clients.

If the server needs to calculate the distance between two alerts a_i and a_j, the formula of Eq. 13 is used. The server needs to calculate the four normalized distance coefficients. The four normalization factors are predefined and available in nonencrypted form. The server calculates the four distance coefficients in encrypted form, so the calculation of the normalized ones, as shown in Eq. 11 for signature coefficient, is feasible because of the homomorphic Paillier property of Eq. 5. Practically, the server calculates the four distance coefficients, shown in Eqs. 7, 8, 9 and 10, in encrypted form.

The calculation of the signature distance coefficient $d^{\text{sig}}(a_i, a_j)$ is conducted by multiplying the two signature vectors a^i_{sig}, a^j_{sig} element by element and then summing up all the elements of the resulting vector. If the two signature vectors are identical, then the result of the procedure is equal to 1; otherwise, it is equal to 0. During this procedure, the server needs to make additions and multiplications between encrypted values. It is possible to commit the additions, because of the homomorphic Paillier property of Eq. 3; however, to commit the multiplications, the server has to outsource some of the processing to the clients randomly. The clients' auxiliary processing for multiplication, as analyzed in Sect. 5.5.1, is used.

The calculation of the class distance coefficient, denoted as $d^{\text{class}}(a_i, a_j)$, is very similar to that of the signature coefficient, as representations of class and signature alert attributes are identical. The same procedure used by the server for the signature coefficient is also used for the calculation of the class distance coefficient $d^{\text{class}}(a_i, a_j)$.

Regarding the calculation of the $d^{\text{ip}}(a_i, a_j)$ distance, the server needs to commit some subtractions which are feasible even if data are encrypted according to Sect. 4.2. Additionally, it shall decide on the absolute value of integers and in this case it should use clients' auxiliary processing, as analyzed in Sect. 5.5.3.

Finally, regarding the time distance coefficient, denoted as $d^{\text{time}}(a_i, a_j)$, and according to Eq. 10, the server needs to commit a subtraction between two integers, which is feasible because of homophobic properties analyzed in Sect. 4.2, and then decide on an absolute value. For the latter, the server uses the absolute value auxiliary processing offered by a randomly selected client.

6.2.3 A typical *K*-medoids round

After the initial step of calculating all values for the DIST matrix, the server can go on with the actual clustering procedure. This is an iterative process that executes the required K-medoids rounds, until the final clusters of alerts are formalized.

The data used during the execution of the *K*-medoids rounds have been formed as matrices and vectors, as described in Sect. 6.1. This enables the server to commit the required calculations for all data points at once. Each one step of the algorithm is practically encoded as an operation on one or two matrices. For example, the step of the calculation of the distance of each point from the centers of all available clusters is encoded as a multiplication between matrices DIST and CC, as shown in Eq. 23. Each row of DIST matrix contains the distances of a single alert from all other alerts. Each column of CC matrix is related to a specific cluster and holds zeros for all elements except the one that corresponds to its center. The multiplication of the two gives a metric, which is the distance of the specific alert from the center of the specific cluster. The same approach holds for all other steps of the *K*-medoids algorithm which are conducted through the equations described in the rest of this section.

Beforehand, the server populates CC matrix with random values. In practice, the server assigns random points as centers for each cluster. It has to be noted that CC is the only nonencrypted matrix, as the server needs to check at the end of each round whether its values have changed with respect to the previous round.

DFC matrix

At the start of each round, the server calculates the DFC matrix that contains the distance of each point from the center of each cluster, by multiplying matrices CC and DIST. The DIST matrix is encrypted, while the CC array is not. The multiplication of the arrays requires multiplications between nonencrypted and encrypted values, which are feasible and additions between encrypted values, which are also feasible. No auxiliary client processing is required at this step:

$$\mathrm{DFC} = \mathrm{DIST} * \mathrm{CC} \tag{23}$$

Every row of the resulting DFC matrix corresponds to one alert and holds *k* values. These are the distances of the alert represented by the row from each one of the centers of the *k* clusters.

BTC matrix

The next step is to construct the BTC matrix from the DFC matrix. The BTC matrix denotes to which cluster an alert belongs to. The server has to find the minimum value in each row of the DFC matrix (excluding zeros) and set the corresponding value of BTC matrix equal to 1, while setting all other elements of the row equal to zero:

$$btc(i_0, j_0) = \begin{cases} 1 & \text{if } dfc(i_0, j_0) \leq dfc(i_0, j), \forall j \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

Multiple comparisons between the values of the row are required, and these cannot be conducted by the server, as the values are encrypted. In this point, the server makes use of the comparison auxiliary processing offered by clients.

CG matrix

The CG matrix is a square matrix that denotes for each alert a_i, which other alerts belong to the same cluster with it. This can be easily calculated from BTC matrix by multiplying it with its transpose matrix $\mathrm{BTC}^{\intercal}$:

$$\mathrm{CG} = \mathrm{BTC} * \mathrm{BTC}^{\intercal} \tag{25}$$

To commit this calculation, the server needs to conduct multiplications and additions between encrypted values. For the multiplications, it has to use clients' multiplication auxiliary processing. The server can then proceed with the required additions, as it is described in Sect. 4.2.

DG matrix

For the next step, the server has to calculate DG matrix that holds the distances of alerts from all other alerts that belong to the same cluster. This matrix is calculated as the Hadamard product [7, 15] of CG and DIST. The server multiplies CG and DIST matrices, element by element.

$$\mathrm{DG} = \mathrm{CG} \odot \mathrm{DIST} \tag{26}$$

In this step, the server needs to use the clients' multiplication auxiliary processing to conduct the multiplications of encrypted elements.

CM vector

The CM vector holds one value for each alert. It corresponds to the cluster center metric for this alert, with respect to the cluster it belongs to. In practice, for each alert, this metric is the sum of the distances from alerts belonging to the same cluster. The CM vector is calculated by summing up the elements of each row of the DG matrix. Only additions are required so the server can calculate the CM vector:

$$cm(i_0) = \sum (dg(i_0, j)), \forall j \tag{27}$$

CCM matrix

CCM matrix holds metrics that are then used to decide the clusters' centers. To construct the CCM matrix, a diagonal matrix of size mxm that holds the CM vector in its diagonal is required. The server constructs the matrix and then multiplies it by BTC matrix. The diag(CM) matrix can be built element by element by the server. For the multiplication of the matrices, the server needs to use the clients' multiplication auxiliary processing:

$$\mathrm{CCM} = \mathrm{diag}(\mathrm{CM}) * \mathrm{BTC} \tag{28}$$

The resulting matrix CCM holds one column for each one of the k clusters. Each column holds metrics for all alerts of the cluster, and these metrics need to be compared, to choose the minimum value. The elements of the column that correspond to alerts and do not belong to the corresponding cluster are equal to zero.

CC matrix

The final step for the round is to recalculate the CC matrix, which holds the center point for each cluster. In order for this to happen, the server needs to pick the minimum value (excluding zeros) in each column of the CCM matrix. The server uses a slightly modified version of the comparison auxiliary processing provided by clients. Specifically, the returned set of values is not encrypted, in order for the server to construct a nonencrypted CC matrix and compare it with the one from the previous round.

This typical *K*-medoids round is repeated, until the resulting CC matrix is identical with that of the previous round. Then, the clustering algorithm has converged and the server can notify clients about the results.

According to the level of collaboration between the partners, the amount of information returned to the clients may vary. The server holds encrypted alerts information in the format denoted in Eq. 20, along with the distribution of alerts into clusters. So it can notify clients only about large clusters, and it can send just the population of the clusters, the identity of clients, the alerts of which make up a cluster, or the actual encrypted data of the alerts in the cluster.

The workflow of the protocol is depicted in the sequence diagram of Fig. 2. The distinct phases of the proposed algorithm (distances calculation, *K*-medoids rounds and cluster information sharing) are shown in the figure. Additionally, the interaction between the server and the clients is clearly noted for every step of the calculations.

7 Experiments

The proposed method offers privacy-preserving clustering for intrusion detection alerts, but it bears an overhead in terms of performance. It requires multiple additional calculations, while a lot of network communication has to take place. To validate the proposed method, the corresponding implementation has been tested by using a network traffic dataset in a twofold approach. We have proved the validity of the method, as it produced the same results as the corresponding clustering procedure did in nonencrypted space. The performance of the algorithm has also been evaluated as both the execution time and the number of calculations server outsourced to clients have been measured.

7.1 Dataset

An intrusion detection-related traffic dataset that would refer to multiple different networks was required to test the proposed system. Ideally, the traffic of these networks should contain artifacts related to different instances of the same malicious activity or/and to similar attacks conducted in more than one of the networks. Because of not being able to find such a dataset, we have opted for employing a dataset that refers to the network of a single organization with over one subnetworks, and to manage the different subnetworks as different networks monitored by different IDSs. Such an experiment setting would provide the input for multiple clients of the proposed system that would then need to collaborate, along with a trusted third-party server instance, to carry out the privacy-preserving clustering workflow described in Sects. 5 and 6. It is significant for the testing procedure to have as many client as possible, to stress the system and to have a balanced mix of normal and malicious traffic, in order to get a normal alerts' rate. The criteria set for finding out an appropriate dataset were:

- Dataset released after 2010.
- Dataset related to at least five different networks.
- Dataset containing information on packet level.
- Dataset containing general malicious traffic (datasets for specific attacks were excluded)

After going through the recently released intrusion detection datasets [27], only two were found to abide with all the aforementioned criteria, ISCX 2012 [29] and TUIDS [12] datasets. Out of the two, the ISCX 2012 dataset was found to be better documented and more concrete. Because of that, it was chosen to be used to test the proposed method.

The dataset used is UNB ISCX 2012 Intrusion Detection Evaluation Data Set [29]. This dataset is based on the concept of profiles, containing detailed descriptions of intrusions, to simulate malicious traffic. It also contains benign traffic created by abstract distribution models for applications, protocols or lower level network entities.

The traffic it contains is related to a network of an organization with five different inside networks and a DMZ zone, accommodating the organization's servers. The approach used to test the proposed inter-organization clustering system was to handle the six different subnetworks as networks that belong to different organizations. Based on this assumption, network traffic of the six networks is required to be handled in a privacy-preserving way throughout the clustering procedure. The selected dataset provides an appropriate test bed for the proposed method.

Practically, the traffic has been split according to the IP ranges of the six different networks into six different traffic chunks. For each one of the different networks, a combination of a Snort [28] sensor and a client of the proposed system has been used. Each one of the six traffic chunks has been used as input for the corresponding Snort sensor. In this way, we have simulated a collaboration between six different organizations, to test the proposed system. The correlation between the traffic of the different networks in the dataset is relatively high, and thus, this makes it appropriate for testing a clustering method.

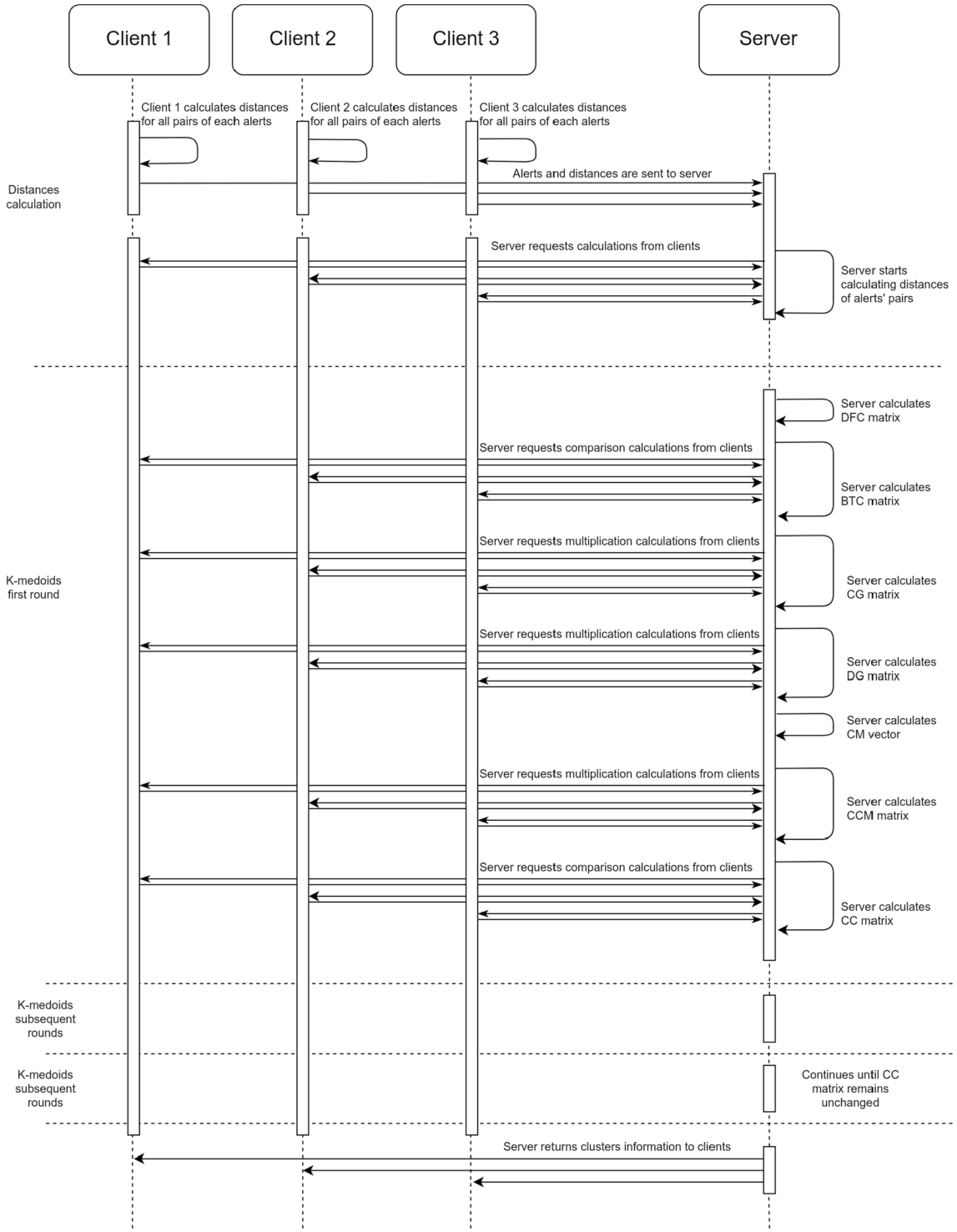

Fig. 2 Sequence diagram

7.2 Experiment setup

Two software implementations have been developed to test the proposed system. We have implemented the client software that runs at each collaborating node, gets as input the alerts, produced by the local intrusion detection sensor, and collaborates with the server. We have also implemented the trusted third-party server component that accepts input from clients and commits the clustering, while it also collaborates with the clients, when it is required.

For implementing the client Python 3.6 has been used, as it provides libraries that enable the development of all the required functionality. The implemented software needs to conduct cryptographic operations, matrix operations, and to efficiently communicate with other parties. Each client of the network along with the trusted third-party client has been executed on a midlevel computer equipped with Intel i5-7200U processor, which consists of four CPU cores. This feature potentially enables the clients to reply to four different calculation requests at the same time.

We have employed multiple different use-case scenarios to assess the performance of the system. Specifically, the number of collaborating nodes has been varied from 3 to 6, while for each one of those cases the number of alerts submitted from each node has been set equal to 10, 60, 30, 90 and 120. Each simulation run corresponds to the clustering procedure for a single time window where a specific number of nodes produce a specific number of alerts. In all these combinations, we have measured the time needed to complete the whole clustering procedure along with the number of operations for which the server requested for the interference of client nodes.

To reproduce the experiment, it is required to split the traffic dataset into six different parts according to the six sub-networks. The next step is to feed the six different traffic sets into a Snort installation to get the produced alerts. Consequently, these alerts shall be used as input to six different client installations that share the same Paillier key pair. The following procedure shall be repeated for each time window:

- Extraction of the four main features for each alert (as described in Sect. 3)
- Calculation of the distance for all alert pairs (as described in Sect. 5.2)
- Encryption of both features and distances (as described in Sect. 4.2)
- Sending of data to the server

Then, the server shall:

- Calculate distances for all alert pairs (as described in Sect. 5 and by using the auxiliary client services as described in Sect. 5.5)
- Build the matrix of distances for all alerts' pairs (as described in Sect. 6.2.2)
- Allocate the alerts to clusters randomly.
- Repeat the *K*-medoids clustering round (as described in Sect. 6.2.3)
- Stop when the clusters remain the same for two consecutive rounds.

7.3 Remote processing estimation

The proposed methodology requires additional processing, to protect the privacy of the alerts submitted for inter-organizational clustering. The main performance overhead is because of the requirement for the server to use remote auxiliary processing offered by clients, presented in Sect. 5.5, when processing encrypted data is not feasible. According to the analysis of the calculations presented in Sects. 6.2.2 and 6.2.3, regarding the construction of the DIST matrix and the execution of a *K*-medoids round, the required remote calculations can be predicted. We assume that the number of clients is n, the number of alerts per client is m_n and the predefined number of clusters is k. The total number of alerts can be calculated as $m = n * m_n$. Additionally, n_{sig} is the number of different possible signature values and n_{class} is the number of different possible class values.

Initially, the server shall calculate the distance between every pair of m alerts, so it commits $\frac{m*(m-1)}{2}$ distance calculations. A portion of these distances have been precalculated in the client sides. Specifically, each client has conducted $\frac{m_n*(m_n-1)}{2}$ calculations for the local alerts, so the actual distance calculations committed by the server are $\frac{m*(m-1)}{2} - \frac{m_n*(m_n-1)}{2}$. For each one of the distance calculations, the remote processing required is depicted in Table 1.

The construction of the DIST matrix happens only once in the beginning of the algorithm, so the overhead is limited. The remote processing required at each *K*-medoids round is related to the actual procedure of the round and is presented in Table 2.

The main calculation issue that is clear from Table 2 is that during execution of *K*-medoids rounds, the number of multiplications required is proportional to the square of total number of alerts m. This means that if the number of alerts per client increases significantly, the number of required multiplications will become very high and will have performance implications to the system. Even if the number of alerts is practically bounded in a real-world scenario, there may be extreme cases where the system will not be able to cope up with the resources requirements. In such cases, organizations may make use of more hardware resources.

Table 1 Remote calculations required for a single distance calculation

	Multiplications	Comparisons	Absolute value
Signature coefficient	n_{sig}	–	–
Class coefficient	n_{class}	–	–
IP coefficient	–	–	4
Timestamp coefficient	–	–	1
Distance calculation	$n_{\text{sig}} + n_{\text{class}}$	–	5

Table 2 Remote calculations required for a *K*-medoids round

	Multiplications	Comparisons	Absolute value
DFC	–	–	–
BTC	–	m	–
CG	$m^2 * k$	–	–
DG	m^2	–	–
CM	–	–	–
CCM	$m^2 * k$	–	–
CC	–	k	–
K-medoids round	$m^2 * (2k + 1)$	$m + k$	-

Theoretically, each client can consume more that one requests for auxiliary calculation at the same time, if it has the hardware resources to process those requests in parallel.

7.4 Results

The execution time for each scenario is shown in Figs. 3 and 4. The execution time consists of the time needed to calculate the DIST matrix and the time needed to execute the required *K*-medoids rounds. The required number of rounds has ranged between 3 to 5, for the experiments executed. Figures 3 and 4 show the required time in seconds for both these steps, for all different combinations for number of clients and number of alerts.

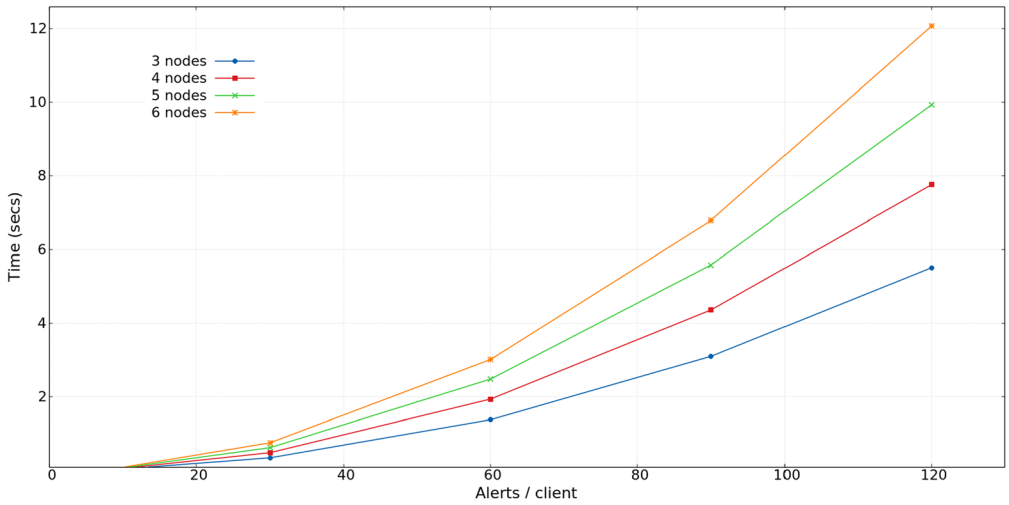

Fig. 3 Time for DIST matrix calculation

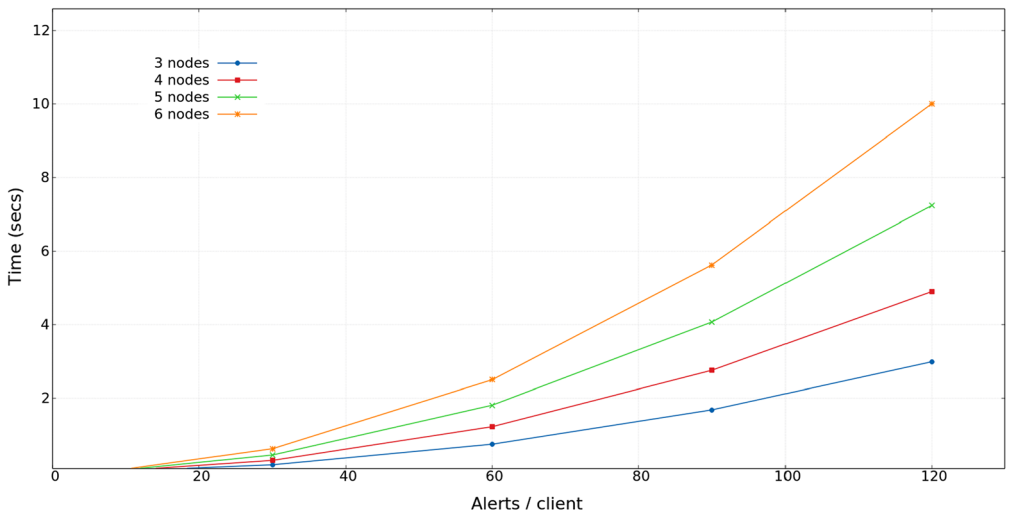

Fig. 4 Average time for a single *K*-medoids round

Table 3 Total execution times for multithreaded experiments

Clients	Alerts	Total execution time (s)			
		1 thread	2 threads	3 threads	4 threads
3	10	0.058	0.030	0.021	0.016
3	30	0.529	0.275	0.192	0.146
3	60	2.121	1.103	0.772	0.586
3	90	4.776	2.484	1.738	1.319
3	120	8.495	4.418	3.092	2.346
4	10	0.087	0.045	0.032	0.024
4	30	0.788	0.410	0.287	0.218
4	60	3.158	1.642	1.149	0.872
4	90	7.112	3.699	2.588	1.964
4	120	12.648	6.578	4.603	3.493
5	10	0.118	0.061	0.043	0.033
5	30	1.069	0.556	0.389	0.295
5	60	4.285	2.228	1.559	1.183
5	90	9.648	5.018	3.511	2.664
5	120	17.156	8.922	6.244	4.738
6	10	0.152	0.079	0.055	0.042
6	30	1.376	0.716	0.501	0.380
6	60	5.510	2.866	2.005	1.522
6	90	12.404	6.451	4.514	3.426
6	120	22.058	11.472	8.028	6.092

The same experiments have been repeated but with more hardware resources in place. The client implementation of the system has been revised, to process in parallel over one request by the server. Specifically, three additional configurations were tested at which the client had two, three and four threads running in parallel, to accept and process over one request for auxiliary processing at the same time. The total execution time results obtained from these experiments are depicted in Table 3.

7.5 Performance analysis

Homomorphic encryption employment penalizes the performance of the approach. As it is depicted in Figs. 3 and 4, and given the hardware used, there are specific delays in the processing of the alerts. For processing to be sustainable in the long run, the execution time of the algorithm must be less that the time window for which the nodes collect the alerts. In this way, the processing of a specific time window will end, before the processing for the next time window begins.

For example, five nodes, each one of which produces 30 alerts per time window, need approximately 1 second, to complete the clustering. This means that the alerts production rate must be limited in order for the system to be able to process each time window as soon as it expires. Specifically, the nodes should produce at most 30 alerts per second or in other words the system is suitable for networks of nodes that produce at most 30 alerts per second.

The typical alert rate in real-world networks is relatively low. While it is related to the volume and the nature of traffic, it is expected that it should not be greater than 1 alert per second. The dataset used throughout the experiments had an average alert rate of 0.304 alerts/s. In practice, this means that our system would easily process the required information. For a scenario of six nodes, ten clusters and a time window of 60 s, the total number of alerts would be around 120; thus, the size of the different matrices (analyzed in Sect. 6) would be 120×120, 120×10 or 120×1. According to execution times shown in Figs. 4 and 3, the processing time would be some seconds; thus, the system would easily process the information produced every 60 s, with no delays.

It must be noted that the hardware used in the experiments has to be taken into account when assessing the execution time results. Organizations that produce alerts in higher rates must use more capable hardware to cope with the required processing. The proposed method is designed to be scalable for nodes that can use over one processing threads. For example, if clients can concurrently execute multiple processing threads, the server can send to them batches of requests for processing of encrypted data. In this way, we can achieve a significant reduction in the execution time. As shown in Table 3, using multiple threads has a significant effect on the performance of the system. Using two threads speeds up the process by a factor of approximately 1.9, using three threads gives a speed up of 2.8 and using four threads gives a speed up of 3.7. There is a small management overhead, but for groups of clients that have higher traffic rates and probably higher alerts' rates, the clients can use additional hardware resources to cope with the demand for calculations.

The proposed system is based on the network communication between the server and the clients. We have conducted the experiments with all the subsystems connected on the same local area network. In a real-world scenario, each of the clients would be installed at different organizations, the network connection between which would be more challenging. In such a scenario, it is expected to have larger connection delays, packet drops or even clients going temporarily offline. These circumstances would hinder the normal operation of the protocol, and the server should be able to overcome any issues that come up.

Irrespective of the reason behind that (network failure, client failure) the server may receive no answer from a client when requesting for the auxiliary processing clients provide. In that case, and after a timeout period, the server shall make the same request to another client, to get the required results. If a client is unresponsive, the server may temporarily exclude it from the list of clients to request auxiliary processing from, to minimize the probability of the failure being repeated.

Theoretically, in a fault-free scenario, and given that the number of clients is n, the total number of requests is n_r and

the average time per request is t_{av}, the time required for the process to complete is :

$$\text{time}_{\text{tot}} = \frac{n_{\text{r}} * t_{\text{av}}}{n} \tag{29}$$

If, for the same setup, there was a failure rate p_{f} (requests to clients failed with p_{f} probability), then the total time required would be :

$$\text{time}^{\text{f}}_{\text{tot}} = \frac{n_{\text{r}} * (1 - p_{\text{f}})t_{\text{av}} + n_{\text{r}} * p_{\text{f}} * (t_{\text{av}} + t_{\text{t}})}{n} \tag{30}$$

where t_{t} is the timeout time that the server waits before re-dispatching a request. The increased time (with delay) with respect to the time required in the fault-free scenario is as follows :

$$\text{delay} = \frac{\text{time}^{\text{f}}_{\text{tot}}}{\text{time}_{\text{tot}}} = \frac{\frac{n_{\text{r}} * (1-p_{\text{f}})t_{\text{av}} + n_{\text{r}} * p_{\text{f}} * (t_{\text{av}} + t_{\text{t}})}{n}}{\frac{n_{\text{r}} * t_{\text{av}}}{n}} \tag{31}$$

And by making all the calculations, the delay is :

$$\text{delay} = \frac{t_{\text{av}} + t_{\text{t}} * p_{\text{f}}}{t_{\text{av}}} \tag{32}$$

In Eq. 32, it is obvious that failure probability is the deciding factor for the effect to the performance of the system. For example, given a relatively safe timeout time value (e.g., $t_{\text{t}} = 10 * t_{\text{av}}$), then a failure probability of 10% (which is an extreme case) would end up with halving the system's alert processing rate. The processing rate of the system for fault-free scenarios is at least one magnitude higher than required in real-world situations. Thus, it can be conducted that the proposed system can withstand the delay induced by failures in a real-world network environment, with no issues.

7.6 Privacy analysis

The main advantage of the proposed system is that organizations can collaboratively cluster their alerts without leaking the relevant data, and avoid any personal private information disclosure, for the organization itself or for its users. To overcome the limitations of the Paillier algorithm, we have opted for involving the clients in the operation. Clients hold the cryptographic keys, so they can decrypt values, commit the required operation and encrypt the result before sending it back to the trusted third-party server.

By using this approach, some information of clients is disclosed to other clients, but this happens in a privacy-respecting manner. As we have analyzed in Sect. 5, the trusted third-party server obfuscates encrypted values before sending those to the clients for processing.

7.6.1 Obfuscation of auxiliary services

Specifically, in the multiplication use case, the server sends values $x + r_x$ and $y + r_y$ to the client instead of x and y. It is impossible for the client to calculate the initial values of x and y as there is no restriction for r_x and r_y.

For the comparison use case, all values are multiplied by a highly composite number r, to hinder the client from getting to know the exact values. One minor drawback for this use case is that client finds out which elements are zero valued. The order of the elements though has been scrambled by the server, so the client can only conclude on the percentage of the zero-value elements out of all elements, but cannot learn to which alerts or clusters these elements may correspond.

Finally, regarding the absolute value use case, the client cannot conclude on the value of b as the submitted numbers have multiple divisors. Practically, the client cannot conclude on the value of z element that the server needs to know the absolute value of.

7.6.2 Proposed approach privacy analysis

During multiplications and absolute value calculations, there is no privacy leakage to the processing clients. The values sent as input from the server are sufficiently obfuscated, and clients cannot conduct any useful information regarding the actual values. The only part of the protocol that may reveal information to the clients is the comparison auxiliary service provided by the clients to the server. This functionality is used during each K-medoids round in the construction of BTC and CC matrices. Server needs to send a vector of elements to a client, in order for the client to pick the minimum value. With the BTC matrix, the elements of the vector are the distances of a single alert regarding all the clusters' centers (rows of DFC matrix). In the case of the CC matrix, the elements of the vector are calculated metrics for deciding the new center of a cluster (columns of CCM matrix).

The server, before sending the vector to the client, multiplies each element by a common random variable r and then scrambles the order of the elements. The client has to decrypt all values to conduct the required processing, so it get access to the obfuscated vector. The client can attempt to calculate the common factors between all elements of the vector and then try to conclude on the value of r. Finding the common divisors of all the elements is a demanding task in terms of resources and may end up with over one candidate values for parameter r. If that is successful, the client will get access to all the values of the vector, but without knowing their proper order (as the order is scrambled).

As the client does not know which is the alert to which the DFC row corresponds to or the cluster to which the CCM column corresponds to, it cannot make any conclusions on

Table 4 Probability of clients concluding information about clusters

Clients	Alerts/client	Probability
3	10	0.19
3	30	0.27
3	60	0.22
3	90	0.26
3	120	0.26
4	10	0.22
4	30	0.18
4	60	0.20
4	90	0.23
4	120	0.16
5	10	0.13
5	30	0.13
5	60	0.10
5	90	0.11
5	120	0.19
6	10	0.08
6	30	0.04
6	60	0.06
6	90	0.05
6	120	0.09

the features of alerts and how those relate to the private information of other clients. The only thing that the processing client can probably do is recognizing that two different requests (in two different *K*-medoids rounds) correspond to the same alert or to the same cluster, as in that case many of the elements should remain the same. That information might give knowledge to the processing client about what percentage of clusters has changed their centers (along with the set of alerts they comprise) between the two rounds. The probability that a client gets a vector for the same alert/cluster over one times is relatively limited as the required number of rounds for *K*-medoids to converge is 3–5. Thus, in a setup with six clients, a client may never process vectors for the same alert/cluster for the second time. Even if that is the case, the information that the client gets is related the procedure and not to the private data of others.

To assess the probability that a client may get access to information regarding how a cluster evolves through *K*-medoids rounds, an experiment has been conducted. For fixed parameters regarding number of clients, number of alerts and number of clusters, the clustering procedure has been repeated multiple times. At each one of these iterations, the actual data points have been varied and the ability of any client to conclude upon information regarding clusters formation has been checked. Through this experiment, we have calculated the probability of such information leakage. The results of the procedure are depicted in Table 4. The table shows the probability that a random client gets information about how a cluster is evolving between two different *K*-medoids rounds. The private data of the clients remain out of reach for other clients.

From the results, it is obvious that as the number of clients increases, the probability that they get vectors for the same alert or the same cluster twice decreases. While there is no direct privacy leakage for clients, it seems that relatively larger sets of clients are safer with respect to this information disclosure to clients.

7.6.3 GDPR limitations

In a real-world scenario, the application of the proposed method would trigger concerns regarding the fact that organizations process data of other organizations. Specifically, under the General Data Protection Regulation (GDPR), this could create an important practical issue. The proposed approach requires organizations to process others' data, but under two significant constraints:

- Data are anonymized; thus, an organization does not have any indication to which of the other organizations the data, being processed, belong to.
- Data are obfuscated; thus, even by getting access to an obfuscated value, the processing organization does not get to know to which real value it reflects.

Under the aforementioned rules, the privacy implications are minimized and it would be more feasible to have participating organizations consent to the specific scheme that would allow others to process a part of their data. Given that organizations give their consent when they accept to become members of this collaborative intrusion detection consortium, there should be no GDPR issues in the long run.

8 Conclusions and future work

The system proposed enables privacy-preserving alert clustering between multiple organizations. Such alerts post-processing may reveal important information about cyber-attacks being conducted and set the organizations able to efficiently protect their networks. To comply with the real-world requirements, the proposed system protects the privacy of the participating organizations. This comes with a cost, as to preserve privacy the system has to bear with a performance penalty. As it is shown in Sect. 7, the rate, at which the system can cluster alerts, depends on the hardware used and it is expected that participating organizations will use sufficient hardware resources.

It has been proven that it is possible to conduct privacy-preserving alerts clustering between different organizations. Given that these organizations are willing to invest in the

required hardware, according to their needs, it is possible to make this collaboration scheme effective and able to process alerts without inducing any significant overhead.

A requirement for the system to function properly is that all participants, the server and the clients, operate according to the protocol. Every participant adheres to the honest but curious model; thus, they send the proper values calculated according to the protocol. At the same time, they may try to get access to available private information regarding others. The privacy of participants is protected by the proposed methodology, as it is analyzed in Sect. 7.6. The integrity of the calculations is protected by the honesty of the participants. If a malicious client (or even a malicious server) attempts to destroy the calculations' workflow, it is easy to do so by sending invalid data to others. Given the fact that data in the system are mainly encrypted, it is challenging to do any integrity checking, to prevent participants from sending invalid inputs.

Regarding future work, we intend to justify that the method presented is appropriate for scaling up with hardware. The system implemented is going to be tested in more demanding conditions. We plan to test the system with higher traffic and alert rates, while using more capable hardware. This approach will reveal any performance bottleneck that our approach may suffer from.

Another improvement to the approach would be to use an alternative encryption algorithm such as the homomorphic encryption scheme BGN [5] that would enable more processing on the trusted third-party side. BGN algorithm may create the need for more resources in the TTP server side, but would significantly limit the required network communications. This would have a positive effect on both performance and privacy protection.

Compliance with ethical standards

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Andreolini, M., Colajanni, M., Marchetti, M.: A collaborative framework for intrusion detection in mobile networks. Inf. Sci. **321**(C), 179–192 (2015). https://doi.org/10.1016/j.ins.2015.03.025
2. Axelsson, S.: The base-rate fallacy and the difficulty of intrusion detection. ACM Trans. Inf. Syst. Secur. (TISSEC) **3**(3), 186–205 (2000)
3. Barry, B.I.A., Chan, H.A.: Intrusion Detection Systems, pp. 193–205. Springer, Berlin (2010)
4. Benali, F., Bennani, N., Gianini, G., Cimato, S.: A distributed and privacy-preserving method for network intrusion detection. In: OTM Confederated International Conferences On the Move to Meaningful Internet Systems, pp. 861–875. Springer (2010)
5. Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Theory of Cryptography Conference, pp. 325–341. Springer (2005)
6. Dara, S., Muralidhara, V.: Privacy preserving architectures for collaborative intrusion detection. arXiv preprint arXiv:1602.02452 (2016)
7. Davis, C.: The norm of the schur product operation. Numer. Math. **4**(1), 343–344 (1962). https://doi.org/10.1007/BF01386329
8. Dermott, A., Shi, Q., Kifayat, K.: Collaborative intrusion detection in federated cloud environments. J. Comput. Sci. Appl. **3**(3A), 10–20 (2015). https://doi.org/10.12691/jcsa-3-3A-2
9. Do, H.G., Ng, W.K.: 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), pp. 1–6. IEEE (2015)
10. Fayi, S.Y.A.: What petya/notpetya ransomware is and what its remidiations are. In: Information Technology-New Generations, pp. 93–100. Springer (2018)
11. Francois, J., Aib, I., Boutaba, R.: Firecol: a collaborative protection network for the detection of flooding ddos attacks. IEEE/ACM Trans. Netw. **20**(6), 1828–1841 (2012)
12. Gogoi, P., Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Packet and flow based network intrusion dataset. In: Parashar, M., Kaushik, D., Rana, O.F., Samtaney, R., Yang, Y., Zomaya, A. (eds.) Contemporary Computing, pp. 322–334. Springer, Berlin (2012)
13. Ho, C.Y., Lai, Y.C., Chen, I.W., Wang, F.Y., Tai, W.H.: Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems. IEEE Commun. Mag. **50**(3), 146–154 (2012)
14. Hong, J., Liu, C.C.: Intelligent electronic devices with collaborative intrusion detection systems. IEEE Trans. Smart Grid **PP**(99), 1-1 (2017). https://doi.org/10.1109/TSG.2017.2737826
15. Horn, R.A.: The hadamard product. Proc. Symp. Appl. Math. **40**, 87–169 (1990)
16. Jin, R., He, X., Dai, H.: On the tradeoff between privacy and utility in collaborative intrusion detection systems-a game theoretical approach. In: Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp, HoTSoS, pp. 45–51. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3055305.3055311
17. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: Ddos in the iot: Mirai and other botnets. Computer **50**(7), 80–84 (2017)
18. Lazarevic, A., Kumar, V., Srivastava, J.: Intrusion Detection: A Survey, pp. 19–78. Springer, Boston (2005)
19. Li, W., Meng, W., Kwok, L.F., Horace, H.: S: Enhancing collaborative intrusion detection networks against insider attacks using supervised intrusion sensitivity-based trust management model. J. Netw. Comput. Appl. **77**, 135–145 (2017). https://doi.org/10.1016/j.jnca.2016.09.014
20. Liang, H., Ge, Y., Wang, W., Chen, L.: Collaborative intrusion detection as a service in cloud computing environment. In: 2015 IEEE International Conference on Progress in Informatics and Computing (PIC), pp. 476–480 (2015). https://doi.org/10.1109/PIC.2015.7489893
21. McHugh, J., Christie, A., Allen, J.: Defending yourself: the role of intrusion detection systems. IEEE Softw. **17**(5), 42–51 (2000)
22. Milenkoski, A., Vieira, M., Kounev, S., Avritzer, A., Payne, B.D.: Evaluating computer intrusion detection systems: a survey of common practices. ACM Comput. Surv. (CSUR) **48**(1), 12 (2015)
23. Morais, A., Cavalli, A.: A distributed and collaborative intrusion detection architecture for wireless mesh networks. Mobile Netw. Appl. **19**(1), 101–120 (2014). https://doi.org/10.1007/s11036-013-0457-8

24. Nicolas, J.L., Robin, G.: Highly composite numbers by srinivasa ramanujan. Ramanujan J. **1**(2), 119–153 (1997). https://doi.org/10.1023/A:1009764017495
25. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 223–238. Springer (1999)
26. Pietraszek, T., Tanner, A.: Data mining and machine learning-towards reducing false positives in intrusion detection. Inf. Secur. Tech. Rep. **10**(3), 169–183 (2005)
27. Ring, M., Wunderlich, S., Scheuring, D., Landes, D., Hotho, A.: A survey of network-based intrusion detection data sets. Comput. Secur. **86**, 147–167 (2019)
28. Roesch, M.: Snort—lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX Conference on System Administration, LISA'99, pp. 229–238. USENIX Association, Berkeley, CA, USA (1999). http://dl.acm.org/citation.cfm?id=1039834.1039864
29. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Comput. Secur. **31**(3), 357–374 (2012). https://doi.org/10.1016/j.cose.2011.12.012
30. Singh, S.S., Chauhan, N.: K-means v/s k-medoids: a comparative study. In: National Conference on Recent Trends in Engineering & Technology, vol. 13 (2011)
31. Sommer, R., Paxson, V.: Outside the closed world: on using machine learning for network intrusion detection. In: 2010 IEEE Symposium on Security and Privacy (SP), pp. 305–316. IEEE (2010)
32. Spathoulas, G.P., Katsikas, S.K.: Reducing false positives in intrusion detection systems. Comput. Secur. **29**(1), 35–44 (2010)
33. Tan, Z., Nagar, U.T., He, X., Nanda, P., Liu, R.P., Wang, S., Hu, J.: Enhancing big data security with collaborative intrusion detection. IEEE Cloud Comput. **1**(3), 27–33 (2014). https://doi.org/10.1109/MCC.2014.53
34. Vasilomanolakis, E., Karuppayah, S., Mühlhäuser, M., Fischer, M.: Taxonomy and survey of collaborative intrusion detection. ACM Comput. Surv. (CSUR) **47**(4), 55 (2015)
35. Vasilomanolakis, E., Krügl, M., Cordero, C.G., Mühlhäuser, M., Fischer, M.: Skipmon: A locality-aware collaborative intrusion detection system. In: 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC), pp. 1–8 (2015). https://doi.org/10.1109/PCCC.2015.7410282
36. Wang, Y., Meng, W., Li, W., Li, J., Liu, W.X., Xiang, Y.: A fog-based privacy-preserving approach for distributed signature-based intrusion detection. J. Parallel Distrib. Comput. **122**, 26–35 (2018)
37. Wang, Y., Xie, L., Li, W., Meng, W., Li, J.: A privacy-preserving framework for collaborative intrusion detection networks through fog computing. In: Wen, S., Wu, W., Castiglione, A. (eds.) Cyberspace Safety and Security, pp. 267–279. Springer International Publishing, Cham (2017)
38. Zhang, P., Huang, X., Sun, X., Wang, H., Ma, Y.: Privacy-preserving anomaly detection across multi-domain networks. In: 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 1066–1070. IEEE (2012)
39. Zhou, C.V., Karunasekera, S., Leckie, C.: Evaluation of a decentralized architecture for large scale collaborative intrusion detection. In: 2007 10th IFIP/IEEE International Symposium on Integrated Network Management, pp. 80–89 (2007)
40. Zhou, C.V., Leckie, C., Karunasekera, S.: Decentralized multi-dimensional alert correlation for collaborative intrusion detection. J. Netw. Comput. Appl. **32**(5), 1106–1123 (2009). https://doi.org/10.1016/j.jnca.2009.02.010. Next Generation Content Networks
41. Zhou, C.V., Leckie, C., Karunasekera, S.: A survey of coordinated attacks and collaborative intrusion detection. Comput. Secur. **29**(1), 124–140 (2010)