PROXY-SECURE COMPUTATION MODEL: APPLICATION TO K-MEANS CLUSTERING IMPLEMENTATION, ANALYSIS AND IMPROVEMENTS

by Erman Pattuk

Submitted to the Graduate School of Sabanci University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabanci University

August, 2010

PROXY-SECURE COMPUTATION MODEL: APPLICATION TO K-MEANS CLUSTERING IMPLEMENTATION, ANALYSIS AND IMPROVEMENTS

APPROVED BY:

Assoc. Prof. Dr. Erkay Savaş	....
(Thesis Supervisor)	
Assist. Prof. Dr. Cemal Yılmaz	....
Assoc. Prof. Dr. Yücel Saygın	....
Assoc. Prof. Dr. Albert Levi	....
Assoc. Prof. Dr. Cem Güneri	....

DATE OF APPROVAL:.......................

PROXY-SECURE COMPUTATION MODEL: APPLICATION TO K-MEANS CLUSTERING IMPLEMENTATION, ANALYSIS AND IMPROVEMENTS

Erman Pattuk

CS, Master's Thesis, 2010

Thesis Supervisor: Erkay Savaş

Keywords: Multi-party Computation, Cell processor, Data mining

Abstract

Distributed privacy preserving data mining applications, where data is divided among several parties, require high amounts of network communication. In order to overcome this overhead, we propose a scheme that reduces remote computations in distributed data mining applications into local computations on a trusted hardware. Cell BE is used to realize the trusted hardware acting as a proxy for the parties. We design a secure two-party computation protocol that can be instrumental in realizing non-colluding parties in privacy-preserving data mining applications. Each party is represented with a signed and encrypted thread on a separate core of Cell BE running in an isolated mode, whereby its execution and data are secured by hardware means. Our implementations and experiments demonstrate that a significant speed up is gained through the new scheme. It is also possible to increase the number of non-colluding parties on Cell BE, which extends the proposed technique to implement most distributed privacy-preserving data mining protocols proposed in literature that require several non-colluding parties.

VEKİL GÜVENLİKLİ HESAPLAMA MODELİ K-MEANS GRUPLAMA UYGULAMASI: UYGULAMA, ANALİZ VE GELİŞTİRMELERİ

Erman Pattuk

CS, Yüksek Lisans Tezi, 2010

Tez Danışmanı: Erkay Savaş

Anahtar Kelimeler: Çok-Partili Hesaplama, Cell İşlemcisi, Veri Madenciliği

Özet

Verinin birden fazla partiye bölünmüş olduğu dağıtılmış veri madenciliği uygulamaları yüksek miktarda ağ üzerinden haberleşme gerektirir. Bu yükten kurtulmak için önerdiğimiz modelde, uzaktan yapılan hesaplamaları güvenli bir donanım üzerinde yerel hesaplamalara dönüştürüyoruz. Partilerin çalışması için vekil ortam olarak Cell BE seçildi. Modelin performansını ölçmek amacıyla, daha önceden oluşturduğumuz güvenli iki-partili hesaplama protokolü üzerinde uygulamalar tasarladık. Hesaplamadaki her parti kendi uygulamasını yazdıktan sonra, imzalayıp şifreleyerek vekil ortama yollamakla yükümlü. Yolladıkları uygulamalar Cell BE işlemcisi üzerinde kendileri için ayrılmış izole durumdaki SPE çekirdeğine yollanır. Uygulamanın çalışması öncesinde, esnasında ve sonrasında herhangi bir bilgi açığa çıkması Cell BE işlemcisinin sunmuş olduğu güvenlik özelliklerinden dolayı çok zordur. Yapmış olduğumuz deneyler, sunmuş olduğumuz modelin büyük hızlanma sağladığı sonucunu ortaya koymuştur. Cell BE platformu içinde yeterli çekirdek olduğu sürece hesaplamadaki parti sayısını arttırmak mümkündür.

Acknowledgements

With all my heart, I wish to express my gratitute to my supervisor Erkay Savaş, for his support, patience and guidance through my graduate education. I wouldn't be able to finish my work without his help.

I would also like to thank Albert Levi, Hüsnü Yenigün, Cemal Yılmaz, Cem Güneri, and Ali Alpar. It was wonderful to work with such brilliant professors.

For their support in my thesis, I would like to thank IBM Turkey. If they had not given Playstation 3 machines, I wouldn't have finished my thesis :)

Special thanks goes to my friends. Erdal Mutlu, Burcu Özçelik, Oğuzcan Mercan, Şeyma Ketenci, Ahmet Onur Durahim, Duygu Karaoğlan, Emine Dumlu, Leyli Javid, Barış Altop, Murat Ergun, Hüseyin Ergin, Oya Şimşek, Cengiz Örencik, İsmail Fatih Yıldırım, Osman Kiraz, Nazım Öztahtacı, Sarp Çakır, Kemal Hatipoğlu, Emre Kaplan and many many others. Without you all my life would be meaningless.

The last and the most important thanks goes to my family. Every member of my family gave me enough support through all my education. Sabahat, Kirkor, Müge, Arman, Serpil and Harudyun Pattuk. Loves and kisses to you all.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Distributed privacy-preserving data mining applications that require the participation of more than one parties that are physically distanced suffer from huge overhead due to network communication. Data to be mined can be partitioned vertically or horizontally, but the fact does not change: exchanging messages via network increases the computation time. The overhead can become larger as the physical distance between parties increases. If the operations involved in data mining applications are performed on multiple data warehouses, then the network communication becomes a major problem, rendering the already time-consuming computations infeasible to complete in a reasonable time frame.

One of the actual reasons behind the network communication overhead can be given as the need of trust. The lack of trust between parties motivates them to physically isolate their data and processing environment from other parties, forcing them to keep their data private and to run their part of the computation on their own servers.

Distributed privacy-preserving data mining applications, nonetheless, require a high degree of interaction among the participating parties that want to keep their data private. Each party needs to make sure that other parties cannot even deduce any information about his/her data from messages exchanged. To guarantee security and privacy requirements, parties need to use secure algorithms/protocols as well as secure and trusted computing platforms. In this respect, secure multi-party computation model where parties are semi-trusted (in the sense they are honest but curious) suits the needs of distributed privacy-preserving data mining applications.

We address these problems by proposing a novel proxy-secure computation model, whereby parties execute their applications on a Cell Broadband Engine Architecture (CBEA) platform that provides a secure and trusted execution environment. Instead of a distributed application where different parts of the application running in physically distanced computers of each participant, these applications execute on trusted hardware where they are isolated from each other by hardware means. Namely, each application is executed on a different core of the architecture, that are physically isolated. Since the applications run on the same integrated circuit, this greatly enhances the performance by replacing network communication with direct memory access operations.

CBEA, whose security features and isolation technology are explained in [1] and [2], provides a secure and trusted hardware platform where participants' codes can run securely and in an isolated fashion. Since each code runs in an isolated core of CBEA and protected by cryptographic techniques and hardware means, no information can leak outside. Furthermore, isolation is implemented by hardware means so that no software based attacks (bugs, trojans, malwares, etc.) can override the protection.

Using the isolation and security properties of the CBEA, parties develop their part of the application on their own, digitally sign and encrypt it, and send it to a CBEA platform which is in the possession of a semi-trusted authority. Since the applications running on the platform are completely free of external threats, they can communicate more efficiently and more securely with each other inside the Cell Processor.

This execution model is also useful in securing cloud computing applications where the security and privacy of applications and data are serious concerns since the data and applications are held on the hardware of third parties. Our proposal is beneficial to alleviate some of these concerns by providing a practical framework.

In order to demonstrate the advantage of the model proposed in this thesis, a distributed data mining application which is time consuming and requires a high degree of interaction is selected. K-means clustering algorithm fulfills these requirements and it is also very common in various branches of computer science. General and modified versions of k-means clustering are described in [3] and [4]. The k-means clustering algorithm that protects the privacy of the participants' data, mentioned in [4], necessitates a secure multi-party communication (SMC) protocol between the parties. In Section 4, an efficient two-party communication protocol that can be used as a primitive in our application is described in details.

1.1 Contribution of the Thesis

In this thesis, we focus on several issues for building a fast and secure platform and framework for massive computational problems. The computations involve participations of more than one party and their private data, where each party owns a part of the data that the operation will be performed on, and doesn't want to give away any information about it by any means.

The first issue we focus on is to assure that a secure and trusted hardware (i.e. CBEA in our work) provides reliable security solutions to software developers that will protect their applications from any hardware or software level of attacks since these application will run on a platform owned by a third party. The second issue is to propose a proxy-secure computation model and a framework, by offering parties to use a trusted execution platform, whereby they can securely deploy and execute their applications. Naturally, trust between parties is a major concern in this setting and in the proposed model, neither contributing parties nor the platform owner can learn a sensitive information which they don't know already. In the model, the parties do not have to trust each other, and during the development stage they do not even have to collaborate except for interfacing issues that allow their applications to talk to each other. Conforming to the protocol steps is sufficient for the computation to succeed, to guarantee the privacy of parties, and to produce the correct results.

The proposed model gives superior performance results compared to the works in [3] and [4], by replacing network communication with direct memory access operations, which greatly accelerates the computation. Moreover, it is shown that security of the application and data is completely preserved in our model by cryptographic techniques and hardware means.

To best of our knowledge, the proposed model and framework that allow a secure multi-party computation to be performed in the same hardware platform are unprecedented.

1.2 Organization of the Thesis

In Section 3, Cell Broadband Engine Architecture is explained with its technical details and security features. Detailed information is given on the types of cores that exist within a Cell processor. In Section 3.2, three different features of CBEA that enable the isolation of an application are explained separately. And in Section 3.3, the key hierarchy used in the authentication and encryption mechanisms in the process of building, deploying and executing the application is explained. Finally, Section 3.4 gives information as to how to develop applications that uses these security features.

In Section 4, a secure two-party computation model is given in details, which constitutes the basis for implementing non-colluding parties in privacy preserving clustering algorithm. Addition, multiplication and comparison operations in the secure two-party computation model that are essential in many privacy preserving data mining applications are defined, and the overhead of every operation is given separately.

In Section 5, an example data mining application that requires massive computations is introduced: k-means clustering algorithms over distributed spatio-temporal data. The k-means clustering algorithm is first explained in its general form in Section 5.1. An extension to this algorithm is presented in Section 5.2, which handles the multi-party and privacy preserving case.

In Section 6, implementation details are given for secure two-party computation model and clustering algorithm on two platforms. First, implementation on a PC is explained in Section 6.1, and then details are given about the implementation in our proxy-secure computational model on CBEA in Section 6.2.

Finally in Section 7, performances of our implementations are given and compared to the results of related works. The results are calculated for secure two-party computation model, as well as for the privacy preserving clustering algorithm.

2 Related Work

The multiplication operation in our secure two-party computation model uses multiplication triples, which represents the three ring elements created by the trusted third party. This technique has been previously used in [5, 6].

In the operation of bitwise comparison, oblivious transfer (OT) described in [7] is used. Previously, works in [8, 9, 10] also use OT in order to perform comparison of bits operations. Moreover, the idea of having a third trusted party in the operations is used in [11, 5, 6].

Previously, several algorithms have been proposed for privacy preserving data mining operations on vertically partitioned data, which uses secure multi-part computation (SMC) models. Work of Vaidya and Clifton [3] is an example of privacy preserving k-means clustering over vertically partitioned data. Vaidya et al. were aware of the communication overhead induced by SMC on large amounts of data. Therefore, they proposed to use a subset of SMC functionalities. However, public key cryptography (PKC) was used on this work, which increased the computation time of their protocol prohibitively.

In another work by Kantarcioglu and Clifton, SMC is used again, but this time they utilized commutative encryption property of RSA encryption [12]. Due to the usage of PKC again in these two works, the computation overhead was high, which caught the attention of Wright and Yang. In their work, they proposed to use secret sharing instead of PKC [13]. Additive secret sharing with SMC has been used in order to compute Bayesian network.

Work by Doganay et al. also focuses on k-means clustering on vertically partitioned data, using SMC and secret sharing [14]. However, their work suffers from high network communication due to the usage of SMC. The number of messages exchanged increases quadratically based on the number of participants (data owners). Yildizli et al. address this issue, and manages to decrease the communication overhead by taking the advantage of non-colluding assumption [4].

In this thesis, we focus on removing the network communication overhead completely by moving the entire computation to a trusted and secure hardware, namely Cell Broadband Engine Architecture (CBEA). This way, the runtime of any data mining application will experience a major decrease, since memory transfer operations are faster compared to long-distance network communication. Previously, privacy preserving clustering on CBEA was proposed in [1]. However, this work is performed on unpartitioned data, while SMC and secret sharing are not used in the process.

3 Cell Broadband Engine Architecture

Cell Broadband Engine Architecture is a multi-core processor architecture, which was designed in 2004. Main reason for the development of CBEA was the need of a new processor for the upcoming Playstation 3 game console, owned by Sony Computer Entertainments Inc.. Together with IBM and Toshiba, a new architecture was developed, whose performance metrics for media processing are far better than existing desktop processors such as 32-bit Intel Architecture processors [15].

Although high performance was the primary goal during the design process of CBEA, another important aspect of CBEA is its hardware-based security features, developed mainly for protecting the intellectual properties of software developers. Nearly none of the existing processor architectures in the market provides a complete hardware-based support for secure computation; in other words security of the data and execution of a program is protected by not hardware means, but mostly software means. The problem with this approach is that software is much more vulnerable to attacks compared to hardware. CBEA focuses on this issue, and proposes an isolation feature of the cores, whereby an isolated core in CBEA separates itself from the rest of the system by hardware means [16].

In this section, a brief information about the technical details of CBEA is given. Two different types of cores are described with their properties and their usage in our project. This is followed by a detailed description of the separate security features supported in the CBEA. Subsequently, key hierarchy used in the security features is explained, and an outline is given on how to design secure applications for CBEA.

3.1 Technical Details

As mentioned in Section 3, three companies participated in the design process of CBEA: IBM, Toshiba and Sony. Sony's primary aim was to use Cell Broadband Engine processor (Cell processor) in their newest game console, i.e. Playstation 3, and other consumer products such as high definition television sets, where intensive data processing is needed [17]. On the other hand, IBM had other plans on the usage of Cell processor. They desired an architecture that can be used in the mainframes, servers, even in the supercomputers [15].

With these defined aims, three companies designed Cell Processor, which features one power processor element (PPE), and eight synergistic processor elements (SPE). Compared to other multi-core processors, PPE and SPEs have different technical properties [15]. For instance, PPE has the authority to use the whole system memory, which is 256 MB in the Playstation 3 configuration of Cell processor, while SPEs have limited memory of 256 Kb. Having different technical properties is not the only difference between the cores, they also differ in their usage style. In a Cell processor, PPE can be considered as the *rider*, while SPEs are *computational work horses.* SPE's technical properties makes it ideal for bulk data processing, where there is an extreme level of data-level parallelism. And if this data processing capability is utilized appropriately, CBEA offers 200 GFlops on single precision values, while 32-bit Intel Architecture processors give a performance of 25 GFlops on single precision [15]. In addition to the performance of a single Cell processor, one can cluster several Cell processors via high-speed connection, and achieve significant floating point and vector processing power [15].

Due to having multiple cores in CBEA, a coherent and fast way of communication between the elements is an important issue. Architects designed what is called Element Interconnect Bus (EIB) to handle data and instruction transfer between cores, main memory, and I/O devices. EIB has the ability to handle 16 simultaneous data transfer requests, called *direct memory access* (DMA). This property improves the parallel processing power of Cell processor, since SPEs can issue DMA requests at the same time. However, there are still some limitations in the DMA requests. The largest amount of data in a DMA request can be 16 KB, while the smallest amount is 16 B for performance concerns [15]. Figure 1 gives an overview of the CBEA, with 8 SPEs, a PPE and EIB that connects the components of CBEA.

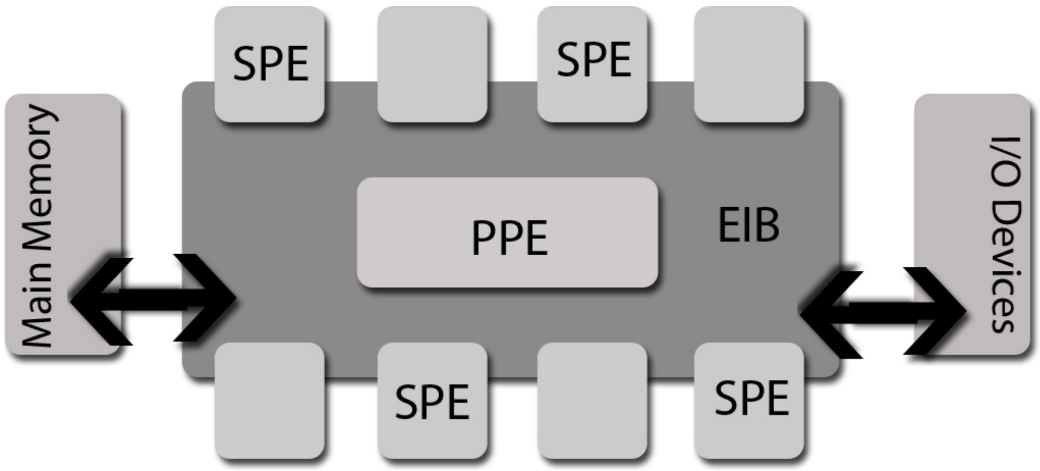


Figure 1: Cell Broadband Engine Architecture Overview

3.1.1 Power Processor Element

Power Processor Element (PPE) is a dual-threaded, dual-issue 64-bit core, with the clock frequency of 3.2 GHz [16]. It has 32 KB L1 instruction and data cache memories, and a 512 KB L2 cache [18, 19]. Designers aimed to optimize clock frequency, and power efficiency by using relatively shorter pipelines, and limited communication delays [15]. Despite having considerably high clock frequency, working only with PPE without utilizing SPEs may not give optimal performance results. In order to achieve better performances, managerial role of the PPE should be kept in mind, and SPEs data processing power must be used.

3.1.2 Synergistic Processor Element

Synergistic Processor Element (SPE) is a 32-bit processor, with a RISC-style 128-bit SIMD instruction set [19]. Every SPE contains 128 128-bit registers to execute SIMD instructions. SPE has separate instruction set from PPE. Its instruction set is optimized for performance on compute-intensive data, by featuring vector instructions to perform the same operation on multiple data [15].

One of the main differences between SPE and PPE is that every SPE has a local memory called Local Store (LS) and no cache. LS of an SPE is used to keep the data and instructions of the program that will run on the SPE, and it has a capacity of 256 KB. Before starting the execution, SPE should transfer the program (set of instructions), and data to its LS by requesting DMA operations from PPE. LS of an SPE can only be used by the owner SPE, other SPEs cannot use it to store instructions. However, some mechanisms are proposed to enable memory transfer between different SPEs. LS of each SPE is mapped in the system memory, which enables the transfer of data between SPEs and PPE [16].

In addition to DMA requests, there are two more alternatives for communication between SPEs and PPE. Mailboxing is the first of these alternatives. Each SPE has four incoming, and two outgoing mailboxes, whereby every mailbox holds a 32-bit data. Signalling is the second alternative that can only be used for synchronization purposes, while mailboxing can be used for data transfer, but with higher overhead compared to DMA.

An SPE can run in three different modes. The first mode is the *normal mode*, in which LS of an SPE running in this mode can be accessed by PPE or other SPEs. While operating on normal mode, other elements in the architecture can issue a DMA request, and get data from that SPE's LS. Although memory of an SPE is limited by 256 KB, size of an SPE program is not limited in normal mode. Overlays can be used and programs with size larger than 256 KB can be loaded into the LS and executed [2].

The second mode is the *isolated mode*, in which SPE isolates itself from the rest of the system using hardware support. An SPE running in isolated mode has full control over its LS, meaning that it can issue DMA requests from/to its LS. On the other hand, other elements in the architecture cannot issue any DMA request from/to the LS of SPE running in isolated mode. DMA request from other elements is the only method of communication that is not allowed in the isolated mode, i.e. SPE can receive mails or signals from the rest of the system in the isolated mode. Restriction of the DMA operation enables the isolation of the program execution from the system, and ensures security of the data and instruction in the isolated SPE. No data can be retrieved from the isolated SPE, not even the contents of hardware performance counters [17].

Program size is also limited on an isolated SPE. The static size of the program and data should not exceed 167 KB, while the runtime application size can be extended to 247 KB [20]. For an SPE application to run in the isolated mode, it must be encrypted and signed after being developed. Before execution it must be first verified and decrypted so that it is ensured that the application has not been altered or compromised. If the program fails to verify, it is not loaded into the SPE and the execution stops.

The last mode is the *emulated isolated mode*. A program running in this mode will be again verified and decrypted before execution. But this time, SPE is not physically isolated from the rest of the system [17]. Emulated isolated mode enables the software developer to debug SPE application, and to read or set program counters, unlike isolated mode. However, just like isolated mode, static program and data size is limited to 167 KB. This mode is only for debugging purposes, and it is not used in general execution.

One final remark should be made on the the number of SPEs in different configurations of CBEA. A regular Cell processor contains 8 SPEs, while the Cell processor in Playstation 3 can only use its 6 SPEs because of power consumption issues. On the hand a Cell Blade contains two Cell processor, in which all 8 SPEs are active in each Cell processor.

3.2 Security Features

In cryptography context, three important security services should be achieved in a platform, system or a program itself. The first service is authentication, in which the authenticity of a program is checked so that unauthorized alterations in the program are detected. After an application has been built and deployed, attackers may try to capture the application and make some changes in it. In order to avoid this kind of malicious modifications, developer may choose to apply digital signatures, which enable the end-user to check the authenticity of the application.

Confidentiality is the second security service, whereby data and application are encrypted so that unauthorized entities cannot access the content. Developers may choose to encrypt their application in order to prohibit other developers from learning the details of the program. Moreover, the processed data may be sensitive or just too valuable to disclose, so that it should be kept secret. In both cases, an encryption key must be used to fulfill encryption requirement.

The last security service can be described as the isolation of an application in a processing environment. Even under the assumption that encryption and authentication mechanisms achieve their objectives and a legitimate application starts running on the platform, it should still be checked that memory addresses read or written by that application can't be accessed by any other application. Operating system is usually held responsible for this control, since it is the supervisor, and it can operate the applications so that isolation aim is achieved.

For all these three security services, various solutions at software level have been proposed. But all these solutions suffer from a major security flaw that can lead to serious problems. A piece of software can be designated as the authenticator by the operating system, and it can be used to authenticate programs before their execution starts. But the problem with designating a piece of software as the authenticator is the vulnerability of the software itself. Once the authenticator application is verified, it can be altered after some time and be used to authenticate malicious softwares.

Encryption mechanism has a different type of problem. A key, used to encrypt data, must be kept safely in the system. This arises the problem of recursive encryption, whereby we have to encrypt the first key with another key so that the system stays secure. No matter how many keys are used, we end up having a key kept in the plaintext form, which is already used to encrypt other keys or data. Compromise of this key will lead to a breakdown in the encryption mechanism, since acquiring it means acquiring the other keys and the data. This key must be kept in a very secure location.

As previously mentioned, operating system can isolate one process from another by software means and it can succeed doing so up to a certain level. But, there is an unrealistic assumption in this approach that the operating system is not compromised and is bug-free. If somehow an attacker gets control of the operating system, one can no longer talk about isolation of the applications. Therefore, in this approach software-only enforced type of isolation cannot be fully trusted.

CBEA provides hardware-based solutions to these problems by introducing the isolated mode of execution of SPE. In section 3.1.2, isolated mode of an SPE is explained briefly. It is mentioned that when an SPE runs in the isolated mode, elements in the rest of the system cannot read data inside the isolated SPE, or send data to it through DMA requests. In the following three sections, the solutions proposed by CBEA to these security problems are explained.

3.2.1 Secure Processing Vault

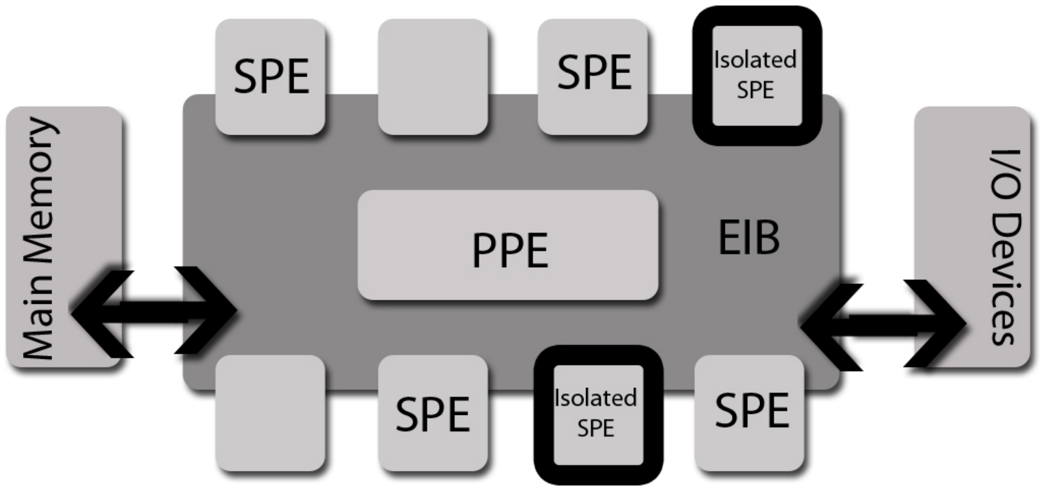


Figure 2: Isolated SPE in CBEA

Secure Processing Vault feature of CBEA is the actual feature that enables the physical isolation of SPE from the rest of the system [1]. When an SPE enters the vault, it disengages itself from the bus not in software, but in hardware means. Figure 2 illustrates two SPEs that are in the isolated mode and separated itself from other processor elements. This separation allows the isolated SPE to discard any DMA request that is originated from any other element in the architecture. Other elements in the system can communicate with the isolated SPE only by mailboxing and signaling, but since these operations can't read from the isolated LS, it is assured that the isolated SPE is the only element that can access its LS [20]. However, this does not imply that the isolated SPE is completely separated from the system. The isolated SPE opens some part of its LS to other elements, so that the parties can communicate. When the isolated SPE issues a DMA in or out request, the data is first gathered on the open part of the isolated SPE's LS, and than based on the request, data is moved to its target [2]. When running in the isolated mode, application should be held responsible for the incoming data. The application should not allow any instruction or data that can risk the data in the isolated LS, or leak the data outside.

Secure Processing Vault feature protects the data and program confidentiality and runtime integrity in the presence of the compromised operating system problem by giving the operating system only one access in the isolated mode, which is the *cancel* command. If an SPE is running in the isolated mode, even the supervisor, PPE, cannot access, modify or read the data in the isolated LS [1]. PPE can only tell the isolated SPE to stop execution. But even in that case, before canceling the execution, isolated SPE deletes all data and instruction in its LS and leaves no trace behind, which means that PPE can't learn anything after issuing cancel command. Therefore, even if the operating system becomes compromised by an attacker, data and application in the isolated SPE are kept safe and secure.

3.2.2 Runtime Secure Boot

Secure Processing Vault keeps the application isolated from the rest of the system; however having only this feature in an architecture is not sufficient.

What if the application itself is attacked and modified, so that when it goes into the isolated SPE, it sends out the sensitive information? The system should check whether the application has been altered during the deployment process or not. Authentication comes into play at this point. There are existing solutions to check the authenticity of a program before starting its execution. Secure Boot Technology is an example of such solutions, whereby starting from the power-on time every piece of code is authenticated through hardware means [2]. The authentication by hardware is performed until authenticating the operating system itself. After the authentication of the operating system, the duty of authenticating new applications is passed from the hardware to the operating system. This approach may seem invulnerable at first sight; but it should be noted that most of the attacks are executed during the runtime, operating system may be compromised after a certain amount of time since the boot, and therefore it can still be used to authenticate malicious software [2].

Runtime Secure Boot is the feature proposed by CBEA, whereby the authentication of an application is done directly by the hardware; more than once, and at any time during the execution of the application [20]. Even if the application is attacked after completing the initial authentication process, this attack can be detected in the later repetitions of the authentication. Moreover, since the authentication is done by hardware means, a break in the authentication chain is caught and necessary measures are taken, which is basically to cancel the execution.

Runtime Secure Boot is implemented in conjunction with the Secure Processing Vault mechanism. If an application wants to work in an isolated SPE, first it is loaded into the isolated part of the LS, and then goes into the authentication process [1]. If the authentication fails, the application is erased from the isolated LS, and execution stops. Otherwise, the application can start in the isolated SPE.

3.2.3 Hardware Root of Secrecy

In order to encrypt data or applications, an encryption key should be used, which reduces the security of the system to the protection of this key. Encrypted data or application is kept safe as long as this key is not captured by any attacker. This issue arises the question on where and how to keep the encryption key. One can choose to encrypt this key with another key, but that does not solve the problem; there should always be a key that is kept in plaintext form. Locating this key, which is called the *root key*, on the hard-disk makes the encryption chain vulnerable to all sorts of attacks, which implies that the root key should be kept somewhere safe. In CBEA, the root key is embedded into the hardware, and can not be retrieved, read or modified by any means [1], which is the basis of the third security feature, Hardware Root of Secrecy.

Before deploying the application, developer may choose to encrypt the data and application with a derivative of the hardware key, so that information is kept secret from any other entities in the process. This encrypted application and data can only be decrypted using the hardware key, which is very hard to read or modify by attackers [2]. The data in the encrypted application may contain other encryption keys that will be used for various purposes, but since the whole package is sent in encrypted form, an attacker learns nothing after capturing this package. Figure 3 gives a representation of an isolated application.

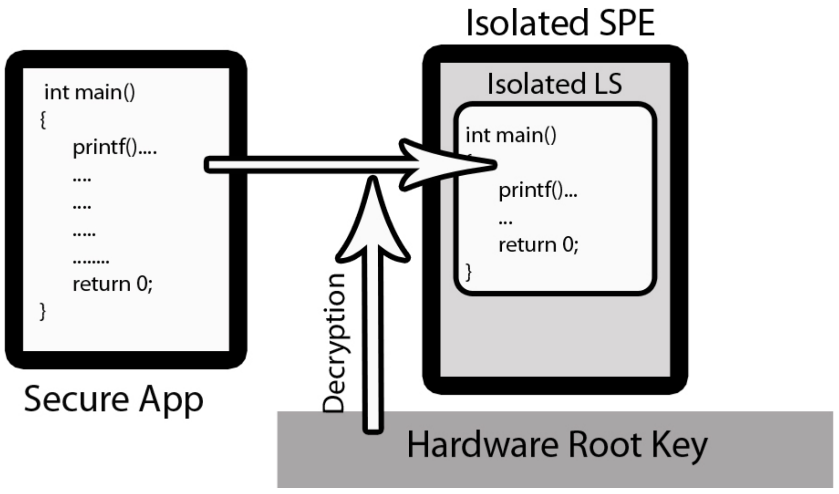


Figure 3: Secure application in isolated SPE

Another aspect of this feature is its use during the execution of an application running in an isolated SPE. If an application is running in the isolated SPE, its initial data and final outcome should also be kept safe from any attackers. When this application tries to write some piece of data to the hard-disk, where every application have access, the outcome is first encrypted using a derivative of the hardware root key, where this derivative key is special to the application itself [20]. This property keeps even the results of the data safe from all attackers. As shown in Figure 4, if this application wants to retrieve previously encrypted data from the hard-disk, data is first fetched and moved to the open area of the isolated LS, decrypted by the derivative key and placed into the protected part of the isolated LS [1].

Similar to Runtime Secure Boot, in order to use the hardware root key,

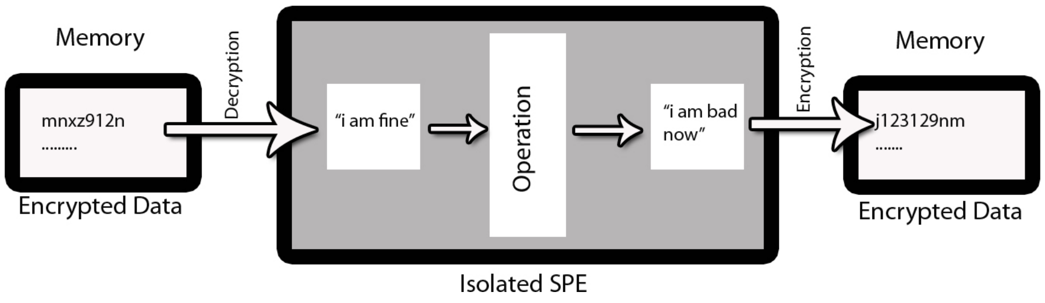


Figure 4: Encrypted storage in an isolated SE

an application should run in the Secure Processing Vault, which also implies that the application must be authenticated beforehand. Unauthenticated applications are not given access to the hardware root key mechanism. It should be noted that even the authenticated applications cannot learn anything about the root key [20].

3.3 Key Hierarchy

In CBEA, the security of the keys that are used throughout the encryption and authentication process relies on the security of the root key. Protecting this key by hardware means (i.e. the root key is hardwired to the processor) makes it safer compared to software means of security [1]. Different keys used in the processes are kept in a format determined by an industry standard, X.509 [20] and their protection is achieved by a key hierarchy whose top is occupied by the root key. In the following two sections, key hierarchy and keys that are used in the authentication and encryption mechanisms are explained.

3.3.1 Application Trust Chain

Three distinct parties participate in the authentication process. The first party is the Root Certificate Authority (Root CA), which may be the manufacturer or the distributor of the Cell processor system. The Root CA has the power to decide which Certificate Authorities (CA) can sign the developers' certificates. The second party in the mechanism is the CA, which can be more than one. Their role is to sign application developers' certificates, so that the applications developed by these verified developers can run in the isolated mode [20]. The last player is the application developer. In order to ensure secure delivery and isolated execution of its application, developer should create a pair of application authentication keys with public and private parts. The public part of the application authentication key should be signed by an approved CA.

In addition to three parties, we can also speak of three key components that are used in the authentication process. The first component is the SPE Secure Loader, which is loaded into the isolated SPE before the application and verifies the authenticity of the application. This component contains the public counterpart of the Root CA key pair [20]. The second component is the loader key ring. It contains the public keys of CA's that are allowed to sign application developers' certificates. This component is only accessed by the SPE Secure Loader. The last component is the application image. It contains the application binary in encrypted form, public counterpart of the application authentication key, and the signature value.

The authentication process starts with the authentication of Root CA's public key, located in the SPE Secure Loader. Once this key is verified, SPE

Secure Loader loads up the loader key ring. At the same time, application is loaded into the isolated SPE. Public key of the CA is verified by the SPE Secure Loader. The next step is to use the verified public key of CA, and authenticate the public key of application authentication key pair. Finally, public counterpart of the application authentication key is used to authenticate the application itself. If any stage of the authentication procedure fails, the execution stops and application is removed from the isolated SPE. Otherwise, application starts executing. Figure 5 gives an outline of the authentication mechanism.

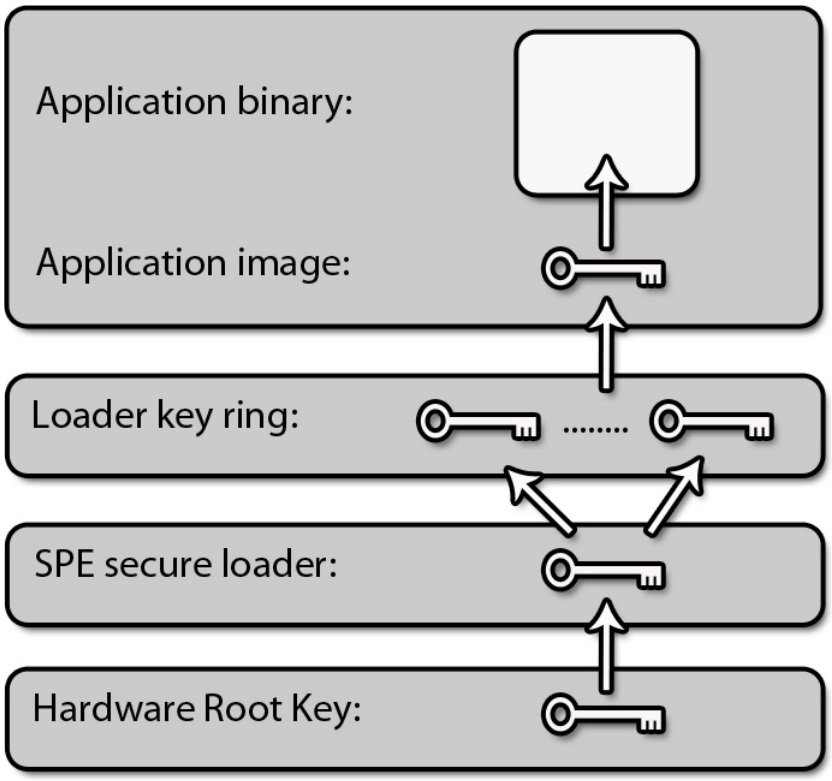


Figure 5: Authentication chain

3.3.2 Application Encryption Chain

The application is encrypted in the build-time, and decrypted in the run-time before the execution starts [20]. At build-time, the application is encrypted by two keys. The application is first encrypted by the private counterpart of the application authentication keys. This encrypted application binary is placed into the application image as described in Section 3.3.1. Finally the whole application image is encrypted using the public counterpart of the application decryption key pair, where the private counterpart of this key is located inside the SPE Secure Loader [1]. Public counterpart of the application decryption key pair comes with the software development kit of CBEA.

In the run-time, application is loaded into the isolated SPE first. After the authentication procedure succeeds, it is decrypted by the private counterpart of the application decryption key pair, followed by the decryption of the application image using the public part of the application authentication key pair. If the decryption mechanism fails at some point, the execution stops just like in the authentication process.

3.4 Building Secure Applications

In order to develop a secure application, which uses the isolation feature of CBEA, developer should first create an application authentication key pair [1]. The public counterpart of this pair should be signed by an authorized CA, so that the application is authorized to run in isolated mode. Once the application authentication keys are created and signed, they are used to create the application image as described in Section 3.3.1. The application can be called a secure application after this point [20]. A very important remark about the secure applications is that it can only run in an isolated SPE. If the developer signs and encrypts its application, it is completely guaranteed that this application will definitely not run in normal SPE mode, where data and instructions of the applications can be retrieved by other elements in the system.

Generating the application authentication keys and building the application image do not suffice for the developer. Communication between the isolated SPE and other elements should be considered and planned in a way that, other elements in the architecture will not have DMA read or write requests to the isolated SPE. Otherwise, bus error will occur, which ends the execution of the application.

4 Simple Secure Two-Party Computation Model

Secure multi-party computation is the process of evaluating an operation between semi-honest parties over a previously agreed protocol [21, 22]. Throughout the evaluation of the operation, parties do not want to reveal any kind of information about their secret data to the other parties involved in the process. Due to being semi-honest in the process, parties obey to the protocol and perform the necessary steps, while trying to gain as much information as possible from the messages received or observed. Because of this fact, messages sent and received between parties should not reveal any kind of information about the secrets held.

In this section, a secure two-party computation protocol is outlined, which contains only two parties; Alice and Bob respectively. Parties in this secure two-party computation model are also semi-honest, which implies that the model should address privacy concerns of the parties. Three operations are defined in this model: Addition, multiplication and comparison. Rivest Oblivious Transfer (OT) is used in order to implement the comparison operation [7]. Comparison is also divided into sub operations, which are the equality of bits, comparison of bits, comparison of numbers, and comparison of secrets.

In the multiplication and comparison operations, there is also a third party (TP), which does not involve in the computation, but serves as a random number generator under a scheme. This party gives random numbers to Alice and Bob according to the operation they want to perform. Just like Alice and Bob, TP is also semi-honest.

The proposed computation model uses additive secret sharing as the security primitive, which is homomorphic with respect to the addition operation [23, 24]. The secret value, on which the operations will be done, can be divided into two pieces additively. For instance, to divide a secret value x into two pieces, one can generate a random number x_0, and calculate $x_1 = x - x_0$, where x, x_0 and x_1 are elements of ring $\mathbb{Z}_N$ and N is an integer. Neither x_0, nor x_1 gives any information about the secret data x since they are simply random numbers independent of x. We define $[x] = (x_0, x_1)$, such that $x_0 + x_1 \equiv x \in \mathbb{Z}_N$, where x, x_0 and x_1 are elements of $\mathbb{Z}_N$. Furthermore, x_0 can be thought as the share of Alice, while x_1 is Bob's for simplicity.

Throughout this work, following notations will be used for computing shares, where c is a constant value, and operations are done in $\mathbb{Z}_N$:

$$\begin{aligned} [x] + [y] &= (x_0 + y_0, x_1 + y_1) = [x + y] && (1) \\ c[x] &= (cx_0, cx_1) = [cx] && (2) \\ c + [x] &= (x_0 + c, x_1) = [x + c] && (3) \end{aligned}$$

In the proposed model, the results are also secretly shared, meaning that at the end of the operation parties need to open their own part of the result to the opposing party in order to learn the actual result.

4.1 Rivest Oblivious Transfer

Oblivious Transfer is a cryptographic primitive, in which the receiver obtains one of the N messages offered by the sender, but learns nothing about the other unchosen messages [21]. OT allows the receiver to choose one of the messages and learn the information in it, while the sender cannot learn the choice of the receiver. In our model, an efficient implementation of OT is needed, and Rivest OT is chosen for its practicality and suitability to our setting [7].

In our implementation, Alice has the role of sender, while Bob is the receiver. Alice has two messages to send to Bob, m_0 and m_1. Bob chooses the message based on its input bit c. Rivest OT starts with the setup phase, where TP generates three random bits r_0, r_1, d, and then calculates a fourth bit according to the formula $r_d = (r_0 \wedge \neg d) \oplus (r_1 \wedge d)$. Then, TP sends r_0 and r_1 to Alice, while r_d and d are sent to Bob. Both parties learn nothing about the opposing party's bits, since the values are randomly created.

After the completion of the setup phase, Bob calculates $e = c \oplus d$, and sends e to Alice. After getting e, Alice first calculates $r_e = (r_0 \wedge \neg e) \oplus (r_1 \wedge e)$ and $r_{einv} = (r_0 \wedge e) \oplus (r_1 \wedge \neg e)$, and then sends $f_0 = m_0 \oplus r_e$ and $f_1 = m_1 \oplus r_{einv}$ to Bob. Finally, Bob computes $m_c = f_c \oplus r_d$.

At the end of the process, Alice learns nothing about c, the input bit of Bob. Moreover, Bob learns nothing about $m_{\neg c}$. As shown in Figure 6, the total overhead of this operation is 4 bits sent by TP, 1 bit sent by Bob, and 2 bits sent by Alice, resulting in 7 bits exchanged in total.

4.2 Addition

Addition is the simplest operation in our model, since there is not any communication between parties and only one single addition is needed. Alice has x_0 y_0, and Bob has x_1 y_1, where parties want to compute $[x] + [y]$. As previously mentioned, only one operation per party is done. Alice calculates $result_0 = x_0 + y_0$, and Bob calculates $result_1 = x_1 + y_1$. All the numbers

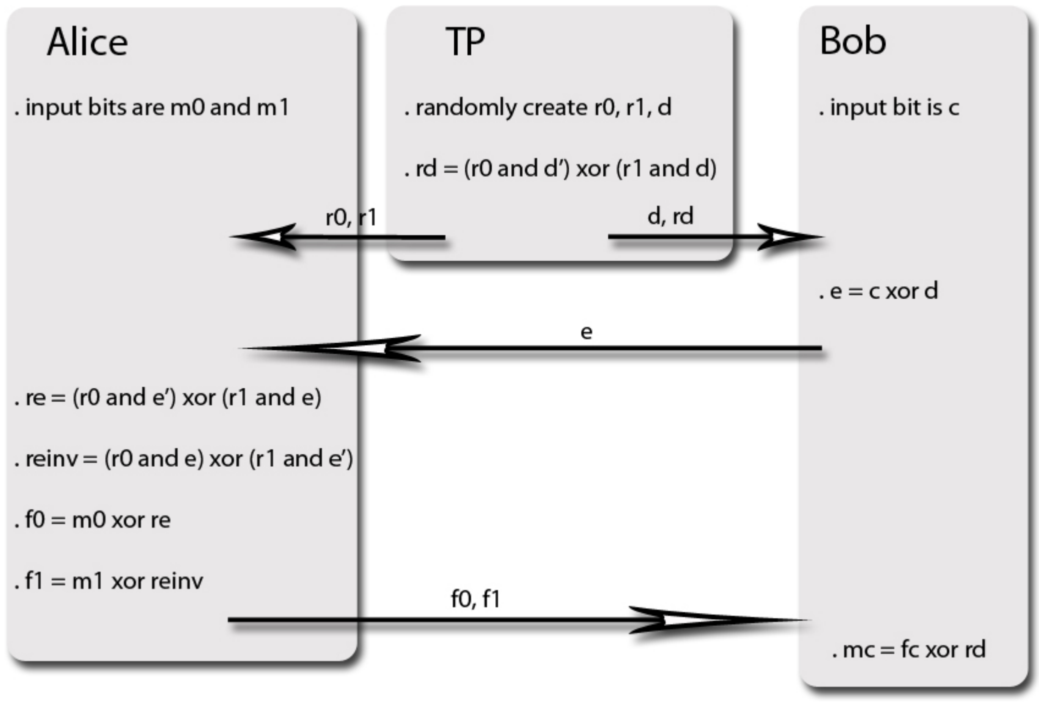

Alice
TP
Bob
. input bits are m0 and m1
. randomly create r0, r1, d
. input bit is c
. rd = (r0 and d') xor (r1 and d)
r0, r1
d, rd
. e = c xor d
e
. re = (r0 and e') xor (r1 and e)
. reinv = (r0 and e) xor (r1 and e')
. f0 = m0 xor re
. f1 = m1 xor reinv
f0, f1
. mc = fc xor rd

Figure 6: Rivest OT

calculated and used in the addition are elements of ring $\mathbb{Z}_N$. Since the parties have only random shares of the result, they need to open their results to the opposing party to learn the real end-result. Addition has no overhead of communication and TP is not used in this operation.

4.3 Multiplication

Multiplication is more complicated compared to the addition operation. Alice has $[x]$, Bob has $[y]$, and they want to compute $[z] = [x][y]$. TP creates two random ring elements $[a]$ and $[b]$, and calculates $[c] = [a][b]$. After that, TP splits $[a]$, $[b]$ and $[c]$ into a_0, a_1, b_0, b_1, c_0 and c_1. Finally, TP sends a_0, b_0, c_0

to Alice, and a_1, b_1, c_1 to Bob.

After getting inputs from TP, Alice calculates $x^1_0 = x_0 - a_0$ and $y^1_0 = y_0 - b_0$, while Bob calculates $x^1_1 = x_1 - a_1$ and $y^1_1 = y_1 - b_1$. Then, they share their results of x^1 and y^1 with the opposing party. Finally, Alice computes its share of the result as $result_0 = c_0 + x^1y^1 + y^1a_0 + x^1b_0$. Bob does the computation of $result_1 = c_1 + y^1a_1 + x^1b_1$. All the values used in the computation are elements of the ring $\mathbb{Z}_N$.

The proposed model computes the multiplication operation correctly as shown below:

$$\begin{aligned} result &= result_0 + result_1 && (4) \\ result &= c_0 + x^1y^1 + y^1a_0 + x^1b_0 + c_1 + y^1a_1 + x^1b_1 && (5) \\ result &= [c] + x^1y^1 + y^1[a] + x^1[b] && (6) \\ result &= ([x^1] + [a])([y^1] + [b]) && (7) \\ result &= [x][y] && (8) \end{aligned}$$

As shown in Figure 7, the communication overhead of the multiplication operation is 6 ring elements sent by TP, 2 ring elements sent by Alice and 2 ring elements sent by Bob, resulting in 10 ring elements exchanged in total.

4.4 Comparison

Comparison is the most complicated operation among operations defined in our security model. It cannot be efficiently implemented using only addition and multiplication. In this section, first comparison and equality check of the bits are explained. Using these two primitives, an algorithm for comparison of two numbers is explained. Finally, an algorithm for comparison of

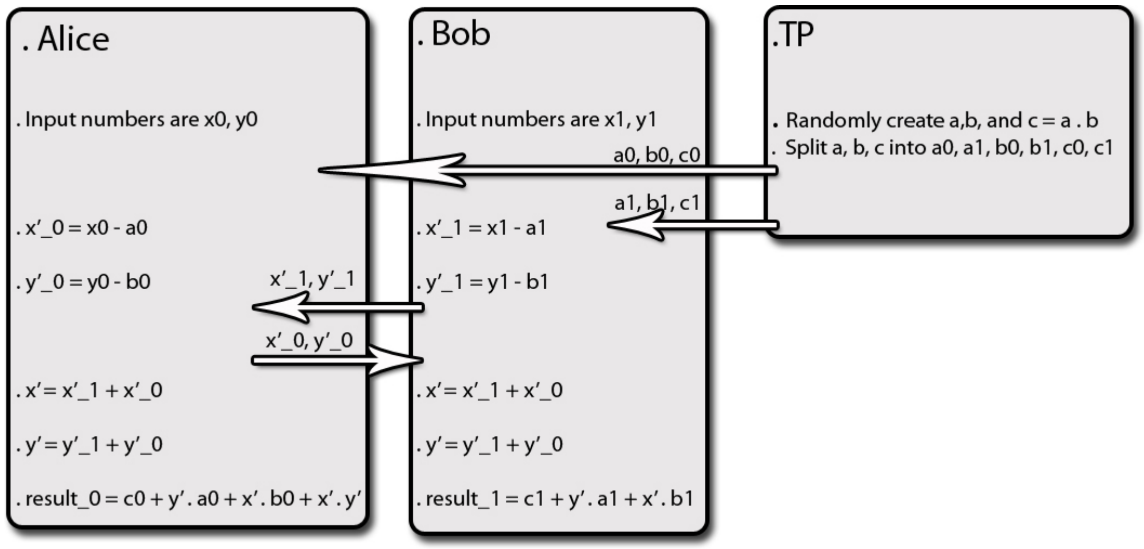

. Alice
. Input numbers are x0, y0
. x'_0 = x0 - a0
. y'_0 = y0 - b0
. x' = x'_1 + x'_0
. y' = y'_1 + y'_0
. result_0 = c0 + y'. a0 + x'. b0 + x'. y'
. Bob
. Input numbers are x1, y1
. x'_1 = x1 - a1
. y'_1 = y1 - b1
. x' = x'_1 + x'_0
. y' = y'_1 + y'_0
. result_1 = c1 + y'. a1 + x'. b1
.TP
. Randomly create a,b, and c = a . b
. Split a, b, c into a0, a1, b0, b1, c0, c1
a0, b0, c0
a1, b1, c1
x'_1, y'_1
x'_0, y'_0

Figure 7: Multiplication

secretly shared values using the previously mentioned comparison operations is provided.

4.4.1 Equality of Bits

Let $a, b \in \mathbb{Z}_2$ be the private bits of Alice and Bob respectively. Two private bits are equal if $a \oplus b \oplus 1$ is 1. So Alice and Bob perform a multiplication operation, where Alice's secrets are $a \oplus 1$ and 1, Bob's secrets are b and 0. If the result of multiplication is 1, then the bits are equal. Otherwise, the result will be 0, which implies that bits are different.

As shown in Figure 8, the overhead of this operation is 1 multiplication, necessitating an exchange of 10 ring elements in total.

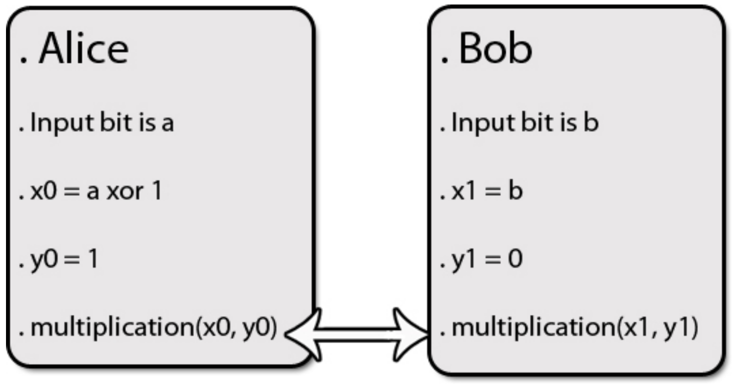


Figure 8: Equality of bits

4.4.2 Comparison of Bits

Let $a, b \in \mathbb{Z}_2$ be the private bits of Alice and Bob respectively. Alice and Bob may want to compute the secret of sharing of the comparison $a > b$, which is 1 if a is bigger than b, 0 otherwise. They can compute the result with one call to OT as follows. Alice creates a random bit z, and sets input messages of OT as z and $z \oplus a$. Bob makes $1 \oplus b$ as its input bit for OT. At the end of the OT process, Alice keeps z as the output, while Bob gets $z' = a(1 \oplus b) \oplus z$. The addition of the results, $z + z' = a(1 \oplus b)$, is 1 if a is 1 and b is 0.

As shown in figure 9, total amount of communication in this process is equivalent to exchanging of 7 bits in total due to one call to the OT.

4.4.3 Comparison of Numbers

Let $a, b \in \mathbb{Z}_n$ be the private numbers of Alice and Bob respectively. In order to calculate the inequality of these numbers, we split the comparison into smaller comparisons recursively. Let $m = \lceil \sqrt{n} \rceil$, we split a and b into a_h, a_l, b_h and b_l, where $a = a_h m + a_l$ and $b = b_h m + b_l$. After this step, it can be

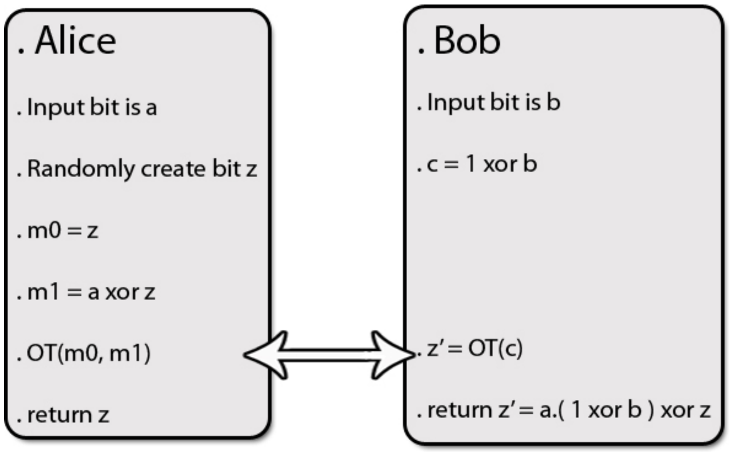


Figure 9: Comparison of bits

stated that $a > b$ if and only if $(a_h > b_h) \vee ((a_h = b_h) \wedge (a_l > b_l))$. In terms of secret sharing, we can write as:

$$[a > b] = [a_h > b_h] \vee ([a_h = b_h] \wedge [a_l > b_l]) \tag{9}$$

At the bottom of the recursion, when $\mathbb{Z}_n$ has the order of $n = 2$, comparison and equality check of the bits are used, which were explained in Section 4.4.1 and 4.4.2. The final result of the number comparison can be computed by opening the shared results to the opposing party.

4.4.4 Comparison of Secrets

Comparison of numbers may seem sufficient in our two-party model. However, this operation cannot be used to compare secret shared values, where Alice and Bob knows some part of the data. An efficient protocol should be designed to compare $[a]$ and $[b]$, where Alice knows a_0 b_0, and Bob knows a_1 b_1. At the end of the comparison, each party gets a secret sharing of the result, $[a > b]$.

Since additive secret sharing is used in our model, the problem of computing $[a > b]$, is actually $a_0 + a_1 > b_0 + b_1$. This problem is also equivalent to computing $a_0 - b_0 + (n-1) > b_1 - a_1 + (n-1)$, where $(n-1)$ is added to prevent any negative operand. At this point, if one of the secrets, $[x]$, is shared such that, $x_0 \geq n - x_1$, the result of the comparison will be wrong. To overcome this issue, following three calculations must be performed:

- If $a_0 \geq n - a_1$ but not $b_0 \geq n - b_1$, compute $a_0 - b_0 + (n-1) > b_1 - a_1 + (n-1) + n$
- If $b_0 \geq n - b_1$but not $a_0 \geq n - a_1$, compute $a_0 - b_0 + (n-1) + n > b_1 - a_1 + (n-1)$
- Otherwise, compute $a_0 - b_0 + (n-1) > b_1 - a_1 + (n-1)$

Since the results of the comparisons $a_0 \geq n - a_1$ and $b_0 \geq n - b_1$ are secretly shared among the parties, all three calculations should be performed to get the true result. The comparison of secrets protocol consists of the following operations in order:

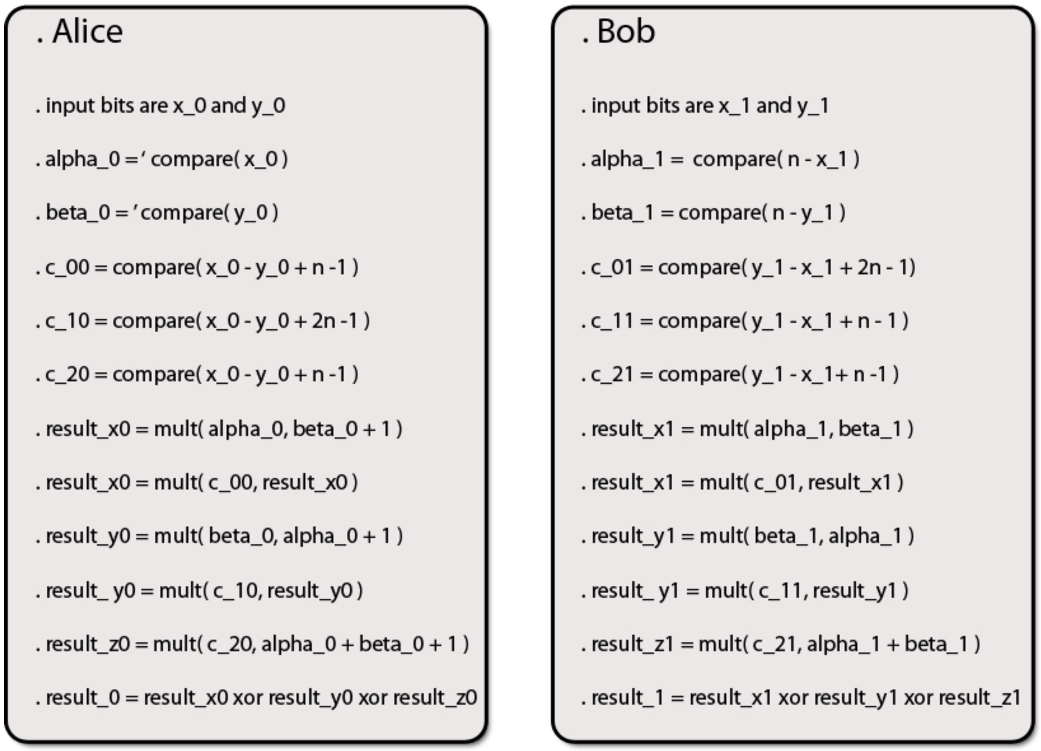
$$[\alpha] = [x_0 \geq n - x_1] = \neg[n - x_1 > x_0]$$

$$[\beta] = [y_0 \geq n - y_1] = \neg[n - y_1 > y_0]$$

$$[c_0] = [x_0 - y_0 + (n-1) > y_1 - x_1 + n + (n-1)]$$

$$[c_1] = [x_0 - y_0 + n + (n-1) > y_1 - x_1 + (n-1)]$$

$$[c_2] = [x_0 - y_0 + (n-1) > y_1 - x_1 + (n-1)]$$



```
. Alice

. input bits are x_0 and y_0
. alpha_0 =' compare( x_0 )
. beta_0 = 'compare( y_0 )
. c_00 = compare( x_0 - y_0 + n -1 )
. c_10 = compare( x_0 - y_0 + 2n -1 )
. c_20 = compare( x_0 - y_0 + n -1 )
. result_x0 = mult( alpha_0, beta_0 + 1 )
. result_x0 = mult( c_00, result_x0 )
. result_y0 = mult( beta_0, alpha_0 + 1 )
. result_ y0 = mult( c_10, result_y0 )
. result_z0 = mult( c_20, alpha_0 + beta_0 + 1 )
. result_0 = result_x0 xor result_y0 xor result_z0
```

```
. Bob

. input bits are x_1 and y_1
. alpha_1 = compare( n - x_1 )
. beta_1 = compare( n - y_1 )
. c_01 = compare( y_1 - x_1 + 2n - 1)
. c_11 = compare( y_1 - x_1 + n - 1 )
. c_21 = compare( y_1 - x_1+ n -1 )
. result_x1 = mult( alpha_1, beta_1 )
. result_x1 = mult( c_01, result_x1 )
. result_y1 = mult( beta_1, alpha_1 )
. result_ y1 = mult( c_11, result_y1 )
. result_z1 = mult( c_21, alpha_1 + beta_1 )
. result_1 = result_x1 xor result_y1 xor result_z1
```

Figure 10: Comparison of Secrets

$$[x > y] = [c_0] \cdot ([\alpha] \cdot ([\beta] + 1)) + [c_1] \cdot ([\beta] \cdot ([\alpha] + 1)) + [c_2] \cdot ([\alpha] + [\beta] + 1)$$

It is important to note that the results of comparison are in $\mathbb{Z}_2$, while the computations for c_0, c_1, c_2 are done in $\mathbb{Z}_{4n}$ to accommodate a possible overflow due to the addition of three numbers (i.e. c_0, c_1, and $c_2 \in \mathbb{Z}_n$). Figure 10 gives an outline of the secret comparison operation. The overall cost of the secret input comparison is 5 comparison of numbers and 5 multiplication operations.

5 K-Means Clustering

Clustering of a set of data into smaller subsets is a commonly used technique in applications such as pattern recognition, statistics, data mining and image processing [25]. The problem consists of partitioning a set of data, into smaller homogeneous groups of data, called a *cluster*, where data points in the same group have closer/similar attributes. Clustering data, and finding the centers of separate clusters can be used in daily life, as well as academic purposes. One can think of a situation, in which a company holds a set of spatio-temporal data of people in a city. The company can use this data to place its advertisements, such that the advertisements are placed on cluster centers. This necessitates an efficient clustering algorithm, since data can grow rapidly in size and become unmanageable.

There are various algorithms proposed to solve the clustering problem, and k-means clustering algorithm is one of the most popular [26]. Briefly, it is simply based on assigning every entry in the data into a cluster, based on its distance from the centers of clusters [26]. However, the problem with this approach so far as the distributed case is concerned is that it assumes the data is owned by only one entity. Several modifications should be applied to the algorithm in order to cluster partitioned data among several data holders.

In a vertically partitioned data, an entry is divided into r parts, and each part is owned by a different entity, where r is the number of entities [4]. During the process of assigning an entry into a cluster, every entity proceeds according to a previously agreed protocol, and shares his/her partial distances so that the actual distance can be calculated. At this point, an entity may choose not to share its data, or partial distance data for privacy concerns since the former is sensitive and the latter may betray some information on the former. Privacy preserving k-means clustering algorithms are used in these circumstances, where general k-means clustering algorithms fail to protect privacy [4].

In this section, two algorithms are explained briefly that efficiently produce clustered data. The first algorithm is the general k-means clustering algorithm, which may be used in the existence of only one entity. Secondly, a privacy preserving k-means clustering algorithm will be given that handles the case of vertically partitioned data, where parties do not want to reveal any information about their data to other parties. Note that the privacy preserving variant of the k-means clustering algorithm is of the focus of this thesis, since it is assumed that the spatio-temporal data is vertically distributed among a set of data owners.

5.1 General K-means Clustering Algorithm

In this algorithm, data is composed of m entries, where each entry consists of t attributes. This set of data is grouped into k different clusters, where cluster n has a cluster center μ_n and μ_n also consists of t attributes. Let μ_c be the c^{th} cluster center, μ_{ci}, $i \in \{0, .., t\}$, represents the i^{th} attribute of the cluster mean.

The algorithm may be composed of fixed or variable number of rounds. Before the first round, cluster centers are randomly initialized, i.e. each attribute of each cluster is given an initial random value. In each round, each entry in the data is assigned to the closest cluster based on a distance metric. Generally, *Euclidean distance* is used to calculate which cluster is the closest [3]. As shown in formula 10, square of a distance between an entry and a cluster center is the sum of the square of sub-distances between the corresponding attributes, or so called *dimensions.* Algorithm 1 illustrates of an efficient way of computing Euclidean distance.

$$||e_x - \mu_y||^2 = \sum_{p=1}^{p=t} ||e_{xp} - \mu_{yp}||^2 \qquad (10)$$

Algorithm 1 Calculate Distance 1

```
Require: entry $e$ is the first parameter
Require: cluster center $\mu$ is the second parameter
Require: $t$ is the number of attributes
1: TempDistance = 0
2: for $i$ from 0 to $t$ by 1 do
3:     TempDistance += $||e_i - \mu_i||^2$
4: end for
5: return $\sqrt{TempDistance}$
```

At the end of a round, every cluster center is updated based on the values of entries the cluster currently contains. Depending on whether the exit criteria is met or not, algorithm continues with another round, or terminates. Figure 11 gives an example set of data, points positioned on an area.

Before starting to cluster data, three cluster centers are positioned randomly on the area, shown as black dots in Figure 12. At the end of the first iteration, data points belonging to each cluster are separated by a delimiter on the area.

The cluster indices of each point can change after each iteration as shown

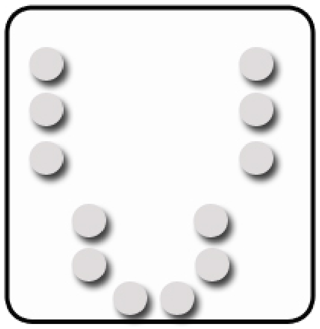


Figure 11: K-means clustering: Initial positions

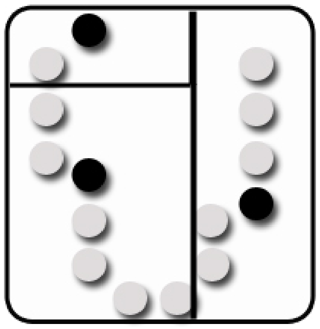


Figure 12: K-means clustering: The end of first iteration

in Figure 13. Some of the data points now belong to different clusters.

The general k-means clustering algorithm is given in Algorithm 2. The performance heavily depends on the initial values of the cluster means [4]. The cluster centers may be initialized very close to each other, which can influence the number of rounds to be computed and therefore, the execution time of the algorithm. Moreover, final result is also affected by the randomness. Consider a case, where a cluster is centered on a highly populated position, while other cluster centers are far away from the data entries.

Algorithm 2 General k-means clustering

```
Require: $m$ is the number of entries
Require: $k$ is the number of clusters
 1: for $c$ from 0 to $k$ by 1 do
 2:     $\mu_c = random$
 3:     $\mu_c^1 = 0$
 4: end for
 5: repeat
 6:     for $x$ from 0 to m by 1 do
 7:         minIndex = 0
 8:         minDistance = CalculateDistance($e_x$, $\mu_0$)
 9:         for $i$ from 1 to $k$ by 1 do
10:             tempDistance = CalculateDistance($e_x$, $\mu_i$)
11:             if tempDistance < minDistance then
12:                 minIndex =$i$
13:                 minDistance = tempDistance
14:             end if
15:         end for
16:         ClusterIndex[$x$] = minIndex
17:         Add values of $e_x$ to $\mu_{minIndex}^1$
18:     end for
19:     for $c$ from 0 to $k$ by 1 do
20:         Calculate $\mu_c$ based on $\mu_c^1$ and number of entries in cluster $c$
21:     end for
22: until Termination criteria is met
```

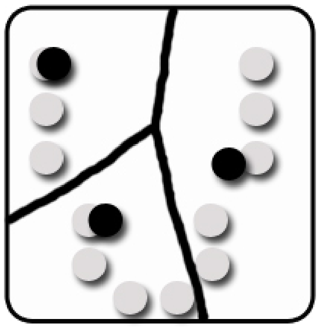


Figure 13: K-means clustering: Final positions

5.2 Privacy Preserving K-means Clustering Algorithm

In order to achieve k-means clustering, where privacy of each entity is conserved, algorithm in [4] is chosen and implemented in this work. The selected algorithm, to some extend, similar to the general k-means clustering algorithm. There are again m data entries and k clusters, where each entry and cluster center consist of t attributes. Moreover, unlike the case in general k-means clustering, there is a number of r entities/data holders. For every entry e_x, $x \in \{0, .., m\}$, party i, $i \in \{0, .., r\}$, holds a subset of the attributes, where e_{xi} is the projection of entry e_x onto the attributes of party i.

General structure of the two algorithms in Section 5.1 and 5.2, namely classical k-means clustering algorithms and its privacy-preserving variant are very similar. However, due to privacy concerns, there are two major differences. The first difference is in the distance calculation step. In general k-means clustering, distance is calculated using Euclidean metric, and the calculated distance reflects the actual result. But in privacy preserving clustering, since each party holds some part of the data, distances are calculated based on projection onto the owned attributes. Formula 10 is transformed into the Formula 11 as follows.

$$||e_x - \mu_y||^2 = \sum_{p=1}^{p=r} ||e_{xp} - \mu_{yp}||^2 \tag{11}$$

Algorithm 3 gives the outline of the privacy preserving k-means clustering algorithm. For each cluster, all parties initialize the cluster attributes that they own in parallel. After that, the closest cluster is computed, and each party updates the cluster mean attributes in parallel.

Algorithm 4 explains how the distance metric is calculated. When a party i, $i \in \{0, ..., r\}$, calls this function, since it has limited number of attributes, it calculates the sub-distance based on the projection of its attributes on the cluster mean and entry.

Algorithm 5 describes as to how the parties compute the closest cluster for entry x, $x \in \{0, ..., m\}$. In the first phase of the algorithm, each party creates an array of size k, *MyDistanceVector*, and calculates the sub-distances between cluster c, $c \in \{0, ..., k\}$, and the current entry based on the attributes it owns. In the second phase, all parties other than party-1 and party-2 use additive secret sharing to divide their distance array into two equal-sized arrays, and send these arrays to party-1 and party-2 respectively. Since party-1 and party-2 get the secret shares of the distance data, they don't learn anything about the actual distance values. In the last stage, as described in Algorithm 6, party-1 and party-2 receive the distance data from other parties, add the received data to their own distance array. Then, using the comparison operation described in Section 4.4, they compute the smallest value, i.e. closest cluster index.

Algorithm 3 Privacy preserving k-means clustering

```
Require: $m$ is the number of entries
Require: $k$ is the number of clusters
Require: $r$ is the number of players
 1: for all $i$ from 0 to $r$ in parallel do
 2:   for $c$ from 0 to $k$ by 1 do
 3:     $\mu_{ci} = random$
 4:     $\mu^1_{ci} = 0$
 5:   end for
 6: end for
 7: repeat
 8:   for $x$ from 0 to $m$ by 1 do
 9:     minIndex = SecurelyComputeClosestCluster($x$)
10:     ClusterIndex[$x$] = minIndex
11:     for all $i$ from 0 to $r$ in parallel do
12:       for $c$ from 0 to $k$ by 1 do
13:         Add values of $e_{xi}$ to $\mu^1_{minIndex-i}$
14:       end for
15:     end for
16:   end for
17:   for all $i$ from 0 to $r$ in parallel do
18:     for $c$ from 0 to $k$ by 1 do
19:       Calculate $\mu_{ci}$ based on $\mu^1_{ci}$ and number of entries in cluster $c$
20:     end for
21:   end for
22: until Termination criteria is met
```

Algorithm 4 Calculate Distance 2

```
Require: cluster index $c$ is the first parameter
Require: entry index $x$ is the second parameter
Require: function is called by party $i$
Require: $t$ is the number of attributes owned by party $i$
1: TempDistance = 0
2: for $j$ from 0 to $t$ by 1 do
3:     TempDistance += $||e_{xij} - \mu_{cij}||^2$
4: end for
5: return $\sqrt{TempDistance}$
```

Algorithm 5 Securely Compute Closest Cluster

```
Require: the first parameter $x$ is the index of the entry
Require: $k$ is the number of clusters
Require: $r$ is the number of players
 1: for all $i$ from 0 to $r$ in parallel do
 2:     for $c$ from 0 to $k$ by 1 do
 3:         MyDistanceVector[c] = CalculateDistance($c$, $x$)
 4:     end for
 5: end for
 6: for all $i$ from 2 to $r$ in parallel do
 7:     Secret share MyDistanceVector into arrays D1 and D2
 8:     Send D1 to party-1
 9:     Send D2 to party-2
10: end for
11: for all $i$ from 0 to 2 in parallel do
12:     for $j$ from 2 to $r$ by 1 do
13:         Recieve my part of party $j$'s distance vector, and add it to MyDis-
            tanceVector
14:     end for
15:     minIndex = SecureFindMinimum(MyDistanceVector)
16: end for
17: return minIndex
```

Algorithm 6 Securely Find Minimum Index

```
Require: the first parameter DistVector contains distance data
Require: Party-1 and Party-2 participate, they have separate distance data
Require: k is the number of clusters, and the size of the array DistVector
1: minIndex = 0
2: for $i$ from 1 to k by 1 do
3:     compResult = Compare($DistVector_{minIndex}$, $DistVector_i$)
4:     Open compResult to other party, and add final result to compResult
5:     if compResult = 1 then
6:         minIndex = i
7:     end if
8: end for
9: return minIndex
```

6 Proxy-Secure Computation Model

In Section 3, technical properties and isolation facilities of CBEA are explained. An application can run in an isolated SPE, as explained in Section 3.2 and 3.4,it is turned to a secure application. In order to do so, an application authentication key pair should be created under a scheme, and then be used to sign and encrypt the application binary and data. By this way, a developer will ensure that the integrity and confidentiality of its application and data are preserved during the deployment, and furthermore throughout the execution and thereafter. As described in section 3.4, a secure application can run only in an isolated SPE, meaning that PPE cannot run the application in an open SPE and then obtain any kind of data from that SPE.

With these features in mind, this thesis proposes a proxy-secure computation model, where parties agree to work on a semi-trusted CBEA platform that may be located in a separate location from all the parties. Each contributing party in the model is given an isolated SPE core on the CBEA platform, so the number of parties is limited by the number of SPEs on the platform. Before deploying their applications, each party creates its own application authentication key pair, signs and encrypts its application. Since, data needs to travel over network before being processed, it should be sent by the party and received by the SPE application in an encrypted form. This forces each party to embed an AES key, which is known only to the owner, and used to decrypt sensitive data before being processed and encrypt the results before sending them back. Secure processing vault and hardware root of secrecy ensures that this AES key is not exposed and acquired by any other entity in any stage of the operation.

All parties, including the CBEA platform owner, are semi-honest, in other words they follow the protocol steps and are honest but curious; namely they try to get maximum information about the secrets of other parties if they leak as a result of protocol/implementation failure or any other means. In other words, information leak can occur only if the underlying secure multi-party computation protocol or its implementation is faulty. As described in Section 4, the proposed multi-party computation model and our framework, if they are followed precisely, do not leak any information through the messages sent and received.

The proxy-secure computation model offers to replace high-latency network communication with memory transfer between isolated SPE cores as much as possible. This way, the overhead on the processing time due to network communication will be significantly decreased, which in turn decreases the total processing time. On the other hand, the model does not aim to decrease the number of packets exchanged during the computation. The number of exchanged packets depends on the nature of the data mining application, and the chosen algorithms.

In order to see improvement in the performance that is gained after replacing network communication with DMA operations, both the secure two-party computation model in Section 4 and privacy preserving k-means clustering algorithm in Section 5.2 are implemented. Implementations are made on two separate platforms. The first platform is the PC platform, where no hardware level of security measure is taken, security being relied on software level of encryption and authentication mechanisms and network communication is used for all interactions between the participants. The second platform is a cluster of two Playstation 3 game consoles, consisted of 2 PPEs and 12 SPEs.

In this section, implementation details on PC platform will be explained at first. Details are given about the network simulator that is used for realistic time measurements. Secondly, detailed information will be provided about the implementation on Playstation 3 cluster.

6.1 PC Implementation

In implementations of basic arithmetic operations in the secure two-party computation model, the number of players is fixed to 3; two players and a trusted party. Each party's application is coded and built separately and independently from each other. Since parties are assumed to work on their own environment, they don't have to encrypt the data they will work on.

On the other hand, in the implementation of the clustering algorithm, there is a lower bound for the number of participants imposed by the algorithm design. There can be at least two players due to the need for two non-colluding parties, while there is not an actual upper limit. Four different types of applications were implemented for four different type of participants. The first two types of implementations, named as party-1 and party-2, implement two non-colluding parties in the secure two-party multiplication model. The third type of implementation, party-N, implements all the other participants (i.e. data holders participating in the protocol) except for two non-colluding parties, namely party-1 and party-2. It calculates the sub-distances of an entry from all clusters, secret share the distance data into two parts and send it to the corresponding parties, party-1 and party-2. Therefore, they do not have to implement functionalities of the secure two-party computation model such as computing secure multiplication, comparison etc. All other participants except for party-1and party-2 implements the same functionality. The last application is for the trusted party, which acts as a random number generator under a scheme. TP application could be developed by a trusted party or by a coalition of participants in the system and needs to be trusted.

In order to get the performance results of data mining application on PC platform, a network application simulator (NAS) is used as described in [4]. NAS is a discrete event simulator, in which bandwidth and latency of the network can be adjusted. The simulator reports accurate timing measurements, by concentrating on point-to-point network communication between the applications. On the other hand, NAS is not used to measure performance of secure two-party computation model. For simplicity, all applications are executed on the same machine, with infinite bandwidth and no latency.

6.2 CBEA Implementation

In CBEA implementation, we use the proxy-secure computation model, in which parties agree to run their secure applications on a remote, trusted CBEA platform which is in the possession of a semi-trusted third party. Each party builds its own application, encrypts and signs it before deploying it to the platform. Moreover, data processed by the application is sent to the platform in encrypted form; either off-line or on-line.

As in the case of PC implementation, the number of parties for the implementations of secure operations in the secure two-party computation model is again limited to 3. Party-1, party-2 and third party (TP) develop secure applications. Party-1 and party-2 receive their shares of sensitive data in encrypted form, and performs decryption before starting to process them. Figure 14 illustrates an outline of the implementation. Applications of Alice and Bob (i.e. party-1 and party2, respectively) are placed into isolated SPEs. Moreover, TP application is also a secure one, and it's placed into an empty isolated SPE.

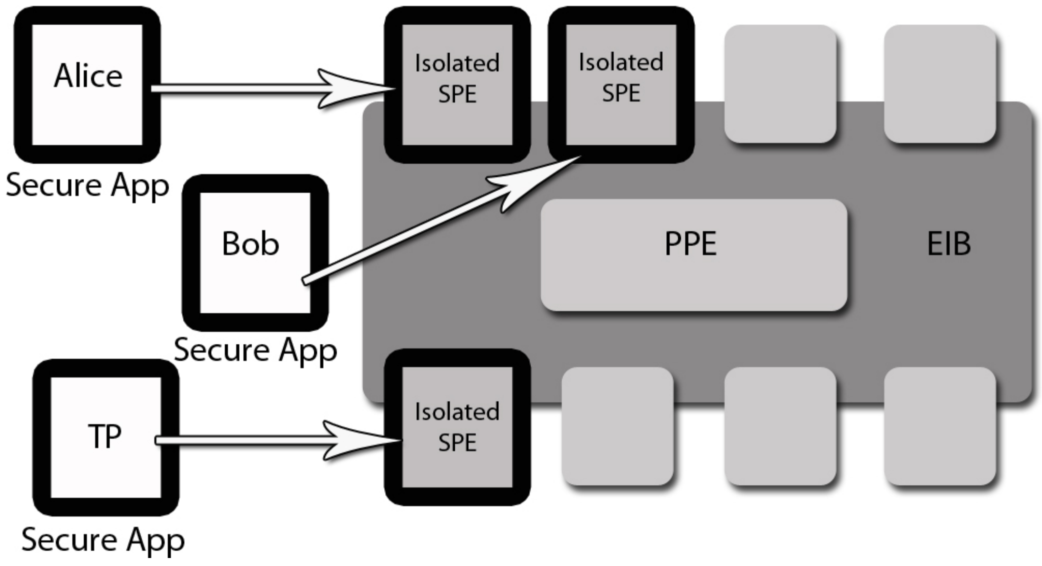


Figure 14: CBEA Implementation: Secure two-party computation model

Similar to the case in PC implementation, the number of parties (besides the TP) should be at least 2 in the clustering algorithm implementation. However, this time, there is an upper limit on the number of maximum players. Since the implementations are developed for a cluster of two Playstation 3, and each Playstation 3 offers 6 SPEs, total number of executing applications can be 12 at most. Trusted party occupies one of these SPEs, which gives an upper limit of 11 players in the clustering algorithm in our implementation. There are again four different type of applications implemented. The first and second applications implement party-1 and party-2, since they are deciding the minimal distance. The third application implements the parties other than party-1 and party-2. The last application is for TP.

Since two cluster nodes are used, and they need to communicate via network, message passing interface (MPI) is used to handle the communication between cluster nodes. SPEs in different Cell processors can not communicate with each other directly. PPEs in two PS3s in the cluster should be synchronized to handle SPE-to-SPE communication in separate machines. Figure 15 illustrates an outline of the implementation scenario. In this example, there are eight (8) players, five (5) on the first PPE, three (3) on the second PPE. In each PS3 console, 2 SPEs cannot be used, since the PS3 game consoles are used, which provide access to only six (6) SPEs in its configuration. All applications are secure applications, meaning that they should work in an isolated SPE.

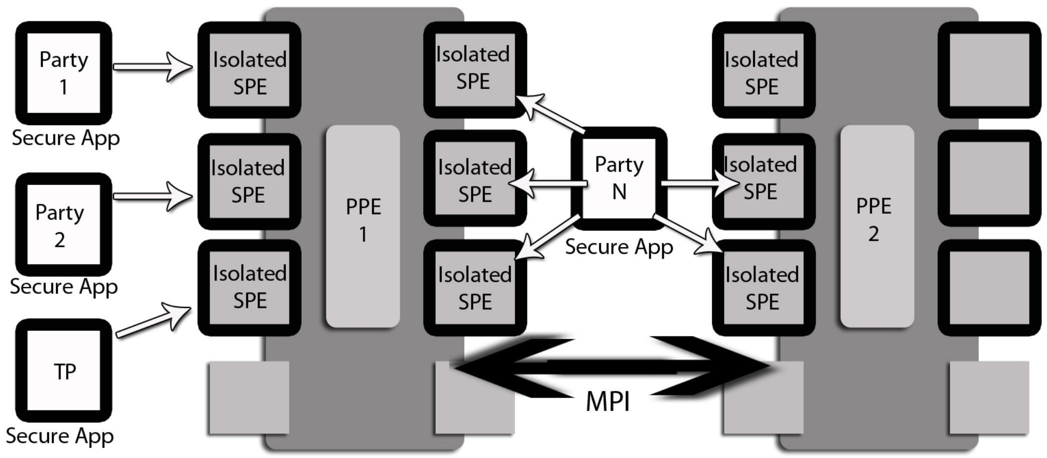


Figure 15: CBEA Implementation: Privacy preserving k-means clustering algorithm

7 Results

As described in Section 6, arithmetic operations in secure multi-party computation model and privacy preserving k-means clustering algorithm were implemented both on PC with a network application simulator (NAS) for desired network latency and bandwidth and on CBEA that offers to implement the proposed proxy-secure computation model. The PC platform that we tested our work on features an Intel Core2 Quad 3 GHz processor with 2 GB of RAM. To simulate the network communication for different latency and bandwidth values, and acquire realistic results, NAS is used as described in Section 6.1.

On the other side, CBEA platform consists of two Playstation 3 game consoles, which are connected to each other via Ethernet and using MPI for inter-core communication across the PS3 consoles. Details about the configuration of a Cell processor are given in Section 3.1. Since the clock frequencies and architectures of two platforms are different, a benchmark program that reflects the computational workload of k-means clustering algorithm is executed on two platforms to check if there is a significant difference in the performances. Calculation of the distance of one entry from all clusters is performed using this program for one million times, in order to benchmark two computing platforms for the same application. When executed on both platforms, program completes in 2.412 s on PC, and in 2.743 s on CBEA. This is natural since a PC is expected to perform better for general-purpose computing tasks than CBEA, whose SPEs are optimized for vector type operations. This results clearly shows that computational advantage of CBEA platform, which is reported in the following, is not due to the (micro-) archi-

Operation	Time (ms)
Multiplication	0.85
Number Comparison	324
Secret Comparison	732

Table 1: Timings for Secure Two-Party Computation Model on PC Platform

tectural superiority of SPEs.

Performance results are measured for both of the algorithms in this thesis. First, results acquired for implementation of the basic arithmetic operations (i.e. multiplication, comparison, etc.) in the computation model are given in the following section. Then, performance outcomes of the distributed k-means clustering algorithm are shown and explained in details.

7.1 Secure Two-Party Computation Model

In order to evaluate the performance of the secure two-party computation model in details, timings are taken for three different operations which are described in Section 4, namely multiplication, comparison of numbers, and comparison of secrets. Since addition requires no interaction in the setting, its timing is not included in the table. Instead, only timings for comparison of numbers, comparison of secrets, and multiplication are included. In all tables, 100 experiments are done, and the average of the best 30 experiments is taken. Moreover, since all applications are executed on the same machine, the latency is zero, and the bandwidth is infinite.

As shown in Table 1, a multiplication operation, which has relatively less communication overhead, takes 0.85 ms on average, while more complicated number comparison takes 324 ms. This is expected, since the comparison contains many number of multiplication operations in itself. Moreover, comparison of secret has a computation time of 732 ms, which is more than two times of a comparison of number operation.

Timing results for the same operations are measured for Cell BE implementation as well: Table 2 lists the timing results which includes the time spent on tasks such as loading threads to SPE and killing it after the computation is completed. Again, the results are average of 30 best of 100 operations. The first column in Table 2 contains the timings when the operands are sent to the SPE one by one, SPE is not in protected mode and data and programs are not encrypted. The second column is serial implementation where the integers are sent to SPEs in groups of 128 pair of integers. The third and fourth columns list the timings of normal and serial implementations, respectively when the SPEs work in isolated mode. The fifth (labeled *Encrypted Serial*) column lists the timings when the data sent to SPEs in serial is encrypted. And finally, the last column gives the timings when the SPEs are in isolated mode and data are encrypted. Note that the SPE threads in isolated mode are encrypted and signed. As one can easily observe from the results in Table 2, the SPE isolation, data and program encryption does not incur a discernible overhead. Serialization of data greatly increase the performance. Secure two-party comparison operations on Cell BE is about two orders of magnitude faster than the same operation in classical setting (*cf.* Table 1 and Table 2).

Table 3 lists the same timing results when the time spent on SPE thread management are excluded. These timings are naturally less than those in

Timing with SPE thread management	Normal (ms)	Serial (ms)	Isolated normal (ms)	Isolated serial (ms)	Encrypted Serial (ms)	Isolated Encrypted Serial (ms)
Multiplication	0.202	0.056	0.480	0.121	0.057	0.124
Comparison of Number	0.301	0.201	0.445	0.258	0.202	0.259
Comparison of Secret	0.789	0.824	0.828	0.886	0.826	0.887

Table 2: Timings for Secure Two-Party Computation Model on CBEA Platform Including SPE Overhead

Table 2. The time spent on SPE thread management becomes less important when larger sets of data is serialized.

7.2 Privacy Preserving K-means Clustering Algorithm

General and privacy preserving k-means clustering algorithms are heavily affected by the initial conditions, i.e. initial cluster centers that are determined randomly. This leads to a variable number of rounds as described in Section 5. In all of our experiments the number of rounds in a clustering algorithm is fixed to six (6).

For the experiments, the same dataset is used, which is a spatio-temporal dataset containing trajectories of private cars in Milan [4]. A subset of this data is actually used, where there are 100 entries, or so called *trajectories*. Each trajectory consists of 500 sample points, and each sample point has one x and one y coordinate. Thus, each trajectory contains 1000 attributes, which

Timing without SPE thread management	Normal (ms)	Serial (ms)	Isolated normal (ms)	Isolated serial (ms)	Encrypted Serial (ms)	Isolated Encrypted Serial (ms)
Multiplication	0.026	0.007	0.254	0.018	0.007	0.017
Comparison of Number	0.091	0.160	0.238	0.173	0.160	0.173
Comparison of Secret	0.495	0.782	0.562	0.801	0.782	0.801

Table 3: Timings for Secure Two-Party Computation Model on CBEA Platform Excluding SPE Overhead

are evenly vertically partitioned among the contributing parties. Therefore, each party in the computation has same or near equal number of sample points. Moreover, results of CBEA implementation includes the time spent for data transfer. Each party puts its data in encrypted form to a web server. Data is downloaded by the PPE using HTTP protocols, and then sent to the corresponding SPE.

Performance of our PC implementation is compared with the results taken from [3] and [4]. These two works also concentrate on privacy preserving k-means clustering over vertically partitioned data.

Figure 16 shows the comparison of PC implementation and CBEA implementation, while the number of clusters is eight (8), and the number of players changes from four (4) to 11. For the PC implementation, bandwidth is 400 kbps, and latency is 6 ms and 3 ms. There is not any change in the number of messages exchanged for the two implementations. The

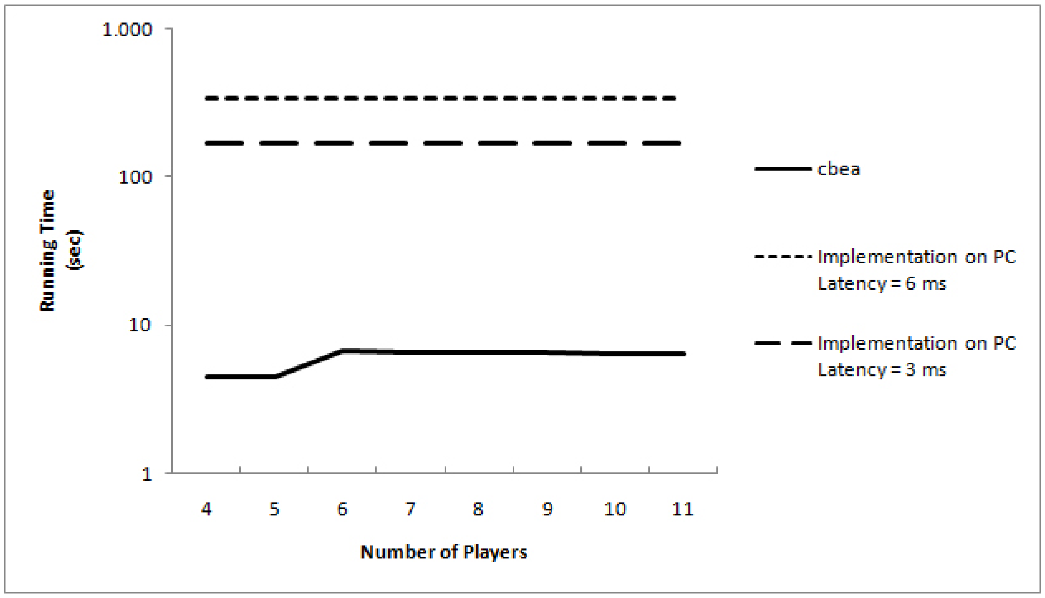


Figure 16: Comparison of CBEA and PC implementation for different number of players

only difference is the replacement of communication via network with memory transfer, due to our proxy-secure computation model. Running time of CBEA implementation is between 4.523 s and 6.39 s, while the running time of PC implementation is around 340.4 s when the latency is 6 ms. Runtime of PC implementation drops down to 171.2 s when the latency is 3 ms. These results show that our proposed model gives nearly 50 times better results compared to implementation on PC.

Figure 17 shows the comparison of PC implementation and CBEA implementation, while the number of players is eight (8), and the number of clusters changes from two (2) to 16. For the PC implementation, bandwidth is 400 kbps, and latency is 6 ms and 3 ms. The results of CBEA implementation changes from 2.434 s to 11.878 s, while the running time PC

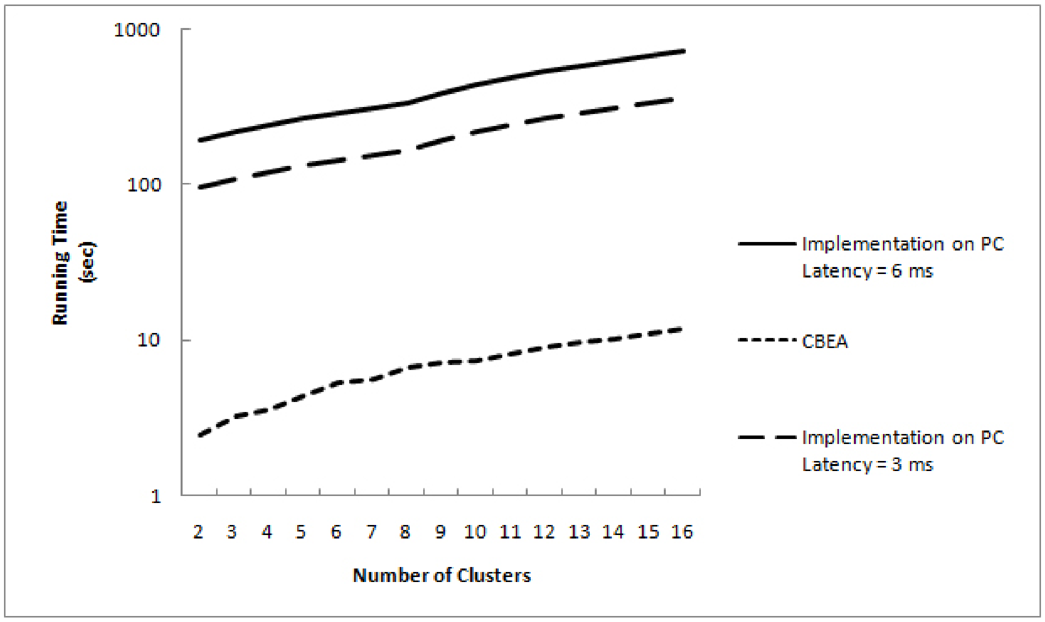


Figure 17: Comparison of CBEA and PC implementation for different number of clusters

implementation changes from 194.70 s to 729.02 s when the latency is 6 ms. If the latency drops to 3 ms, then the PC implementation completes in 98.23 s to 365.10 s. Once again, our model performs much better compared to the same implementation on PC platform. Running time the proxy-secure computation model is nearly 70 times better than the PC implementation.

Figure 18 shows the comparison of three PC performances and one CBEA performance, while the number of clusters is eight (8), the number of players is between four (4) and 11. For PC implementations bandwidth is 400 kbps, and latency is 6 ms. It can be seen that the work of Yildizli et al. performs better than our implementation on PC and the work of Vaidya and Clifton. However, comparing the PC implementations with the CBEA

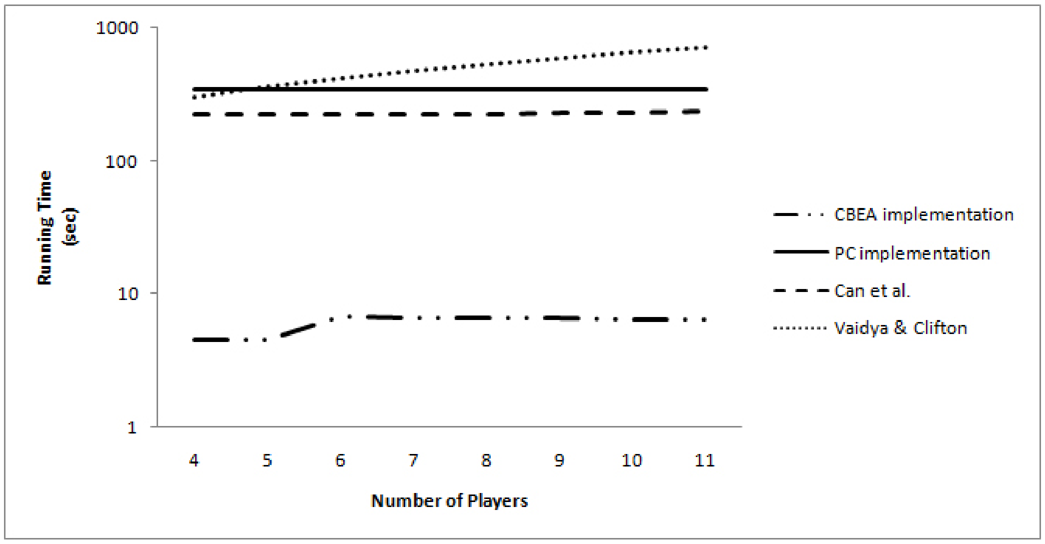


Figure 18: Comparison of PC implementations, k = 8, r = [4-11], bandwidth = 400k bps, latency = 6 ms

implementation shows that our proxy-secure computation model gives much better results. The best PC implementation, work of Yildizli et al., performs the computation between 224.56 s and 236.51 s, while running time of our model changes from 4.52 s to 6.39 s.

Figure 19 shows the comparison of three PC implementation performances and one CBEA performance, while the number of players is eight (8), and the number of clusters varies from two (2) to 16. For PC implementations, bandwidth is 400 kbps, latency is 6 ms. Again work of Yildizli et al. performs better than the other two PC implementations. However, when we compare our CBEA implementation with all three PC implementations, the figure shows that our model gives nearly 60 times better results than the work of Yildizli et al..

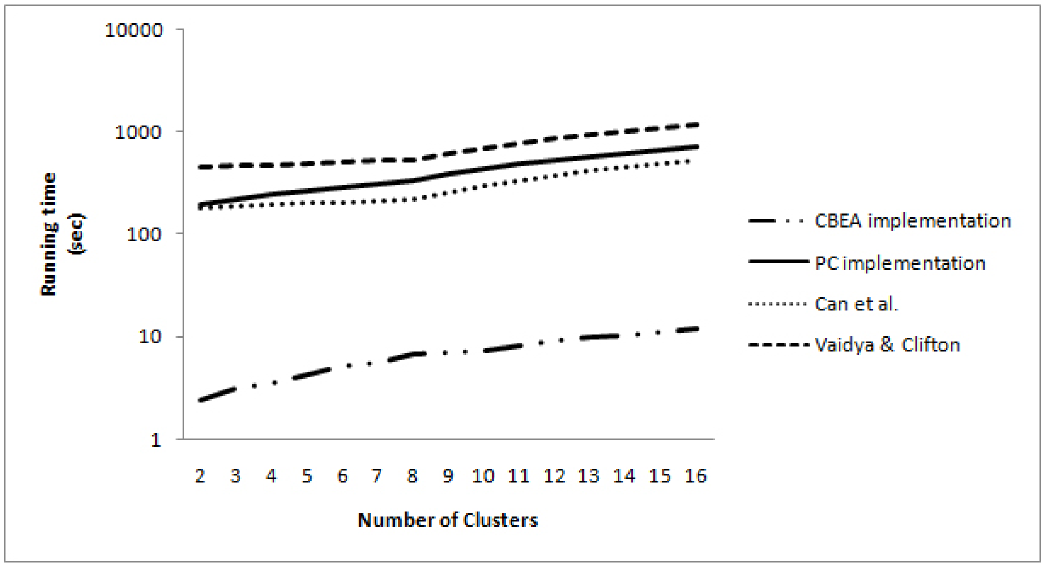


Figure 19: Comparison of PC implementations, k = [2-16], r = 8, bandwidth = 400 kbps, latency = 6 ms

Figure 20 shows the results, where the number of players and clusters is eight (8), bandwidth is 400 kbps and latency varies between 24 and 120. Since changes in latency will not have any effect on the result of CBEA implementation, we fixed CBEA implementation's result to 6.641 s, which is the runtime for eight (8) clusters and eight (8) players. Once again, figure shows that our proxy-secure computation model gives better results compared to PC-based solutions.

Secure two-party computation model and privacy preserving k-means clustering algorithm that are used in this thesis exchange only integers of 32-bits. Since the size of the packages is small, a change in the bandwidth of the network will have minor effects on the runtime. Figure 5 in **??** shows a similar work, where changing bandwidth has minor effects on the runtime.

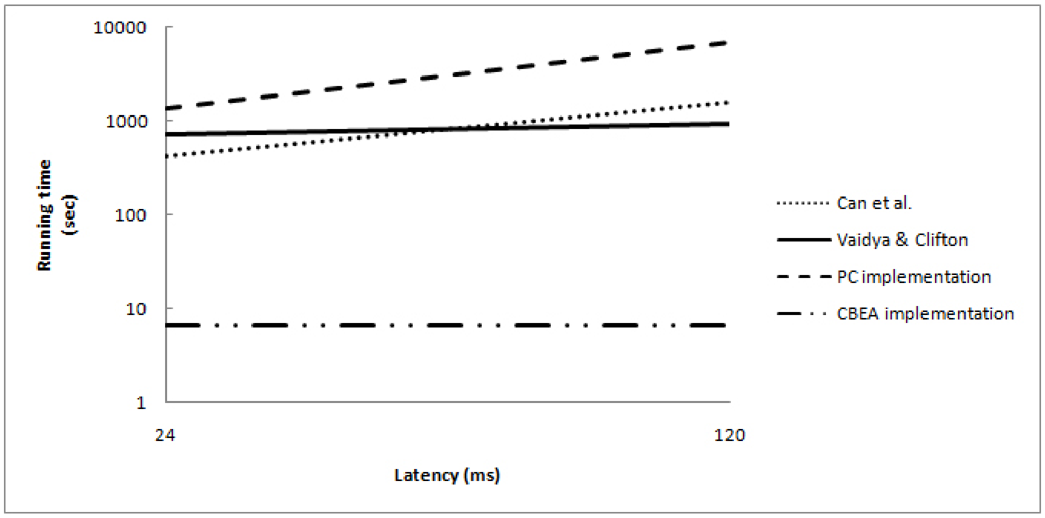


Figure 20: Comparison of PC implementations, k = 8, r = 8, bandwidth = 400kbps, latency = [24-120] ms

All these results show that our implementation is not the best on the PC platform. However, our aim is not to prove that our scheme performs better in all platforms. Our aim is to show that the model we propose, working securely on a CBEA platform, greatly enhances the performance by replacing network communication with direct memory access operations.

Figure 21 shows the implementation results of distributed k-means clustering operation on a cluster of two Playstation 3 game consoles. The total number of players has an upper limit of 11 since we can use total of 11 SPE cores available in our cluster.

Performance for two different cases, where the number of clusters is eight (8) and 16, are shown in Figure 21. For both cases, there are two interesting points to mention. The first point is the decrease in the runtime, when the number of players increase from four (4) to five (5), and from six,(6) to

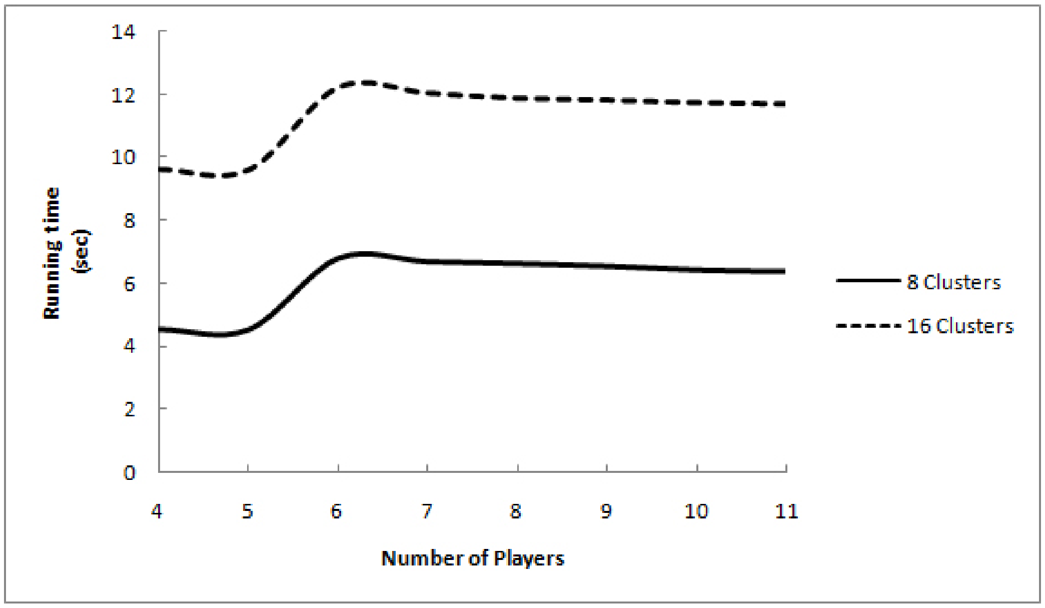


Figure 21: Comparison of CBEA implementation, k = [8-16], r = [4-11]

11. When the number of players is four (4) or five (5), all computations are done on a single machine. Since all SPEs can perform DMA operation simultaneously, increasing the number of players does not increase the runtime; resulting in better utilization of available in EIB. On the other hand, slight decrease can be explained by the fact that data is split into even smaller subsets as the number of players increases. This, naturally, reduces the computational load of each SPE, leading to a slight decrease in the overall computation time.

A similar situation also occurs while the number of players increases from six (6) to 11. Since the communication between the cluster nodes depends on the number of clusters, an increase in the number of players does not affect the communication overhead. The second interesting point is the increase in the runtime, while the number of players increases from five (5) to six (6), which is expected, since communication between the two cluster nodes that happens through Ethernet contributes to the total execution time.

The runtime of CBEA implementation, when the number of clusters and players are eight (8), is 6.641 s, which is far better than the value of 222.49 s of Yildizli et al.'s work with the same configuration and 533.37 s of Vaidya and Clifton's work.

Figure 22 shows the performance of the CBEA implementation, where the number of clusters change from two (2) to 16, and number of players are five (5), eight (8) or 10. Again the case, where the number of players is five (5), gives better results compared to other two cases, due to the fact that all computations are performed on a single machine. When the number of players exceeds five (5), computations expand to two machines and the runtime increases by 2 s approximately.

Once again, CBEA implementation gives far better results than any PC implementation. When the number of clusters is 16, and number of players is eight (8), the runtime is 11.88 s, while implementation of Yildizli et al. terminates in 490 s and implementation by Vaidya and Clifton does in 1100 s.

These results prove that our proxy-secure computational model provides far better results than the existing PC platform solutions. In most of the cases, our work gives at least 20 times improvement in performance compared to the work of Yildizli et al., and Vaidya and Clifton. In addition to increased performances, our model assures that privacy of data and application is preserved throughout the execution. Once the applications and data

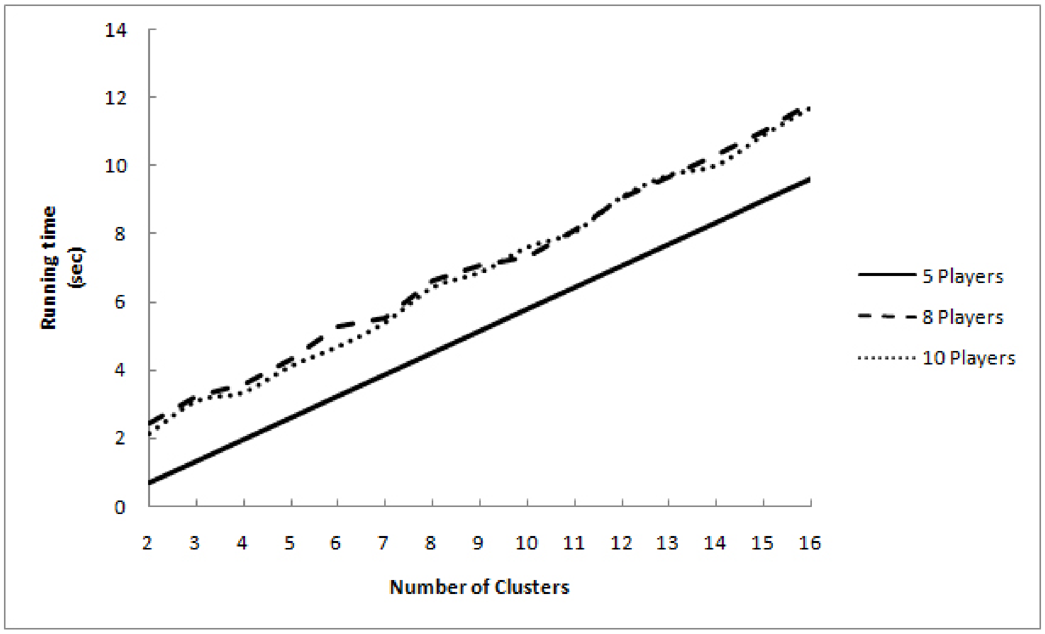


Figure 22: Comparison of CBEA implementation, k = [2-16], r = [5-10]

are sent to the CBEA platform in encrypted form, there is no possibility of data leakage to hostile entities.

8 Conclusion

In this thesis, we propose a generic proxy-secure computation model for secure multi-party computation applications, where the participants want to protect their private data used as input to the computation. In the proposed model, parties can develop, sign and encrypt their application, and send it to a trusted computing platform along with their private data in encrypted form where the private data can be sent off-line as well as on-line.

We selected to use CBEA (Cell Broadband Engine Architecture) by IBM, Sony and Toshiba which is the only platform that provides the required security features such as hardware-based process isolation, runtime secure boot that authenticates programs any time by hardware means before execution, and hardware root of secrecy. Thanks to the security features offered in CBEA, no information can be extracted from the encrypted application and data, and no runtime data is compromised unless intended for sharing. In our security model all the involved parties are assumed to be semi-trusted; they are honest but curious, namely they follow the protocol steps, but eager to exploit leaked information. We assume that CBEA platform is in the possession of a third party, so called supervisor, and even when the supervisor of CBEA platform is compromised, data leakage to outside is prevented.

One other important advantage of our model is that parties develop their applications independently, eliminating the need for collaboration during development process. They only need ensure that their applications can talk to each other and implements a specific interface conforming to the underlying protocol and algorithm.

The computation/application of each party is delegated to a separate SPE core which is isolated and therefore protected. By working on a CBEA platform, applications will communicate using direct memory access operations, which is much faster than network communication, and does not consume the network bandwidth.

In order to demonstrate the usefulness of the proposed model, we selected a distributed data mining application that requires high level of interaction between the data owners that holds partial private data used in the computation. We implemented privacy-preserving distributed k-means clustering algorithm in our model that processes spatio-temporal data obtained from real world. The spatio-temporal data consists of 500 trajectories from Milan taxi cabs, which is vertically partitioned to varying number of data owners/parties. Data owners possess only a portion of every trajectory and they want to learn which cluster each trajectory belongs to after the computation without revealing their parts of trajectories to others.

Privacy-preserving distributed k-means clustering algorithm makes use of our model and secure arithmetic operations. The algorithm necessitates exchange of prohibitively high number of messages, and the total computation time increases to an unacceptable levels if conventional, high-latency network communication is used. Our model alleviates this problem by turning network communication to mostly core-to-core communication within the processor that takes advantage of point-to-point, high-bandwidth, low-latency communication infrastructure.

Our implementation results, both for basic arithmetic operations and for complicated data mining application, show considerable speedup values for each case. Furthermore, our model is especially useful for cloud computing applications where a third party hosts the data and applications of other parties. Our model not only solves the security and privacy problems associated with cloud computing applications but allows secure interaction between different parties that want to perform certain collaborative computation in the same cloud.

References

[1] H. Wang, H. Takizawa, and H. Kobayashi, "A performance study of secure data mining on the cell processor," *Eighth IEEE international symposium on cluster computing and the grid*, 2008. `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04534275`.

[2] K. Shimizu, H. P. Hoftsee, and J. S. Liberty, "Cell broadband engine processor vault security architecture," *IBM Journal of Research and Development*, vol. 51, 2007.

[3] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," *In KDD 2003: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 206–215, 2003.

[4] C. Yıldızlı, T. B. Pedersen, Y. Saygın, E. Savaş, and A. Levi, "Distributed privacy preserving clustering via homomorphic secret sharing and its application to (vertically) partitioned spatio-temporal data," Accepted.

[5] W. Du and M. J. Atallah, "Secure multy-party computation problems and their applications: a review and open problems," *In proceedings of the 2001 workshop on New security paradigms*, pp. 13–22, 2001.

[6] I. Damgard and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," *CRYPTO 2007*, pp. 572–590.

[7] R. L. Rivest, “Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initialize,” August 1999. `http://people.csail.mit.edu/rivest/Rivest-commitment.pdf`.

[8] I. F. Blake and V. Kolesnikov, “Strong conditional oblivious transfer and computing on intervals,” *Lecture Notes in Computer Science*, vol. 3329, pp. 515–529, 2004.

[9] I. Damgard, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, “Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation,” *LNCS*, vol. 3876, pp. 285–304, 2006.

[10] K. Ohta, “Multiparty computation for interval, equality, and comparison without bit-decomposition protocol,” *In proceedings of the 10th international conference on Practice and theory in public-key cryptography*, pp. 343–360, 2007.

[11] Y. Lindell and B. Pinkas, “Secure multiparty computation for privacy-preserving data mining,” *The journal of Privacy and Confidentiality*, vol. 1, pp. 59–98, 2009.

[12] M. Kantarcioglu and C. Clifton, “Privacy-preserving distributed mining of association rules on horizontally partitioned data,” *IEEE Trans. Knowl. Data Eng.*, vol. 16(9), pp. 1026–1037, 2004.

[13] R. Wright and Z. Yang, “Privacy-preserving bayesian network structure computation on distributed heterogeneous data,” *In Kdd 2004: Pro-*

ceedings of the tenth acm sigkdd international conference on knowledge discovery and data mining, pp. 713–718, 2004.

[14] M. C. Doganay, T. B. Pedersen, Y. Saygin, E. Savas, and A. Levi, "Distributed privacy preserving k-means clustering with additive secret sharing," *In Pais 2008: Proceedings of the 2008 international workshop on privacy and anonymity in information society*, pp. 3–11, 2008.

[15] A. K. Nanda, J. R. Moulic, R. E. Hanson, G. Goldrian, M. N. Day, B. D. D'Amora, and S. Kesavarapu, "Cell/b.e. blades: Building blocks for scalable, real-time, interactive, and digital media servers," *IBM Journal of Research and Development*, vol. 51, 2007.

[16] H. P. Hofstee, "Power efficient processor architecture and the cell processor," *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.5650&rep=rep1&type=pdf`.

[17] K. Shimizu, D. Brokenshire, and M. Peyravian, "Cell broadband engine support for privacy, security, and digital rights management applications," 2005. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.2001&rep=rep1&type=pdf`.

[18] N. Costigan and M. Scott, "Accelerating ssl using the vector processors in ibm's cell broadband engine for sony's playstation 3," `http://eprint.iacr.org/2007/061.pdf`.

[19] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, 2005.

[20] *Security Software Development Kit 3.0: Installation and User's Guide*, 2007. `https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/AEBFE7D58B5C36E90025737200624B33/$file/CBE_Secure_SDK_Guide_v3.0.pdf`.

[21] C. Crépeau, G. Savvides, C. Schaffner, and J. Wullschleger, "Information-theoretic conditions for two-party secure function evaluation," *Advances in Cryptology*, vol. 4004, 2006.

[22] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, May 2004.

[23] J. C. Benaloh, "Secret sharing homomorphisms: Keeping shares of a secret (extended abstract)," *Advances in Cryptology*, vol. 263, pp. 251–260, 1987.

[24] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22(11), pp. 612–613, November 1979.

[25] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, pp. 451–461, 2003.

[26] K. Alsabti, S. Ranka, and V. Singh, "An efficient k-means clustering algorithm," `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.44.518&rep=rep1&type=pdf`.