University of Nevada, Reno

Privacy Preserving Cyber Threat Intelligence Sharing Framework for Encrypted Analytics.

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Computer Science and Engineering

by

Ignacio Astaburuaga

Dr. Shamik Sengupta - Thesis Advisor
August 2022

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

Ignacio Astaburuaga

entitled

Privacy Preserving Cyber Threat Intelligence Sharing Framework for Encrypted Analytics

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Shamik Sengupta, Ph.D.
Advisor

Dave Feil-Seifer, Ph.D.
Committee Member

Hanif Livani, Ph.D.
Graduate School Representative

David W. Zeh, Ph.D., Dean
Graduate School

August, 2022

Abstract

This research focuses on the creation of an encrypted Cyber Threat Intelligence (CTI) sharing framework that supports encrypted data analytics with privacy preservation. It aims to support analytical computation in a centralized node without allowing that node to see any of the plain-text data.

To enable privacy preservation of the data and its users, we structured the data into a graph structure that allows traversal over the encrypted data. We used Ciphertext-Policy Attribute-Based Encryption (CPABE), Deterministic Encryption (DE), and Order Revealing Encryption(ORE) to ensure end-to-end encrypted sharing of Cyber threat data.

In this work we also cover CYBersecurity information EXchange with Privacy (CYBEX-P) and CYBEX-P with Encrypted Analytics, the precursor projects on which the framework is based.

Our research aims to solve one of the biggest problems that CTI sharing has: securing the privacy of the data once it leaves the user's premises. We focus on eliminating attack surfaces present in centralized systems, that is, the attack surface attackers had over the Backend and the surface the Backend has against the system. We also focused on maintaining as many capabilities of a CTI sharing platform, that is, CTI sharing and centralized analytics.

Dedication

This thesis is dedicated to my family: They are my support structure that keeps me going. My mother, who has sacrificed herself tremendously so our family can flourish to the success we have now. My father, who has worked endlessly to provide for the me and my family throughput the years. My brothers, for the support and encouragement they have given me throughout the years.

Acknowledgments

I wish to thank my advisor, Dr. Shamik Sengupta, and my friend, Tapadhir Das, for their support throughout this process.

This material is based upon work supported by the National Science Foundation (NSF) Grant No. 1739032.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Developing a strong cybersecurity posture is paramount to achieving higher levels of security within an organization. Organizations should perform Cyber Threat Intelligence (CTI) sharing to achieve said posture. Cyber Threat Intelligence is information, data, and experience that can help level the playing field. CTI can include system logs, network traffic, malware signatures, and reports about the cyber world around us. Sharing allows organizations to learn and stay current in the continuously evolving cyber battlefield. It allows organizations to learn and identify current threats, leading to successfully defending from attacks. Although it is widely known in the cybersecurity community that sharing said information is highly beneficial for identifying and preventing threats, sharing CTI has had little traction over the past few years. There are three main reasons why companies choose not to share:

1. The risk associated with sharing data

2. There is no universal way of sharing said heterogeneous data, preventing analytics, TAHOE aims to solve this problem [1]

3. Lack of features in CTI platforms

    - Manual workflow
    - No data aggregation to support analytics in the first place

Companies do not participate in such platforms because sharing has many associated risks. For an information-sharing platform to be successful and have an excellent user base, it has to maintain specific characteristics that outweigh the possible downsides, such as maintaining the data safety, reliability, and correctness for a large user base with lots of data ingestion. Most sharing platforms have some missing components, reducing either the usability or security of the data and privacy. For example, most only focus on the sharing data but not the aggregation or analysis of said data; the input and output formats of data are not standardized. In most cases, the security of the data in the system and transit is inherently vulnerable due to bugs in the system. Sharing sensitive information into an untrusted sharing platform becomes very risky for companies as their information may or may not be handled correctly down the pipeline.

Additionally, the data and its analytics are very appealing to malicious actors for an array of reasons. For example, learning about the internals of a company, such as, possible active vulnerabilities. Most companies do not want to share their data because it is simply too risky; existing solutions significantly lack security features for protecting the data, the privacy of the data and the companies. One of the main concerns for companies that do share is data privacy [2].

This research focuses on developing a framework that would allow end-users more control over their data and their analytics by eliminating the exposure of the data itself to everyone, while allowing authorized end-users to generate meaningful unencrypted-equivalent analytics reports over the encrypted aggregated data. Unencrypted-equivalent reports (analytical results) are generated fully encrypted, and only if the end-user is authorized will he be able to translate them to a meaningful plain text. The plain text version of the encrypted report equates to a report generated in a traditional CTI sharing platform with analytics, for example, CYBEX-P [1].

The objective of this framework is to achieve storage and processing of data with end-to-end privacy. However, this work does not focus on data communication/transmission schemes (as the strength comes from the underlying encryption schemes), focusing only on the underlying mechanism for encryption and processing of data to achieve analytics as if the data were unencrypted.

1.2 Gap of current solutions

This research focuses on the gap in available CTI sharing and analytics solutions with encryption. Furthermore, this research focuses on methods that avoid using the processing party as a trusted and integral part of the system, allowing for end-to-end encrypted CTI sharing and data analytics with access control. We initially solved the privacy problems with encryption, as we worked towards analytics we realized that doing analytics over the data at the centralized node was not possible. We ultimately worked around this limitation by separating the data into two layers, a data relationship layer and a data encryption layer. More details explained in section 4.2.

Most CTI sharing platforms work by sending their data to the centralized system, where it is processed and later analyzed. In this common scenario, while it has security against hackers, the centralized systems can bypass any security mechanisms as they have complete control over the sensitive information. Ultimately privacy of the data lies in the trust that we have with that centralized party, whether they are acting correctly and have no bugs or vulnerabilities. We propose a mechanism that allows the submitting users to have complete control over their data before it leaves their premises, not allowing any unauthorized party to view the data, including the centralized party.

Most research in this area focuses on developing CTI sharing platforms but ultimately puts aside the dangers of having to trust a centralized party. This research goes beyond current solutions and introduces the use of multiple encryption primitives to achieve fully end-to-end CTI sharing. Our framework cuts out the centralized party's abilities to maliciously use the data while maintaining the ability to aggregate and analyze data from multiple users. Our approach gives users the ability to establish access control rules on their data, giving them ultimate control over their shared information.

One of the main goals of this research was to establish a way to have the same layout as a traditional CTI sharing platform, that is, users manually submit, query, and analyze the data. We automate all three steps in our improved workflow while allowing users to manually/programmatically handle data if needed. Our framework establishes a more automated workflow after the initial setup. Our improved workflow allows the framework to handle the machine work, allowing security engineers to focus on what is important, analyzing and interpreting the data to improve their organization's security.

This research aims to establish a framework to overcome one of the main

reasons companies do not want to contribute their data and participate in CTI sharing: the risks involved in sharing data. However, this research does not focus on other aspects of CTI sharing like the number of users, quality of contributions, attribute management, and key management.

Organizations choose not to participate in CTI sharing platforms because platforms do not offer rich features; most platforms only offer data download. Lack of features means that it is up to each user to download, store, aggregate, and analyze the data, which requires storage, processing power, and knowledge. For this reason, in this research, we work to address these three limitations that organizations have and develop a framework that has a central node that can aggregate, store, and analyze data, so users do not have to.

1.3 Motivation

Security breaches can have a devastating effects to companies and its users. They often handle sensitive information, such as users' names, Social Security numbers, dates of birth, and contact information. Another example are companies' trade secrets, like Google's indexing algorithm. There are also considerations related to operational processes; for example if a company's inventory system went offline, it would mean that the company would not be able to operate. Another consideration is the aspect of liability that can come from a breach settlement to costumers, each costumer could be entitled to money and credit monitoring which can add up quickly depending on the number of costumers affected by a breach. This particularly affects businesses with a large costumer base because they may not be able to pay the settlement leading to bankruptcy. There are also physical security concerns related to cybersecurity,

some systems are connected with the physical world and can have catastrophic effects on population, for example nuclear power plants, power grids, and water plants. Generally, these cyberphysical systems have connections to the internet, increasing their attack surface making them more vulnerable to external attacks. One notable recent example, in 2021 a hacker gained access to a Florida water treatment plant and changed the amount of chemicals that went into the water treatment, essentially poisoning it. This example is not related to CTI sharing but stresses why breaches should be minimised.

There are plenty of examples where CTI sharing could have prevented the breaches. For example, the companies Baltimore, Marriott, Uber and Ticketmaster [3] all possess a common characteristic. Each of these major attacks were seen somewhere else and organizations had poor intelligence ingestion and adaptation to threats, which led to successful breaches. These could generally be prevented by having a mature cybersecurity posture that shares and ingests CTI data.

CTI sharing does not automatically solve vulnerabilities or prevents attacks, but it opens the channel for communication about threats. The sooner organizations can detect and share information about threats, the sooner organizations can react and adapt to new attacks.

1.4 Terminology

1.4.1 Cyber Threat Intelligence

Cyber Threat Information is any information that can help an organization identify, assess, monitor, and respond to cyber threats. Cyber Threat Information

includes indicators of compromise; tactics, techniques, and procedures (TTPs) used by threat actors; suggested actions to detect, contain, or prevent attacks; and the findings from the analyses of incidents. Specific examples of cyber threat information include indicators of compromise, system artifacts, TTPs, security alerts, reports, and recommended security tool configurations [4, 5]. Most organizations often produce CTI to share internally. The problem with only sharing internally is that the knowledge is significantly smaller and limited compared to external sharing. Internal sharing also means that CTI data is created in silos and other organizations can not benefit from it, for example if there is a hack in one company then other companies wouldn't know what works against that type of attack.

1.4.2 Honeypot

Honeypots are either a software that runs on a server or are the whole server. They are nodes that look legitimate on the network but have no real business; therefore, if someone interacts with it, they are most likely malicious. Their goal is to detect and learn from attackers. Honeypots can be deployed within a legitimate network in hopes to detect attacks within it. For the purposes of this thesis we used honeypots to gather attack command sequences from attackers. We then used the data to test our framework.

1.5 Content

The following chapters of this thesis are as follows: In Chapter 2 we discuss related works, requirements, assumptions, and possible approaches to our proposed framework detailed in Chapter 4. Chapter 3 introduces CYBEX-P, the

brother project that this thesis work aims to improve upon, the initial work done to support the work described in Chapter 4. Chapter 3 also discusses early work done with CYBEX-P with Privacy Preservation, the first implementation and the algorithms used to support it. Chapter 4 introduces a new framework that expands in the previously mentioned work. The new framework utilizes similar primitives but achieves analytics. Lastly, conclusion and future work are provided in the last chapter of this thesis.

Chapter 2

Related works

2.1 Related Work

In [6] the authors aim to solve the same problem our research solves, that is, current CTI sharing platforms do not provide means to preserve privacy for the users. However, in their research, they only try to mediate privacy preservation agreements between parties. They do so by layering degrees of preservation using predetermined actions/functions of preservation, for example, plain-text, anonymity, sanitization, and homomorphic encryption. Their approach is to determine the level of preservation needed by each party and adjust the data privacy to that specific level. Our research completely differs from theirs in the solution to a common problem; we achieve privacy by equipping users with fine access control to preserve data privacy, which also includes the centralized party, while at the same time enabling data analytics. Their research is related to determining appropriate degrees of protection and applying actions to the data. Depending on the actions applied to the data, its confidentiality may

still be at risk. Furthermore, their system does not allow analytics after their protective actions are applied.

Chadwick et al. [7] expanded on the above framework for selecting appropriate levels of privacy preservation and applying different respective privacy preservation mechanisms. Neither of which details any practical implementation details on preserving privacy while maintaining analytical power over the data.

Most platforms work by having a centralized party gather data from multiple sources, making it available to its users. On some platforms, users can contribute data. Then, other users can request all of the data within a certain timestamp (which are generally not related to the data but to when the data was submitted to the system). In systems where data is gathered from the public domain, it tends to be of bad quality. Data from organizations tend to be overly obfuscated/sanitized, making it significantly less helpful. Often, organizations generate quality CTI data; unfortunately, this data is commonly only shared internally within the organization and not with other organizations.

To the best of our knowledge, there is no other platform or framework that attempts to achieve usability, privacy, and analytics, all in a single CTI sharing platform. Our framework is the first step in supporting all these functions. Other research and platforms attempt to solve other problems, but not the technical problem of the data sharing logistics for maintaining its privacy and achieving analytics simultaneously. Because we aggregate data, we can achieve rich features like analytics compared to platforms that only upload and download information. Our framework allows the users to submit and aggregate from multiple sources and analyze while preserving their privacy.

2.2 Requirements, assumptions & possible approaches

In order to solve the problem of privacy, we have turned to encryption mechanisms. There are a few ways to apply off-the-shelf components to solve the privacy issues with CTI sharing platforms. We first have to establish a few system requirements:

- The system must have a similar flow to current CTI sharing platforms, or better/simpler
- The system must be able to aggregate data from multiple sources and
- Generate reports based on aggregated data, essentially showing the potential of diverse CTI data. For our purposes we focused in a similar approach to TAHOE [1] as it allows great flexibility in report generation
- Ingest heterogeneous data
- Allow user control over their submitted data, including reports
- Data must be safe from the centralized party and any other unauthorized users, including at rest and in transit

Trusted parties use the system, and the data is assumed to be correct in nature; however, this does not mean that users can not act maliciously and share encrypted data.

We explored the following possible solutions to solve the privacy problem:

1. Encrypt all the pieces of data independently using CPABE. This approach did not give us the flexibility that was required to achieve any meaningful analytics. This specific approach is further discussed in the next chapter.

2. Add encryption layers to the database and implement everything as usual but with an encrypted database. This approach does not allow flexibility in encryption and decryption of the data as users would have to share keys. The decentralized system would not be able to generate analytics reports on a large amount of aggregated data if it does not give access to the data. A possible solution to this problem is homomorphic encryption [8, 9]. For our purposes, homomorphic encryption does not work. This is because we would not give the central party keys, which means that if we wanted to do some filtering of the data for analytical purposes, the central party would not be able to discern relevant results and would be forced to return the calculation results of every possible permutation of the query with every data in the system to the querying user, this could take multiple iterations of back and forth and is also very expensive in data transmission, storage, and computation for the user. Homomorphic encryption was not a feasible solution for our requirements.

3. Use of an off-the-shelf graph database and encryption of the data and its links. This approach did not work as modern graph DBMSs can not handle queries over the whole data efficiently (further discussed in section 2.3).

4. Implement custom graph data structure, apply encryption to the data and relations, and have full control over development and implementation. This is the approach we took in this research. We started with MongoDB as our NoSQL database of choice and implemented a graph structure described in section 4.1.3. By implementing our own encrypted graph structure, it allows us to do efficient database queries while maintaining the data privacy, allowing the generation of analytical reports at the central party while giving the end-user full control of their data (at initial encryp-

tion time).

Off-the-shelf components are not suitable for solving the problem, and a custom solution had to be implemented. We recognized that data had to be encrypted or placed in an equivalent encrypted format. In our research, we placed data in an encrypted graph database allowing us to traverse it encrypted. We implemented it in this manner so that the following is met:

1. The analytics agent can not learn about the plain-text data from the encrypted data

2. Be able to retrieve and calculate information (i.e. traverse the graph) without knowing about the underlying data. The approach we used was to use encryption for the relationship information that is:

    - deterministic in nature to be able to retrieve the information by the required user
    - and random to avoid the analytics agent from learning any information about the relationship data

    The encryption primitive is further discussed in section 3.3.

3. Allows the analytics engine to query and filter the data without knowing what it is or make sense of it

We formed the framework to incorporate the following requirements:

- There must be a centralized party. Single organizations are said to have small computational power and would not like to join if they have to do the work for computing analytics over the aggregated data. Therefore the centralized party must be able to compute some results for the end querying user.

- Data is said to be of known types and structures. This means that a conversion layer must exist, later referred to as Collector-client
- Data is encrypted before leaving the premises
- Submitting users should have control over their data
  - In our framework, the user has control when they submit their data

This system assumes that users only trust themselves and the parties they allow to view their data. This is strictly enforced using encryption, which is why this system heavily relies on a Key Management Service (KMS). The KMS is said to be trusted. To use even the most basic features of the framework, users must be enrolled with the KMS/system as they will need system encryption/decryption and system keys to operate. The system relies on the KMS and assumes that the security comes from underlying encryption mechanisms.

2.3 Challenges

The work presented in this thesis is divided into two sections: the early supporting works, chapter 3, and the successor framework in chapter 4.

In our early works, we design the platform to use direct encryption over each piece of data so that the Backend could not compromise the privacy of the data. Using encryption meant that we were trading usability for privacy. The biggest challenge with using direct encryption was finding encryption primitives that allowed comparability around the encrypted data that allowed us to create analytical queries. It is widely known that homomorphic encryption can be used to compute results with encrypted data, and in our system, we explored the use of such encryption primitive. Homomorphic encryption is not suitable for our

purposes as the Backend will require decryption keys to use the results of the encrypted computations for efficient analytical reports of the data. Using only encryption mechanisms for CTI processing left us with a CTI sharing platform that only shared. This problem lead us to rework the solution. Ultimately we developed a system that uses a graph data structure to hold relationship data with a separate value data layer. Queries were implemented by traversing the graph of relationships formed by the encrypted CTI data.

In our early design, we explored two possible solutions to relationship traversal. One was to use an unencrypted graph database and add an encryption layer. The second solution was to use an encrypted graph database. Each solution had its limitations; for the unencrypted database approach, we had to create an encryption mechanism that allowed the comparison of encrypted data. The encrypted approach limited the ability of the Backend system to do analysis on the data. Some implementations only allowed calculations to be done in a temporary decrypted state.

The first approach gave us the most flexibility without compromising security. The use of out-of-the-box graph database solutions like Neo4j did not work. The most common use case and goal for off-the-shelve graph databases is to find relationships between two nodes as fast as possible. They focus on a minimal number of paths, such as finding the shortest path given two nodes. Our problem can be modeled as finding all the possible related nodes from one node to any other node while saving the paths, a problem that is not compatible with graph databases' objectives. Neo4j can not handle this task even with the smallest amounts of data; it was not designed to find all possible paths. Even other research related to encrypted graph traversal only focus on the shortest path problem [10]. Our proposed solution is building a simple data structure

that allows fast traversal of the relations of the data. More on the proposed database schema and how we solved this problem in section 4.1.3.

We also considered using an encrypted database and storing the data in a graph structure. Popa et al. [11, 12] developed an encrypted SQL-like encrypted database. Their research addresses two main threats which we also have in our system. The centralized Backend that runs the database and the possibility of adversaries that could take control (fully or partially) over the system, the database, and ultimately the data. However, this approach was not suitable for our problem as the Backend has no way of interacting with the encrypted data; only the end-users do; therefore, there is no centralized analysis generation. This approach also does not protect the data from other users, as mixing access control with this kind of database restriction always prevented analytics, so only one or the other could exist in this configuration. Similar incompatibility would happen with using encrypted graph databases [13–16] and encrypted no-SQL databases [17].

2.4 Privacy Preservation

When we talk about privacy preservation, we mean that the organizations will not be able to access data from one another and that we will protect against outsiders, including the centralized Backend. This ensures that we have preserved the privacy of the data. Another essential point to consider is the privacy of the users themselves. We aim to protect both by implementing access control over the data, users can submit pre-sanitized data to further protect themselves, but it is highly discouraged as they have fine control over who can see their data. Essentially, they have complete control over their privacy. The access control

is defined even before data is encrypted with CPABE on the user's premises. This guarantees that the access control can not be tampered with later on at any point of the pipeline. This also means that if the user submitting the data encrypts data with a policy where he does not have the suitable attributes to decrypt it, he will not be able to access that data either.

We aim to achieve privacy preservation by relying on two mechanisms—the implementation of access control and decryption attribute assignment. A robust access control mechanism will ensure that the data is protected. The submitting user will have to establish who is able to access data, regardless of how other parties could gain access to the data, via the system or from an adversaries point of view, stolen. Our framework assumes two significant points: the user will be able to carefully select who can access their data in the access control policies and that the access control attributes are correctly assigned.

We have assumed that our Key Management System (KMS) will act as a trusted party that can vouch for everyone's attributes. It is its job to verify every single organization before joining. Each organization can further be broken down into smaller groups or users that the KMS will also have to verify and check attributes. Attribute checking is a crucial component for the system as it is used to maintain data privacy via access control. Attribute checking, however, offloads the privacy problem to access control and attribute management.

The system's users can be anyone who will submit or retrieve data to the system. Generally speaking, in CTI sharing platforms, users are not encouraged or allowed to only retrieve data from the system as it degrades the system's utility, as it is a function of quality data stored in it. In most platforms, users must also submit meaningful data or pay for the data if they are only retrieving it.

Our framework explicitly requires users to sign up with the system because CPABE attributes need to be verified by the KMS for decryption purposes. In terms of KMS and keys, organizations can further split up into subgroups. For example, a single organization can sign up as a single entity that submits data with hundreds of users who can query data; each independent user will get a decryption/private key with their corresponding attributes.

Chapter 3

CYBEX-P with Privacy

In this chapter we will focus on the origin of the framework later described in this thesis. The first platform that we researched and developed was named Cybersecurity Information Exchange with Privacy (CYBEX-P). CYBEX-P had privacy as one of its main objectives, which is achieved through access control oversight for everyone by having a centralized party manage access to the data. As a parallel thread, we developed our initial implementation of a stripped down version of CYBEX-P that focused on removing the access the centralized party had over the data. We called this parallel project Encrypted Analytics (EA). This chapter focuses on the Encrypted Analytics. Encrypted Analytics is the precursor and foundation of the more advanced Encrypted Analytics Framework. We will also discuss details and implementation of both Encrypted Analytics Framework and its precursor, CYBEX-P Encrypted Analytics project.

3.1 Background on CYBEX-P

Cybersecurity Information Exchange with Privacy (CYBEX-P) is a platform that we researched and developed intended to share CTI and analyze it to generate insightful reports about the data while maintaining privacy of its users. Privacy was achieved by having good separation of modules and strong nodes with high security. We divided the platform into modules that lived inside of two zones. Two zones meant that the non-secure zone could potentially be compromised without compromising the security of the system. The whole system relies on three assumptions: that the secure side of the system will not be compromised, that the centralized Backend will not be malicious in any kind of way, and that the system does not mind the Backend having access to the data.

We designed the CYBEX-P platform in modules. These modules live in different locations depending if the specific modules are trusted. The modules will live in three different locations. The first is the organizations premise, the other two are located in the CYBEX-P servers. The modules located in CYBEX-P premises are divided into two more categories, the DMS and the secure inside zones. These two zones worked in preserving privacy by moving all the mission critical privacy preservation to the secure side, for example, viewing data, storing, aggregating, analyzing, and reporting about the data is all done by the secure side of the Backend. In our CYBEX-P implementation, the segmentation was also done in a physical manner, there were five servers that defines these zones. One server was in the DMZ which hosted the systems API and web interface, and the rest in the secure side: two database servers, one analytics/report generating server, and one ingestion and aggregation server. We designed the platform in two or more zones, with the secure zone being the important one. The secure zone allows us to control the privacy of the data very strictly. Trust-

ing and ensuring this single point is secure implies the system is secure.

CYBEX-P can do data collection, analysis, privacy preservation, report/alert generation. The functions are supported by the six software modules we developed: the Input, API, Frontend, Archive, Analytics, and Report modules. These modules are composed of one or more sub-modules from Fig. 3.1.

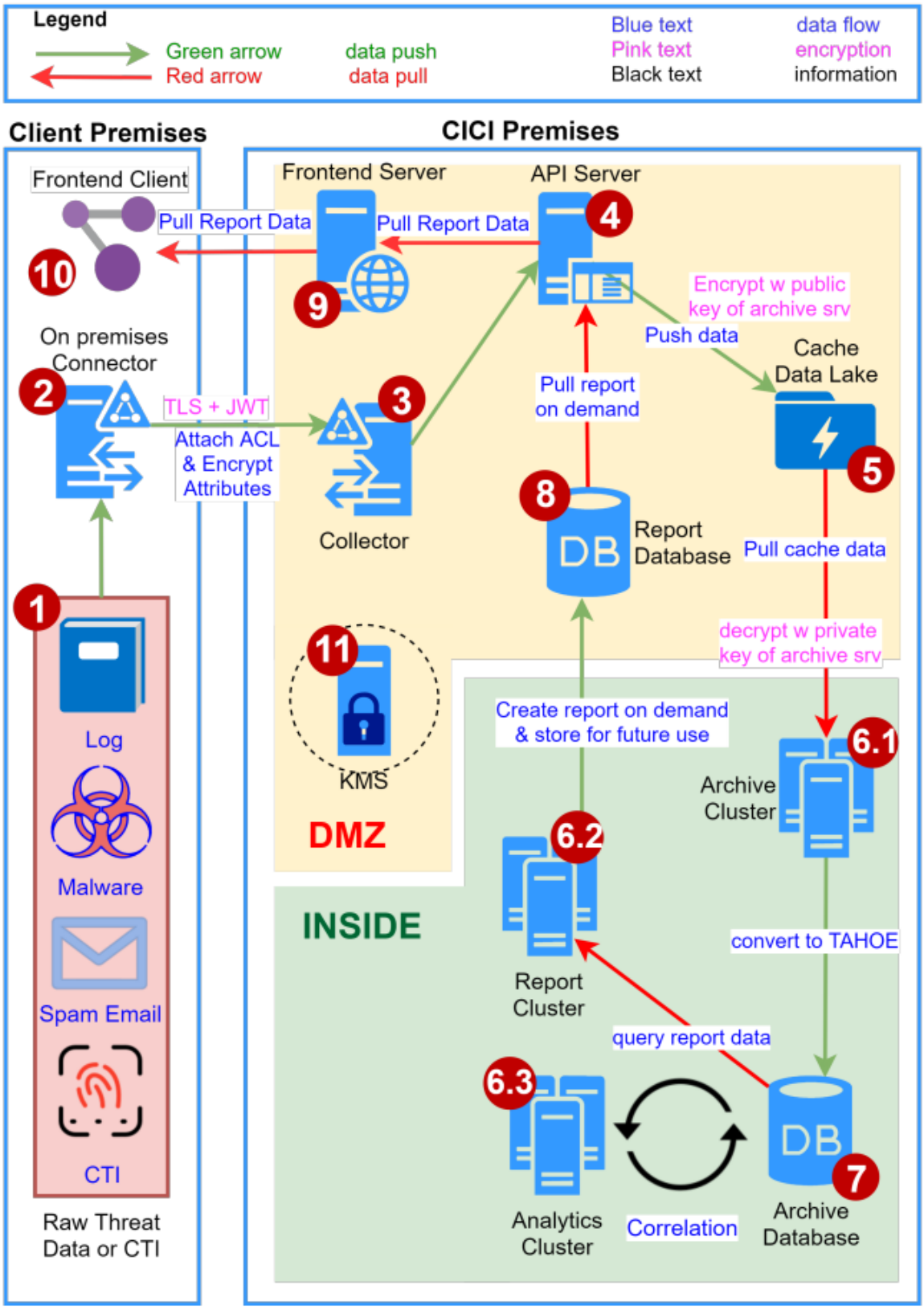

Figure 3.1: System architecture of CYBEX-P along with the Data Flow [1].

The Input module is composed of Collectors. Collectors are responsible for collecting data from users. One Collector unit is located in the web interface of the CTI sharing platform. Collectors can also be deployed where necessary like user premises. In our implementation of CYBEX-P, we deployed Collectors in our in-house honeypots.

The Archive module is located in the secure side and it is in charge of decrypting raw data and preparing the raw data for conversion to TAHOE Raw format, where it gets aggregated and stored in the main database. This step is the hardest to achieve because data types are heterogeneous, the standardization to TAHOE format allowed the system to work over any kind of data.

The Analytics module worked by converting TAHOE Raw data format to proper TAHOE format. This translates to interpreting the data and extracting proper meaning of it; the analytical modules will run against new data and create new data from it. For example, data from our in-house honeypots came from logs, these logs had connections to other events. This is where the modules will do actions like linking all the preexisting data, extract domains and resolve IP addresses, and generate connections between other pieces of data.

The report module will take a user's query and generate reports based on their query, for example, given an Attribute, find all of the related events (up to x degree) that happened between these dates.

The API is simply an interface for communicating between the DMZ and the secure zones. As a security measure no connections are allowed to go inside the secure side, all connections are outgoing from the secure side into the DMZ. This asynchronous communication method was the hardest challenge to implement as it limited functionality, all synchronous activities had to be broken down into a reverse connection/communication or else they will impact

the security of the system.

The flow the data takes across CYBEX-P is as follows:

1. Collect Data from either:
    - Public sources, ingested by Backend directly
    - Private sources, submitted by each party
2. Archive data, preserve privacy temporarily until the data gets processed
3. Aggregate data & Preserve Privacy
4. Analyze data
5. Alert & Report

In combination to Zone separation, the security comes from processing and manipulating data in the Secure Zone only. Anything that enters and leaves the Secure zone is encrypted in some way. In conjunction to the separation of zones and manipulation of data, privacy is preserved by identifying user's level of desired privacy preservation and strictly ensuring their data's privacy is ensured inside the secure zone by the Backend. It is paramount that the Backend behaves correctly to ensure all these properties.

More information on the CYBEX-P project can be found in the main paper [1], documentation can be found on the projects website [18], code can be found on its GitHub [19], and more on the user interface [20, 21].

3.2 CYBEX-P Encrypted Analytics

Initially, we explored and developed the CYBEX-P project in two threads. The first thread focused on the creation of a CTI sharing platform that aimed to

solve many of the problems with cyber-threat information sharing, like usability, aggregation, analysis, and privacy.

CYBEX-P intended to solve the privacy issue by anonymizing the data at ingestion time in the Backend. It did so by encrypting the data at the users' premises using the Backend public key. When the data is ready to be ingested for aggregation and analysis, it will get decrypted, stripped of ownership information and aggregated. It could later be further analyzed by the many post-processor modules CYBEX-P has. It maintains privacy by having one secure module manipulate data and enforce privacy.

CYBEX-P focused on solving multiple of the issues that CTI sharing platforms have and are reasons why organizations do not choose to join and share. The most notable one is what is done with the data once it gets ingested into the system. CYBEX-P maintained privacy by strictly enforcing Access Control Lists (ACL). Users submitting the data have to maintain an ACL list of allowed organizations to view their data. Centralized privacy preservation was sufficient to defend against misuse by enrolled members but it was not strong in regards to defend against attacks inside the Backend. The Backend had complete access to all the data. By extension, compromising the Backend means that the attackers also have access to all the data, bypassing the privacy.

The second thread of the CYBEX-P research dealt with maintaining as many capabilities of the main CYBEX-P project while keeping the data secure from the Backend, that is, CTI sharing and analytics. In the following paragraphs we will focus on discussing the second thread of research, CYBEX-P with Encrypted Analytics, hereafter called Encrypted Analytics (EA).

Our first implementation of Encrypted Analytics (EA) started by trying to mirror the goals of the main CYBEX-P project, share data and allow for analyt-

ics in the Backend, but with even stricter privacy measures. The main objective for the EA research was to achieve CTI sharing while maintaining computational capabilities for report generation at the Backend. The goal level of privacy preservation for this project was to also eliminate both the attack surface attackers had over the Backend and the surface the Backend had against the system. In a nutshell, privacy means not giving unauthorized users access to the system or data, including the system administrators and Backend.

The codebase for CYBEX-P with Encrypted Analytics is divided into modules. Each module is to be located on each part of the system as seen in Figs. 4.1 and 4.2. That is, Collector modules are located in organizations' premises, Backend is located in the system servers, and Query-client is located at the user's premises. The KMS module is independent to the system. Data flow and system architecture are identical for CYBEX-P with Encrypted Analytics and the Encrypted Analytics Framework. Both are described in more detail in the next chapter. The main distinction between EA and the Framework is how data is stored.

3.3 Encryption Primitives

In this section we detail how each primitive is used and what they achieve. These primitives apply to both the CYBEX-P with Encrypted Analytics and the Encrypted Analytics Framework, which are both discussed throughout this thesis.

3.3.1 Ciphertext-Policy Attribute-Based Encryption

Ciphertext-Policy Attribute-Based Encryption (CPABE) is responsible for the enforcement of access control and granular data access over data fields. When a user first inputs data into the system, the data will be separated into two layers, the data layer and the relationship layer. CPABE will be encrypting the data layer. Each piece of data that leaves a user's premise will be encrypted using CPABE as per equation 3.1. Users submitting data have the ability to create encryption policies POL for the encryption of $Data$, more on encryption policies in section 3.3.4. In a nutshell, most data gets also encrypted with CPABE using the systems public key $PUBkey$ and an encryption policy POL. The encryption key is said to be public to the system because it is used to encrypt the data, the key is shared among all the encrypting users. The data can later be decrypted only by private keys that meet the encryption policies requirements (determined at encryption time), any key that does not meet the policy's requirements will fail at decrypting the data. Every user in the system has a unique CPABE key tied to their organization's attributes (not to be confused with Attributes from section 4.1.3). Key generation is carried out at the KMS, where it will generate two keys, the public key which can be used to encrypt, the master key which in conjunction with the public will be used to create private keys, and lastly private keys that will be created using the public and master keys as well as a list of attributes that the key will be able to decrypt.

The decryption process can only be carried out of the private key decrypting the encrypted cipher contains and satisfies all the attributes of the encryption policy. Neither the key or the cipher can be modified to bypass this restriction and its enforced by the encryption primitive. The math behind the primitive ensures the access control and any decryption that does not meet the policy

will simply output garbage. Policies specify what attributes will be allowed to decrypt the data and can also include *AND*, *OR*, and parenthesis *()* operators, where attributes can be combined with these operator to make more complex access policies. For example “REASERCH and ITOPS”, in this case a key would have to have both attributes in order to decrypt data with this policy. Multiple keys that partially satisfy the policy can not used to decrypt the data, even if they collectively satisfy the policy.

$$C = CPABE\_ENC(PUBkey, POL, Data) \tag{3.1}$$

Exactly what CPABE policy is used for encryption is up to each organization. As far as primitives go, every piece of data will be encrypted using Deterministic Encryption and CPABE unless otherwise overwritten by the encryption policy, that is describes in section 3.3.4.

3.3.2 Deterministic Encryption

Deterministic Encryption (DE) is used as a one way function for data, that is tied to a specific deployment, much like a hash with a secret salt. The purpose to use DE is that it allows the system to create indexable Identifiers (IDs) for each piece of data without compromising the confidentiality of the data. DE is used over hashes to ensure the ID generation is tied to this specific system (prevent collisions with other deployments) and to prevent brute force attacks on the data as it is encrypted using the system’s public DE key (public to system users, not outsiders). This does not prevent insider brute force attacks as they will have the encryption key.

We use a single symmetric encryption algorithm to achieve what we have

called Deterministic Encryption throughout this paper. Asymmetric encryption is used to achieve one way functionality. When the key for the asymmetric algorithm gets created, the private key is immediately discarded and overwritten in memory by the KMS. Discarding the private key ensures that the one way functionality is maintained throughout the lifetime of the public key. We assume here is that our primitive is at least equally secure as the underlying encryption primitives, therefore using and choosing modern and secure encryption primitives is key for this system to maintain those properties. We also rely on the KMS's trustworthiness for creating and immediately disposing the private key. If both of these assumption are met we can confidently say this is a secure algorithm. Enforcing these two properties are not beyond the focus of this paper.

Value data is encrypted using both DE and CPABE, DE generated the unique ID for each piece of data in the system for searching and retrieval purposes. IDs are also used as a relationship pointers. CPABE is used to encrypt the value of the data and are referenced using their unique IDs which allows for retrieval by the querying parties. The encrypted IDs are only for retrieval and data lookup, since IDs are encrypted in nature they protect the data. In the event that an attacker were to get access to the database everything will be encrypted: the IDs and the value data. It is only when the data is in the hands of an end user with an appropriate CPABE private key that they will be able to decrypt the value data layer if they possess the right attributes. The relationships are simply inferred from the pointers and the decrypted data is arranged in the same format.

3.3.3 Order Revealing Encryption

Order Revealing Encryption (ORE) is used to encrypt data by which the submitting users feel/desire their data should be filterable by. Usually submitting users will setup their encryption policies to allow this additional piece of encrypted data to be generated for some of the attributes in their data. ORE encryption can be used in any attribute that is integer based or that can be converted into integer, this includes things like dates and time (not limited to). The encryption process is determined by the Collector at ingestion time based on the encryption policy that the organization has setup. If a an organization recognizes that their data would benefit more users if it was filterable by time for example, then they can enable ORE encryption over the Attribute, which would allow other users to filter based on that. Equation 3.2 shows how the data gets encrypted, $C1$ is generated at ingestion time by the Collector, $C2$ is generated by the Query-client, and equations 3.3 are carried out by the Back-end for data filtering. It is important to note that encrypting using the ORE mechanism reduces the privacy of the data, making it subject to brute force attacks by insiders.

$$\begin{aligned} C1 &= ORE\_ENC(PUBkey, Attrib) \\ C2 &= ORE\_ENC(PUBkey, Query) \end{aligned} \tag{3.2}$$

$$
\begin{aligned}
C1 &= C2 \\
C1 &> C2 \\
C1 &\geq C2 \\
C1 &< C2 \\
C1 &\leq C2
\end{aligned}
\tag{3.3}
$$

3.3.4 Encryption Policies

Encryption Policies determine how the data is going to be encrypted at a given Collector. It allows for granular control over access of fields/attributes. A CPABE policy is established prior to sending data. Policies only affect how the data is encrypted, the data later leaves the premises and those policies are ultimately enforced by encryption.

Encryption policies are written in terms of how the data will be encrypted. Users have the ability to choose what data is going to be encrypted with fine precision. The system uses regular expressions to match attribute labels. Regular expressions allows the user to match more than they know about their data. For example, if they have a field that they may know about and would like better control over but don't necessarily know what the field could be named (which is unlikely because they should know their data, out of scope), they can use regular expression to encapsulate more matches with wildcards.

Policy matching is done according to the Collector's configuration file (referred to as encryption policy in this paper), an example configuration can be seen in Fig. 3.2. It specifies rules to match fields in each record. Each rule has an associated CPABE policy that will be applied to the field if matching. If

no rule matches, the default policy is applied. One can use an exact match or regex match for a field. Encryption is done field-wise. Aside from encryption policy, the policy administrator must also specify what field will be indexed for encrypted searching.

```
policy:
 default:
   # If no other policy matches below
   #  Then this CPABE policy will be used
   abe_pol: "default_public"
 index:
   # These two will match fields with to
   #  create searchable encrypted fields
   #  this will force better indexing
   #  on fields (optimization)
   exact:
     - match: "ipv4"
     - match: "ipv6"
   regex:
     - match: ".*src.*"

# The following rules will match specific
#  fields if a field matched it will be
#  encrypted with the corresponding rule
exact:
 - match: "event_id"
   abe_pol: "itops_AND_ciso"
 - match: "count"
   abe_pol: "default_public_AND_research"
regex:
 - match: ".*ssn.*"
   remove: True
 - match: ".*timestamp.*"
   abe_pol: "itops"
```

Figure 3.2: Example encryption policy file, in YAML format.

If a user decides that they don't care about certain data then they can let it be encrypted using default policies. These default policies are still user defined and they catch any piece of data not matched with fine control.

Every piece of data will be encrypted using DE for indexing as it creates data IDs. Data by default will be encrypted using CPABE using the organization's default CPABE policy if the data does not directly match any other encryption policy. If the data does match a policy, the Collector will use that policy for CPABE instead. This means that data that is not explicitly tagged for removal will be encrypted using default settings. This means that the Collector will usually send the data to the Backend encrypted by DE and CPABE. These default settings are set by each submitting organization.

When specifying that a data is accessible by certain organizational attributes, the policy will establish what decrypting attributes will have access to a particular label. For example, consider the label "source_IP" with CPABE policy "RESEARCH or ITOPS". The person querying must be either "RESEARCH" or "ITOPS" to decrypt that field. CPABE policies are done in a field-by-field manner.

Data can also be marked for removal, which means that the Collector will make sure to not send said data to the Backend. This feature is useful if data is too sensitive to be sent, or the data is not useful so it is removed.

3.3.5 CPABE attributes

Access control attributes are labels used by the Ciphertext-Policy Attribute-Based Encryption (CPABE). These attributes are the basis of the access control mechanism; they are used to specify what and who can decrypt the data (field-wise). Encryption policy acts like boolean logic, where a user either possesses all of the required attributes or not (in their private CPABE key) to decrypt the data. Access control attributes can range from trivial things like company names, to groups or collections names, to broad of attributes. For example,

Google can have attributes like *SearchEngine*, *Google* and more specific groups/users can have attributes like *Research*, *IT*, *CISO*, etc.

For access control to work properly, both the encryption policy's attributes and the attributes in the private key need to match; therefore, it is important that attribute distribution and usage is done consistently. It is paramount that attribute creation and usage is standardized. The KMS is responsible for creating these attributes; therefore, a mechanism to keep track of assigned attributes is key to successfully decrypting data. Without knowing what attributes are available in the system it is not possible to make the data useful to any of the users, as using an incorrect attribute or non-existent attribute will result on inaccessible data. In our framework, implementation of the KMS provides users with a list of current attributes in the system to address this issue. Since access control uses CPABE we can encrypt data using non-existing attributes that may later be assigned. This will temporarily create inaccessible data, at least until the attributes used to encrypt the data are assigned to someone.

3.3.6 Data Structure & Flow

The data, as previously mentioned, flows from the Collector to the Backend, and later on to the user. Starting at the Collector, the data will be encrypted field by field using DE and CPABE using the encryption policies set by the organization submitting the data.

The main distinction between CYBEX-P with Encrypted Analytics and Encrypted Analytics Framework is the data structure that is used to store the data, as seen in algorithms 1, 2 and algorithms in Chapter 4. In EA the data gets separated field by field, encrypted using the encryption policy set by the user, and also encrypted using Deterministic Encryption. Data then gets bun-

dled into its respective Events and Sessions, where it can later be retrieved by the Backend by searching in the database for Events or Sessions with a specific DE cipher. Results could be Attributes/Events/Sessions. Once data is gathered by the Backend, it is up to the querying party to try to decrypt the rest of the fields in the results as they are subject to the CPABE set by the submitting organization. As each data will be encrypted by different organizations, there is no need to do deduplication, as the CPABE data will always result in different encrypted outputs. Events and Sessions are always assumed to be unique. Therefore this system does not deduplicate data, as there is nothing to deduplicate by nature. In the Framework, the architecture and data flow are the same as EA. However, the data structure for data is a graph of relationships layered with an encrypted data layer. More on this structure in the next chapter.

Algorithm 1 Data encryption & storage algorithm overview, CYBEX-P Encrypted Analytics

Input: Raw data
Output: Encrypted Data

```
1: Collect new data
2: Separate each new bundle/Event/Session of incoming data to its simplest components
3: for all Att in components/Attributes do
4:     Encrypt Attribute with DE
5:     Encrypt Attribute using CPABE
6:     if Attribute == type(int) AND ORE in Policy(Att) then
7:         Encrypt Attribute using ORE
8:     end if
9: end for
10: Discard plain-text data
11: Combine all of the encrypted data into a single unit
12: Send data for data storage to Backend
13: Backend will append bundle to database
```

Algorithm 2 Data query algorithm, CYBEX-P Encrypted Analytics

Input: Encrypted query
Output: Decrypted Data
1: Encrypt query's Attribute using DE
2: **if** Integer Filtering desired **then**
3: Encrypt integer/time ranges using ORE
4: **end if**
5: Send encrypted value to Backend
6: Backend searches for all records that contain $enc(Att)$
7: **if** Integer Filtering desired **then**
8: Backend filters results using ORE comparison
9: **end if**
10: Return results to user
11: User's Query-client will attempt to decrypt records if able using CPABE
12: Reconstitute record bundles

3.4 Capabilities & Data Input

CYBEX-P with Encrypted Analytics is capable of preserving privacy by fully encrypting the data. It uses multiple encryption primitives to store the data. Each encryption primitive adds a new layer of usability to the data, allowing for simple functionality. The first functionality that this project achieves is that it can ingest heterogeneous data, store with privacy preservation mechanisms, and retrieval of the data. Retrieval of the data can also be done with queries over the data.

In the implementation of our first CYBEX-P Encrypted Analytics project, our system had a relaxed input type. The data was encrypted field by field and stored as a bundle. That is, a whole event will be dissected into Attributes, each attribute will get encrypted, and then all of the Attributes of that event will get stored together in a single unit. This meant that looking for an attribute, the Backend will search for all bundles of data containing the encrypted Attribute. Any DBMS supports this kind of query, which made it very efficient. Looking for a particular attribute in the database yields whole bundles of data, and not

how they are related to one another or other pieces of data. We achieved simple lookup queries on any specific attribute, count of specific Attribute/Events/Sessions queries, and integer filtering queries. Time and integer/float range filtering queries can also be stacked with any other query.

These queries allowed us to produce a complete CTI sharing platform with privacy preservation even on its core. EA, however, could not do any more advanced analytics on the data, only lookups. As far as data ingestion goes, CYBEX-P Encrypted Analytics can ingest any type of data. For example, it supports some of the data types NIST recognizes as CTI, indicators of compromise, system artifacts, Tactics, Techniques, and Procedures (TTPs), security alerts, reports, and recommended security tool configurations [4, 5].

This however is not true for the EA framework, the EA framework only supports certain types of data. It supports all the basic queries EA support and new relational queries. The new relational queries are based on a graph data structure which represents relationship information. Due to this data structure relational queries are only supported if ingesting information with such relationships. Encrypted Analytics frameworks also supports data aggregation and which enables data fusion across a variety of heterogeneous and unstructured data sources.

3.5 Obstacles with first implementations

CYBEX-P had big assumptions about the security of the system, the Secure zone had to be secure else the privacy guarantee fell apart. We addressed that by implementing a platform that did not rely on the Backend for data privacy. EA used encryption primitives that where carefully selected to add more usability

to the system, but it was not enough, it only achieved basic queries over the data as the encryption restricted what calculations the Backend could do. The data structure was very simple and could be completely handled by the DBMS. This also means that the data can also be decentralized. The encryption and data structure made it extremely hard to do any meaningful analytics, resulting in a platform for sharing the data with some analytics like counts, searches, and integer filtering. For this reason is that a redesign of the system was in need.

3.6 Summary

CYBEX-P is complex in its design, as it operated in the very principle that there is a trusted side and an unstructured side. Everything in the unstructured side could be compromised and the data will still be safe. This separation by data handling assumes the trusted side is secure. If we challenge this assumption we see that the privacy of the model is compromised.

It is important to note that in reality, this centralize party has access to all the data and controls all of it, whether the mechanisms for privacy are working as intended are up to this centralized party, and any malfunction or malicious behavior could and will compromise the security of the system. This also means that there is a single point of failure for security. Access to the Secure zone means access to all the data, violating privacy for all the users and data.

We aimed to solve this problem by introducing strict Access Control measures enforced by encryption at ingestion time. This however made it very difficult to produce complex analytical insights of the data at the Backend. Further information about the CYBEX-P Encrypted Analytics project, implementation, code, and documentation can be on its official website [22].

In CYBEX-P, data maintained a reference to the ACL list. With the ACL, when data is queried, data would only be returned to the querying user if they had access to the data. All of this happens at query time, meaning ACLs and data submission could be edited/ingested at different times. In contrast to the Encrypted Analytics framework in 4, the framework uses a similar mechanism to maintain privacy, that is, access control. However, access control is enforced by encryption at submission time and can not be modified later. As data is no longer in plain text, the Backend can not decrypt it. This furthers the strength of privacy by rendering the data unusable to outsiders.

In combination with the previously mentioned work, we developed a new Encrypted Analytics Framework that enables privacy preserving CTI sharing with encrypted analytics. The framework address all the aforementioned issues by introducing a graph data structure to store some of the information.

The new framework borrowed most of the structure from the EA project and sub-components, where most constraints remained the same. The major and only difference is how the data is stored. Storing the data in a different structure changed how we conducted analytics, achieving something that to our knowledge has not been done before. The following chapter will reintroduce the components that make up the framework, how data flows from component to component, how the data achieves its core functionality of aggregating data to support encrypted analytics and lastly a guided example on how the data would get processed on a live deployment.

Chapter 4

Encrypted Analytics Framework

4.1 Architecture

4.1.1 Overview

In the project model, no party trusts the underlying transportation and central storage mediums. This project assumes that the Key Distribution Service (KMS) is a trusted entity that sharers can rely on to verify and adequately assign attributes to new platform members.

The system has four parts as shown in Fig. 4.1 which correspond to the actions of each module in Fig. 4.2.

Collector

The Collector is in charge of encrypting the data with the encryption policy set by the organization that wants to share the data. The Collector module itself has two parts, the encryption module and the Collector-client which gather

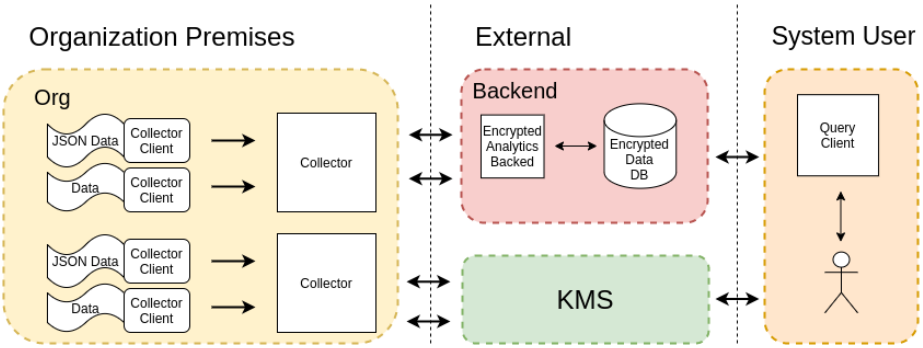

Figure 4.1: System architecture, shows interaction paths.

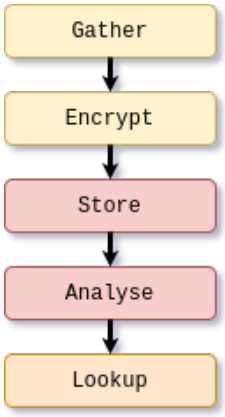

Figure 4.2: Overview of basic interaction principles of the system.

data from user endpoints. The encryption module encrypts any data sent to it by one or more gatherers and ships it over encrypted to the Backend server for aggregation and storage.

Backend

The Backend/API/DB collects the encrypted data and stores it into a database. When a user queries the Backend using an encrypted version of the query, the Backend returns applicable encrypted data using encryption primitives related to both value data encryption and relationship encryption layers.

Query Client

The Query-client handles queries from a user. The Query-client handles all tasks necessary to carry out a query: the encryption of the query, the API call, possible encrypted analytics (client-side analytics), and data decryption if possible. The user querying will only be able to decrypt data that the original encrypting user desired; this is enforced at encryption time with CPABE policies.

Key Management System (KMS)

The KMS handles key creation for new system members, attribute assignment, and key distribution. New keys are generated based on a user's/organization's attributes.

The KMS is used by the Collector to get the public keys when encrypting, by the administrator when setting up the Collector's policy to determine available attributes, and by the Query-client to get private keys for decryption purposes. The KMS server is not needed at run-time if keys are preloaded and stored locally by each module.

4.1.2 System layout & data flow

In Fig. 4.3 we see the flow of information from end-user contributors to end-users that query for data. In a nutshell users gather their data, convert it to a standardized format, encrypt it and send it over to the Backend; the Backend then stores the data and aggregates it. Users can later request the data from the Backend, where it gets analyzed based on queries made. Once it reaches its end-user, it can be decrypted if decryption policies allow it.

In Fig. 4.1 we see the system layout. It is composed of a single Backend,

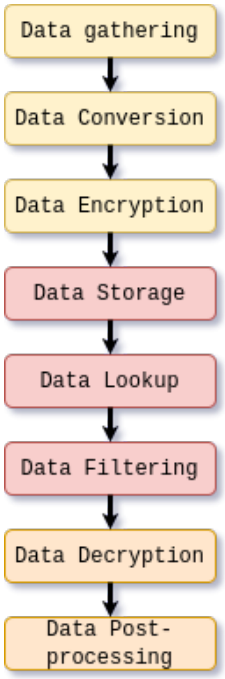

Figure 4.3: Overview of data manipulation throughout the system modules

a single KMS and multiple Collectors and Query-clients. Each organization is allowed to have multiple Collectors to suit their encryption needs. The system was designed this way for data safety, later described in section 4.3. Organizations can also have more than one Query-client.

In our framework we have a single Backend, responsible for data storage and retrieval, users can send or request encrypted data to and from. In this paper, Organization/users are broken down into their two action components, submission and retrieval of data, as seen in Figs. 4.1 and 4.4. For the sake of simplicity, Figs. 4.1 and 4.4 only show a single Collector and a single Query-client, they can belong to a single organization or separate organizations. Keep in mind that this framework supports multiple users with multiple Collections and Query-clients.

It is important to note that data gathering and data encryption (as depicted in Figs. 4.1,4.2, 4.3, 4.4) are done in conjunction by a single entity/organiza-

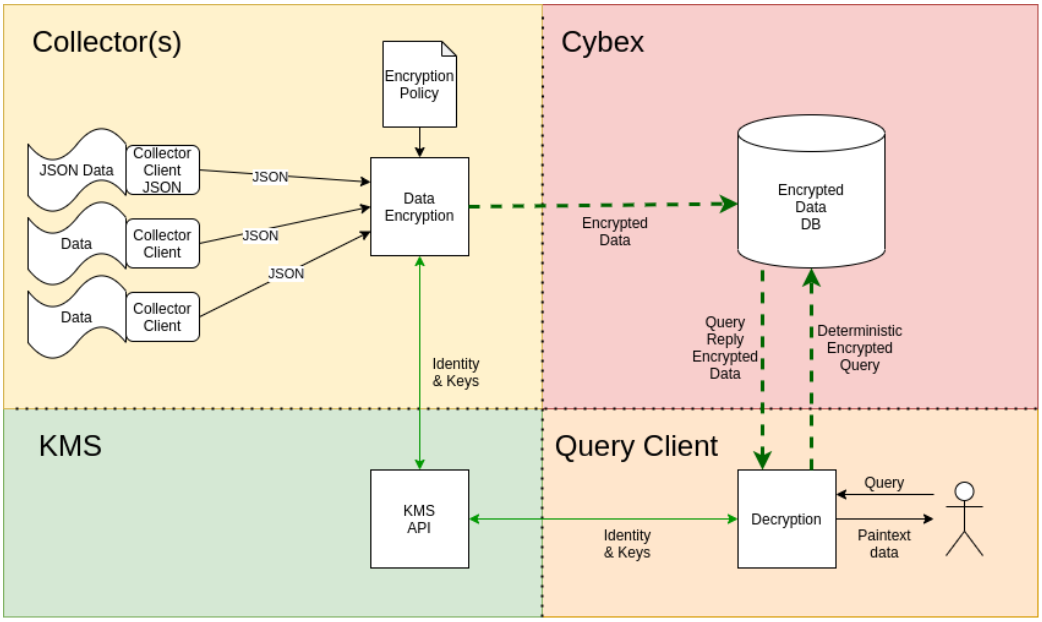

Figure 4.4: Interaction of system components and data.

tion as they are coupled to a specific organization's needs. On the other hand data, data submission is not limited to a single organization; multiple organizations can submit and contribute. In other words, data gathering and encryption (policies, not algorithms) are subject to each organization. Data submission can be done by any of the participating organizations. A single organization is not responsible for all data encryption, each independent submitting organization is.

4.1.3 Data Layout

The data is divided into three types of data, which later get converted into two encrypted layers. These three layers include session, events, and attributes. Figure 4.5 shows the relationship between these three data types. Sessions are made up of events, events are made up of attributes. Attributes are made up of labels and data. The two encryption layers are the relationship layer

which is made up of the relationship between sessions and events (links specifically), and the data value layer, which is the encrypted data of the attributes and events. Figure 4.5 gets converted into Fig. 4.6. The relational data gets encrypted with one way deterministic encryption (DE), while Attributes get encrypted with Ciphertext-Policy Attribute-Based Encryption(CPABE), both later described in section 3.3. In Fig. 4.6 components unique identifiers (ID), generated by the one way deterministic encryption algorithm (DE), are represented by the arrows/links (ID of the Attribute or Event, destination of arrow). IDs are also stored with the attributes for direct retrieval of encrypted data, and inside relational data which is further described in section 4.2.

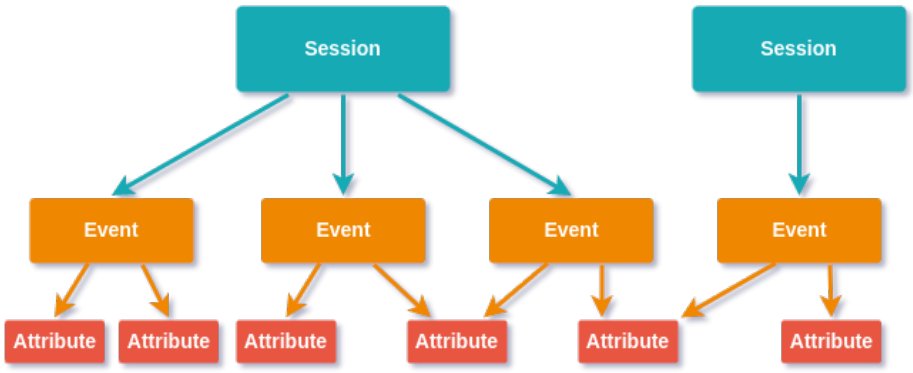

Figure 4.5: Shows three main data types and how they are related to each other. That is, sessions, events and attributes.

Attributes

The most common piece of data is attributes, they are simple label-value pairs that represent something. For example, an IP address can be represented as the label *IP* and value *1.1.1.1*. Attributes do not contain more information aside from their label and value. For example, *1.1.1.1* is the IP address of Cloudflare's DNS server but this particular information is not represented by an attribute, possibly represented by a higher piece of data like an event or session. In our

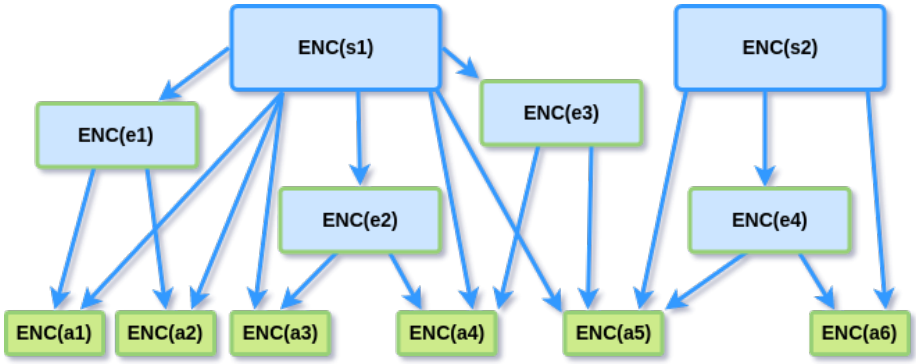

Figure 4.6: Shows the relationship encryption layer in blue and the data value layer in green. Relationship information is encrypted with DE and value data is encrypted with CPABE.

database, Attributes are stored as pairs of labels and values. They do not store which events they are related to; Events and Sessions store what attributes they have a relationship with. This allows for data deduplication as multiple events can be related to the same attribute, and also decouples attribute values from Events and Session which will later allow us to create two new encryption layers, later described in section 4.2.

Events

Events represent something that has happened. This is not to be confused with sessions. Events only represent a single happening, for example a file download or a user login. Each event is described by one or more attributes. For our purposes there is a requirement that each event has at least one attribute describing the event type. Event's attributes describe or give more information about the event itself. For example, an event of type *login* may contain a timestamp and an source IP, a download event may have foreign server IP, filename, file. Events are not to be confused with sessions.

Sessions

Sessions describe what events are connected in a series of events. For example a user may have logged in, ran a command to check for root and then tried downloading a file before finally trying to logout. Sessions are just a collection of Events.

The data is assumed to have relational information when it is submitted by the users. Relational data is what a session is and it is key that it is known before submission. This does not mean that we must know how an event from a particular session is related to a session/event form another user, but instead that in a particular session we know that some events are related to each other. This relational information is usually automatically generated by the software logging the data. In the event that data does not inherently contain such relationships in the specified format, the Collector-client should be developed to convert data on-the-fly before sending it over to a Collector for encryption, as shown in Fig. 4.4. Collectors have a specified input data format and Collector-clients should adhere to said format; in our implementation that format is JSON.

4.2 Data submission and aggregation

Submission of data starts at each organization. Each organization determines the importance of the data/data fields they are sharing. With that information, they create an encryption policy. The organizations write the encryption policy in terms of the fields and their respective encryption policy, i.e., fields and attributes that can decrypt the said fields. The decryption of information is enforced using attribute based encryption (CPABE), not to be confused with Attributes from our data types. Encryption attributes refer to characteristics or

groups an organization belongs/represents, for example an organization may be part of *RESEARCH* and *DEVELOPMENT*, but may not be represented by *FARMING*, these attributes are determined by the KMS at enrollment time.

In section 4.1.1 we describe the data flow between certain subsystems, in section 4.1.2 we describe how the data is generated, formatted, encrypted, aggregated. The Collector is responsible for structuring the data for encryption before the data gets appended into the encrypted database by the Backend. The Collector will first convert the data into a standardized format according to it's structure and meaning. Here, we assume that the data is in a standardized structure, as shown in Fig. 4.5 and that, in this case, the Collector knows exactly how to extract that structural/relational meaning. This assumption is critical for doing data encrypted data analytics at the processing Backend. Once the Collector has standardised the data into the particular data format, e.g. as seen on Fig. 4.10[D1], the Collector can start processing the semantics of the data to create a graph structure.

The encryption procedure uses multiple encryption primitives, later described in section 3.3. One can use unique encrypted reproducible IDs for each piece of data to create the relationship information. This information can be created by the Collector software at ingestion time and can later be recreated in a similar fashion to retrieve data in the direct query procedure. The process for doing related query is more involved. The following paragraphs will go over how the value data and relationship encryption layers are formed.

This relationship information is stored encrypted (not double encrypted, the generation of IDs is already encrypted) in a graph database in graph format. This allows operations like graph traversal allowing for analytical questions to be asked about the data, which is what this framework intends to support.

In the following paragraphs we will separate relationship/links from the value data encryption scheme, we will be using Fig. 4.7 as an example. Ultimately, one can use any graph traversal algorithm to traverse the encrypted data, achieving advanced queries over the data without knowing the data or the relationships between them. We will later include another piece of data in Fig. 4.13 to demonstrate the data traversal algorithm, aggregation, and querying.

Each piece of data is converted to two major critical pieces of data. The first data that is created is in the node/link format as seen in Fig. 4.11, i.e. it's unique ID, to be able to link things into a graph. The second piece of data is the encrypted data itself, for the end user to decrypt. In this system we can have multiple users input similar data.

We handle similar data in the following format: node data will only be create once per piece of unique data, this means that graph traversal information will not be duplicated.

The data (data value) and relationship information (node data or graph data) work on two different layers, which are decoupled. This allows us to encrypt them in completely different encryption mechanisms. The real values of the data and field name data can be encrypted using mechanisms like CPABE which grants us access control capabilities. The relationship data is encrypted using irreversible (one way) Deterministic Encryption (DE) encryption that uses the system's encryption secret, therefore deterministic but only for valid system users for their particular system deployment.

The real value of the data is encrypted based on user's desired encryption policy. If we have the scenario where two users submit the same IP for example, it will result in only one data node being created and two pieces of value encrypted data. Node data and encrypted value data are completely indepen-

dent from relational data (data ID), and it is not necessary to know the real or encrypted value (CPABE encrypted) of the data to traverse the relationships in the graph. This makes it possible to traverse the graph/relationships without knowing the data.

Information in the value data layer is encrypted using CPABE with an encryption policy that is up to the submitting party to decide. This also means that leaves are encrypted with different encryption keys. If two entities submit the same value data it will result in one unique Identifier (ID) with two value data leaves. Relational information will be able to relate to said ID. Later in the retrieval process, querying users will be able to pull either data form the database using that specific ID, depending on their decryption capabilities (organization attributes), i.e. whether they are allowed or not.

With a single pass (traversal search) of the database we can establish multi-directional information leaf to event and event to leaf, without knowing what the leafs are. We can fully traverse multiple levels of the relationships to answer many analytic queries. Our two layer solves the problem of excessive processing times from off the shelf graph databases, they are not able to obtain all possible relationships and all their paths in a useful timely manner, they are not design with this problem in hand, they are designed to find a single shortest path.

By using only one-way deterministic encryption for the relationship graph creation and full value data encryption mechanism (actual encryption described in section 4.3), we can achieve a full range of relational queries by simply traversing the graph. This comes with a caveat, information can be learned by the analytics engine. For example lets say link1 is usually related to link2, a malicious entity will not be able to know to what they point to (what is the

original value of the data, that is, the leaf) but will be able to know that both obfuscated links may be related or possibly of interest. As previously stated, link IDs are derived from the data itself and are directional towards the leaves from events. The value data and any information relating the link is safe, the encrypted links themselves are not. In our framework we don't see this as a trade-off between data usability and privacy. Further research needs to be conducted on whether relationships between encrypted links without knowing what they point to can lead to information leak.

In the current state of the research, we concluded that by including access control mechanisms it is not possible for the Backend to conclude any tangible statements about the data. We will use a high level example to demonstrate.

For example in the malicious context of machines, if we, in plain text ask about the data for the relationship about the probability of a given machine to be malicious based on the IP address location (for example China). Without having direct access to the leaf (IP, location, malicious rating) of the graph it is not possible to calculate the statistic. Using the same example, if we asked how many machines located in China are malicious versus not (ie probability of being malicious give the location is China) without using China or Malicious context, we can not convert China and Malicious to their corresponding identifiers. The graph database stored their identifiers which for the sake of this example we will call ID1 and ID2, we are still able to evaluate the probability of ID2 given ID1. In our framework we are redacting the data by encrypting it with CPABE to control who has access to the original data, in this case the value of the encrypted data that ID1 and ID2 point to (China, malicious). This is essentially the gist of the two layers of data, node/relationship data and data value. This also grants options like allowing users to retrieve queries like prob-

ability of ID2 given ID1, but not probability of malicious given the location is China. Users with the correct CPABE access will be able to decrypt the data layer into the latter but not authorized users will only be able to see the former.

The user querying will be able to know what ID1 and ID2 are as they are the ones that generate the deterministic IDs for the query by encrypting the query values malicious and China in this example. It is important to note that this example is only one type of query, other queries may return other types of output. Regardless of the output style, all data is returned and only made available in an encrypted format. For example, if we ask in what context does a certain thing (ID3) happen?, this query could be equivalent to finding all the related neighbors and paths on the graph. This query will return all related paths associated with ID3. Extracting further meaning from "in what context does ID3 happen" will not make any sense unless we can decrypt what ID3 points to, which is subject to the encryption policy of the submitting parties.

Query traversal is done the following way. A user can encrypt the data to find its link ID which can then be passed to the encryption analytics engine. The encrypted analytics engine will then traverse the graph and collect the appropriate results for the user. This includes things like related queries or any other query. Then the user can also download all the appropriate encrypted value data for that particular query and if able to decrypt the data. As stated above, the user querying knows the base parameters for the query, he will at least know them about the encrypted information, but nothing more. Any other information will be secured by access control, any further meaning will be subject to CPABE encryption policies.

The encrypted analysed engine will do a graph traversal on the encrypted data using the encrypted value passed by the user querying. Some partial ana-

lytics can still be done at the analytics engine as explained in previous sections. The user may be able to decrypt results even further into plain text. Only at the fully decrypted stage the data will be in its most meaningful state.

4.3 Encryption usage

This section details the procedures that each module carries out and what encryption primitives they use to achieve the overall goal of the system as detailed in sections 1.1, 4.1.1, 4.2. Further on encryption primitives in section 3.3.

Collector client

Algorithm 3 explains the flow of data from collection point to delivery. They collect data from a data generation node, where each Client is specialized to conform to each data retrieval method and input data format. Collector-clients are the only modules that are specialized to their input data structure, other modules are generic in the sense that the code stays the same regardless of the data, where others it's the configuration that changes. Each Collector-client's implementation will vary from data to data, with the common goal to collect data from a single node, standardized it (as specified in Fig. 4.5), and prepare it's format and structure for encryption. Then it transfers the information to a Collector unit. Collector-clients are made to be reused if they ingest the same data (format and semantics), meaning the same implementation can be deployed across multiple data generating nodes.

Algorithm 3 Algorithm for Collector-client logic

Input: Raw data
Output: Standardized data
1: Collect new data
2: Build relationships if not explicitly present
3: Convert to Collector's input format
4: Pass data to previously selected Collector

Collector

It will ingest data as it arrives and encrypt it. There could be many Collectors units configured within an organizations premises. Each Collector can only carry one encryption policy, therefore Collector-clients will be configured to send data to a specified Collector, this is to prevent sensitive information from being incorrectly encrypted and sent outside of the premise. The design solves the problem by reducing the chance of error by creating a pipeline that will always see the same kind of attributes where a policy is made specifically for that data, more is discussed about policy in section 3.3.4. For example, there could be a single Collector encrypting server connection logs, where data can come from two nodes with different formats but the data is similar. Collector-clients will help with the format conversion and Collectors will do its only job, ensuring that the records are properly encrypted before leaving the premise. One would not want to send for example web server logs to this Collector, as it is only properly configured to match connection logs. The default policy for unknown attributes in a record is to submit with maximum restrictions on the data. This should prevent usage of the data while allowing for generic analytics using the encrypted relational data without using the value data. The Collector will act according to algorithm 4.

Algorithm 4 Algorithm for Collector logic

Input: Standardized data
Output: Encrypted data

On startup validate configuration
if Keys not preloaded **then**
    Retrieve system and private keys from KMS
end if
Perform key testing
loop
    Get data d from Collector-client
    Extract all attributes A from data
    for all attributes a in A **do**
        Encrypt a based on encryption policy
        Generate unique Deterministic Encryption (ID) of a
    end for
    Build event objects E and session objects S
    for all event e and session s in $E \cup S$ **do**
        Generate unique Deterministic Encryption (ID)
    end for
    Remove traces of unencrypted data from memory
    Send objects, attributes and any extra encrypted values as per policy to the Backend
end loop

KMS

The KMS does not handle any data; it's only job is to generate system, public and private keys for users to use. It's logic is described in algorithm 5. It will only distribute to keys to legitimate members of the system. System keys are only used for encrypting data with coarse access control, meaning only holders of the system keys will be able to query the system without being able to decrypt the data, allowing for generic system queries. Public and private keys are used to encrypt the data with granular control over direct access to the data. Algorithm 5 is described in terms of the query the user will make, but in reality the Query-client will request keys from the KMS and not pass the query to the KMS.

In reality there are major assumptions that have to be followed for this system to be secure. The underlying security of the system comes from the encryption mechanisms used and proper key management, the KMS must be able to securely generate, store, and distribute keys. This include things like picking cryptographically secure keys. The KMS must also correctly verify new users as well as authenticate current users before distributing their updated keys.

Users can have the option to store their keys, removing the need to constantly contact the KMS for keys, this also means that if a user was given new attributes they will have to fetch the new key from the KMS. This research is did not explore key management. Things like key revocation and re-generation were not considered.

Centralized Backend

The centralized Backend is responsible for ingesting encrypted data, storing it, and later retrieval of said data. The analytics is mostly done by the centralized

Algorithm 5 Algorithm for KMS logic

```
On startup validate local configuration
if First startup then
    Generate random numbers for public keys initialization
    Generate public DE, ORE and CPABE system keys (public to legitimate users)
end if
loop
    for all New users do
        Generate CPABE private key
    end for
    for all Users wanting to encrypt do
        Send DE, ORE and CPABE system keys
    end for
    for all Users wanting to query do
        if Coarse query then
            Send DE system key only back to user
        else if Fine query then
            Send system DE key and corresponding CPABE private key to user
        end if
        if Query also is range filtered then
            Send ORE key to user
        end if
    end for
end loop
```

Backend, also referred to as analytics engine in this paper. The Backend will first ingest the data it receives and store it depending on the type of data it is, that is, if its an attribute, relationship. The storage procedure is described in algorithm 6.

Algorithm 6 Algorithm for Backend ingestion logic

```
Input: Encrypted data
for all Items received do
    if Is relationship then
        Store relationship into database
    else if Is attribute then
        Lookup attribute in database using ID
        if ID exists in database then
            Append encrypted value to ID in database
        else
            Insert encrypted value into database using ID
        end if
    end if
end for
```

The storage of the data is straight forward as the graph generation is done at encryption time by the Collectors. This allows for the data from multiple users to be aggregated and later analysed. The relationship data is simply stored in the Backend for later usage, as they link to attribute data. Attribute data on the other hand, has two fields, the unique attribute identifier and the value. The unique identifier is made from the original attribute label and the data, which is then encrypted using DE with system keys for a one-way encryption process. The data itself is encrypted using the policies desired by the encrypting user. When two independent users encrypt/submit the same attributes into the system, they both generate the same unique identifier for that attribute, but generate completely different encrypted value for the data of that particular attribute. For this reason is that the backend will store both the unique identifier for that particular data and multiple encrypted values for that data.

It is worth noting that the unique identifier is tied to the value of that data, for example is there are two IP addresses then even though they are both IPs they will yield different unique identifiers, but if they are that same type of IP and same value then they will generate same identifiers, the resulting encrypted value will be different (as they are encrypted by different policies). This allows for querying users to be able to reach to this attribute easily (either by direct or relational queries) and for them to be able to decrypt any of the possible encrypted values. For example, if company A and B had submit the same piece of data, company A denies access to company C, company B grants access to company C. Regardless of value encryption company C can still traverse the tree to search around the encrypted data (as it uses the relationship database, without knowing its meaning), and in this case since there are multiple encrypted results, and company B explicitly granted access to that data, company C can still decrypt it.

The Backend is also responsible for data analytics and filtering. Relationship information is stored in two layers, this means that relationship/relative queries can be accomplished in a single query to the database. The Backend is responsible for implementing analytics. In our research we focused in achieving supporting analytics and not the analytics themselves, we provide a sample analytics like relational searches, described in section 4.4. The logic behind the query/analytics is described in algorithm 7.

Algorithm 7 Algorithm for Backend query logic

Input: Query: Item ID, query parameters
Output: Encrypted data, results

```
1:  loop
2:      Receive unique ID from Query-client
3:      if Direct query lookup then
4:          Lookup ID in database
5:          Append returned items to results set
6:      else if Analytics query then
7:          Run graph traversal algorithm on relationships
8:          Perform other analytics
9:          Append resulting items to results set
10:     end if
11:     if Time based or integer filtering desired then
12:         Filter results set based on ORE parameters
13:     end if
14:     Return results
15:
16: end loop
```

We assume that sessions are unique, even if two sessions are identical they will differ in time and therefore are not completely unique, for that reason is that we simply add the session/relationship data to our graph database and not deduplicate it.

An assumption here is that this framework protects the data against missuse of the information, not the availability of the data. There are other mechanisms that deal with enforcing data availability. Essentially, the frameworks assume that the Backend will not be a sinkhole and will not tamper with the generated results. Result tampering in this case would mean to include more or less encrypted data to the results, not editing data as it would not be possible by the Backend. We consider this to be out of scope.

Query client

The Query-client will preform similar operations as the Collector but to the query data. It will grab the user's query, which will usually start at an attribute. It will apply DE encryption algorithm using the system's public keys and will pass the encrypted result along with what kind of query the user wants to the Backend. It will handle all encryption and decryption for the user. Query-client logic showed in algorithm 8.

Algorithm 8 Algorithm for Query-client logic

Input: Query: plaintext, query parameters
Output: Decrypted data, results

```
Get query from user
Encrypt query value with DE system keys
if Time or integer filtering query then
    Encrypt filter integer with ORE system keys
end if
Send encrypted values and query parameters to Backend
Wait for results from Backend
if Direct query then
    Decrypt all data with CPABE private key
else if Advanced query then
    Read graph results
    Request value data for pertinent IDs in graph
    Decrypt value data using CPABE private key, if access is permitted
    Reconstruct graph
end if
Interpret results
Convert to user readable(or machine) format
Return results to user
```

ID generation & Queries

The unique identifiers that are used across the system to index and search data are DE encrypted. The ID is generated by encrypting the SHA256 hash of concatenating the data type and value data (equation4.1). This ensures that it will

be unique in the system regardless of what kind of data is being inputted. This approach however does requires the Collector to know what kind of data it is; this does not mean that it need to know which data type it is. The Query-client can simply generate the ID based on the information passed in the query and by inferring type by looking at the type of query. Query-client and the Backend can simply use these IDs as if they where the real data without knowing what the data is.

$$\begin{aligned} dat &= concat(data\_type, data\_value) \\ DIGEST &= SHA256(dat) \\ ID &= DE\_ENC(PUBkey, DIGEST) \end{aligned} \tag{4.1}$$

4.3.1 Creating relationship information

The relationship algorithm is very simple. This is because one of our main assumption, the data inputted to the system has to have the relational information built into it or generated at collection time. This is at the Collector-client stage; therefore, the process for building the relationships is out of scope. However the system still needs to encrypt the relationships. To do that the Collector creates a graph of the data, which then processes to create IDs for every relationship, this is done so using the data where it points to, then collects the information in a single node in the database. These nodes are defined as follows: Event nodes hold pointers to its Attributes and Sessions hold pointers to its Events and Attributes. Since IDs are generated using the same procedure by every user in the system, aggregation is inherently done. Therefore the only step needed to aggregate data is for the data to live in the same database.

Using graphs will allows us to store the relational information about the data. Unique identifiers for the data will allows us to standardize the data retrieval process and also allows us to traverse the relational information.

4.3.2 Query Types

We have differentiated two categories of queries. Exact queries that search for items on the database by directly accessing an item using its ID and returning it. For example, if a user has an IP address, he can search for all events that this item is a part of, i.e. directly related. This includes things like attribute counting and statistics about the data, where a simple lookup of the data is enough to retrieve the desired information.

On the other hand, there is relational queries that leverage the graph structure to find related event based on related information. In Figs. 4.5 and 4.6 we see that data that has one common piece of information gets linked up in the Backend. Relational queries help support analytical queries over the data and it was the main goal of this research, supporting said queries. This group of queries include queries like, given an Attribute find all the related events/session that this particular Attribute is related to (Fig. 4.6 for example), or even queries like find how two items are related. The usefulness of this type of queries comes from the fact that multiple users submit data and that the data is aggregated automatically, exposing never seen before relationship between common data. This becomes useful when looking at CTI that may not be useful when separate but very useful when connected. In the case of our framework aside from having the benefits of a CTI sharing platform that aggregates the data it also has the benefits of the same analytical queries in an privacy preserving manner.

Data in our framework is structured in two layers, the data value layer which supports direct type queries and the relationship layer which in conjunction with the data layer support the relational type queries.

We can further differentiate queries in two other categories, coarse access and fine access queries. Coarse access queries do not require access the value data layer, which means there are more generic queries. Fine access queries are subject to the CPABE polices as they decrypt the data value layers.

Aside from these types of queries, we can filter data using Order Revealing Encryption at the Backend. The user sends an ORE encrypted vale or values to the Backend in conjunction to the query parameters, and the Backend can filter results based on those ORE encrypted value(s). This is only supported on value data that has the ORE encrypted layer, therefore using this feature will result on only searching data with the ORE layer. The Backend can be configured to be inclusive or exclusive of the data that is missing the ORE layer, for example if we look at data between today and tomorrow, the system will return data between to those timestamps, with the option enabled to include data that does not have the ORE layer, the Backend will also send data that does not have the ORE layer that also meets the query criteria. This ORE-less data can later be further filtered out at the Query-client if value data is decryptable, this however comes at the cost of data transfer, space, and time. ORE benefits the Query-clients as the filtering is done at the Backend, which is one of the assumptions this project was built upon, the fact that clients will not have enough resources to do processing themselves.

4.4 Framework example

In this section we will demonstrate the framework through a guided example. The examples in this section are fabricated to demonstrate the analytical capabilities of the framework. To demonstrate that the framework can in aggregate and search over data we will replicate simplified data from a local server. In Fig. 4.7 shows a sample of a chain of events carried out in a server.

Consider the session data in Fig. 4.7:

```
{
    "type": "session",
    "events": [
        {"type": "Login",
         "IP": "78.220.235.199"},
        {"type": "File Download",
         "command": "wget example.com/virus.exe",
         "file_name": "virus.exe"},
        {"type": "Command execution",
         "command": "./virus.exe"}
    ]
}
```

Figure 4.7: Example input data by user, represents a session to a server. It contains all three levels of data, sessions, events and attributes.

it is generated directly from a data generating node. Depending on the node configuration the node can send it directly to a Collector-client or the Client can copy it from a specified location. Once the Collector-client gets a hold of new data it will enforce structure and basic requirements like whether relationships are clearly identifiable. In this example the requirements are met, so the Client passes the data along to the Collector.

The Collector, in this example, is configured to run with the configuration shown in Fig. 3.2. The data in Fig. 4.7 is separated into three groups of components, attributes, events and sessions as shown in Fig. 4.10. In this

particular piece of data we only have one session with multiple events and multiple attributes. The Collector will first take all the attributes and encrypt them using DE, this will generate unique IDs for each piece of data, then it will take the value data of the attributes and match each specific rule to each label for those pieces of data and determine what CPABE policy should be applied. Once the CPABE policy is determined it will encrypt the attributes using their respective policies, as seen in Figs. 4.8 and 4.9. In this case, the labels *type*, *IP*, *command*, and *file_name* are going to be encrypted using the default policy as they did not explicitly matched any of the rules. The default CPABE policy for this particular configuration is *default_public*, meaning anyone holding a CPABE private key with in this system with that attribute will be able to decrypt those attributes. In the event that a rule would match a policy then it would be encrypted using that instead. It is important to note that encrypting does not require to have special attributes assigned in the private CPABE key, this is because only the private key is used in the encryption process. Any attribute can be assigned as an CPABE encryption policy, whether any user possesses those attributes or not.

After encrypting the value data layer it will, for each Event grab the IDs for every Attribute in it and create a pseudo object with all of the IDs in it. Then it will create another pseudo object for the session and add all of the Attributes' IDs and Event IDs into that object, This can be seen in Fig. 4.11, the plain text value data was kept in the structure to show the relationships between the IDs and Events/Session, however this would not be included in the end result and the resulting data would look like Fig. 4.12.

After encryption, the Collector will send the data to the Backend where it is stored. All the encrypted value data is stored in one database and all the

```
{
  "data_type": "attribute",
  "type": "Login"
}
{
  "data_type": "attribute",
  "IP": "78.220.235.199"
}
{
  "data_type": "attribute",
  "type": "File Download"
}
{
  "data_type": "attribute",
  "command": "wget example.com/virus.exe"
}
{
  "data_type": "attribute",
  "file_name": "virus.exe"
}
{
  "data_type": "attribute",
  "type": "Command execution"
}
{
  "data_type": "attribute",
  "command": "./virus.exe"
}
```


Figure 4.8: Represents the unencrypted value data.

relationship data (for example Fig. 4.12) in another database.

Now lets consider another piece of data (Fig. 4.13) that is submitted by another node. This particular data comes from one of their firewalls and contains related information to the data in Fig. 4.7. This network log data will get converted into data shown in Fig. 4.14 via the same steps as the previously shown data but with a different encryption policy. Then the data shown in Fig. 4.15 is sent to the Backend. This data simply gets appended to the database, no

```
{
  "data_type": "attribute", "uniq_id": 2458,
  "cpabe_enc": [0x19...741]
}
{
  "data_type": "attribute", "uniq_id": 1335,
  "cpabe_enc": [0x43...092]
}
{
  "data_type": "attribute", "uniq_id": 8720,
  "cpabe_enc": [0x92...342]
}
{
  "data_type": "attribute", "uniq_id": 6181,
  "cpabe_enc": [0x16...493]
}
{
  "data_type": "attribute", "uniq_id": 1428,
  "cpabe_enc": [0x56...398]
}
{
  "data_type": "attribute", "uniq_id": 2098,
  "cpabe_enc": [0x29...282]
}
{
  "data_type": "attribute", "uniq_id": 7952,
  "cpabe_enc": [0x01...906]
}
```


Figure 4.9: Represents the encrypted value data.

queries or processing necessary as it is already in the graph format.

```
{
    "type": "session"
}
{
    "type": "event"
}
{
    "type": "Login",
    "IP": "78.220.235.199"
}
{
    "type": "File Download",
    "command": "wget example.com/virus.exe",
    "file_name": "virus.exe"
}
{
    "type": "Command execution",
    "command": "./virus.exe"
}
```

Figure 4.10: Example of what data would look like after separating the data is separated into its core components.

```
{
    "type": "session",
    "events": [
        {"type": "File Download",
         "IP": "78.220.235.199",
         "file_name": "virus.exe"},
        {"type": "Connection established",
         "IP": "78.220.235.199"},
        {"type": "File Upload",
         "IP": "78.220.235.199",
         "file_name": "database.sql"}
    ]
}
```

Figure 4.13: Another example of input data to the system. This data would represent a network traffic log.

The Backend now has now appended both relationship data from both sub-

missions from Figs. 4.12 and 4.15. The value data layers are merged, this is done by searching for all the submitted value data IDs in the database and checking if they exist, in the case of the first data submission in our example, all of them do not exist so the Backend simply adds the values to the database. The second submission of value data is looked up, where the fields that have previously been added will be appended to previously submitted IDs and the ones that have not been seen before are just added, this results in the data shown in Fig. 4.16.

The value data in the Backend can be accessed using its unique identifier. The decryption process will involve checking each piece of data is decryptable or not by given a set of attributes. Checking for decryptability is not computationally expensive, as the binary data has the attributes required to be decrypted in plain text, meaning the check can be checked by anyone. The attributes being stored with the encrypted data is not a risk, as attributes are just labels and the KMS can solve this problem by using a random sequence of characters to represent each attribute/characteristic. There are two ways to check the data, each approach has benefits and risks. The first approach is to download the data to the user and have the Query-client compare it, it is not computationally expensive but it will make the query take longer as more data needs to be downloaded and temporarily stored by the user. The second approach is for the Query-client to pass along what attributes the key holder has, and have the Backend return data that meets the query and the decryptability criteria, this however reveals the attributes of the key-holder. If sharing the attributes of their organization is not important then it is not a privacy violation. Ultimately the decision is up to the querying user/organization.

It is important to note that any of the actions in the framework are com-

pletely asynchronous and can happen at anytime. In this paper and this example we walk through the possible actions in the same flow the data would take but in reality there will be many users interacting with the system at once.

Now the data there exists data in the Backend, we can perform queries. We will first describe the procedure for exact type queries. We will be searching for information relating to a specific IP, *78.220.235.199* in our case. We will first demonstrate how the system would handle this case. Every request starts by getting the query value from the user and then generating the unique identifier by using the deterministic encryption algorithm. The ID generation step requires a specific data type, in our specific example this is *Attribute* since the IP is an Attribute. The algorithm will produce the ID *1335*. Note that the value is deterministic for the encryption key, a different key will produce a different value.

If we wanted to find out all the events that contain this IP, we would send the ID and our request. The Backend will then proceed to search the database for all Events containing that IP, the query would look as follows: *{“data_type": “event", “refs": 1335}*. For this particular example, this query will result in returning all the data in our relationship database. The database query returns only the relationships, which are sent back to the Query-client for interpretation. This is the point where each query differentiates from each other, for example, if we ask about the number of related events, the results are are not directly from the database back to the user, the Backend will instead count (done by the DBMS) and return a single number to the user. If we ask for the data itself then the relationship resulting graph and the corresponding encrypted value data is sent back to the user. The Query-client can then reconstruct the result buy decrypting the data. As mentioned earlier, there may be multiple

encrypted value data ciphers (CPABE) for a single piece of data, the user can decide whether to carry out the encrypted value data filtering in the Backend or in the Query-client, regardless of the choice, the Client will then proceed to decrypt the value data and link it up according to the graph showing the relational information. After this reconstruction step the result data is ready displayed to the user. In this example, six events are reconstructed as shown in Fig. 4.17. In this we assume that the querying user is allowed to decrypt the data as per access control policies, meaning has the right decrypting attributes in his CPABE key. If the user did not have all the right attributes in his CPABE key then he may not be able to conclude in to the same results.

Generally, asking the right query is key to getting good results. In this particular example, the results for the related query may not be that useful on their own and follow up queries may be needed to conclude an investigation. As an investigator we may realize that *databse.sql* is the name of a database file and it is tied to a file upload and somehow it is related to the IP. We now move to the related queries.

```
"events": [
    {"type": "Login",
     "IP": "78.220.235.199"},
    {"type": "File Download",
     "command": "wget example.com/virus.exe",
     "file_name": "virus.exe"},
    {"type": "Command execution",
     "command": "./virus.exe"},
    {"type": "File Download",
     "IP": "78.220.235.199",
     "file_name": "virus.exe"},
    {"type": "Connection established",
     "IP": "78.220.235.199"},
    {"type": "File Upload",
     "IP": "78.220.235.199",
     "file_name": "database.sql"}
]
```


Figure 4.17: Results for querying all events related to the IP *78.220.235.199*.

We may want to know exactly is the file related to the IP. If we remember, we have two records, one is a log of the actions taken on a server, the other is a firewall log. To continue our investigation we will use the name of the file *databse.sql*. We can do a related query to find all the paths from the file name to the IP. The initial procedure is always the same, the Query-client will take the value and generate its unique ID (*2765* in this case). Then we send our request to the Backend along with the ID of the IP and the ID of the file. The Backend then will conduct a query on the database for all the sessions related to file. The Backend then applies a graph traversal algorithm to find all the paths from the file name ID to the IP ID. The Backend will then return the relationship paths. In reality multiple path can be returned and is up to the investigator to interpret the results. In this particular example we will see that there will see multiple paths within the same session and another that will link the upload event to the firewall log event. Depending on the parameters of the query, the Backend will

return the paths, the related session and events (as in the relationship graph), and the encrypted value data. All of of this information will allow the Query-client to reconstruct the results by decrypting the data which is subject to the access control policies.

This was a very basic example on how the graph can be traversed to find information from the relationships. The analytics engine in the Backend can be programmed with even more advanced queries. The graph is constructed in a way that most queries can be resoled in one to two queries to the relationship database, one query to the value data database, and depending on the query a graph traversal. We can even asks queries relating more than one degree of search, in this case the degree of relation desired desired in a query is proportionally tied to the number of queries made to the database.

It is important to note that this example is small and only works with a very small data set, in reality results could be inherently large depending on the query, results can be human readable but are designed to be interpreted or filtered by machines.

```
{ "data_type": "session", "uniq_id": 8500,
  "refs": [6245,2268,7511]}

{ "data_type": "event", "uniq_id": 6245,
  "refs": [2458,1335]}

{ "data_type": "event", "uniq_id": 2268,
  "refs": [8720,6181,1428]}

{ "data_type": "event", "uniq_id": 7511
  "refs": [2098,7952]}

{ "data_type": "attribute", "uniq_id": 2458,
  "type": "Login"}

{ "data_type": "attribute", "uniq_id": 1335,
  "IP": "78.220.235.199"}

{ "data_type": "attribute", "uniq_id": 8720,
  "type": "File Download"}

{ "data_type": "attribute", "uniq_id": 6181,
  "command": "wget example.com/virus.exe"}

{ "data_type": "attribute", "uniq_id": 1428,
  "file_name": "virus.exe"}

{ "data_type": "attribute", "uniq_id": 2098,
  "type": "Command execution"}

{ "data_type": "attribute", "uniq_id": 7952,
  "command": "./virus.exe"}
```

Figure 4.11: Example of unencrypted structure of Session. The unique identifier ID is generated based on the data and will Deterministically Encrypted using system secrets. This however does not represent the real order in which the system encrypt information, this is just a visual way to demonstrate the grouping of data.

```
{
  "data_type": "session", "uniq_id": 8500,
  "refs": [6245,2268,7511]
}
{
  "data_type": "event", "uniq_id": 6245,
  "refs": [2458,1335]
}
{
  "data_type": "event", "uniq_id": 2268,
  "refs": [8720,6181,1428]
}
{
  "data_type": "event", "uniq_id": 7511
  "refs": [2098,7952]
}
{
  "data_type": "attribute", "uniq_id": 2458
}
{
  "data_type": "attribute", "uniq_id": 1335
}
{
  "data_type": "attribute", "uniq_id": 8720
}
{
  "data_type": "attribute", "uniq_id": 6181
}
{
  "data_type": "attribute", "uniq_id": 1428
}
{
  "data_type": "attribute", "uniq_id": 2098
}
{
  "data_type": "attribute", "uniq_id": 7952
}
```

Figure 4.12: Example of encrypted relationship information ready to be transferred to the Backend.

```
{
  "data_type": "session", "uniq_id": 46547,
  "refs": [65356,23982,93723]
}
{
  "data_type": "event", "uniq_id": 65356,
  "refs": [8720,1428,1335]
}
{
  "data_type": "event", "uniq_id": 23982,
  "refs": [943,1335]
}
{
  "data_type": "event", "uniq_id": 93723
  "refs": [3575,1335,2765]
}
{
  "data_type": "attribute", "uniq_id": 1335,
  "IP": "78.220.235.199"
}
{
  "data_type": "attribute", "uniq_id": 8720,
  "type": "File Download"
}
{
  "data_type": "attribute", "uniq_id": 1428,
  "file_name": "virus.exe"
}
{
  "data_type": "attribute", "uniq_id": 943,
  "type": "Connection established"
}
{
  "data_type": "attribute", "uniq_id": 2765,
  "file_name": "database.sql"
}
{
  "data_type": "attribute", "uniq_id": 3575,
  "type": "File Upload"
}
```


Figure 4.14: Relationship layer of the network traffic example. This figure also shows the information associated with each ID, in reality the value data layer will not be send in plain text.

```
# Encrypted Relationship data layer
{
  "data_type": "session", "uniq_id": 46547,
  "refs": [65356,23982,93723]
}
{
  "data_type": "event", "uniq_id": 65356,
  "refs": [8720,1428,1335]
}
{
  "data_type": "event", "uniq_id": 23982,
  "refs": [943,1335]
}
{
  "data_type": "event", "uniq_id": 93723
  "refs": [3575,1335,2765]
}
# Encrypted value data layer
{
  "data_type": "attribute", "uniq_id": 1335,
  "cpabe_enc": [0x83...462]
}
{
  "data_type": "attribute", "uniq_id": 8720,
  "cpabe_enc": [0x38...638]
}
{
  "data_type": "attribute", "uniq_id": 1428,
  "cpabe_enc": [0x43...398]
}
{
  "data_type": "attribute", "uniq_id": 943,
  "cpabe_enc": [0x19...582]
}
{
  "data_type": "attribute", "uniq_id": 2765,
  "cpabe_enc": [0x84...954]
}
{
  "data_type": "attribute", "uniq_id": 3575,
  "cpabe_enc": [0x02...573]
}
```

Figure 4.15: Encrypted value data and relationship layers of the network traffic example, ready to be submitted to the Backend.

```
{
  "data_type": "attribute", "uniq_id": 1335,
  "cpabe_enc": [0x43...092, 0x83...462]
}
{
  "data_type": "attribute", "uniq_id": 8720,
  "cpabe_enc": [0x92...342, 0x38...638]
}
{
  "data_type": "attribute", "uniq_id": 1428,
  "cpabe_enc": [0x56...398, 0x43...398]
}
{
  "data_type": "attribute", "uniq_id": 2458,
  "cpabe_enc": [0x19...741]
}
{
  "data_type": "attribute", "uniq_id": 6181,
  "cpabe_enc": [0x16...493]
}
{
  "data_type": "attribute", "uniq_id": 2098,
  "cpabe_enc": [0x29...282]
}
{
  "data_type": "attribute", "uniq_id": 7952,
  "cpabe_enc": [0x01...906]
}
{
  "data_type": "attribute", "uniq_id": 943,
  "cpabe_enc": [0x19...582]
}
{
  "data_type": "attribute", "uniq_id": 2765,
  "cpabe_enc": [0x84...954]
}
{
  "data_type": "attribute", "uniq_id": 3575,
  "cpabe_enc": [0x02...573]
}
```

Figure 4.16: Example of encrypted value data layer, as stored in the Backed after merging multiple submissions.

Chapter 5

Conclusion and Future Work

5.1 Conclusion & Future Work

We have create a framework for Cyber Threat Intelligence sharing to support for encrypted analytics while preserving integrity and confidentiality of the data from external attackers. In our model we assumed that parties did not want to or did not have the computation power to carry out analytics on their premises; therefore, these calculations are done in the centralized Backend. In our model we established the Backend as a completely untrusted entity and should only be able to calculate analytics without seeing details about the data. The Backend is considered to be an outsider of the system, with the privilege to access encrypted data for analytical computations. We achieve security against the centralized party and any other foreign entities by separating the data into two mayor types, the value data layer and the relationships. This allowed us to treat each layer independently to apply different encryption methods to each layer. We explored alternative solutions, but none could provide all the flexi-

bility needed to achieve analytic on the centralized Backend while maintaining the data secure. The degree on which our system can do data analytics is determined by the level of access control placed on the data itself, and each user experience will differ from others as submitting parties can specify who can see what part of their data.

We have defined and implemented a framework for working with encrypted CTI. As far as the kind of queries the framework can answer, we have implemented all of the exact queries/searches, counts, integer/time range filtering, access control, and some basic related queries of degree one. The framework can answer questions like finding similar attack chains, relating indicators of compromise to other events in the system, finding out how these related events are actually related.

This work takes steps the first steps in the field towards encrypted privacy preserving Cyber Threat Information sharing platform for preserving privacy against the central processing entity and other external attackers. Loosely this framework allows to the usage of encrypted data as if it was plain-text, there are many queries and data analysis that is possible that we did not cover in this research. Access control and data analytics are the two main requirements from our CTI sharing framework, which made it difficult to implement queries. Ultimately, each new query has to be split into core steps where they get implemented carefully and then separated for each module (Collector, Backend, Query client).

Relationship data itself is derived from the data using one way deterministic encryption, and it is considered safe from preying eyes. Value data is encrypted using CPABE and it allows access control over the data. The combination of the two layers allows for simple queries like counts and direct lookups, and more

advances queries like relational question about the data, for example give an IP, how many Sessions have similar characteristics as the parent event of this attribute, and how are they related. In the future we would like to dig deeper into the data and create new analytical queries using this framework.

Future work includes key revocation and re-encryption in the event of revocation. This research only focuses on the usage of such technologies to achieve analytics on encrypted data for CTI sharing purposes. There are many research works out there detailing revocation and re-encryption mechanisms for the encryption primitives, our focus would be on how to apply these mechanisms without compromising the data at re-encyption time.

We have also, found possible shortcoming with the framework. Since the relationship data is made in a way that allows encrypted traversal over the graph, an adversary can theoretically learn something from traversing it. One of the questions that we did not answer in this research was if an adversary has something to learn from this specific relationship layer. The data itself is encrypted and it is safe because it is not used for graph traversal. The question arises, does an adversary have enough information to infer any new piece of information? For example if analytics agent learns that probability of *redacted2* given *redacted1*, without knowing what the encrypted labels/IDs mean or point to?

We also identified possible points of failure of the system. Just like any one-way function where the attacker has direct access to the function, our DE function is susceptible to insider attacks, as they will have the key (Backend does not have keys). This allows the possibility initiating brute force attacks on the data. This would work by encrypting possible data using the DE functions with the system key and comparing the output to encrypted value(s), if

the encrypted values match then it means the inputs are the same, essentially revealing what the original data was. This of course will entirely depend on the input space of the data, for example if we are talking about IP addresses versus a big binary file, the IP is clearly at a disadvantage. Future work will look into the feasibility of such attacks by insiders as it has the potential to violate the privacy against other system users.

Further research has to be conducted into analysing CipherPath [23] algorithms which allows to find shortest paths between to nodes in an encrypted graph. This however, only covers shortest paths, and for similar reasons to Neo4j and other graph DBMSs, they only try to find shortest paths in a timely manner, leaving all possible paths running extremely inefficiently.

We have also identified that our framework supports parallel calculation and storage. Since data can live anywhere there is high calculation and storage capabilities, the data can be duplicated into multiple locations. These locations can act as mirrors and can reduce loads on a single server. This also by extension means that the data can be stored decentralized and calculated elsewhere, this however does not mean it will be suitable for things like blockchain or smart contracts, as the price for using such systems will completely outweigh the benefit as computing in blockchain is very expensive.

One completely yet to be explored avenue is the decentralized storage. The only reason for the centralization is for analytical computations, someone has to do it, in our framework we assume that someone will take on that responsibility, but in reality it could be anyone. Choosing the centralized party to compute the calculations for the tree traversal and of the data lookups also poses a problem that we did not explore, the fact that this party could attack the data at some points: At submission, the Backend can blackhole the data and delete it, it

could also choose to return other encrypted records not corresponding to a users query, even blackhole queries. For this reason, it is important to choose a trusted Backend administrator or have the Backend have something at stake for fulfilling the Backend roles. Decentralization could mean faster query times as more data and computational mirrors are closer and available, and problems related to Backend could be mitigated.

Bibliography

[1] F. Sadique, I. Astaburuaga, R. Kaul, S. Sengupta, S. Badsha, J. Schnebly, A. Cassell, J. Springer, N. Latourrette, and S. M. Dascalu, “Cybersecurity information exchange with privacy (CYBEX-P) and TAHOE - A cyberthreat language,” *CoRR*, vol. abs/2106.01632, 2021. [Online]. Available: https://arxiv.org/abs/2106.01632

[2] D. Shackleford, “Who’s using cyberthreat intelligence and how?” 2015. [Online]. Available: https://cdn-cybersecurity.att.com/docs/SANS-Cyber-Threat-Intelligence-Survey-2015.pdf

[3] “How famous cyber security breaches could have been prevented.” [Online]. Available: https://studyonline.ecu.edu.au/blog/how-famous-cyber-security-breaches-could-have-been-prevented

[4] C. Johnson, M. Badger, D. Waltermire, J. Snyder, and C. Skorupka, “Guide to cyber threat information sharing,” Oct 2016. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-150/final

[5] C. S. Johnson, L. Feldman, and G. A. Witte, “Cyber threat intelligence and information sharing,” May 2017. [Online]. Available: https://www.nist.gov/publications/cyber-threat-intelligence-and-information-sharing

[6] W. Fan, J. Ziembicka, R. de Lemos, D. Chadwick, F. Di Cerbo, A. Sajjad, X.-S. Wang, and I. Herwono, “Enabling privacy-preserving sharing of cyber threat information in the cloud,” in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, June 2019, pp. 74–80.

[7] D. W. Chadwick, W. Fan, G. Costantino, R. de Lemos, F. Di Cerbo, I. Herwono, M. Manea, P. Mori, A. Sajjad, and X.-S. Wang, “A cloud-edge based data security architecture for sharing and analysing cyber threat information,” *Future Generation Computer Systems*, vol. 102, pp. 710–722, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19300895

[8] C. Dengler, “Homomorphic encryption,” Ph.D. dissertation, Universitat Heidelberg.

[9] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, ser. STOC ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 169–178. [Online]. Available: https://doi.org/10.1145/1536414.1536440

[10] X. Meng, S. Kamara, K. Nissim, and G. Kollios, “Grecs: Graph encryption for approximate shortest distance queries,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 504–517. [Online]. Available: https://doi.org/10.1145/2810103.2813672

[11] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, “Cryptdb: Protecting confidentiality with encrypted query processing,” in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 85–100. [Online]. Available: https://doi.org/10.1145/2043556.2043566

[12] R. A. Popa, N. Zeldovich, and H. Balakrishnan, “Guidelines for using the cryptdb system securely,” *IACR Cryptology ePrint Archive*, vol. 2015, p. 979, 2015. [Online]. Available: https://eprint.iacr.org/2015/979

[13] N. Aburawi., A. Lisitsa., and F. Coenen., “Querying encrypted graph databases,” in *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,*, INSTICC. SciTePress, 2018, pp. 447–451.

[14] N. N. Aburawi, “Cryptdb mechanism on graph databases,” Ph.D. dissertation, University of Liverpool, 2020.

[15] N. Aburawi, F. Coenen, and A. Lisitsa, “Traversal-aware encryption adjustment for graph databases,” in *Proceedings of the 7th International Conference on Data Science, Technology and Applications*, ser. DATA 2018. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2018, p. 381–387. [Online]. Available: https://doi.org/10.5220/0006916403810387

[16] P. Xie and E. P. Xing, “Cryptgraph: Privacy preserving graph analytics on encrypted graph,” *CoRR*, vol. abs/1409.5021, 2014. [Online]. Available: http://arxiv.org/abs/1409.5021

[17] M.-H. Shih and J. M. Chang, “Design and analysis of high performance crypt-nosql,” in *2017 IEEE Conference on Dependable and Secure Computing*, Aug 2017, pp. 52–59.

[18] F. Sadique, I. Astaburuaga, and A. Cassell, “Cybex-p project site.” [Online]. Available: https://cybex.cse.unr.edu/

[19] “Cybex-p github.” [Online]. Available: https://github.com/CYBEX-P

[20] A. Cassell, T. Das, Z. Black, F. Sadique, J. Schnebly, S. Dascalu, S. Sengupta, and J. Springer, “Sharing is caring: Optimized threat visualization for a cybersecurity data sharing platform,” in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, 2021, pp. 1–8.

[21] A. Cassell, “Navigating cyberthreat intelligence with cybex-p: Dashboard design and user experience,” Ph.D. dissertation, University of Nevada, Reno, 2021, copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2021-09-15. [Online]. Available: https://unr.idm.oclc.org/login?url=https://www.proquest.com/dissertations-theses/navigating-cyberthreat-intelligence-with-cybex-p/docview/2563493096/se-2?accountid=452

[22] I. Astaburuaga, “Cybex-p encrypted analytics.” [Online]. Available: https://cybex.cse.unr.edu/docs/encrypted-analytics/

[23] G. Bramm. and J. Schütte., “cipherpath: Efficient traversals over homomorphically encrypted paths,” in *Proceedings of the 17th International*

Joint Conference on e-Business and Telecommunications - SECRYPT,, INSTICC. SciTePress, 2020, pp. 271–278.

[24] S. Lai, X. Yuan, S.-F. Sun, J. K. Liu, Y. Liu, and D. Liu, "Graphse^2: An encrypted graph database for privacy-preserving social search," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, ser. Asia CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 41–54. [Online]. Available: https://doi.org/10.1145/3321705.3329803

[25] H. T. Lee, S. Ling, J. H. Seo, H. Wang, and T.-Y. Youn, "Public key encryption with equality test in the standard model," *Information Sciences*, vol. 516, pp. 89–108, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025516322290

[26] E. Charles, M. Samuel, N. Roger, and O. Daniel, "System and method of data collection, processing, analysis, and annotation for monitoring cyber-threats and the notification thereof to subscribers," Mar 2002.

[27] H. Sudo, K. Nuida, and K. Shimizu, "An efficient private evaluation of a decision graph," in *Information Security and Cryptology – ICISC 2018*, K. Lee, Ed. Cham: Springer International Publishing, 2019, pp. 143–160.

[28] S. Kulkarni, "Cryptographic algorithm using data structure using c concepts for better security," in *2015 International Conference on Pervasive Computing (ICPC)*, Jan 2015, pp. 1–3.