



SAPIENZA
UNIVERSITÀ DI ROMA

SCHOOL OF ENGINEERING
DEPARTMENT OF COMPUTER AND SYSTEM SCIENCES “ANTONIO
RUBERTI”

MASTER THESIS IN COMPUTER SCIENCE
ACADEMIC YEAR 2010/2011

“Collaborative Environment for Cyber Attacks Detection:
The Cost of Preserving-Privacy”

Pasquale Fimiani

SUPERVISOR: Prof. Roberto Baldoni
FIRST MEMBER: Dr.ssa Giorgia Lodi

to my parents and my friends

Contents

1	Introduction	1
1.1	Main Problem	1
1.2	Motivation	2
1.3	Thesis Outline	3
2	Related Work	4
2.0.1	Collaborative Enviroment under Trusted Third Party	4
2.0.2	Secure Multi-Party Computation	5
2.0.3	Privacy Preserving Data Mining	7
2.0.4	Airavat: Security and Privacy for MapReduce	7
2.0.5	Sharemind	8
2.0.6	SEPIA: Privacy-preserving Aggregation of Multi-Domain Network Events and Statistics	9
3	Background	11
3.1	River, Shamir and Adleman (RSA) public-key cryptography	11
3.2	MapReduce	12
3.3	Hadoop	13
3.3.1	Hadoop Distributed File System (HDFS)	15
	NameNode	16
	DataNodes	16
	HDFS Client	18
3.4	HIVE - Data Warehouse Using Hadoop	19
3.4.1	Data Model and Type System	19
3.4.2	Query Language	20
3.4.3	Data Storage	21
3.4.4	System Architecture and Components	22
3.4.5	Thrift Server	24
3.5	Esper - Complex Event Processing	24

3.5.1	CEP and relation databases	25
3.5.2	Esper engine	26
3.5.3	EQL	27
4	A privacy preserving architecture for collaborative processing	29
4.1	System Model	29
4.2	1st Architecture - Proxy Based (Data Spread)	31
4.2.1	How it works	31
4.2.2	Architecture Analysis - Pros vs Cons	32
4.3	2nd Architecture - Matrix Proxy Based	32
4.3.1	How it works	33
4.3.2	Architecture Analysis - Pros vs Cons	33
4.4	3rd Architecture - Asymmetric Key Proxy Based	34
4.4.1	How it works	34
4.4.2	Architecture Analysis - Pros vs Cons	35
5	A Case of Study: Man-In-The-Browser (MITB)	37
5.1	Cyber Attacks in Financial Scenario	37
5.1.1	An introduction to Man-In-The-Browser (MITB) Attacks	38
	Evolution of Man-in-the-Browser	39
5.1.2	How it works: Points and Method of attack	39
5.1.3	MITB: a detailed example	41
5.1.4	Nowadays: Mitigation Strategies	45
5.1.5	A new approach: Transaction Monitoring	48
6	Implementations and Performance Evaluation	49
6.1	Short Analysis of Log Files	49
6.2	System accuracy - evaluation parameters	51
6.3	First Approach - Hadoop/Hive	51
6.3.1	Scenario 1: four participants - daily logs	54
6.3.2	Scenario 2-3: four participants - monthly/3months logs	55
6.3.3	System Performances with or without Privacy	56
6.4	Second Approach - From Hadoop/Hive to ESPER	59
7	Discussion	65
A	RSA Number Theory	67
A.0.1	Proof of the Main Result	68

B Two-factor Authentication	69
C Advanced Standard Encryption	71
C.0.2 Attacks on AES	72
D Configure virtual machines with Hadoop and Hive	73
D.1 Running Hadoop on Single Node Cluster	73
Prerequisites	73
conf/*-site.xml	75
D.2 Running Hadoop on Multi Node Cluster	76
D.2.1 Networking	76
E Java Code	81
List of Figures	107
List of Tables	108

Chapter 1

Introduction

Human communications have been revolutionized by the advent of the Internet. Public and private service providers are increasingly transferring their services online for improved availability and convenience. Low startup barriers, high consumer privacy and global outreach of this open network have made it possible to run completely online businesses and services. Unfortunately, these benefits have attracted all kinds of adversaries. The number of cyber crimes are increasing at an alarming rate. According to MacAfee, “corporations around the world face millions of cyber-attacks everyday [15]”. Collaborative computing is now the trend to face this problem in most industries particularly in banking, finance, insurance, and healthcare. Most of the collaborative tasks are performed with internal or external parties. When two or more collaborators within a collaborative framework want to jointly perform a collaborative task, they need to share their private data with their counterparts.

1.1 Main Problem

Privacy is frequently discussed during the development of collaborative systems under distributed environment. The balance between privacy protection and data sharing among collaborators is now becoming crucial. A privacy preserved system normally can ensure that it has a series of secure mechanisms for privacy protection. For distributed environments, much work has been focused on designing specific information sharing protocols [26]. There are two important considerations to be taken before any collaborative task can be performed. If privacy is not a major concern, each collaborator can send their private data to a Trusted Third Party (TTP). The TTP functions as the central repository or data warehouse to perform the collaborative tasks. This is an ideal approach to support most collaborative tasks if the data being shared is not sensitive information (e.g., sharing of project member’s name, email address, contact numbers, and etc). If privacy is a concern, none of the collaborators should reveal their private data (in clear) to any party including the TTP (e.g., sales anal-

ysis, product costs, stocks, transaction's informations). It is clear that collaborative tasks are easy and straightforward if the privacy protection for those shared data is not taken into consideration. However, in most real life cases, the privacy concern cannot be ignored. The disclosure of private data could have a serious and negative impact on the data owner. When datasets are distributed on different sites, there is a need to find a balance between security and privacy protection. Database operations such as union or intersection computation, equijoin, and aggregation are important operations that can be used to support the secure data sharing process. For example, intersection computation is used to find the common value for different distributed datasets while revealing only the intersection [22].

1.2 Motivation

For these reasons, it has been decided in the context of this thesis to design a privacy preserving architecture for collaborative processing, with the main goal to investigate the cost of preserving privacy. The designed architecture, uses third parties and cryptographic algorithm to allow the participants to correlate keeping them private unless certain condition are verified. Our architecture is based on log analysis, these logs must be provided by system's participants. In absence of real data (we have simulated bank's logs) we have developed a Java software that produce log files.

With the aim to verify the goodness of the proposed solution a Man-in-the-Browser(MITB) attack has been used as a case of study. Man-in-the-Browser has been chosen because it has an high spread and a lack of adequate defense and detection tools. For the implementation and performance evaluation we have proposed two different approach.

In the first approach it has been used the Apache **Hadoop** framework combined with Apache **Hive** data warehouse software. It has been created an Apache Hadoop cluster and logs have been loaded on Hadoop Distributed File System (HDFS). HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable computations. Data on HDFS have been queried through HIVE software.

In the second approach we have replaced Hadoop/Hive with **ESPER** engine. ESPER is a component for complex event processing (CEP), available for Java. Our goal is to achieve a near real-time monitoring by keeping the good level of detection accuracy and error level reached with the previous solution.

1.3 Thesis Outline

Rest of this thesis is organized as follows.

Chapter two This chapter describes several approaches and protocols that have been proposed for distributed collaborative environments with privacy preserving mechanisms.

Chapter three This chapter describes the technological and theoretical background useful to understand our work. We present in detail Map Reduce, Hadoop, Hive, RSA algorithm and ESPER engine.

Chapter four This chapter describes in detail the architecture developed. We present all the steps performed to reach the final architecture that we used during the testing phase.

Chapter five This chapter describes in detail the use case chosen: Man-in-the-Browser. We analyze how this attack works, we present a detailed example and some possible mitigation strategies. We introduce what we means for *transaction monitoring*, an idea used by our architecture.

Chapter six This chapter describes how we have developed our architecture with different technologies. We present, also, the result of simulation performed.

Chapter seven This chapter analyze result obtain with simulation performed, we analyze pros and cons of our solution and we discuss some possible future works.

Appendix Appendix go into details of some arguments like: River-Shamir-Adleman(RSA) number theory, Two-Factor Authentication, Advanced Standard Encryption (AES) and a very short tutorial to understand how configure virtual machines with Hadoop and Hive. Appendix could help to understand better some elements of this thesis.

Chapter 2

Related Work

When data is distributed in multiple locations, the access of private data by collaborators cannot be avoided. With privacy concerns, this data must be protected during the collaborative tasks. Whenever private data is being used, there will be a potential risk for privacy breaches. Several approaches and protocols have been proposed for data privacy and security protection. The use of collaborative environment for protection of critical infrastructures was investigated in [28]

2.0.1 Collaborative Environment under Trusted Third Party

One of the direct implementations for multi-party computation is the use of a trusted third party (TTP). Under this approach, all collaborators are connected to a central site. With the existence of a Trusted Third Party, multi-party computation can be performed easily. Each collaborator reveals its private data to the TTP and allows the central site to compute the final result. This final result can be then broadcasted to all collaborators. As showed in

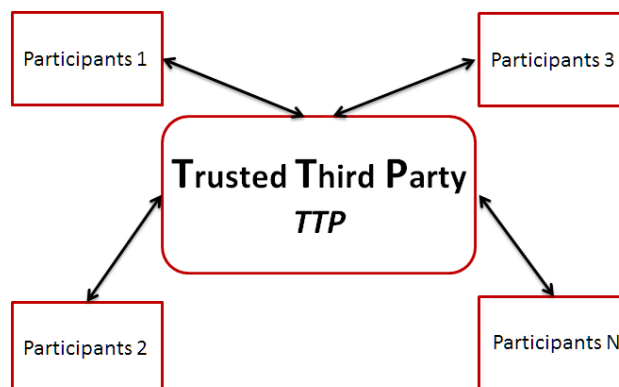


Figure 2.1: Trusted Third Party (TTP) Model

figure 2.1, the Trusted Third Party serves as central repository that will perform most of the

operations. The construction of a central repository enables collaborators to share their data with other collaborators. However, this approach is viewed as insecure for collaborative works because the level of trust between collaborators and the trusted party is always unacceptable. The *TTP model* has been criticized to be too risky because it is always a single point target for malicious adversaries. Trusted Third Party is an extremely valuable target for malicious parties because it holds most of the private data from the collaborators.

2.0.2 Secure Multi-Party Computation

Secure Multi-party Computation (SMC) is an important method used to protect shared data. The concept of multi-party computation is to enable several parties jointly contribute their private data as the inputs for a specific computation function¹. Only the final result is revealed to all parties. Secure two-party computation was first studied by Andrew C. Yao in 1982, in terms of the millionaire's problem [42]. According to the millionaire's problem, Alice and Bob are two millionaires, who are trying to find out who is richer, without revealing information about their wealth. Yao proposed a two party protocol, that solves the problem with the given constraints. The solution of the millionaire problem lead to a generalization to multi-party protocols. Since then, many multi-party computation protocols have been proposed. As proved by Goldreich et al. in [23], there exists a secure solution for any functionality which can be represented as a combinatorial circuit. However, the circuit evaluation is somewhat inefficient for a large number of parties because the cost for large inputs can be very high.

The main objective of SMC is to perform secure computations in the absence of a Trusted Third Party. However, without a TTP, current generic multi-party computations cannot support concurrent operations. Most of the approaches required collaborators to perform their operations in a sequential way. No parallel operations can be executed because each collaborator needs to wait for their direct neighbor to send them the data before they can perform their roles (e.g., data permutation, encryption, and etc). In the absence of a TTP, a secure multi-party computation model is shown in Fig 2.2. Assuming that there are 4 collaborators: Participants 1 (P1), Participants 2 (P2), Participants 3 (P3) and Participants 4 (P4), who would like to jointly compute for a specific function. P1 is the party who initiates the computation protocol. Here are some drawbacks in the general multi-party computation model:

It does not support concurrent operations No concurrent operations can be performed by all collaborators. For instance, P3 needs to wait for P1 (initiator) before it can start

¹The computation functions can be a scalar product of two-party, set union of multi-party, and other set operations.

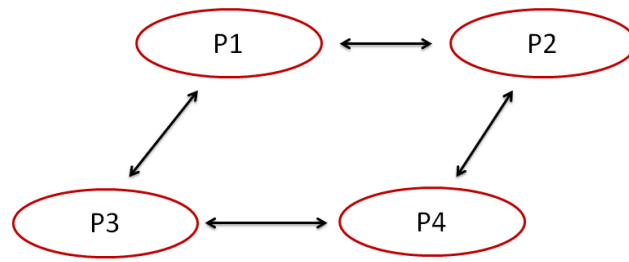


Figure 2.2: Secure Multi-Party Computation Model

the computation. At the same time, P2 must wait for P1 and P3 before any operation can be performed.

Possible single point failure problem During the execution of the protocol, failure of any collaborator can cause the termination of the overall protocol. When collaborators are relying on each other, the computation can be terminated if one of them fails to perform their tasks. If the computation is aborted, the protocol needs to be restarted from the beginning.

Possible collusions among collaborators Since each collaborator has at least two direct connected neighbors, any two participants can collude together to find out the secret data of their common neighbor.

Two-party computation problem If only two collaborators are involved in the computation, either party can derive the private data of their counterpart based on his own input and the final output.

Chance to act maliciously In the existing design, all collaborators are connected in a ring. When the final output is broadcast to all collaborators, there exists a chance for any party to act maliciously by adding noise or alter the original result before sending it to the next party.

Scalability and flexibility problem Once the protocol is started, disconnection of any participant will terminate the computation protocol. The arrangement of all participants is another issue that can limit the flexibility of the current SMC approach.

High execution costs When the size of the accumulated data is large, the communication and computation costs will increase.

2.0.3 Privacy Preserving Data Mining

Privacy Preserving Data Mining (PPDM) can be considered as an extension of the existing data mining process. Besides the knowledge extractions from a large volume of data, PPDM also ensures that all sensitive information is being protected. There are several approaches that have been proposed to solve the PPDM problems. In 2000, Agrawal and Srikant proposed the first PPDM approach based on the data randomization and perturbation technique [3]. Since then, there have been many discussions and researches based on this technique [37, 2]. As mentioned in [13], this technique is somewhat efficient, but there exists a tradeoff between data accuracy and privacy protection. When the data is strongly protected, the accuracy of the mining results cannot be accurate [1]. Cryptography based techniques, such as Secure Multi-party Computation (SMC) which was introduced in [41], have been widely studied in PPDM. Under this approach, a group of collaborators want to compute a function with their private inputs while keeping their inputs private. Only the final result to the query will be learned by the data miner, and no extra information should be revealed [23, 12]. Lindell and Pinkas [17] examine the SMC problem based on cryptography techniques.

2.0.4 Airavat: Security and Privacy for MapReduce

Airavat [24], is a a system developed at University of Texas at Austin for distributed computations which provides end-to-end confidentiality, integrity, and privacy guarantees using a combination of mandatory access control and differential privacy. Airavat is based on the popular MapReduce(see 3.2) framework. Airavat enables the execution of trusted and untrusted MapReduce computations on sensitive data, while assuring comprehensive enforcement of data providers' privacy policies. To prevent information leaks through system resources, Airavat runs on SELinux [20] and adds SELinux-like mandatory access control to the MapReduce distributed file system. To prevent leaks through the output of the computation, Airavat enforces differential privacy using modifications to the Java Virtual Machine and the MapReduce framework. Airavat cannot confine every computation performed by untrusted code. For example, a MapReduce computation may output key/value pairs. Keys are text strings that provide a storage channel for malicious mappers. In general, Airavat cannot guarantee privacy for computations which output keys produced by untrusted mappers. MapReduce computations that necessarily output keys **require trusted mappers**.

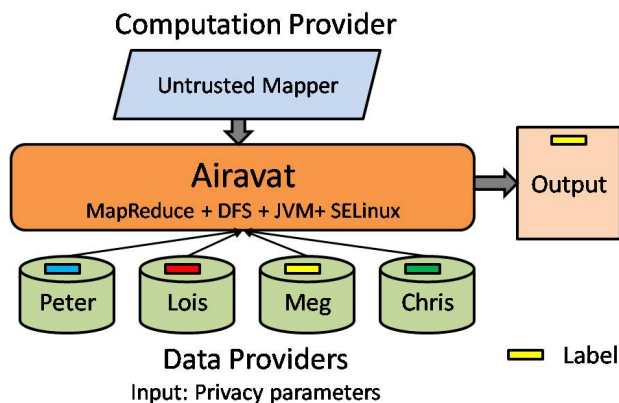


Figure 2.3: Airavat: Security and Privacy for MapReduce

2.0.5 Sharemind

Sharemind is a secure multi-party computation framework designed with a strong focus towards speed, scalability and ease of application development. Sharemind uses secret sharing to split confidential information among several computing nodes denoted as miners. The data donors do not need to trust the miners provided that the number of corrupted miners colluding with each other is always less than a prescribed threshold, t , where the number of computing parties, $n > 3t$. The framework achieves provable security in semi honest model for information theoretically secure communication channels. In practice, models with three to five miner nodes in semi honest setting are common, where three miner models are comparatively communication-efficient. As Sharemind is strongly concentrated in improving processing efficiency in terms of speed, current implementation of the framework consists of three miner nodes. Sharemind uses 32 bit integers as input to achieve efficiency in local computation time. Since Shamir secret sharing scheme does not work over 32 bit integers, an additive secret sharing scheme is used by the framework. Figure 2.4 depicts the Sharemind framework. As said earlier, the framework consists of three miners. All secret values provided by the data donors are shared to the miners by the additive secret sharing. The framework consists of three computing parties defined as miners. The miners stores the shares of secret values in a secure storage. If the input values are not confidential then they are replicated to each miners in a publically accessible storage. Sharemind implements its MPC functionality using constant round share computing protocols. The share computing functions provided by the framework are addition, multiplication and comparison operations. Several privacy preserving application have been designed using Sharemind. The Sharemind framework relies on secure multi-party computation, but it also introduces several new ideas for improving the efficiency of both the applications and their development process. However, the current implementation has several restrictions, most notably it can

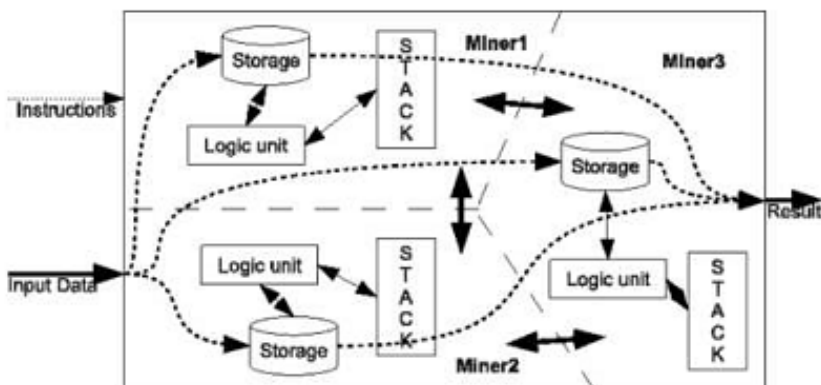


Figure 2.4: Deployment diagram of Sharemind

use only three computing parties and can deal with just one semi-honest adversary.

2.0.6 SEPIA: Privacy-preserving Aggregation of Multi-Domain Network Events and Statistics

SEPIA [31] is a library for efficiently aggregating multi-domain network data using Multi-Party Computation. The foundation of SEPIA is a set of optimized MPC operations, implemented with performance of parallel execution in mind. A typical setup for SEPIA is depicted in Figure 2.5 where individual networks are represented by one input peer each. The input peers distribute shares of secret input data among a (usually smaller) set of pri-

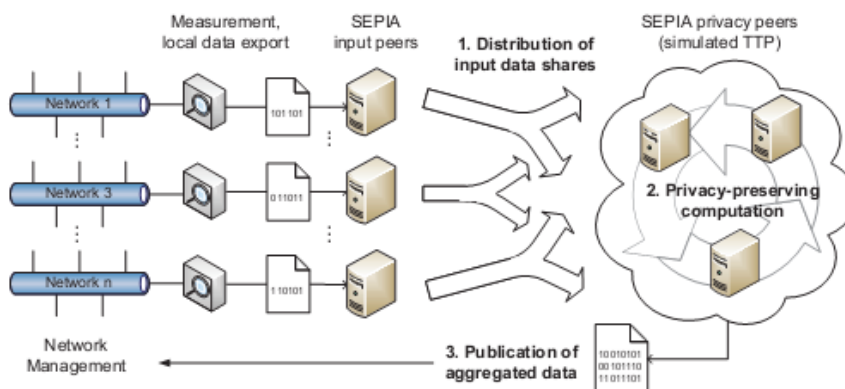


Figure 2.5: Deployment scenario for SEPIA

vacy peers using Shamir’s secret sharing scheme [32]. The privacy peers perform the actual computation and can be hosted by a subset of the networks running input peers but also by external parties. Finally, the aggregate computation result is sent back to the networks.

It has been adopted the semi-honest adversary model, hence privacy of local input data is guaranteed as long as the majority of privacy peers is honest. Although the number of privacy peers m has a quadratic impact on the total communication and computation costs, there are also m privacy peers sharing the load. That is, if the network capacity is sufficient, the overall running time of the protocols will scale linearly with m rather than quadratically. On the other hand, the number of tolerated colluding privacy peers also scales linearly with m . Hence, the choice of m involves a privacy-performance tradeoff. The separation of roles into input and privacy peers allows to tune this tradeoff independently of the number of input providers. SEPIA doesn't work well with large set of privacy peers, due to Shamir's secret sharing scheme.

Chapter 3

Background

This chapter describes the technological and theoretical background useful to understand our work.

We present in detail Map Reduce, Hadoop, Hive, RSA algorithm and ESPER engine.

3.1 River, Shamir and Adleman (RSA) public-key cryptography

RSA (which stands for Rivest, Shamir and Adleman who first publicly described it) is an algorithm for public-key cryptography [27] It is the first algorithm known to be suitable for signing as well as encryption, and was one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be sufficiently secure given sufficiently long keys and the use of up-to-date implementations.

It is very simply to multiply numbers together, especially with computers. But it can be very difficult to factor numbers. The square-root of a number with 400 digits is a number with 200 digits. The lifetime of the universe is approximately 10^{18} seconds - an 18 digit number. Assuming a computer could test one million factorizations per second, in the lifetime of the universe it could check 10^{18} possibilities. But for a 400 digit product, there are 10^{200} possibilities. This means the computer would have to run for 10^{176} times the life of the universe to factor the large number. It is, however, not too hard to check if a number is prime. If it is not prime, it is difficult to factor, but if it is prime, it is not hard to show it is prime.

So, RSA works like follow:

Find two huge prime numbers, p and q . Keep those two numbers secret (they are the private key), and multiply them together to make a number $N = pq$. That number N is basically the public key. It is relatively easy to get N , it is just a multiplication of p and q , but also

if N is known, it is basically impossible to find p and q . To get them, is needed to factor N , which seems to be an incredibly difficult problem. But exactly how is N used to encode a message, and how are p and q used to decode it? Below is presented an example. In a real RSA encryption system, the prime numbers are huge.

In the following example, suppose that person A wants to make a public key, and that person B wants to use that key to send a message to A. Suppose that the message B sends to A is just a number. We assume that A and B have agreed on a method to encode text as numbers. Here are the steps:

1. Person A selects two prime numbers. We will use $p = 23$ and $q = 41$ for this example.
2. Person A multiplies p and q together to get $pq = (23)(41) = 943$. 943 is the “public key”, which tells to person B.
3. Person A also chooses another number e which must be relatively prime to $(p - 1)(q - 1)$. In this case, $(p - 1)(q - 1) = (22)(40) = 880$, so $e = 7$ is fine. e is also part of the public key, so B also is told the value of e .
4. Now B knows enough to encode a message to A. Suppose, for this example, that the message is the number $M = 35$.
5. B calculates the value of $C = M^e \pmod{N} = 35^7 \pmod{943}$.
6. $35^7 = 64339296875$ and $64339296875 \pmod{943} = 545$. The number 545 is the encoding that B sends to A.
7. Now A wants to decode 545. To do so, he needs to find a number d such that $ed = 1 \pmod{(p - 1)(q - 1)}$, or in this case, such that $7d = 1 \pmod{880}$. A solution is $d = 503$, since $7 * 503 = 3521 = 4(880) + 1 = 1 \pmod{880}$.
8. To find the decoding, A must calculate $C^d \pmod{N} = 543^{503} \pmod{943}$.

In appendix A the number theory beyond RSA algorithm has been described, with proof of correctness and main fundamental theorems.

3.2 MapReduce

MapReduce [10] is a programming framework popularized by Google and used to simplify data processing across massive data sets. With MapReduce, computational processing can occur on data stored either in a filesystem (unstructured) or within a database (structured). There are two fundamental pieces of a MapReduce query:

Map The master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes that smaller problem, and passes the answer back to its master node.

Reduce The master node then takes the answers to all the sub-problems and combines them in a way to get the output - the answer to the problem it was originally trying to solve.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The runtime system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. The user of the MapReduce library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows to handle lists of values that are too large to fit in memory. Figure 3.1 shows an execution overview of MapReduce.

3.3 Hadoop

Today, we're surrounded by data. People upload videos, take pictures on their cell phones, text friends, update their Facebook status, leave comments around the web, click on ads, and so forth. Machines, too, are generating and keeping more and more data. The exponential growth of data first presented challenges to cutting-edge businesses such as Google, Yahoo, Amazon, and Microsoft. They needed to go through terabytes and petabytes of data to figure out which websites were popular, what books were in demand, and what kinds of ads appealed to people. Existing tools were becoming inadequate to process such large data sets. Google was the first to publicize *MapReduce*, a system they had used to scale their data processing needs. This system aroused a lot of interest because many other businesses were facing similar scaling challenges, and it wasn't feasible for everyone to reinvent their own proprietary tool. Doug Cutting¹ saw an opportunity and led the charge to develop an

¹Douglas Read Cutting is an advocate and creator of open-source search technology. He originated Lucene and, with Mike Cafarella, Nutch, both open-source search technology projects which are now managed through the Apache Software Foundation.

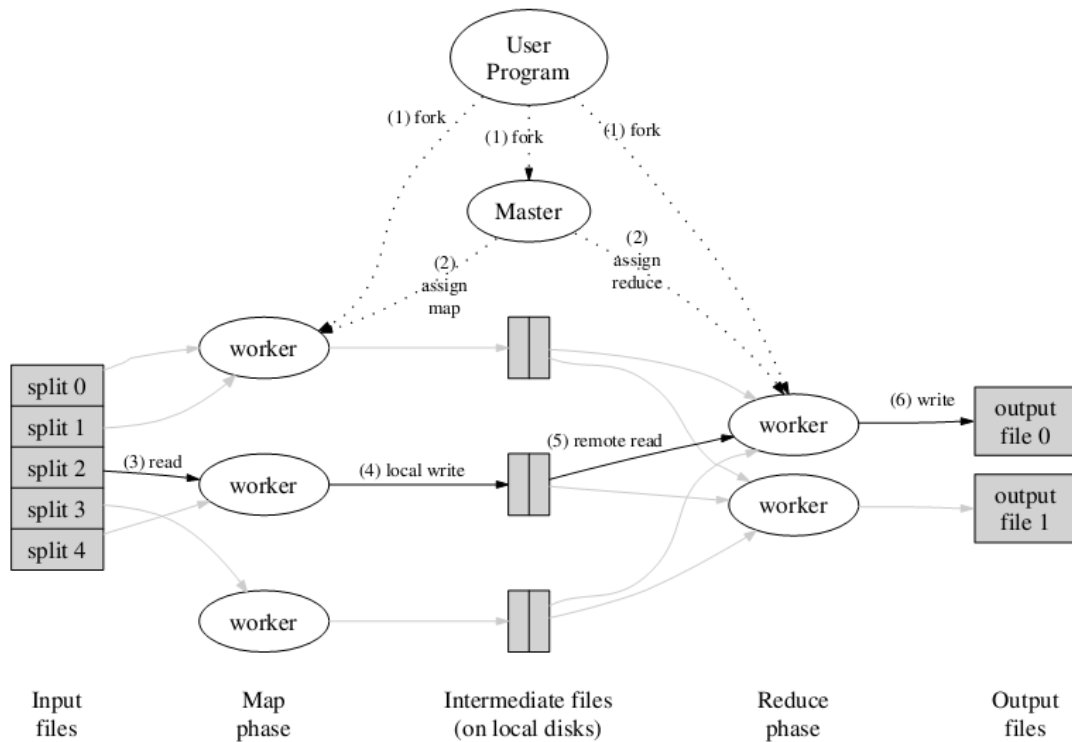


Figure 3.1: Map-Reduce Execution Overview

open source version of this *MapReduce* system called **Hadoop**, Yahoo and others rallied around to support this effort. Today, Hadoop is a core part of the computing infrastructure for many web companies, such as Yahoo, Facebook, LinkedIn, and Twitter . Many more traditional businesses, such as media and telecom, are beginning to adopt this system too. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. The key distinctions of Hadoop are that it is

Accessible Hadoop runs on large clusters of commodity machines or on cloud computing services.

Robust Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.

Scalable Hadoop scales linearly to handle larger data by adding more nodes to the cluster.

Simple Hadoop allows users to quickly write efficient parallel code.

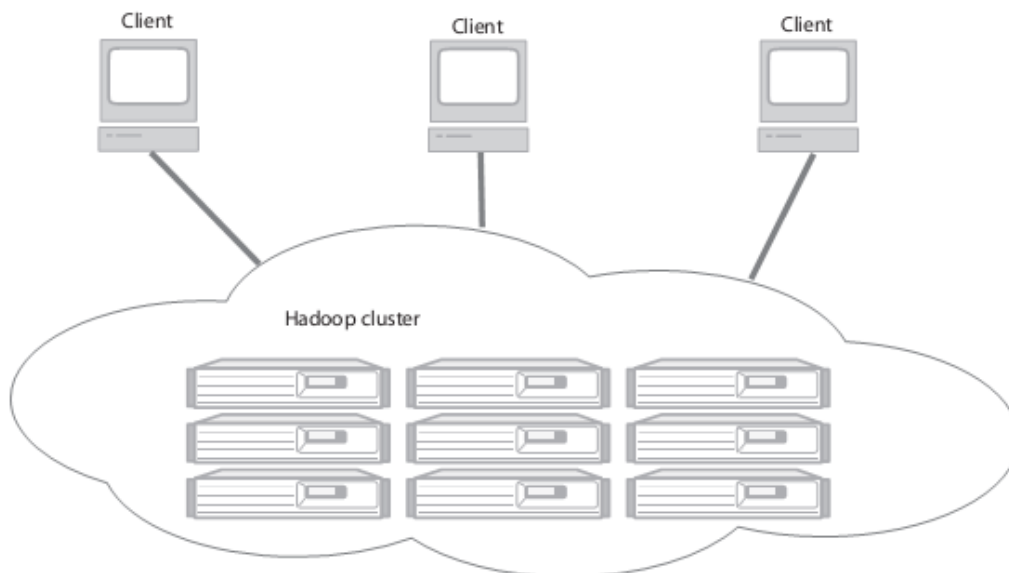


Figure 3.2: A Hadoop cluster has many parallel machines that store and process large data sets. Client computers send jobs into this computer cloud and obtain results.

Figure 3.2 illustrates interactions with an Hadoop cluster. Data storage and processing all occur within this “cloud” of machines. Different users can submit computing “jobs” to Hadoop from individual clients, which can be their own desktop machines in remote locations from the Hadoop cluster. Not all distributed systems are set up as shown in figure 3.2.

One of the most important Hadoop’s characteristic is its distributed file system.

3.3.1 Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size.

HDFS is the file system component of Hadoop. While the interface to HDFS is patterned after the UNIX file system, faithfulness to standards was sacrificed in favor of improved performance for the applications at hand. HDFS stores file system metadata and application data separately. As in other distributed file systems, like PVFS [25, 38], Lustre and GFS [29, 19], HDFS stores metadata on a dedicated server, called the NameNode. Application’s

data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-based protocols. Unlike Lustre and PVFS, the DataNodes in HDFS do not use data protection mechanisms such as RAID to make the data durable. Instead, like GFS, the file's content is replicated on multiple DataNodes for reliability. While ensuring data durability, this strategy has the added advantage that data transfer bandwidth is multiplied, and there are more opportunities for locating computation near the needed data.

NameNode The HDFS namespace is a hierarchy of files and directories. Files and directories are represented on the NameNode by *inodes*, which record attributes like permissions, modification and access times, namespace and disk space quotas. The file content is split into large blocks (typically 128 megabytes, but user selectable file-by-file) and each block of the file is independently replicated at multiple DataNodes (typically three, but user selectable file-by-file). The NameNode maintains the namespace tree and the mapping of file's blocks to DataNodes (the physical location of file data). An HDFS client wanting to read a file, first contacts the NameNode for the locations of data blocks comprising the file and then reads block contents from the DataNode closest to the client. When writing data, the client requests the NameNode to nominate a suite of three DataNodes to host the block replicas. The client then writes data to the DataNodes in a pipeline fashion. The current design has a single NameNode for each cluster. The cluster can have thousands of DataNodes and tens of thousands of HDFS clients per cluster, as each DataNode may execute multiple application tasks concurrently. DFS keeps the entire namespace in RAM². The *inode* data and the list of blocks belonging to each file comprise the metadata of the name system called the image. The persistent record of the image stored in the local host's native files system is called a checkpoint. The NameNode also stores the modification log of the image called *the journal* in the local host's native filesystem. For improved durability, redundant copies of the checkpoint and journal can be made at other servers. During restarts the NameNode restores the namespace by reading the namespace and replaying the journal. The locations of block replicas may change over time and are not part of the persistent checkpoint.

DataNodes Each block replica on a DataNode is represented by two files in the local host's native file system. The first file contains the data itself and the second file is block's metadata including checksums for the block data and the block's *generation stamp*. The size of the data file equals the actual length of the block and does not require extra space to round it up to the nominal block size as in traditional file systems. Thus, if a block is half full it needs only half of the space of the full block on the local drive. During startup each DataNode

²Random Access Memory

connects to the NameNode and performs a handshake. The purpose of the handshake is to verify the namespace ID and the software version of the DataNode. If either does not match that of the NameNode the DataNode automatically shuts down. The namespace ID is assigned to the file system instance when it is formatted. The namespace ID is persistently stored on all nodes of the cluster. Nodes with a different namespace ID will not be able to join the cluster, thus preserving the integrity of the file system. The consistency of software versions is important because incompatible version may cause data corruption or loss, and on large clusters of thousands of machines it is easy to overlook nodes that did not shut down properly prior to the software upgrade or were not available during the upgrade. A DataNode that is newly initialized and without any namespace ID is permitted to join the cluster and receive the cluster's namespace ID. After the handshake the DataNode registers with the NameNode. DataNodes persistently store their unique storage IDs. The storage ID is an internal identifier of the DataNode, which makes it recognizable even if it is restarted with a different IP address or port. The storage ID is assigned to the DataNode when it registers with the NameNode for the first time and never changes after that.

A DataNode identifies block replicas in its possession to the NameNode by sending a *block report*. A block report contains the block id, the generation stamp and the length for each block replica the server hosts. The first block report is sent immediately after the DataNode registration. Subsequent block reports are sent every hour and provide the NameNode with an up-to-date view of where block replicas are located on the cluster. During normal operation DataNodes send *heartbeats* to the NameNode to confirm that the DataNode is operating and the block replicas it hosts are available. The default heartbeat interval is three seconds. If the NameNode does not receive a heartbeat from a DataNode in ten minutes the NameNode considers the DataNode to be out of service and the block replicas hosted by that DataNode to be unavailable. The NameNode then schedules creation of new replicas of those blocks on other DataNodes. Heartbeats from a DataNode also carry information about total storage capacity, fraction of storage in use, and the number of data transfers currently in progress. These statistics are used for the NameNode's space allocation and load balancing decisions. The NameNode does not directly call DataNodes. It uses replies to heartbeats to send instructions to the DataNodes. The instructions include commands to:

- replicate blocks to other nodes
- remove local block replicas
- re-register or to shut down the node
- send an immediate block report

The NameNode can process thousands of heartbeats per second without affecting other NameNode operations.

HDFS Client User applications access the file system using the HDFS client, a code library that exports the HDFS file system interface. Similar to most conventional file systems, HDFS supports operations to read, write and delete files, and operations to create and delete directories. The user references files and directories by paths in the namespace. The user application generally does not need to know that file system metadata and storage are on different servers, or that blocks have multiple replicas. When an application reads a file, the HDFS client first asks the NameNode for the list of DataNodes that host replicas of the blocks of the file. It then contacts a DataNode directly and requests the transfer of the desired block. When a client writes, it first asks the NameNode to choose DataNodes to host replicas of the first block of the file. The client organizes a pipeline from node-to-node and sends the data. When the first block is filled, the client requests new DataNodes to be chosen to host replicas of the next block. A new pipeline is organized, and the client sends the further bytes of the file. Each choice of DataNodes is likely to be different. The interactions among the client, the NameNode and the DataNodes are illustrated in Figure 3.3 Unlike conventional file systems, HDFS provides an API that exposes the locations of a file

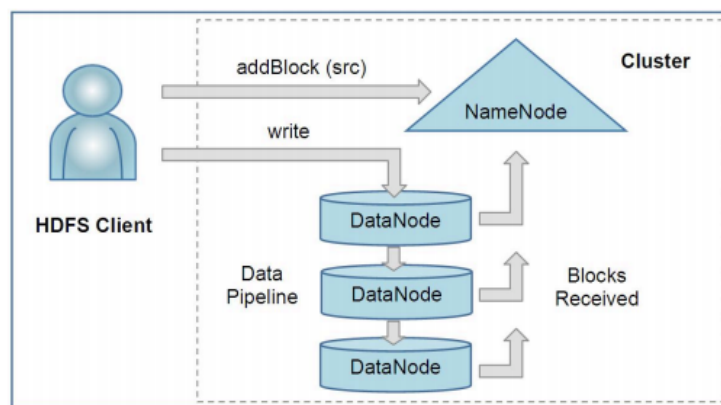


Figure 3.3: HDFS clients

blocks. This allows applications like the MapReduce framework to schedule a task to where the data are located, thus improving the read performance. It also allows an application to set the replication factor of a file. By default a file's replication factor is three. For critical files or files which are accessed very often, having a higher replication factor improves their tolerance against faults and increase their read bandwidth.

3.4 HIVE - Data Warehouse Using Hadoop

Hadoop lacked the expressiveness of popular query languages like SQL and as a result users ended up spending hours (if not days) to write programs for even simple analysis. *Hive* [14] is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems.

Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

Hive structures data into the well-understood database concepts like tables, columns, rows, and partitions. It supports the major primitive types like integers, floats, doubles and strings, as well as complex types such as maps, lists and structures. The latter can be nested arbitrarily to construct more complex types. In addition, Hive allows users to extend the system with their own types and functions. The query language is very similar to SQL and therefore can be easily understood by anyone familiar with SQL.

3.4.1 Data Model and Type System

Similar to traditional databases, Hive stores data in tables, where each table consists of a number of rows, and each row consists of a specified number of columns. Each column has an associated type. The type is either a primitive type or a complex type. Currently, the following primitive types are supported:

Integers bigint(8 bytes), int (4 bytes), smallint (2 bytes), tinyint (1 byte). All integer types are signed

Floating point numbers float (single precision), double (double precision)

String

Hive also natively supports the following complex types:

Associative arrays map<key-type,value-type>

Lists list<element-type>

Structs struct<file-name: field-type,... >

These complex types are templated and can be composed to generate types of arbitrary complexity. For example, list<map<string, struct<p1:int, p2:int>>> represents a list of associative arrays that map strings to structs that in turn contain two integer fields named

p1 and p2. These can all be put together in a create table statement to create tables with the desired schema. For example, the following statement creates a table t1 with a complex schema:

```
CREATE TABLE t1(st string, fl float, li list<map<string,
struct<p1:int, p2:int>>>);
```

The tables created in the manner described above are serialized and deserialized using default serializers and deserializers already present in Hive. However, there are instances where the data for a table is prepared by some other programs or may even be legacy data. Hive provides the flexibility to incorporate that data into a table without having to transform them, which can save substantial amount of time for large data sets.

3.4.2 Query Language

The Hive query language(HiveQL) comprises of a subset of SQL and some extensions. Traditional SQL features like from clause sub- queries, various types of joins - inner, left outer, right outer and outer joins - cartesian products, group byes and aggregations, union all, create table as select and many useful functions on primitive and complex types make the language very SQL like. In fact for many of the constructs mentioned before it is exactly like SQL. This enables anyone familiar with SQL to start a hive cli³ and begin querying the system right away. Useful metadata browsing capabilities like show tables and describe are also present and so are explain plan capabilities to inspect query plans. There are some limitations e.g. only equality predicates are supported in a join predicate and the joins have to be specified using the ANSI join syntax such as

```
SELECT t1.a1 as c1, t2.b1 as c2
FROM t1 JOIN t2 ON (t1.a2 = t2.b2);
```

instead of more traditional

```
SELECT t1.a1 as c1, t2.b1 as c2
FROM t1 JOIN t2 ON (t1.a2 = t2.b2);
```

Another limitation is in how inserts are done. Hive currently does not support inserting into an existing table or data partition and all inserts overwrite the existing data. Hive allows users to interchange the order of the *FROM* and *SELECT/MAP/REDUCE* clauses

³command line interface

within a given sub-query. This becomes particularly useful and intuitive when dealing with multi inserts. HiveQL supports inserting different transformation results into different tables, partitions, HDFS or local directories as part of the same query. This ability helps in reducing the number of scans done on the input data as shown in the following example:

```
FROM t1
  INSERT OVERWRITE TABLE t2
  SELECT t3.c2, count(1)
  FROM t3
  WHERE t3.c1 <= 20
  GROUP BY t3.c2

  INSERT OVERWRITE DIRECTORY '/output\_dir'
  SELECT t3.c2, avg(t3.c1)
  FROM t3
  WHERE t3.c1 > 20 AND t3.c1 <= 30
  GROUP BY t3.c2

  INSERT OVERWRITE LOCAL DIRECTORY '/home/dir'
  SELECT t3.c2, sum(t3.c1)
  FROM t3
  WHERE t3.c1 > 30
  GROUP BY t3.c2;
```

In this example different portions of table t1 are aggregated and used to generate a table t2, an HDFS directory(/output_dir) and a local directory(/home/dir on the user's machine).

3.4.3 Data Storage

While the tables are logical data units in Hive, table metadata associates the data in a table to HDFS directories. The primary data units and their mappings in the HDFS name space are as follows:

Tables A table is stored in a directory in HDFS

Partitions A partition of the table is stored in a sub-directory within a table's directory.

Bucketes A bucket is stored in a file within the partition's or table's directory depending on whether the table is a partitioned table or not.

As an example a table *test_table* gets mapped to `<warehouse_root_directory>/test_table` in HDFS. The `warehouse_root_directory` is specified by the `hive.metastore.warehouse.dir` configuration parameter in *hive-site.xml*. By default this parameter's value is set to `/user/hive/warehouse`.

A table may be partitioned or non-partitioned. A partitioned table can be created by specifying the `PARTITIONED BY` clause in the `CREATE TABLE` statement as shown below.

```
CREATE TABLE test\_part(c1 string, c2 int)
PARTITIONED BY (ds string, hr int);
```

In the example shown above the table partitions will be stored in `/user/hive/warehouse/test_part` directory in HDFS. A partition exists for every distinct value of *ds* and *hr* specified by the user. Note that the partitioning columns are not part of the table data and the partition column values are encoded in the directory path of that partition (they are also stored in the table metadata). The Hive compiler is able to use this information to prune the directories that need to be scanned for data in order to evaluate a query. Pruning the data has a significant impact on the time it takes to process the query. In many case respects this partitioning scheme is similar to what has been referred to as list partitioning by many database vendors ([18]), but there are differences because in this case the values of the partition keys are stored with the metadata instead of the data.

The final storage unit concept that Hive uses is the *Buckets*. A bucket is a file within the leaf level directory of a table or a partition. At the time the table is created, the user can specify the number of buckets needed and the column on which to bucket the data. In the current implementation this information is used to prune the data in case the user runs the query on a sample of data e.g. a table that is bucketed into 32 buckets can quickly generate a 1/32 sample by choosing to look at the first bucket of data.

3.4.4 System Architecture and Components

The following components are the main building blocks in Hive:

Metastore The component that stores the system catalog and metadata about tables, columns, partitions etc.

Driver The component that manages the lifecycle of a HiveQL statement as it moves through Hive. The driver also maintains a session handle and any session statistics.

Query Compiler The component that compiles HiveQL into a directed acyclic graph of map/reduce tasks.

Execution Engine The component that executes the tasks produced by the compiler in proper dependency order. The execution engine interacts with the underlying Hadoop instance.

HiveServer The component that provides a thrift interface and a JDBC/ODBC server and provides a way of integrating Hive with other applications.

CLI Clients components like the Command Line Interface (CLI), the web UI and JDBC/ODBC driver.

WebInterface Extensibility Interfaces which include the SerDe and ObjectInspector interfaces already described previously as well as the UDF(User Defined Function) and UDAF(User Defined Aggregate Function) interfaces that enable users to define their own custom functions.

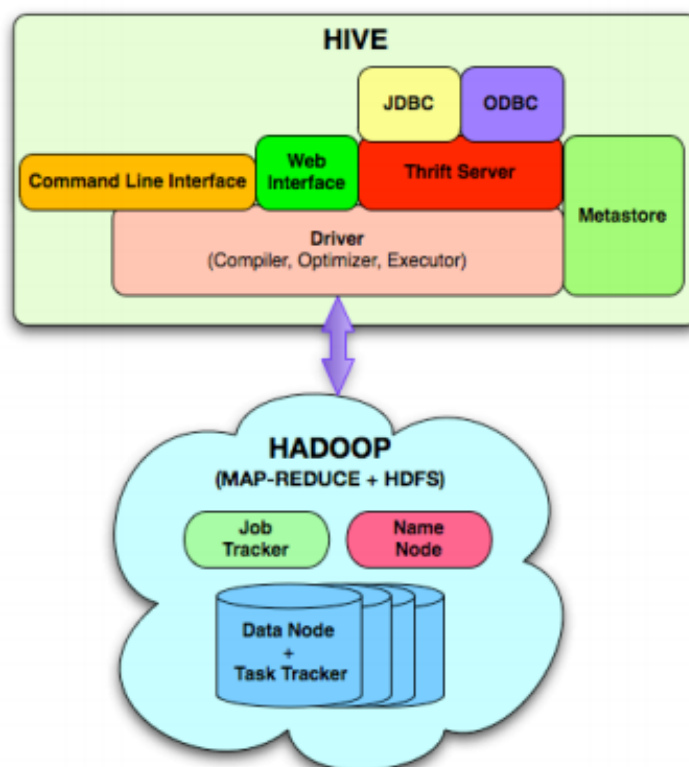


Figure 3.4: Hive System Architecture

A HiveQL statement is submitted via the CLI, the web UI or an external client using the **thrift**, **odbc** or **jdbc** interfaces. The driver first passes the query to the compiler where it goes through the typical parse, type check and semantic analysis phases, using the metadata stored in the Metastore. The compiler generates a logical plan that is then optimized through

a simple rule based optimizer. Finally an optimized plan in the form of a DAG of map-reduce tasks and hdfs tasks is generated. The execution engine then executes these tasks in the order of their dependencies, using Hadoop.

3.4.5 Thrift Server

Thrift is a software library and set of code-generation tools developed at Facebook to expedite development and implementation of efficient and scalable backend services. Its primary goal is to enable efficient and reliable communication across programming languages by abstracting the portions of each language that tend to require the most customization into a common library that is implemented in each language. Some key components have been identified during developing phases:

Types A common type system must exist across programming languages without requiring that the application developer use custom Thrift datatypes or write their own serialization code. That is, a C++ programmer should be able to transparently exchange a strongly typed STL map for a dynamic Python dictionary.

Transport Each language must have a common interface to bidirectional raw data transport. The specifics of how a given transport is implemented should not matter to the service developer. The same application code should be able to run against TCP stream sockets, raw data in memory, or files on disk

Protocol Datatypes must have some way of using the Transport layer to encode and decode themselves. Again, the application developer need not be concerned by this layer. Whether the service uses an XML or binary protocol is immaterial to the application code. All that matters is that the data can be read and written in a consistent, deterministic matter.

Versioning For robust services, the involved datatypes must provide a mechanism for versioning themselves. Specifically, it should be possible to add or remove fields in an object or alter the argument list

Thrift Server has been used to submit query to Hive using Java programming language.

3.5 Esper - Complex Event Processing

Esper is a component for complex event processing (CEP), available for Java as Esper, and for .NET as NEsper. Esper and NEsper enable rapid development of applications that process large volumes of incoming messages or events. Esper and NEsper filter and analyze

events in various ways, and respond to conditions of interest in real-time. The Esper engine has been developed to address the requirements of applications that analyze and react to events. Some typical examples of applications are:

Business Project management and automation

Finance Algorithmic trading, **fraud detection**, risk management

Network Application monitoring and intrusion detection

Sensor Network RDIF reading, air traffic control

What these applications have in common is the requirement to process events (or messages) in real-time or near real-time. This is sometimes referred to as complex event processing (CEP) and event stream analysis, complex event processing (CEP) delivers high-speed processing of many events across all the layers of an organization, identifying the most meaningful events within the event cloud, analyzing their impact, and taking subsequent action in real time. Key considerations for these types of applications are throughput, latency and the complexity of the logic required

- High throughput
- Low latency
- Complex computations

3.5.1 CEP and relation databases

Relational databases and the standard query language (SQL) are designed for applications in which most data is static and complex queries are less frequent. Also, most databases store all data on disks (except for in-memory databases) and are therefore optimized for disk access. To retrieve data from a database an application must issue a query. If an application need the data 10 times per second it must fire the query 10 times per second. This does not scale well to hundreds or thousands of queries per second. Database triggers can be used to fire in response to database update events. However database triggers tend to be slow and often cannot easily perform complex condition checking and implement logic to react. In-memory databases may be better suited to CEP applications than traditional relational database as they generally have good query performance. Yet they are not optimized to provide immediate, real-time query results required for CEP and event stream analysis.

3.5.2 Esper engine

The Esper engine works like a database but instead of storing the data and running queries against them, it allows application to store queries and run data through.

Response are in near-real time when conditions that match queries occur. The execution model is thus continuous rather than only when a query is submitted. Esper architecture is shown in figure 3.5 Esper provides two principal methods or mechanisms to process events:

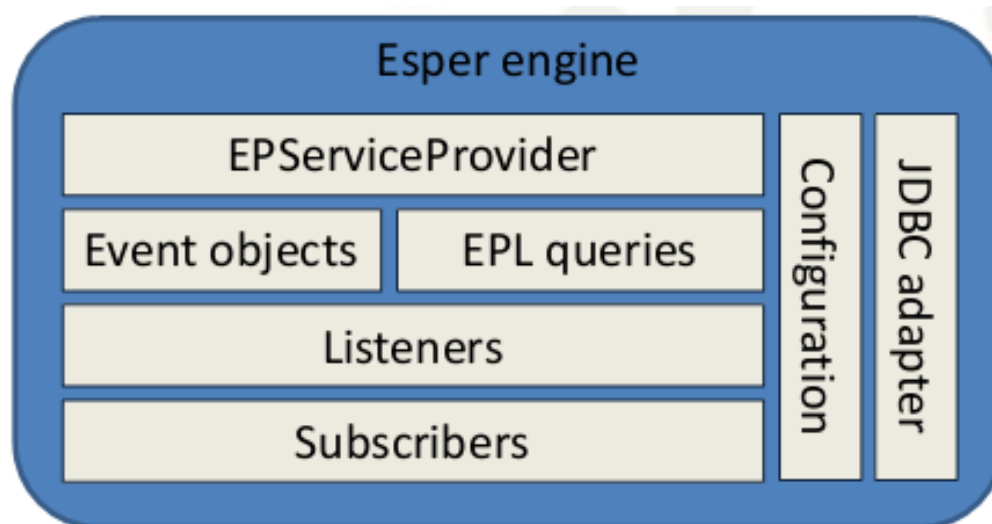


Figure 3.5: Esper System Architecture

event patterns and *event stream queries*.

Event pattern language Event pattern language specify an expression-based event pattern matching. Underlying the pattern matching engine is a state machine implementation. This method of event processing matches expected sequences of presence or absence of events or combinations of events. It includes time-based correlation of events.

Event stream queries Event stream queries address the event stream analysis requirements of CEP applications. Event stream queries provide the windows, aggregation, joining and analysis functions for use with streams of events. These queries are following the EQL syntax. EQL has been designed for similarity with the SQL query language but differs from SQL in its use of views rather than tables. Views represent the different operations needed to structure data in an event stream and to derive data from an event stream.

Esper provides these two methods as alternatives through the same API.

3.5.3 EQL

EQL statements are used to derive and aggregate information from one or more streams of events, and to join or merge event streams. EQL is similar to SQL in its use of the *select* clause and the *where* clause. Where EQL differs most from SQL is in the use of tables. EQL replaces tables with the concept of event streams.

EQL statements contain definitions of one or more views. Similar to tables in an SQL, views define the data available for querying and filtering. Some views represent windows over a stream of events. Other views derive statistics from event properties, group events or handle unique event property values. Views can be staggered onto each other to build a chain of views. The Esper engine makes sure that views are reused among EQL statements for efficiency.

EQL queries are created and stored in the engine, and publish results as events are received by the engine or timer events occur that match the criteria specified in the query. Events can also be pulled from running EQL queries. For example, see the following statement

```
select avg(price) from Stock where symbol='HP'
```

EQL queries follow the below syntax

```
[insert into insert into def]
select selectlist
from streamdef [as name] [, streamdef [asname]] [, ...]
[where searchconditions]
[group by groupingexpressionlist]
[having groupingsearchconditions]
[output outputspecification]
[order by orderbyexpressionlist]
```

The EQL statements's most important elements are listed below

Select The *select* clause is required in all EQL statements, it can be used to select all properties via the wildcard *, or to specify a list of event properties and expressions. Also it defines the event type (event property names and types) of the resulting events published by the statement, or pulled from the statement.

From The *from* clause is required in all EQL statements. It specifies one or more event streams. Each event stream can optionally be given a name by means of the *as* syntax.

Where The *where* clause is an optional clause in EQL statements. Via the where clause event streams can be joined and events can be filtered. Comparison operators =, <

, >, >=, <=, !=, <>, is null, is not null and logical combinations via and and or are supported in the where clause. The where clause can also introduce join conditions.

Aggregate Functions The *aggregate* functions are *sum*, *avg*, *count*, *max*, *min*, *median*, *stddev*, *avedev*, they can be used to aggregate functions to calculate and summarize data from event properties.

Group By The group by clause divides the output of an EQL statement into groups. Can be created group by one or more event property names, or by the result of computed expressions. When used with aggregate functions, group by retrieves the calculations in each subgroup.

Chapter 4

A privacy preserving architecture for collaborative processing

4.1 System Model

The idea beyond our system is to use information provided by system's participants (through log files) to retrieve useful information about possible cyber attacks. The use of log files raise a problem: **privacy**. Log files, in fact, usually contains private information such as IP addresses, userIDs . . . , business organizations are historically reluctant to divulge such information to other companies. This kind of issue can be seen as a particular case of *secure multi-party computation*.

Secure multi-party computation is as follows:

n players, P_1, P_2, \dots, P_n wish to evaluate a function, $F(x_1, x_2, \dots, x_n)$, where x_i is a secret value provided by P_i .

The goal is to preserve the **privacy** of player's input and player's themselves and guarantee the correctness of the computation.

In our scenario the term *privacy* contains more than one meaning. Our goal was to achieve a **Three dimensional privacy** (see figure 4.1) Privacy in our system requests that **Low identity** and **No linkability** properties must be satisfied.

Identity It's degree to which personal information are visible to participants that not own them - GOAL: decrease the amount of identity into the system

Linkability It's degree to witch provided data are linkable to participants - GOAL: decrease the amount of linkability into the system

Observability is the impact (or "**the cost**") of previous requirements on the system. Before to proceed with the presentation of the architecture we informally highlight prevalent

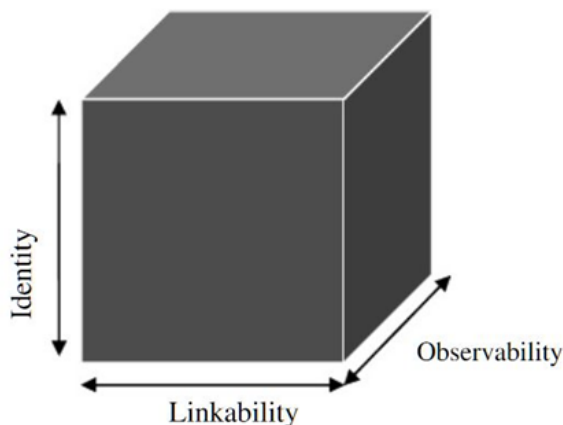


Figure 4.1: Three dimensional privacy

Adversary Models.

With the term *adversary model* we mean the set of assumptions, explicit and implicit, which have been made with regards to the adversary in any given situation. While there is precedence for using such a definition of adversary modelling (e.g., [40, 30]), it is not widely used in literature. There are two major adversary models for secure computation: *Semi-honest model* and *Fully malicious model*.

Semi-Honest Model Semi-Honest Model (a.k.a. honest-but-curious) means that parties follow the protocol but try to infer as much as possible from the values (shared data), also combining their information.

Fully malicious model Dishonest parties can deviate from the protocol and behave arbitrarily

In our experiment it is assumed that participants follow a **Semi-Honest Model**.

Different architectures have been analyzed: using symmetric encryption only, or a combination of asymmetric and symmetric encryption.

Build a system that guarantees our goal is trivial if a **trusted third party** T is added to the computation. Simply, T collects all the inputs data from the participants, do the correlation and announces the result, but in this scenario we have considered a trusted party not acceptable. So, in the architecture's design it is assumed that third parties must be more than one and majority of them is **honest** while the others *could* be **semi-honest** and can collaborate (among themselves or with participants).

4.2 1st Architecture - Proxy Based (Data Spread)

The first architecture analyzed is shown in figure 4.2.

GW_x is the “Participant’s X gateway”, PU_s are the “Processing Units”, P_{xy} are the third

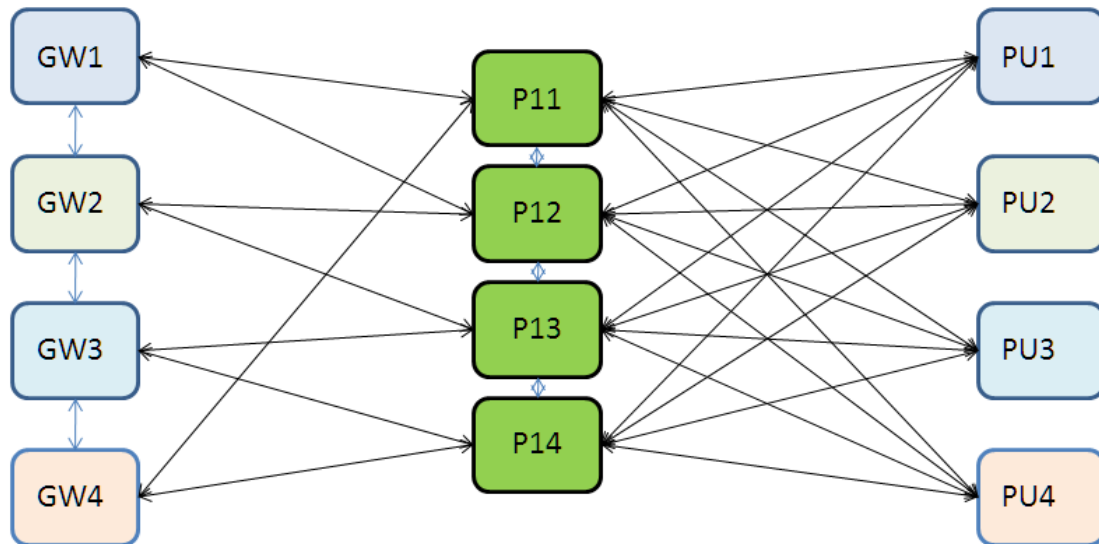


Figure 4.2: Proxy Based (Data Spread) - Overview

parties.

Gateways can communicate among themselves as proxies, also gateways communicate with proxies and proxies communicate with processing units. Before showing how this architecture works let’s see the role of its different components

Gateways They belong to participants, gateways own the data and send them to the system

Proxies They assure No-Linkability of data with their owner

Processing Units They do the correlation among data and produce a result

4.2.1 How it works

Let’s see how this system works

1. Gateways calculate a key \mathbf{k} from a common seed (now all gateways have the same key \mathbf{k})
2. Every Gateway encrypts its data with \mathbf{k} and sends them to a subset of proxies (fixed a priori)

3. Proxies encrypt data with k_1 (it's the same for all of them) and send data to Processing Units
4. Processing Units do the computation and send back data for decryption

k and k_1 are **AES-128 bit** symmetric keys. We have chosen a size of 128-bit because it is strong enough for a Brute-Force Attack (see Appendix C for further details).

4.2.2 Architecture Analysis - Pros vs Cons

Let's analyze some possible scenario to identify pros and cons of this architecture

All components follow the protocol If all system's components follow the protocol, this architecture guarantees *identity* using double encryption (one from Gateways and one from Proxy), also *No-Linkability* is assured, because proxies mask data to Processing Units

Proxies collaborate among themselves If proxies collaborate among themselves they can't obtain information because they can't break gateways's encryption

Proxies collaborate with Gateways If a proxy (at most two of them - from system model majority of proxies are honest) collaborates with a gateway (i.e. give k_1 to it), gateway can't obtain data of all others participants thanks to particular data spread chosen

Proxies collaborate with Processing Units If at least one proxy decides to collaborate with a Processing Unit (i.e. give k_1 to it) and Processing Unit is in the same domain of Gateway (so PU has the key k), Processing Unit can obtain all data of others participants (even if they are not suspicious) because now it has both necessary key for decryption

So, to have a sufficient degree of security Processing Units **must** be external to Gateway domain and trusted and, as said above trusted party are not acceptable. There's another reason to consider this architecture unsuitable for our goal, it's **hard** to calculate *a priori* data spread (this problem can be seen a particular case of Set Cover [39]).

For these reasons a new architecture has been analyzed.

4.3 2nd Architecture - Matrix Proxy Based

The second architecture analyzed is shown in figure 4.3 We have decided to add a second column of proxies to create a sort of "matrix of proxies".

, removing the data spread.

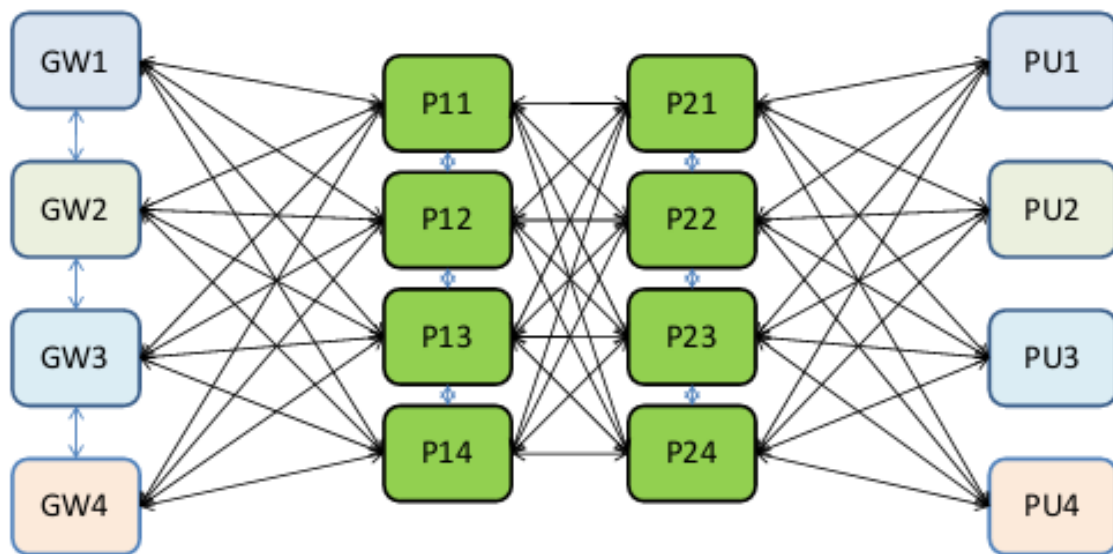


Figure 4.3: Matrix Proxy Based - Overview

4.3.1 How it works

System, now, works as follow

1. Gateways calculate a key \mathbf{k} from a common seed (now all gateways have the same key \mathbf{k})
2. Every Gateway encrypts its data with \mathbf{k} and sends them to all first column's proxies
3. First column's proxies encrypt data with the same key $\mathbf{k1}$ and send them to second column's proxies
4. Second column's proxies encrypt data again with a key $\mathbf{k2}$ (common among themselves) and send data to Processing Unit

$\mathbf{k}, \mathbf{k1}$ and $\mathbf{k2}$ are AES 128-bit keys.

4.3.2 Architecture Analysis - Pros vs Cons

Let's analyze some possible scenario to identify pros and cons of this architecture

All components follow the protocol If all system's components follow the protocol, this architecture guarantees strong *identity* using triple encryption (one from Gateways and two from Proxies), also *No-Linkability* is assured, because proxy mask data to Processing Units

Proxies collaborate among themselves If proxies decides to collaborate among themselves they can't obtain information on data because they can't break gateways encryption

Proxies collaborate with Processing Unit Now, in contrast to previous architecture, if a proxy decides to collaborate with a Processing Unit, PU can't obtain data unless one proxy for every column decides to communicate its respectively key to it. So we have reduced the possibility that a Processing Unit gets no-malicious data

Proxies collaborate with Gateway If at least one proxy (of the first column) decides to collaborate with a Gateway, it can obtain all data of others participants.

The removal of data spread simplifies system architecture but creates a possibility to break system privacy by the gateways with a proxy's help.

We have analyzed others similar architectures to avoid these problems but finally we have noticed that in all of them there was always a single point of failure (one of the proxies - it could have all data). A possible solution is to move encryption phase from proxies to gateways, but symmetric encryption **doesn't allow** that, because the key used for encryption is *the same* used for decryption.

So, we have designed a third architecture, based on a mix between symmetric and asymmetric encryption that we've chosen as final architecture for our system and used for our experiments.

4.4 3rd Architecture - Asymmetric Key Proxy Based

The final architecture is shown in figure 4.4 We have decided to change some elements in comparison to the previous architectures, first of all the number of required proxies was reduced and now Processing Units can be co-located in the same domain system of Gateway.

4.4.1 How it works

This architecture works as follow

1. One of the Gateways (i.e. leader for this round) starts MPC round. It generates an *OperationID* and sends it to the others participants.
2. Gateways calculate a key k from a common seed (this seed must be different from *OperationID* - now all gateways have the same key) and encrypt data

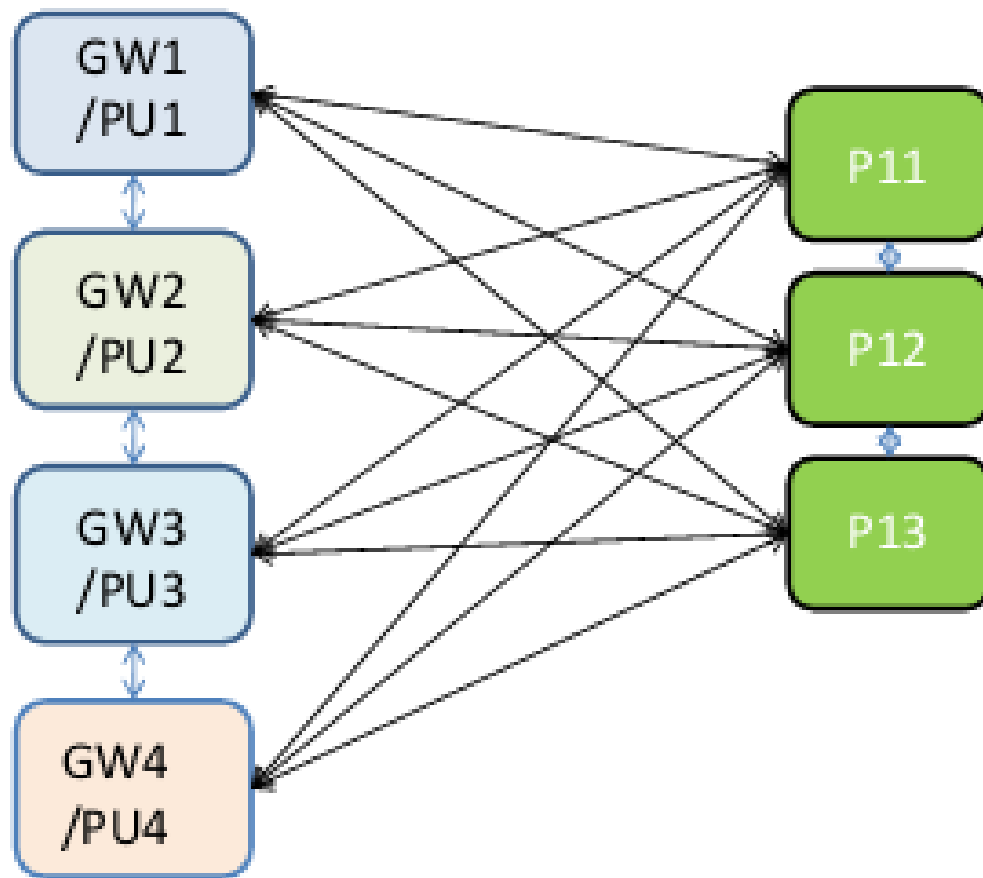


Figure 4.4: Asymmetric Proxy Based - Overview

3. Every gateway sends to Proxies the *OperationID* asking a key \mathbf{k}_x (x is the number of the proxy)
4. Proxy x receives from a gateway an *OperationID*, it checks if this *OperationID* has been processed yet, if not, it generates a couple of public/private *RSA* (1024-bit) key and sends the public key to gateway
5. Gateway encrypts data with public key in chain (first proxy's key, second proxy's key and third proxy's key)
6. Now gateways can exchange data (through proxies that can do a data shuffle) and Processing Units can start the computation

4.4.2 Architecture Analysis - Pros vs Cons

Let's analyze some possible scenario to identify pros and cons of this architecture

Table 4.1: RSA encryption and decryption execution timings

Text Size	Encryption	Decryption
128 bits	0.0549	0.0549
256 bits	0.1098	0.1098
512 bits	0.2197	0.1648
1K	0.3846	0.3296
2K	0.7142	0.6593
5K	1.7032	1.7032
10K	3.4020	3.4020

All components follow the protocol If all system's components follow the protocol, this architecture guarantees a very high degree of *identity* using four encryption (one from Gateways and three from Proxies), also *No-Linkability* is assured, because proxy mask data to Processing Units

Proxies collaborate with gateway/processing units In this case, the assumption that majority of proxies are honest means that only one proxy can collaborate, so Gateways neither Processing Units can derive information from encrypted data (they would break at least three RSA 1024-bit encryption - it could be considered infeasible)

The use of the asymmetrical cryptography allows to avoid the problem of collaboration between proxies and gateways/processing units seen in previous architecture. However, the introduction of this new encryption scheme is not costless in fact asymmetric encryption is slowest than symmetric encryption [6], table 4.1 (taken from [7]) shows RSA encryption and decryption execution timings. With this new architecture the decryption phase is a reverse chain, a gateway must ask first to third proxy then to the second and so on to decrypt with their private key. This behavior can raise a problem if a proxy fails decryption phase, a good (and pretty simple) solution is to use two column that works in parallel (with different keys). **This two parallel computation must produce the same result.**

Chapter 5

A Case of Study: Man-In-The-Browser (MITB)

5.1 Cyber Attacks in Financial Scenario

One of the first home banking services using personal computers was Postbank, which introduced the Girotel application in 1986. In the very beginning an external computing device called Viditel was required, which had to be connected to both a telephone line and the user's monitor. Using this system a user could initiate transactions locally and transmit these to the bank over a telephone line in a batch. Eventually, the Viditel device was replaced by a fat-client [21] software application which had a similar operating procedure. Other banks followed with similar applications such as Fortis Bank's PC Banking and ING Bank's Interactive Banking. Not only competitiveness amongst banks was a motive to introduce these home banking applications. For banks major benefits of these remote banking services are the reduced personnel and administrative costs.

Nowadays, internet banking has become established technology. All major banks offer advanced internet banking services that offer domestic and international payment transactions and alerting services using e-mail or SMS. Some banks take remote banking even further. The establishment of internet banking is also expressed by its large user base. In 2006, 68% of all internet users regularly used internet banking services [36]. For the coming years a considerable growth in the number of users of internet banking is expected. After that the number of users will likely remain stable [5].

Right from the introduction of internet banking, security of these systems has been subject for debate. Fortunately, all major banks introduced strengthened authentication and security measures. Especially so-called two-factor authentication (see Appendix B) systems became popular right from the introduction of internet banking. Using such a system, initi-

ation of an internet banking session or a transaction requires access to an additional device (often called security calculator), which can generate temporal access codes.

5.1.1 An introduction to Man-In-The-Browser (MITB) Attacks

A man-in-the-browser attack is designed to intercept data as it passes over a secure communication between a user and an online application. A Trojan embeds itself in a user's browser and can be programmed to activate when a user accesses specific online sites, such as an online banking sites. Once activated, a man-in-the-browser Trojan can intercept and manipulate any information a user submits online in real-time. A number of Trojan families are used to conduct MITB attacks including Zeus, Adrenaline, Sinowal, and Silent Banker. Some MITB Trojans are so advanced that they have streamlined the process for committing fraud, programmed with functionality to fully automate the process from infection to cash out. Additional capabilities offered by MITB Trojan developers include:

- HTML injection to display socially engineered pages (i.e. injecting a field into a page asking for the user's ATM pin number in addition to their username and password).
- Real-time integration of Trojans with mule account databases to aid in the automated transfer of money.
- The ability to circumvent various two-factor authentication systems including CAP/EMV, transaction signing, iTANs, and one-time password authentication

MITB Trojans commonly perform what is known as "session hijacking" - abusing a legitimate user's session with the site being accessed while the user is logged into their account. By hijacking a session in this way, all actions performed by the Trojan actually become part of the user's legitimate session such as conducting a malicious activity (i.e. a fraudulent money transfer, changing a postal address) or even injecting JavaScript code that can then perform this automatically.

MITB attacks are not contained to one region or geography. They are a global threat, affecting all regions of the world. However, they are especially prevalent in areas where two-factor authentication is densely deployed. Today, MITB attacks are increasing in their deployment and scale: in the UK, banks are suffering an increasing number of MITB attacks. One financial institution alone reported a loss of £600,000 as a result of a single attack by the PSP2-BBB Trojan. European countries such as Italy, Germany, the Netherlands, Spain, France, and Poland have deployed two-factor authentication in the last few years, which have attracted a rise in the numbers of MITB attacks in these regions. Germany has been particularly hard hit by an abundance of MITB attacks as it is one of the few successful paths to commit online banking fraud in the country. Banking innovations such as the Single Euro

Payments Area [4] (SEPA) and pressure to deliver faster payments have also increased exposure to transaction fraud. The increased ease and speed of moving money is advantageous for legitimate transactions, but reduces the flexibility to investigate and prevent suspicious transactions, also, U.S. financial institutions are attacked by MITB; however, the threat has been mainly confined to commercial banking or high net worth customers. Because one-time password authentication is not very common amongst consumers in the U.S., MITB attacks against the general consumer public are less common compared to the volume experienced by consumers in Europe. However, as security defenses increase and the ability to infect more machines with MITB Trojans increases the expected number of attacks on US retail banking institutions is also expected to rise.

Evolution of Man-in-the-Browser

MITB Trojans are part of the natural evolution of online fraud. Before the introduction of strong authentication, online criminals could gather information to commit fraud through phishing attacks or standard Trojans that did not intervene with a user's online activity or transactions. Due to increased consumer awareness and stronger online security mechanisms, fraudsters had to update their methods and tools to overcome two-factor authentication. The idea of MITB attacks was conceived primarily for the purpose of circumventing strong authentication.

5.1.2 How it works: Points and Method of attack

The new trojan technology is technically more advanced than prior generations by way of combining Browser-Helper-Objects, Browser Extensions, and direct Browser manipulation techniques, by the way nowadays trojans can do the following

- Modify the appearance of a website before the user sees it.
- Modify the data entered by the user before it is encrypted and sent to the server. This modification is not visible / detectable by the user or the server.
- Change the appearance of transactions returned by the server back to the version that the user expects

There're many points of attack in a internet browser that could be used by trojan to manipulate the user interaction with a website. The main points of attack are:

- **Browser Helper Objects** \mapsto these are dynamically-loaded libraries (DLLs) loaded by Internet Explorer / Windows Explorer upon start-up. They run inside IE, and have

full access to IE and full access to the DOM tree, etc. *Developing BHOs is very easy.*

- **Extensions** \mapsto similar to Browser Helper Objects for other Browsers such as Firefox (hereafter, both will be referred to as extensions). *Developing Extensions is easy.*
- **UserScripts** \mapsto Scripts that are running in the browser (Firefox/Greasemonkey+Opera). *Developing UserScripts is very easy.*
- **API-Hooking** \mapsto this technique is a Man-in-the-Middle attack between the application (.EXE) and the DLLs that are loaded up, both for application specific DLLs such as extensions and Operating System (OS) DLLs. For example if the SSL engine of the browser is a separate DLL, then API-Hooking can be used to modify all communication between the browser and the SSL engine. *Developing API Hooks is difficult.*
- **Virtualisation** \mapsto running the whole operating system in a virtual environment, to easily bypass all the security mechanisms. *Developing Virtualisation attacks is difficult.*

Below is shown in a simple way how an attack may progress (for a more detailed example see 5.1.3):

1. The trojan infects the computer's browser
2. The trojan installs an extension into the browser configuration, so that it will be loaded next time the browser starts.
3. At some later time, the user restarts the browser.
4. The browser loads the new extension (trojan) without prompt user.
5. The extension registers a handler for every page-load.
6. Whenever a page is loaded, the URL of the page is searched by the extension against a list of known sites targeted for attack.
7. The user logs in securely to for example `http://server_site_url/login.php`
8. When the handler detects a page-load for a specific pattern in its targeted list (for example `https://server_site_url/do_bank_transfer.php`) it registers a button event handler.
9. When the submit button is pressed, the extension extracts all data from all form fields through the DOM interface in the browser, and remembers the values.

10. The extension modifies the values through the DOM interface (i.e. IBAN).
11. The extension tells the browser to continue to submit the form to the server.
12. The extension tells the browser to continue to submit the form to the server.
13. The server receives the modified values in the form as a normal request. The server cannot differentiate between the original values and the modified values, or detect the changes.
14. The server performs the transaction and generates a receipt.
15. The browser receives the receipt for the modified transaction.
16. The extension detects the `https:///server_site_url/account/receipt.php` URL, scans the HTML for the receipt fields, and replaces the modified data in the receipt with the original data that it remembered in the HTML.
17. The browser displays the modified receipt with the original details.
18. The user thinks that the original transaction was received by the server intact and authorised correctly.

5.1.3 MITB: a detailed example

To analyze in detail MITB attack, let's see an example from Germany. The victim, a customer of Deutsche Postbank, is using the online banking system. This victim's system became infected with malware by visiting a *legitimate* web site that allows people to download free screensavers.

After the malware was installed, the user transacted with his online bank-still oblivious that anything was wrong-while the attacker viewed the screenshot 5.1 below. This screenshot shows the number of times the malware was accessed and by the type of browser, giving the attacker easy manageability and tracking of the attack toolkit. This particular screenshot is from the *LuckySploit* attack toolkit. Once the malware was successfully installed, the attacker was able to remotely control the infected computer from the control panel. The following screenshot 5.2 is an example of different parameters that can be set, such as the range from minimum to maximum amounts. This is an important setting, because banks have different limits on which transactions are allowed or will be validated. They might garner the needed information, so desired parameters can be set - from monitoring the installation to seeing which banks are being used to scan the infected computer logs. These options need to be set for each installation. With the attacker setting all their required

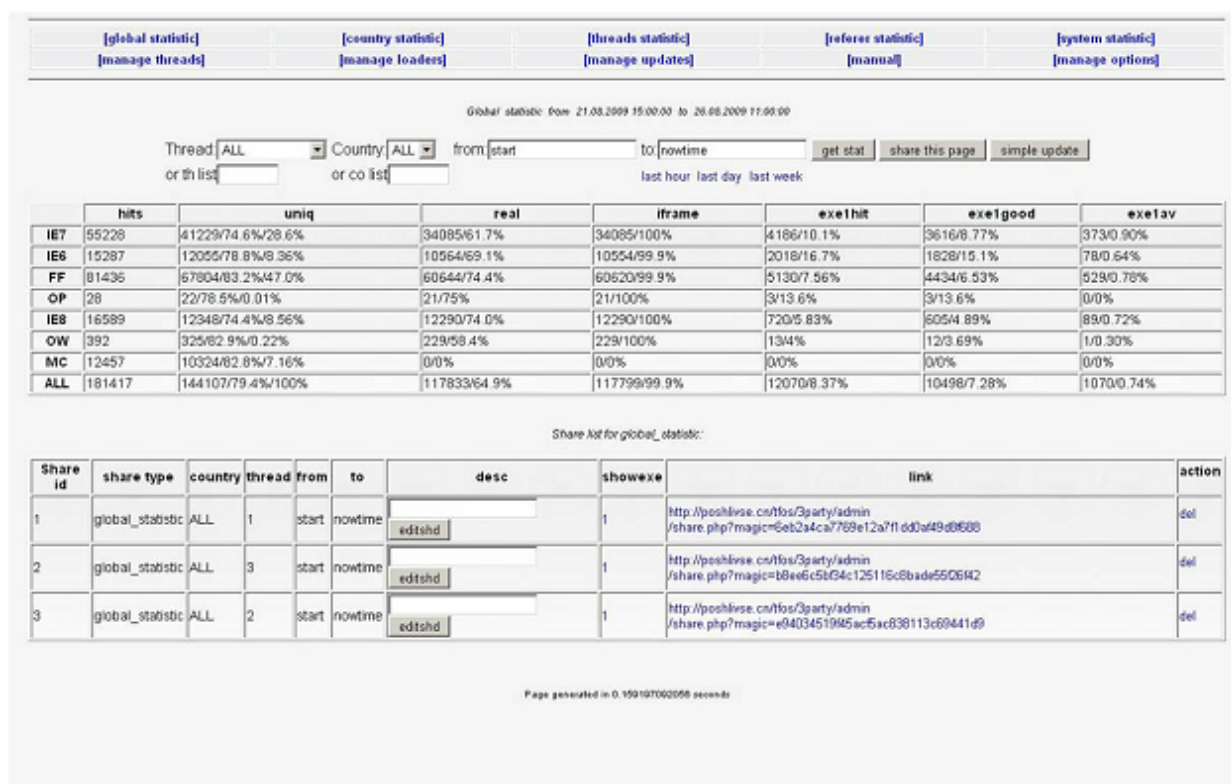


Figure 5.1: Screenshot of trojan's admin panel

parameters, the installation is ready to be deployed. At this time, the victim is still unaware of this infection. Note: the victim's antivirus scanner was running in the background and did not detect this infection. The attack is now ready to execute.

The victim logs into his online banking system to transfer €53,94, but the malware running on the user's PC changes the amount to €8576,31 and changes the destination to the drop name or *Money Mule*¹. If the online banking system requests validation of this transfer, the malware will rewrite the pages changing the transfer amount back to victim's original amount of €53,94. The user is completely unaware and believes the transfer occurred as normal, and thus validates the transfer. Even if the victim returns to view the bank statement or activity after the transaction, the malware will automatically modify the user's view of the transaction or statement to appear as though only the amount of €53,94 was transferred. Below is the transaction balance as it appears for the user, which shows the original amount. The log that the attacker sees 5.4 shows the real story along with other information that has

¹Cybercriminals need to hide their tracks as much as possible. In order to do this, they use unsuspecting middlemen called Money Mules. These are people that the stolen funds get transferred to first; the cyber-criminal might use several of these Money Mules to further muddle the trail. All the money transfers are simple wire transfers supported by any bank.

NAME:	POST1
DROPSTATUS:	ENABLED <input type="button" value="v"/>
IF ACC BALANS "<":	-1
MIN_AMOUNT:	4000
MAX_AMOUNT:	15000
LEAVE % (default 23):	5
CORRECT (default 100):	0
DROPNAME:	VIKTOR KLOSE
KONTONUMMER:	103009706
BLZ:	60010070
C1:	Ref Num 995345
C2:	Ref Num 995345
C3:	Ref Num 995345
C4:	Ref Num 995345
COMMENT:	Tigr
BROWSER:	ie: <input checked="" type="checkbox"/> explore: <input checked="" type="checkbox"/> firefox: <input checked="" type="checkbox"/> mozilla: <input checked="" type="checkbox"/> avant: <input checked="" type="checkbox"/> maxthon: <input checked="" type="checkbox"/> MyIE: <input checked="" type="checkbox"/>

Figure 5.2: Configuration Options for the Attacker

been captured, such as login names, pass codes, etc. This log is how the attacker can monitor and track what has been happening with this particular installation, how the funds will be received, which Money Mules are being used, and so on. At this point, one might think the attacker has covered most ways in which a user would access his account information but it goes even further than that. Even if the user downloads a PDF version of their bank statement, the Man-in-the-Middle malware can change the PDF information to re-create the PDF file, disguising the true transaction amount. Only when the bank account runs out of funds or when the user gets a physical statement or calls his bank is the deception uncovered.

The main crucial details that emerge from this example include:

- The transaction is carried out from the victim's PC

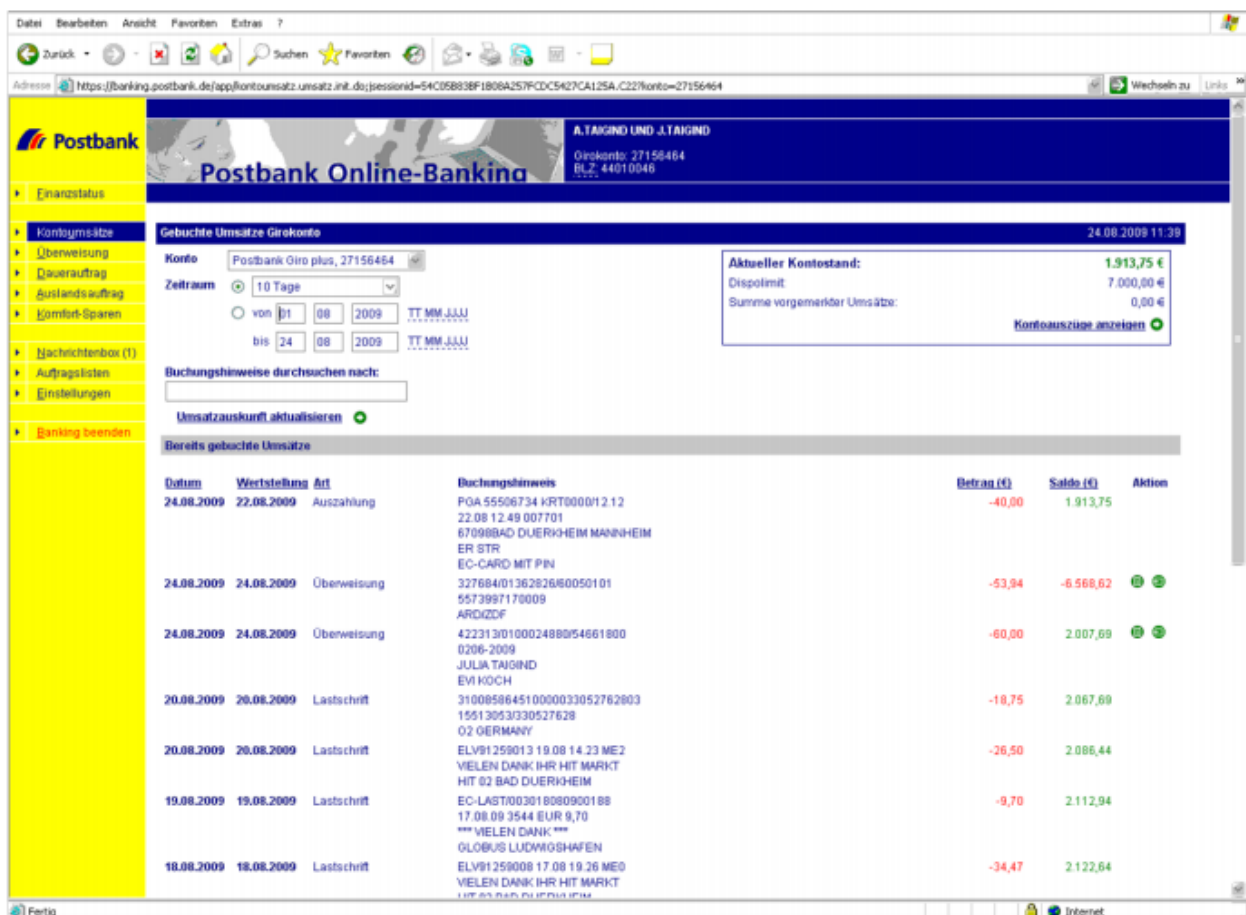


Figure 5.3: Transaction Balance

- The money is sent to a local bank account
- Each fraud transaction has a different (semi-random) amount that tries to circumvent different traits banks look for in detecting fraudulent transactions
- The transaction conforms to the restrictions imposed on the compromised account by the bank
- The amount is lower than the threshold of transactions that need to be further examined
- Money Mules' accounts are only used to a certain degree (no more than X amount of Euros in a certain period)

```

09:39:04 2009-08-24 GMT FJFAFJP1HAWOHNCAIN
NAME=POST1
USERHOST=postbank.de
USERACC=27156464
USERPASS=q3q2b
BALANS=2027.69
INET_LIMIT=15000.00
DISPO_LIMIT=7000.00
MAXBETRAG=
BLZ=60050101
TRUEAMOUNT=53,94
AMOUNT=8576,31
%DROP_BLZ%=|LBBW/BW-BANK STUTTGART|
%DROPNAME%=|VIKTOR KLOSE|
%KONTONUMMER%=|103009706|
%BLZ%=|60010070|
%C1%=|Ref Num 995345|
%C2%=|Ref Num 995345|
%C3%=|Ref Num 995345|
%C4%=|Ref Num 995345|
COMMENT: Tigr
EXINF=
DATE: 24.08.2009
VERSN: iexplore.exe 6.0.2900.2180
IP: 77.1.142.125
*****

```

Figure 5.4: Attacker trojan's log

5.1.4 Nowadays: Mitigation Strategies

This section analyzes some actually strategies used by organizations to contrast MITB attacks, in particular we focus on pros and cons of these strategies to understand why MITB is still dangerous.

Hardened Browser All browser vendors should make sure that Userscripts, Extensions and BHOs can't be easily run on SSL protected websites. To create a secure browser, must be disallowed any extensions / browser helper objects, and compile it into one static binary.

PROS

- Could be made available on every Desktop additionally to the user's normal insecure browser, so as to be used in parallel when high security sites are visited.
- Better usability than a Secure Live-Distribution
- Only few changes are needed to current industry standard browsers, mostly to strip them down and hard-compile them.

CONS

- No reliable way for the server to identify the use of a hardened browser, and differentiate between secure and insecure client access.
- There is substantial work in creating parallel browser distro.

Unwriteable Distribution An evolving solution from the open source world is that of Live-Distributions, being distributions of client operating systems running from read-only media such as CDROMs, DVDs or USB pens. This allows the client to bootup securely, and to reboot securely any time it is needed.

PROS

- A client achieves a very high security grade, and can likely withstand all currently validated threats.

CONS

- It is a severe usability problem for the users to reboot their computer. For many users, loosing from one to three minutes, and interrupting their entire workflow completely will be unacceptable.
- The approach assumes that there is no trojan in the platform BIOS (eg. rootkit)

Virtual Machine A suggested variation for *Unwriteable Distribution* usability problem is to run the secure operating system in a virtual machine. The risk on the other hand is that attacks that are used against the browsers in the local environment on the host system can also spread into the guest system by affecting the live system or the harddisk image.

PROS

- It raises the cost of the attack
- Usability is better than a Secure Live-Distribution

CONS

- Usability is not as good as a Hardened Browser
- It will probably not be secure for long. If sufficiently popular, it can be challenged and broken easily as VM-manipulation techniques are already quite sophisticated both for positive and negative purposes. It is likely that local attacks can be automatically applied to guest-applications from outside the VM.

Smart Card Reader A theoretical solution for a secure client would be the Class 4 Smart-Card reader. Although not available, the device characteristics and security profile are relatively well understood, and appropriate. Class1 Readers are just interfaces without any security mechanisms, Class2 Readers have a keyboard, but no display, Class3 Readers have a small display, Class4 Readers would have to have a secure viewer

PROS

- It is based on secure hardware

CONS

- It is an additional device to the PC
- There are no such devices available on the market yet
- Drivers would need to be installed on the computer
- High price

Transaction based Applications Transaction-based systems have the problem of authorisation of the transaction, as opposed to authentication of the user. Is the proposed transaction really the transaction the user wanted done that way, or has it been modified in flight (as MITB do)? The main idea is to use uncorrelated channels (e.g. PC and phone).

PROS

- The two channels - web page and cell/mobile - are uncorrelated, so the attacker would have to achieve control over both channels. Two attacks on unrelated platforms is much more expensive.

CONS

- The process is very dependent on the application.
- Telephone, SMS, FAX and Pager can have cost ramifications for the server-operators.

There are also others and different approach with their pros and cons, it is not our goal to analyze all of them.

5.1.5 A new approach: Transaction Monitoring

The success of the MITB attacks highlight the false sense of security that many types of authentication solutions can give IT/Security teams within organizations. In the case of MITB, deploying advanced authentication solutions like smartcards or PKI have long been considered sufficient protection against identity theft techniques. However, since the MITB attack piggybacks on authenticated sessions rather than trying to steal or impersonate an identity, most authentication technologies are incapable of preventing its success. What emerges by the analysis of current scenario is that MITB attacks are hard to detect without transaction monitoring and protection. ***Transaction monitoring*** refers to an organization's ability to monitor and identify suspicious post-login activities, a capability most often provided by a risk based fraud monitoring solution.

Our collaborative environment works to aim the goal of monitor and analyze user activities after a proper login from server's log files provided by collaborative's system participants.

Chapter 6

Implementations and Performance Evaluation

With the purpose to evaluate the goodness of the proposed architecture in 4.4 some experimental tests have been made, particularly we tried to evaluate the impact of Privacy-Preserving mechanism (*Observability*) on the performances and on the reliability of the system.

As explained above, our system is based on log analysis. We want to test our architecture against Man-in-the-Browser attack, so the main idea is: starting from data provided by participants (logs) build a blacklist of suspected IBAN¹, they will correspond to the Mule Accounts, and can thus be blocked.

Let's see how these logs are organized.

6.1 Short Analysis of Log Files

We wrote a java program for random generation of log files. They are characterized by this structure:

```
(source_ip,destination_ip,start_date,end_date,HTTP
operation,url,user_session_id,iban[optional])
```

For every user session there're four lines in the log, two lines for login/logout operations and the others two are randomly chosen from a list of four operations

1. check_balance

¹The International Bank Account Number (IBAN) is an international standard for identifying bank accounts across national borders with a minimal of risk of propagating transcription errors. It was originally adopted by the European Committee for Banking Standards (ECBS), and was later adopted as an international standard under ISO 13616:1997 and now as ISO 13616-1:2007. The official IBAN registrar under ISO 13616-2:2007 is SWIFT.

2. check_movement
3. do_bank_transfer
4. pay_bill

Only two of them require an IBAN number (bank transfer and pay bill), in the others two IBAN was set as “n/a”. The code presented in E.1 shows code written to generate log files. To decide a properly size of log files the bank’s account situation in Italy has been analyzed. In Italy there are about 14 million bank’s accounts (4 million companies [11] and 10 million individuals), these accounts are managed by 700 different banks([34]) for a total of 20000 users for every bank. Using the *Indicatore Sintetico di Costo (ICS)*([35]) provided by *Banca d’Italia* users/firms profiles were created (see table 6.1) to estimate number of daily, monthly and every 3 and 6 months operations, so four types of logs have been created. Users’ profiles have been used to estimate a *threshold*, it represent the maximum number of transactions

Table 6.1: Indicatore Sintetico di Costo (ICS)

user type	num_operations	period
standard user	<1	daily
firms	1	daily
standard user	2	monthly
firms	20	monthly
standard user	6	3months
firms	60	3months
standard user	12	6months
firms	>120	6months

received by an IBAN (every IBAN belong to an user/firm) in a fixed period, after which it can be regarded as suspicious.

As said before, private information are contained into log files, figure 6.1 shows them (in red). These information **must** remain private unless certain conditions are verified.

```

204.97.171.187 1.1.1.1 2011-08-10 11:42:50.2 2011-08-10 11:43:56.2 GET /test/pay_bill.php HTTP/1.1 http://1.1.1.1/test/pay_bill.php
SOURCE IP DESTINATION IP SESSION ID IBAN

```

Figure 6.1: Log

6.2 System accuracy - evaluation parameters

In order to assess the system's performances four values have been considered:

1. True Positive (TP)
2. False Positive (FP)
3. True Negative (TN)
4. False Negative (FN)

(i) **True Positive** (TP) represents the number of IBAN marked as suspicious and that they are true suspect; (ii) **False Positive** (FP) represents the number of IBAN marked as suspicious but that they are honest (they are a *detection error*); (iii) **True Negative** (TN) represents the number of IBAN that are not marked as suspect and that are truly honest; (iv) **False Negative** (FN) represents the number of IBAN that are not marked as suspect but that they are (*missed detection*).

With these values we computed two metrics:

Detection Accuracy (DA): $\frac{TP}{TP+FN}$

Error Level (EL): $\frac{FP}{FP+TN}$

These tests have been realized using two different approaches:

First approach We have used the framework Hadoop/Hive mainly assembling us on the *Detection Accuracy* and *Error Level*. In the section 6.3 we present further details.

Second approach We have used the *Esper* engine and focused our attention in achieving better time performances, possibly real-time detection. (maintaining the same *Detection Accuracy* and *Error Level* reached with the first approach). In section 6.4 we present in detail how *Esper* has been used.

6.3 First Approach - Hadoop/Hive

There were used Virtual Machines (see Appendix D) to simulate an Hadoop/Hive cluster. Communication between gateway and proxies has been realized through Java Socket. In particular there were created SSL socket to perform **secure** communication. Every proxy has been realized as a multithread server (see E.2), it is in listening mode on a specific port, when a client (gateway) ask for a connection, proxy starts a new thread that manage the client and the communication with it. Also gateways have been realized as multithread

component (see E.3). When a client starts it creates one thread for every proxy that must be contacted, every thread manages communication with its related proxy and encryption operation.

The encryption phase is one of the most important step to guarantee privacy. We have developed two different Java class, the first one manages the AES encryption used by gateways and the second one manages RSA encryption (listings E.4 and E.5 show these class). As mentioned in section 6.1, in absence of real data, we have developed a Java program that automatically creates log files to use as database for the tests. We have also created another Java program that allows us to submit query to Hive through Thrift Server, this program take as input three parameter

num_participants - INTEGER it is possible to set easily the number of participants and so the number of log files that must be analyzed.

threshold - INTEGER it is possible to set the desiderated threshold. This parameter allows to easily perform tests varying the threshold

privacy - BOOLEAN Through this parameter we can communicate to the system if log files are in clear or not.

Figure 6.2 shows UML class' diagram of this software. Before submit a query through Thrift server it has been necessary to setup a connection with it. The code in E.6 shows how we had setup it using JDBC² HiveDriver. As said before, we tried to automate the process of query submission. Code E.7 shows a snippet of written code that interact with Thrift server, this code submit a query on loaded logs, without using Hive command line interface.

During tests we hypothesized an attack spread of 5% [8]. This information has been merged with the users's profile showed in table 6.1 to obtain the estimated maximum number of operation for a single user, so we have derived possible threshold.

Three different scenarios have been analyzed, using three types of logs

- Daily logs
- Monthly logs
- 3 Months logs

Before being able to performs our tests it has been necessary to prepare the structures useful to our purpose. Remembering the structures of log files (see 6.1), an Hive table has been created with the following statements

²Java DataBase Connectivity, commonly referred to as JDBC, is an API for the Java programming language that defines how a client may access a database.

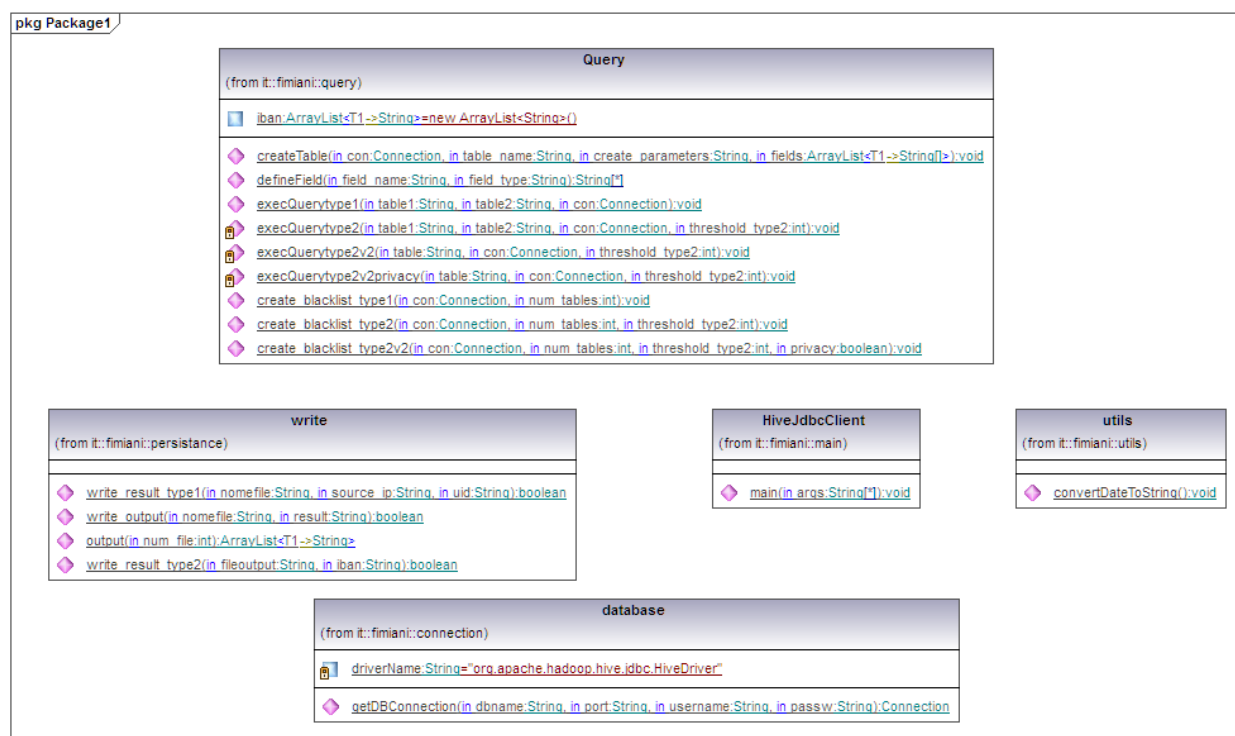


Figure 6.2: Class Diagram

```

CREATE TABLE evaluation (source_ip STRING, dest_ip STRING,
    start_date STRING, end_date STRING, op STRING,
    url STRING, uid STRING, iban STRING) PARTITIONED
    BY (bank STRING) ROW FORMAT DELIMITED FIELDS
    TERMINATED BY '\t';
  
```

Listing 6.1: Hive CREATE TABLE

Hive's *CREATE TABLE* statement is very similar to SQL-statements, the difference resides in the possibility, provided by Hive, to create some *partitions* inside the table. In this case we have created one partition for every log file. Moreover, this EQL-statements allows us to specify how the data are organized inside the text file (we specify that files are organized in rows and fields in the same row are separated by the special character “*tab*”), allowing so, subsequently, to load the data into the table directly from log files through a bash³ script that we have written.

³Bash is a Unix shell written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell (sh).

```
#!/bin/sh
../hive-0.7.0/bin/hive -e "LOAD DATA LOCAL INPATH
'/home/fimiani/hive-0.7.0/mydata/bank1.txt' OVERWRITE
INTO TABLE evaluation PARTITION (bank='bank1')"

../hive-0.7.0/bin/hive -e "LOAD DATA LOCAL INPATH
'/home/fimiani/hive-0.7.0/mydata/bank2.txt' OVERWRITE
INTO TABLE evaluation PARTITION (bank='bank2')"

../hive-0.7.0/bin/hive -e "LOAD DATA LOCAL INPATH
'/home/fimiani/hive-0.7.0/mydata/bank3.txt' OVERWRITE
INTO TABLE evaluation PARTITION (bank='bank3')"

../hive-0.7.0/bin/hive -e "LOAD DATA LOCAL INPATH
'/home/fimiani/hive-0.7.0/mydata/bank4.txt' OVERWRITE
INTO TABLE evaluation PARTITION (bank='bank4')"
```

Listing 6.2: HDFS load files

6.3.1 Scenario 1: four participants - daily logs

Two tests have been done. In the first one, we have done experiments using “*clear logs*”, *clear logs* means that no privacy mechanisms have been enabled, in the second one we made test after that sensible information have been crypted. So, we have compared system’s performance with and without privacy to evaluate the cost of preserving it.

A *daily log* consists of 6665 rows in a txt file. Figures 6.3 shows *Detection Accuracy* achieved in tests performed. These graphs show, clearly, that our system is able to achieve a very good detection accuracy if the threshold is properly chosen, so, as expected, the choice of threshold is a critical aspect. It can be also visible that Privacy-Preserving mechanisms don’t effect the *Detection Accuracy*, if we use the same threshold, we get the same percentage of accuracy with or without privacy.

Figure 6.4 shows the *Error Level* reached by our application, with or without Privacy-preserving mechanism enabled. Also in this case, preserving privacy don’t change the reached value respect to no-privacy tests.

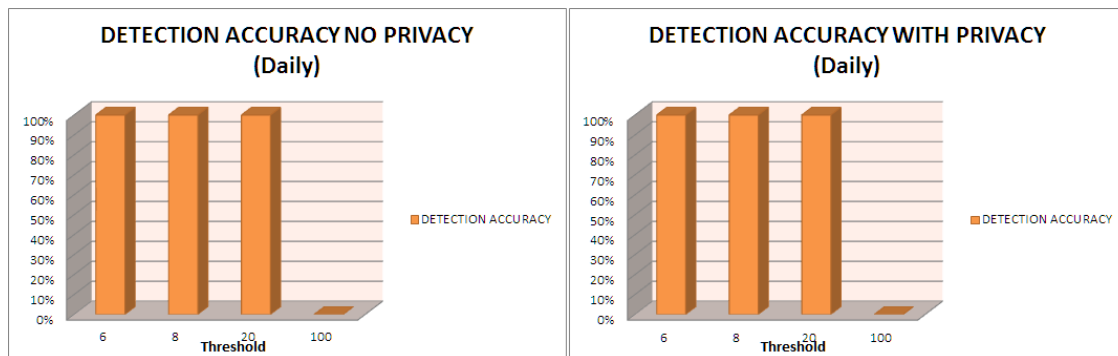


Figure 6.3: Detection Accuracy with or without Privacy (daily)

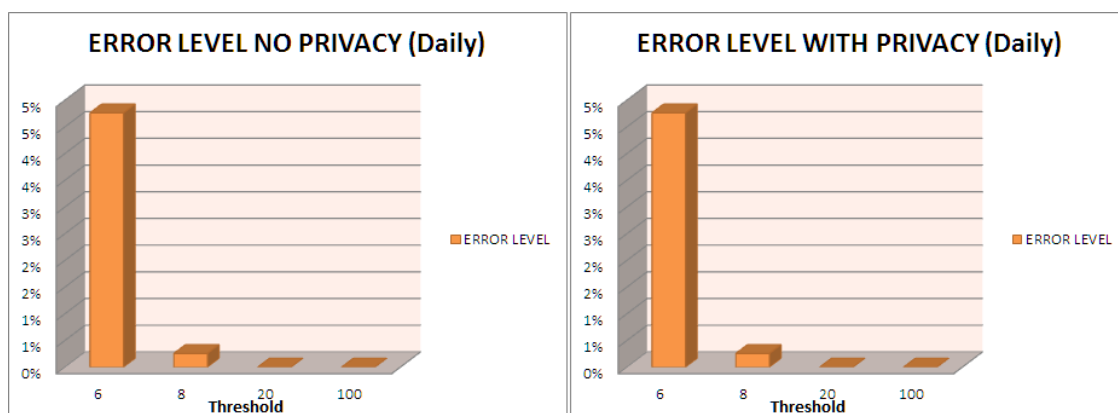


Figure 6.4: Error Level with or without Privacy (daily)

6.3.2 Scenario 2-3: four participants - monthly/3months logs

We decided to test our system with bigger logs respect to the first scenario. As previous, also in this one comparative tests have been done to evaluate the cost of preserving privacy. New logs consist of about 26656 (monthly) and 80000 (3months) rows respectively for every file.

The procedure used to perform the tests is exactly the same as previous, figures 6.5 , 6.6, 6.7, 6.8 show obtained result. As previous, tests allow us to state that privacy mechanisms don't affect the Detection Accuracy neither Error Level.

. We have done also some simulations to evaluate query time and error level in relation of the number of participants. Figure 6.9 show that the query time increase linearly with the number of participants. Figure 6.10 show how, with more participants (and thus with more available data), the Error Level decreases, as expected. In next section we analyze the cost of privacy preserving on system performance.

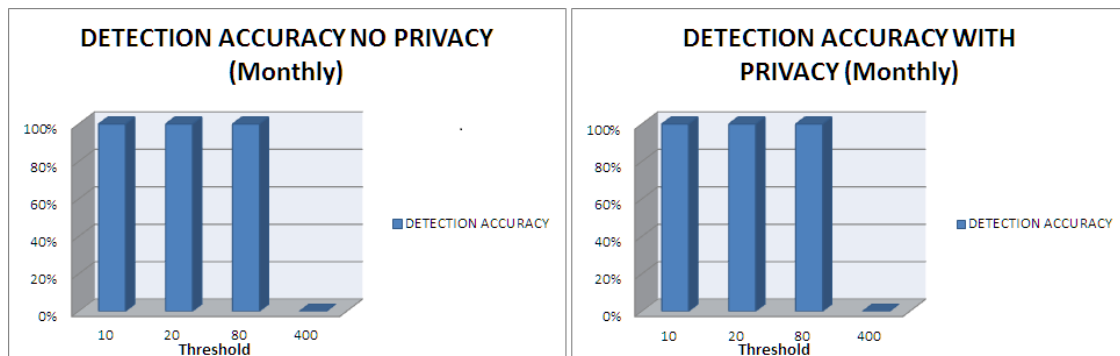


Figure 6.5: Detection Accuracy with or without Privacy (monthly)

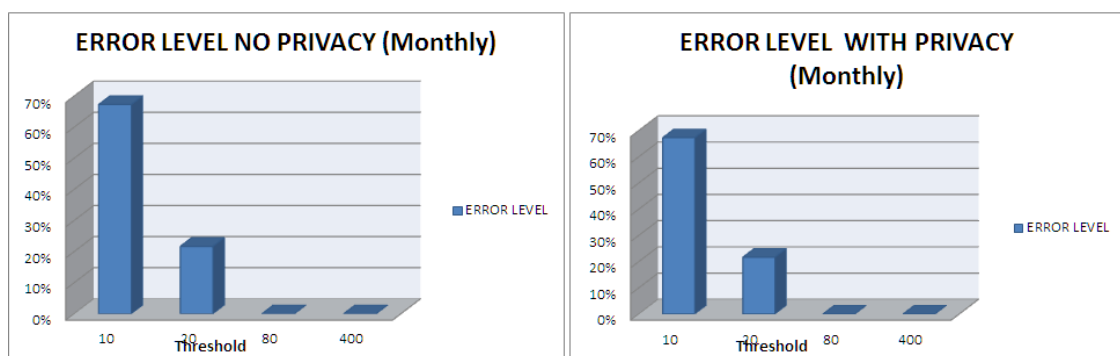


Figure 6.6: Error Level with or without Privacy (monthly)

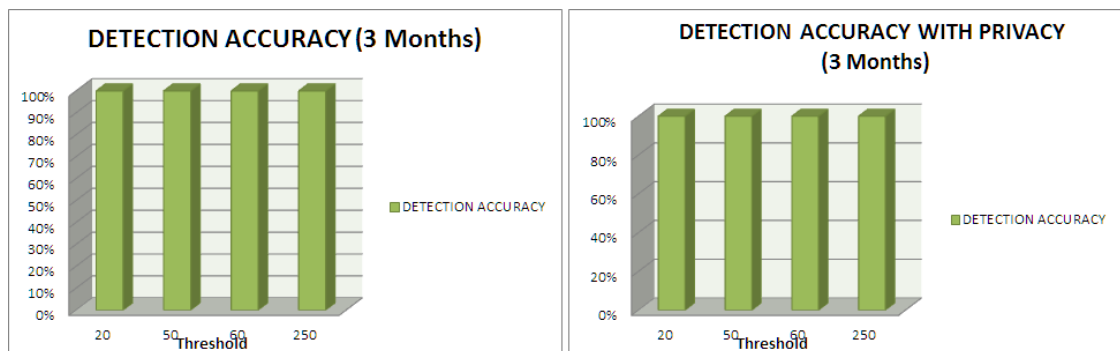


Figure 6.7: Detection Accuracy with or without Privacy (3 months)

6.3.3 System Performances with or without Privacy

Our architecture with the use of Hadoop and Hive allows to preserve privacy **without losing Accuracy and without increasing Error Level** respect to no-privacy behaviour. In this section we analyze the impact of privacy on system performance. Figure 6.11 shows total execution time of difference simulations with Hadoop/Hive, this graph clearly shows that

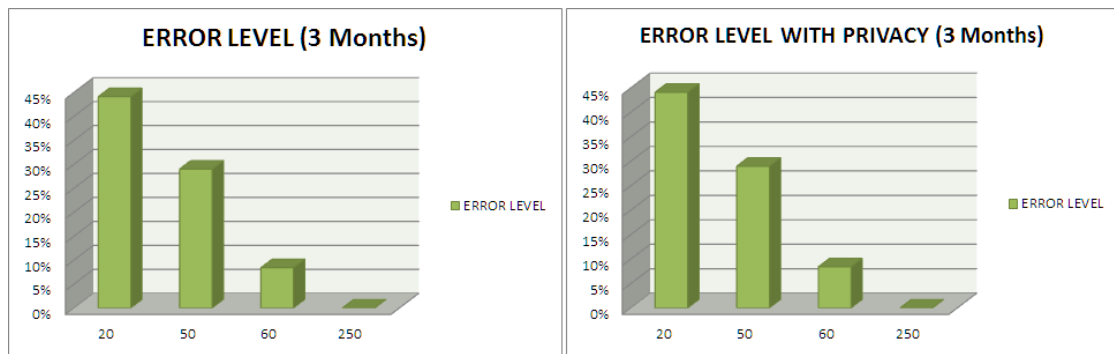


Figure 6.8: Error Level with or without Privacy (3 months)

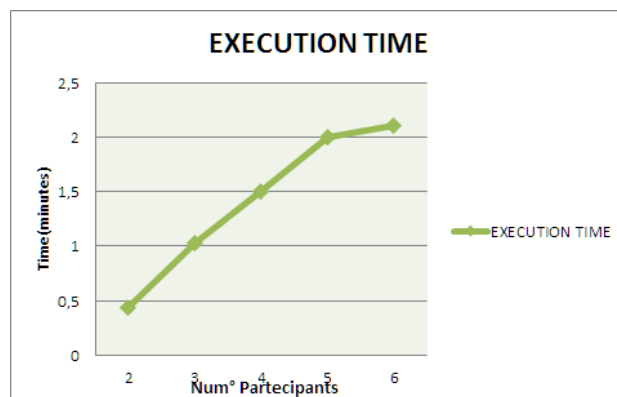


Figure 6.9: Query Execution Time (using 3 months logs no privacy)

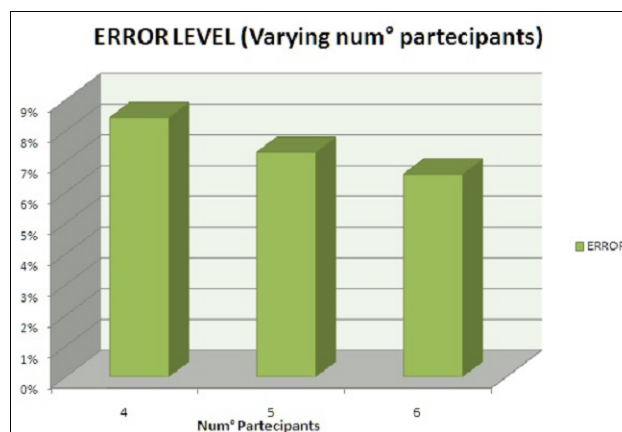


Figure 6.10: Error Level Trend

privacy-preserving mechanisms have a big impact on execution time. Figure 6.12 shows in detail how time is consumed during an execution with privacy mechanisms enabled. The analysis of these result state that *Encryption* and *HDFS Loading* have a big impact on

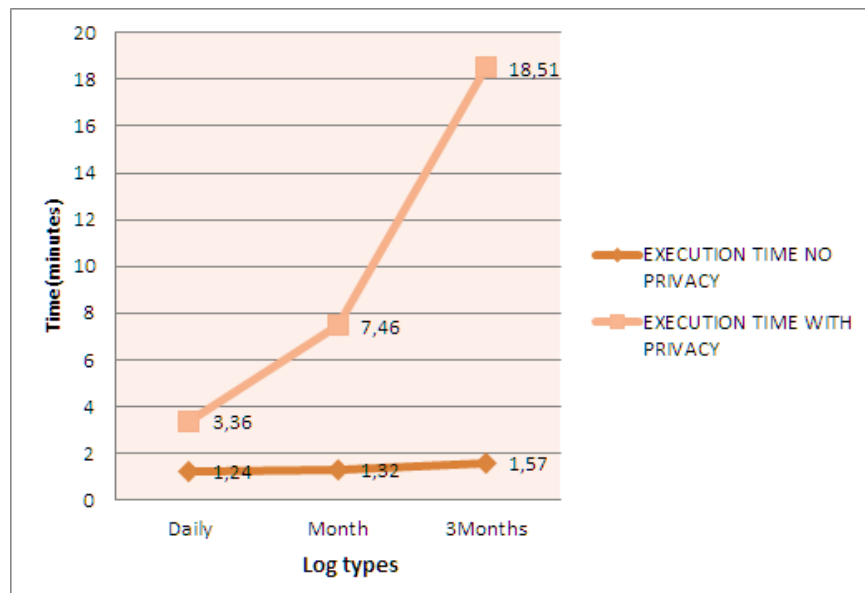


Figure 6.11: Total Execution Time

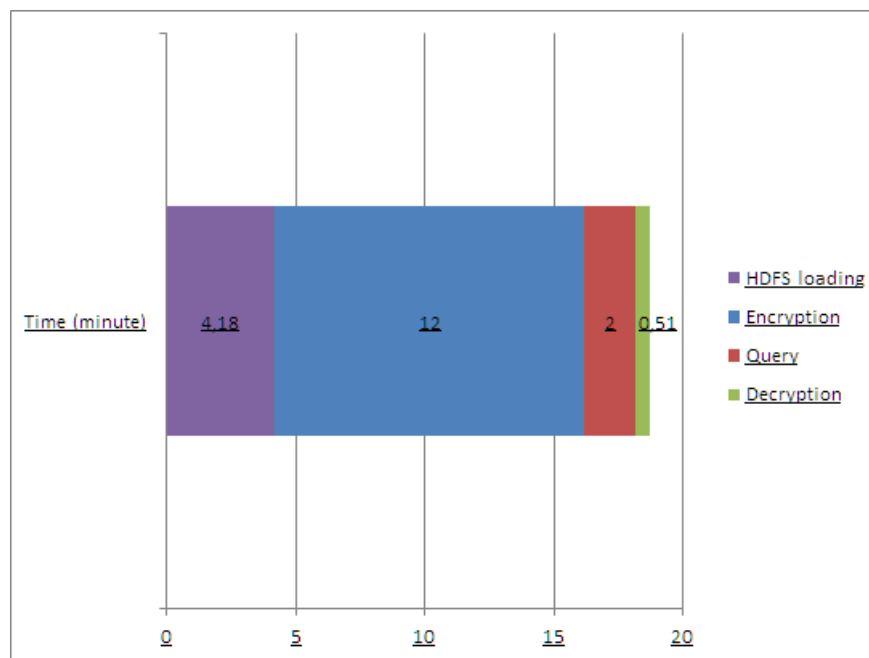


Figure 6.12: Total Time Elapsed (3 months with privacy)

performances while decryption time doesn't affect total time because blacklist are often characterized by a small number of items (see 6.13 for the different impact of Encryption and Decryption).

To understand if HDFS Loading time is bounded by MySQL database (used by Hive to store metadata), a loading test on MySQL was performed. During this test data have been

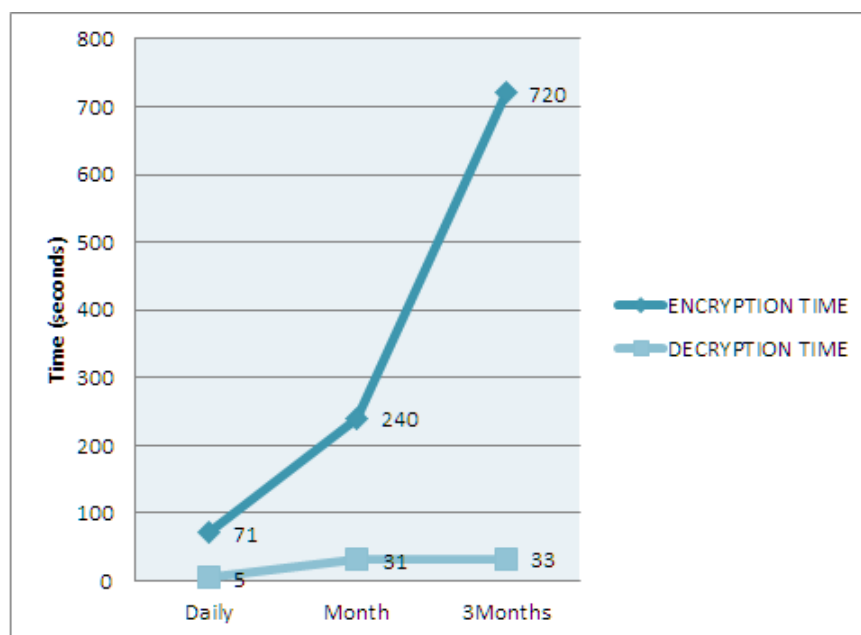


Figure 6.13: Encryption/Decryption impact on total time

inserted directly into a MySQL table through this statement.

```
LOAD DATA INFILE 'combined_daily_plain.txt'
INTO TABLE evaluation FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n';
```

Listing 6.3: MySQL load files

As figure 6.14 shows, HDFS loading time is totally used by MySQL to insert data into its tables.

We decided to understand if MySQL can be used not only for store metadata, but also for querying them directly, without using Hadoop/Hive. Figure 6.15 shows that with our data MySQL outperforms Hadoop/Hive. However, Hadoop/Hive allows to have a processing time constant, while with mySQL we have a processing time increasing with data. The size of logs files has been also investigated to analyze if it changes when privacy is enabled. As shown in figure 6.16, the logs' size after encryption is, more or less, twenty-three/seven times than original.

6.4 Second Approach - From Hadoop/Hive to ESPER

Results coming from tests described in the previous section enables us to state that the proposed architecture, implemented through the use of Hadoop/Hive, provides a high degree of

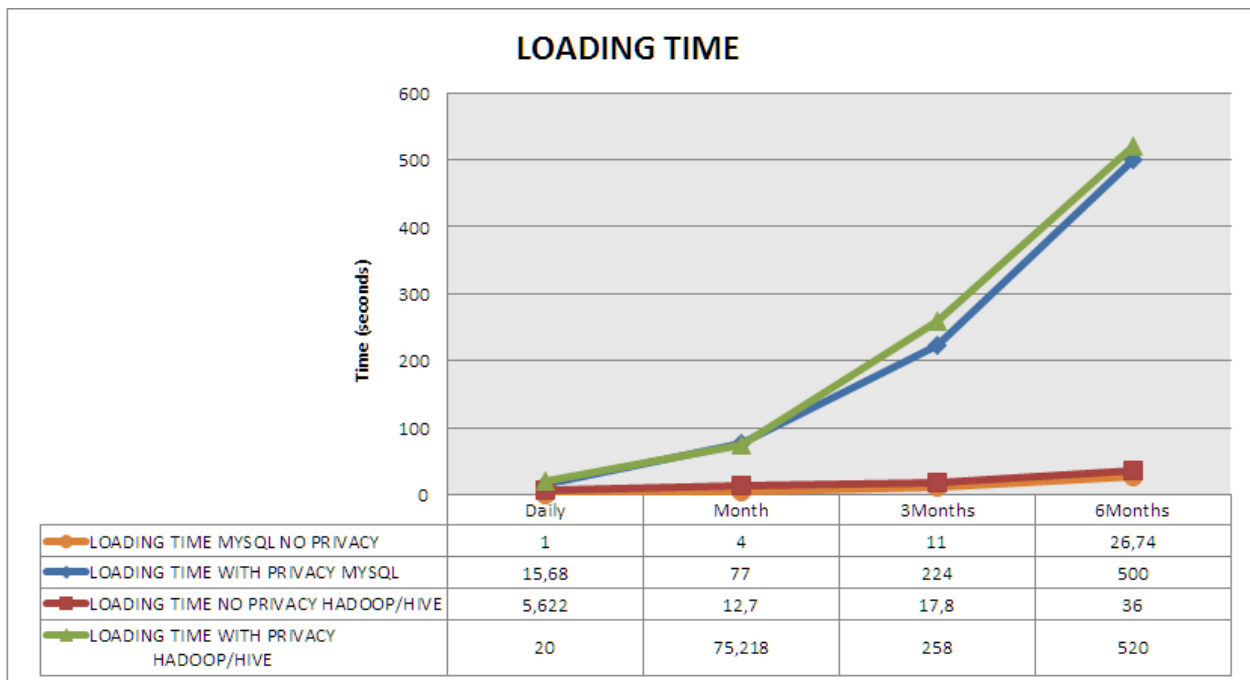


Figure 6.14: MySQL vs Hadoop/Hive Loading Time

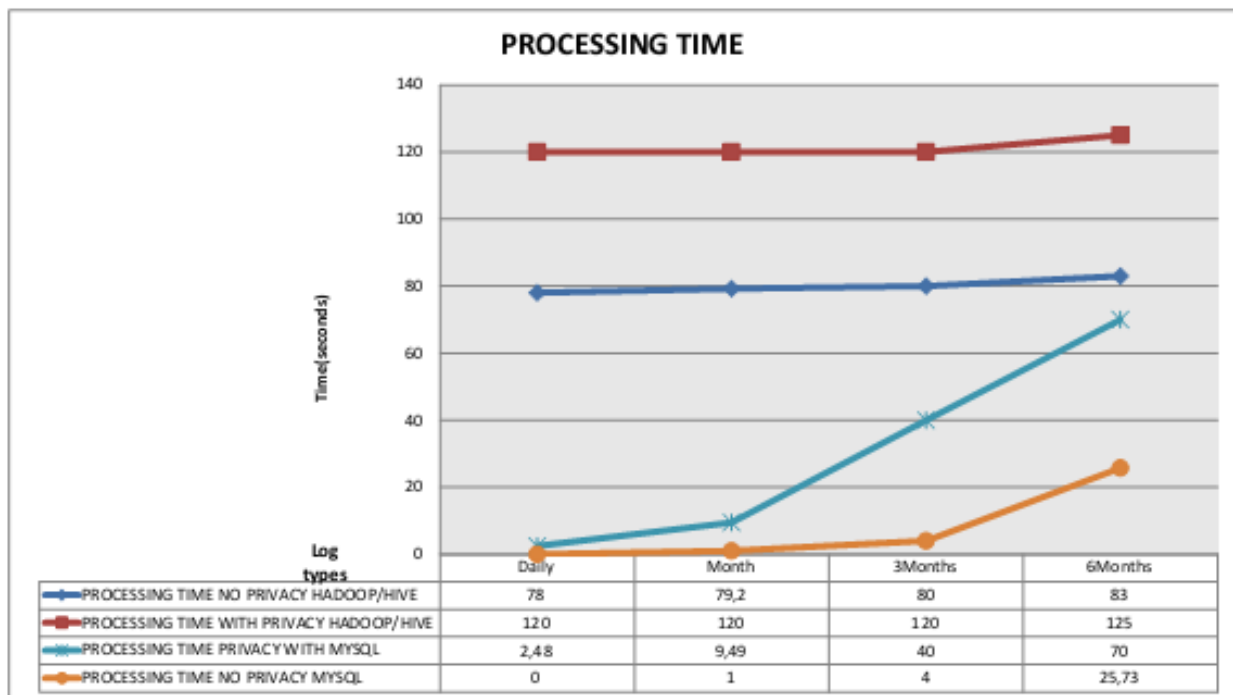


Figure 6.15: MySQL vs Hadoop/Hive Processing Time

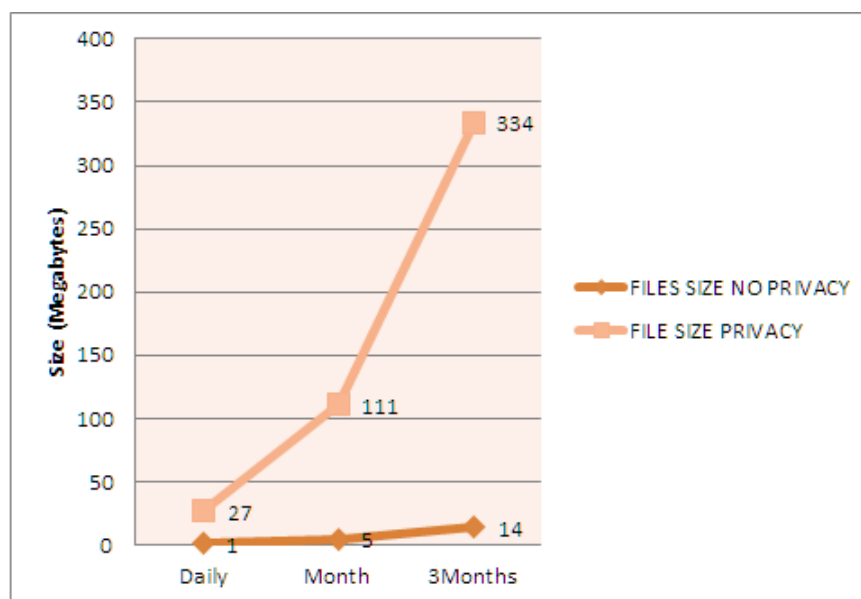


Figure 6.16: Logs' size before/after Encryption

detection and a low level of error, paying, however, this achievements in terms of resources and time required that make it unsuitable for real-time monitoring. Also, to get a result a large amount of data is necessary, because the resources required by Hadoop/Hive does not allow under any circumstances to make a real-time processing.

Fore these reasons it was decided to replace Hadoop/Hive with the ESPER engine. ESPER, as described in 3.5, is a component for complex event processing (CEP), available for Java, it enables rapid development of applications that process large volumes of incoming messages or events. It filter and analyze events in various ways, and respond to conditions of interest in real-time, its use for protection of critical infrastructures has been analyzed in [16].

The architecture remains the same, but now the log files are not loaded on HDFS but directly processed by ESPER engine. We have done some modification to gateways java code, because now the gateway encrypt data field by field and not file by file and send them to ESPER engine directly. Listing E.8 shows the new version of the gateway. The main idea is that when a participant process an operation, it crypt data, through the use of proxies, and then sends it to the ESPER engine that performs a query on them to check if the threshold is reached. Data have been sent to ESPER engine through a Java socket as shown in the code below:

```

outt.writeObject(new LogEvent(source_ip,destination_ip,start_date,end_date,
    http_operation,url,user_session_id,iban));

```

Listing 6.4: How to send data to Esper engine

The `LogEvent` is a Java class (see E.9) that represents data provided by participants. The ESPER engine is always in listening mode as shown by the following code.

```
ConfigurationSocketAdapter adapterConfig = new ConfigurationSocketAdapter();
    SocketConfig socket = new SocketConfig();
    socket.setDataType(DataType.OBJECT);
    int port = 9001; // listening port
    socket.setPort(port);
    adapterConfig.getSockets().put("SocketService", socket);
    // start adapter
    EsperIOSocketAdapter socketAdapter = new EsperIOSocketAdapter(
        adapterConfig, "10.79.61.54");
    socketAdapter.initialize();
    socketAdapter.start();
    while(true);
```

Listing 6.5: Esper engine listening socket

When the `socketAdapter` receive a connection the `EventListener` starts and performs the query.

```
EPServiceProvider epService = EPServiceProviderManager.getProvider("
    10.79.61.54", configuration);
epService.getEPAdministrator().createEPL("insert into evaluation select
    iban from Log").addListener(new UpdateListener() {
    int i = 1;
    public void update(EventBean[] ebs, EventBean[] ebs1) {
        for (EventBean eventBean : ebs) {
            System.out.println("Iban ricevuto("+i+"): "+eventBean.get("iban"));
            i++;
        }
    }
});
epService.getEPAdministrator().createEPL(" "
    + "select iban, count(iban) from evaluation where iban <> 'n/a' group by
        iban having count(iban) > 40").addListener(new UpdateListener() {

    public void update(EventBean[] ebs, EventBean[] ebs1) {
        for (EventBean eventBean : ebs) {
            if(!((String)eventBean.get("iban")).equalsIgnoreCase("n/a")){
                System.out.println("***** ");
            }
        }
    }
});
```

```
System.out.printf ("time: %1$ta, %1$td %1$tb %1$tY, %1$tI:%1$tm:%1
    $tS.%1$tL %1$tp %1$tZ%n", SntpClient.getTime());
System.out.println("ALERT COMPUTATION RESULT:"+eventBean.get("iban")
    +" count:"+eventBean.get("count(iban)"));
System.exit(0);
}
}
}
});
```

Listing 6.6: Esper engine

If the threshold specified in the query is reached an *EventBean* starts and a *ALERT COMPUTATION RESULT* is printed.

Some simulations have been performed to analyze if ESPER may be an option for real-time monitoring against cyber attacks. These simulations shown that ESPER guarantees a *Detection Accuracy* and an *Error Level* equal to the previous solution. To evaluate the real-time performances we have introduced a new parameter: **Detection Latency**.

Detection Latency has been defined as time interval between gateway processes the transaction that will exceed the threshold (in clear) and the time when ESPER's engine rises the ALERT. Detection Latency, so, includes the time used to perform data encryption.

For the measure of *Detection Latency* has been used an NTP server, deployed on one of the gateways. All the components ask time to NTP server (see Listing E.10), in this way we can be sure that there are not error of synchrony. Figure 6.17 shows the *Detection Latency* achieved by our system.

Detection Latency is constant and near real-time. Time is consumed to process NTP server request and to encrypt data.

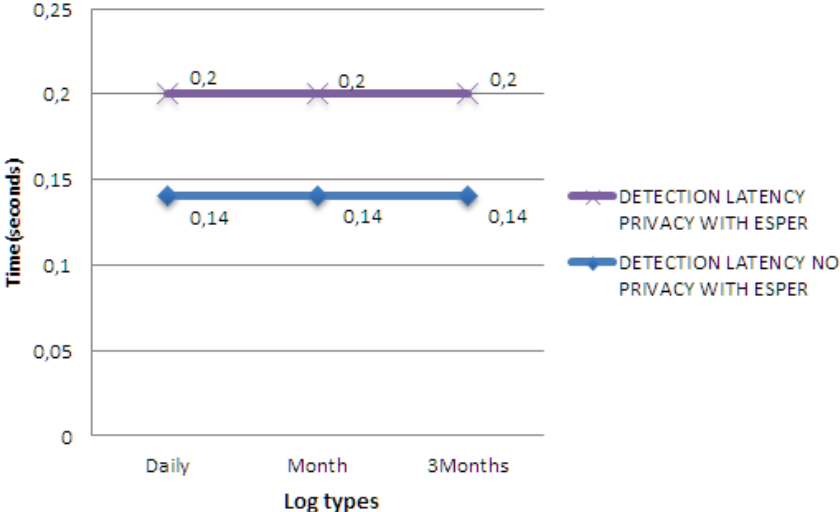


Figure 6.17: Detection Latency Esper

Chapter 7

Discussion

The wide spread of broadband connections around the world has led to an exponential increase of the number of services available via the web. The increase of transactions over the network has inevitably led to an increase of interest by attackers against online activities, often perceived as a profitable source of income thanks to the inexperience of the users and the lack of suitable forms protection by service providers. If we analyze the cyber attacks of the recent years it is easy to realize that they are no longer limited to one geographical area or against a single entity but they spread like wildfire around the world, exploiting any existing vulnerability. For this reason today in order to defend themselves and to prevent such attacks, companies must necessarily work together, gather all the information at their disposal to obtain sufficient data to plan a defense strategy. This type of approach is known as *Collaborative Environment* and it is becoming a standard in the fight against cyber crime. The participation to a *Collaborative Environment* often leads to the acceptance of some compromise by participants; one of these is the need to disclose sensitive information to other system components. One of the adopted solutions nowadays is the use of a **trusted third party** to entrust all those data; it (TTP) will be use for computing a result and then for providing it to all the participants. Although this solution allows to get to reliable results more easily, most of the firms are reluctant to entrust their data to a third party that must be trusted blindly.

The idea behind this thesis is aimed at trying to overcome this limitation by creating an architecture that makes use of third-party honest (or semi-honest) but not necessarily trusted. To do it we have used different tools such as **AES** symmetric encryption algorithms or **RSA** asymmetric encryption; it has been also built a structure of communication for sensitive information that also guarantees the privacy of the participants. Our main goal was to quantify the “*cost*” of the mechanisms introduced to preserve privacy. With the word “*cost*” we mean the impact that privacy preserving mechanisms have both in terms of accuracy and performance of the system. For this reason, tests have been carried out in parallel

with and without the privacy preserving mechanisms enabled, so we can have a clear and direct comparison. In order to simulate a true “field test”, it has been decided to use one of today’s most popular cyber attack as a use case: *Man-in-the-Browser*. The motivations that led us to choose such an attack not only reside in its spread, but also in the lack of adequate defense and appropriate monitoring tools, however our architecture can be used and adapted for different types of cyber attacks.

We have explored two different approaches, one based on the use of open source framework *Hadoop* supported by the framework *Hive*, the other one based on the engine *ESPER*. As shown in detail in Chapter 6, the tests done show that our architecture can have a **high detection capability** and a **very low error rate** if the fundamental parameter of the system (the *threshold*) is properly set. For these reasons we have tried to draw up a table that helps in the choice of this parameter (6.1). This positive achievement can be seen in both the first and the second approach, although they are inspired by two different philosophies of processing: *Hadoop/Hive* are designed to scale very well with the increasing data available, they may all be used exclusively for reporting operations because the processing times are significant and also need to have a fair number of aggregate data to provide a reliable result; *ESPER* allows to carry out a true real-time monitoring, so that the participants can have an immediate alert in case of attack and can take the appropriate counter-measures without losing the **high detection capability** and the **very low error rate** seen in the *Hadoop/Hive* solution.

There is no doubt that the work done must be seen as a first step closer to a definitive solution, but right now we can provide an “*embryonic*” defense mechanism that allows the companies to protect themselves from cyber-attack while both information and participants are privacy-preserved. The developed system, however, is not without flaws: first of all the heavy dependence on log files, the system is designed to work well with the log files structured in a certain way, changing them means having to change the code of the architecture (but not its structure), also the system at the present time is unable to guarantee complete privacy against *malicious models*.

Future developments will focus above all on these two aspects, trying to make the architecture independent from how the information is provided and particularly robust against a malicious scenario.

Appendix A

RSA Number Theory

RSA cryptography is based on the following theorems:

Theorem 1 (Fermat's Little Theorem). *If p is a prime number, and a is an integer such that $(a, p) = 1$, then*

$$a^{p-1} = 1(\text{mod } p)$$

Proof. Consider the numbers $(a \cdot 1), (a \cdot 2), \dots, (a \cdot (p-1))$, all modulo p . They are all different. If any of them were the same, say $a \cdot m = a \cdot n(\text{mod } p)$, then $a \cdot (m - n) = 0(\text{mod } p)$ so $m - n$ must be multiple of p . But since all m and n are less than p , $m = n$.

Thus $a \cdot 1, a \cdot 2, \dots, a \cdot (p-1)$ must be rearrangement of $1, 2, \dots, (p-1)$. So modulo p , we have

$$\prod_{i=1}^{p-1} i = \prod_{i=1}^{p-1} a \cdot i = a^{p-1} \prod_{i=1}^{p-1} i,$$

so $a^{p-1} = 1(\text{mod } p)$ □

Theorem 2 (Fermat's Theorem Extension). *If $(a, m) = 1$ then $a^{\phi(m)} = 1(\text{mod } m)$, where $\phi(m)$ is the number of integers less than m that are relatively prime to m . The number m is not necessarily prime.*

Proof. Same idea as above. Suppose $\phi(m) = n$. Then suppose that the n numbers less than m that are relatively prime to m are:

$$a_1, a_2, \dots, a_n$$

The $a \cdot a_1, a \cdot a_2, \dots, a \cdot a_n$ are also relatively prime to m , and must all be different, so they must just be a rearrangement of the a_1, \dots, a_n in some order. Thus:

$$\prod_{i=1}^n a_i = \prod_{i=1}^n a \cdot a_i = a^n \prod_{i=1}^n a_i,$$

modulo m , so $a^n = 1(\text{mod } m)$. □

Theorem 3 ((Chinese Remainder Theorem)). *Let p and q be two numbers (not necessarily primes), but such that $(p, q) = 1$. Then if $a = b \pmod{p}$ and $a = b \pmod{q}$ we have $a = b \pmod{pq}$.*

Proof. Proof: If $a = b \pmod{p}$ then p divides $(a - b)$. Similarly, q divides $(a - b)$. But p and q are relatively prime, so pq divides $(a - b)$. Consequently, $a = b \pmod{pq}$. (This is a special case with only two factors of what is usually called the Chinese remainder theorem but it is all we need here.) \square

A.0.1 Proof of the Main Result

Based on the theorems above, here is why the RSA encryption scheme works. Let p and q be two different (large) prime numbers, let $0 < M < pq$ be a secret message¹, let d be an integer (usually small) that is relatively prime to $(p - 1)(q - 1)$, and let e be a number such that $de = 1 \pmod{(p - 1)(q - 1)}$. (We will see later how to generate this e given d .) The encoded message is $C = M^e \pmod{pq}$, so we need to show that the decoded message is given by $M = C^d \pmod{pq}$.

Proof. Since $de = 1 \pmod{(p - 1)(q - 1)}$, $de = 1 + k(p - 1)(q - 1)$ for some integer k . Thus:

$$C^{de} = M^{de} = M^{1+k(p-1)(q-1)} = M \cdot (M^{(p-1)(q-1)})^k$$

If M is relatively prime to p , then

$$M^{de} = M \cdot (M^{p-1})^{k(q-1)} = M \cdot (1)^{k(q-1)} = M \pmod{p}$$

By the extension of Fermat's Theorem giving $M^{p-1} = 1 \pmod{p}$ followed by a multiplication of both sides by M . But if M is not relatively prime to p , then M is a multiple of p , so equation 1 still holds because both sides will be zero, modulo p . By exactly the same reasoning,

$$M^{de} = M \cdot M^{q-1} = M \pmod{q}$$

If we apply the Chinese remainder theorem to equations 1 and 2, we obtain the result we want: $M^{de} = M \pmod{pq}$.

Finally, given the integer d , we will need to be able to find another integer e such that $de = 1 \pmod{(p - 1)(q - 1)}$. To do so we can use the extension of Fermat's theorem to get $d^{\phi((p-1)(q-1))} = 1 \pmod{(p-1)(q-1)}$, so $d^{\phi((p-1)(q-1))-1} \pmod{(p-1)(q-1)}$ is suitable value for e . \square

¹If the message is long, break it up into a series of smaller messages such that each of them is smaller than pq and encode each of them separately.

Appendix B

Two-factor Authentication

Two-factor authentication (TFA, T-FA or 2FA) is an approach to authentication which requires the presentation of two different kinds of evidence that someone is who they say they are. It is a part of the broader family of multi-factor authentication, which is a defense in depth approach to security. From a security perspective, the idea is to use evidences which have separate range of attack vectors (e.g. logical, physical) leading to more complex attack scenario and consequently, lower risk.

Two factor authentication implies the use of two independent means of evidence to assert an entity, rather than two iterations of the same means. “Something one knows”, “something one has”, and “something one is” are useful simple summaries of three independent factors. In detail, these factors are:

- what the requestor individually knows as a secret, such as a password
- what the requesting owner uniquely has, such as a passport, physical token, or ID-card
- what the requesting bearer individually is, such as biometric data, like a fingerprint or face geometry

It is generally accepted that any independent two of these authentication methods is ***two-factor authentication***. Traditional hardware tokens, SMS, and telephone-based methods are vulnerable to a type of attack known as the man-in-the-middle, or MITB attack. In such an attack the fraudster impersonates the bank to the customer and vice versa, prompting the victim to divulge to them the value generated by their token. This means they do not need to be in physical possession of the hardware token or telephone device to compromise the victim’s account, but only have to pass the disclosed value on to the genuine website within the time limit. Citibank made headline news in 2006 when its hardware token-equipped business customers were targeted by just such an attack from fraudsters based in

the Ukraine. Such an attack may be used to gain information about the victim's accounts, or to get them to authorise a transfer of a different sum to a different recipient than intended.

Appendix C

Advanced Standard Encryption

AES is the Advanced Encryption Standard, a United States government standard algorithm for encrypting and decrypting data. The standard is described in Federal Information Processing Standard (FIPS) 197 [33].

AES is a symmetric block cipher with a block size of 128 bits. Key lengths can be 128 bits, 192 bits, or 256 bits;¹ called AES-128, AES-192, and AES-256, respectively. AES-128 uses 10 rounds, AES-192 uses 12 rounds, and AES-256 uses 14 rounds.

The main loop of AES² performs the following functions:

- SubBytes()
- ShiftRows()
- MixColumns()
- AddRoundKey()

The first three functions of an AES round are designed to thwart cryptanalysis via the methods of “confusion” and “diffusion”. The fourth function actually encrypts the data. Diffusion means patterns in the plaintext are dispersed in the ciphertext. Confusion means the relationship between the plaintext and the ciphertext is obscured. A simpler way to view the AES function order is:

1. Scramble each byte (SubBytes).
2. Scramble each row (ShiftRows).
3. Scramble each column (MixColumns).

¹The Rijndael algorithm supported additional key lengths and block sizes which are not supported in AES.

²SubBytes() is called once before the first AES round. The final AES round omits the MixColumns() function

4. Encrypt (AddRoundKey).

A term associated with AES is “the State”, an *intermediate cipher*, or the ciphertext before the final round has been applied. AES formats plaintext into 16 byte (128-bit) blocks, and treats each block as a 4x4 State array. It then performs four operations in each round. The arrays contains row and column information used in the operations, especially MixColumns() and Shiftrows().

C.0.2 Attacks on AES

The most successful attack on AES to date is the **Square Attack**, based on the Square Cipher, which was also created by the authors of Rijndael. It *exploits the byte-oriented structure of Square cipher*. This attack is also valid for Rijndael, as Rijndael inherits many properties from Square [9]. The Square Attack is faster than a brute force attack for AES using six rounds or less. For seven rounds or more, brute force attacks are the fastest known attacks. AES uses 10-14 rounds, based on the key length. Brute forcing AES-128 (smallest key length) is unlikely to be practical in the foreseeable future. According to NIST³, “Assuming that one could build a machine that could recover a DES key in a second (i.e., try 2^{55} keys per second), then it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key.”

³National Institute of Standards and Technology

Appendix D

Configure virtual machines with Hadoop and Hive

In this appendix, the required steps for setting up a Hadoop cluster using the Hadoop Distributed File System (HDFS) on Ubuntu Linux will be described.

D.1 Running Hadoop on Single Node Cluster

This experiment has been tested with the following software version:

- Ubuntu Linux 11.04
- Hadoop 0.20.2
- Hive 0.7.0

Prerequisites

Sun Java Hadoop requires a working Java installation. To install it on Ubuntu

```
$ sudo apt-get install sun-java6-jdk
```

After installation, make a quick check wheter Sun JDK is correctly set up:

```
user@ubuntu :~# java -version
javaversion "1.6.020"
Java (TM) SE Runtime Environment (build1.6.0_2-b02)
Java HotSpot (TM) ClientVM (build 16.3-b01,mixed mode ,
sharing)
```

SSH Hadoop requires SSH access to manage its nodes. If cluster doesn't have the ssh requisite software will need to install it.

```
$ sudo apt-get install ssh
$ sudo apt-get install rsync
```

Now, we have to generate an SSH key for the user:

```
hduser@ubuntu:~$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
9b:82:ea:58:b4:e0:35:d7:ff:19:66:a6:ef:ae:0e:d2 hduser@ubuntu
The key's randomart image is:
[...snipp...]
hduser@ubuntu:~$
```

Now, we have to enable SSH access to our local machine with this newly created key.

```
hduser@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >>
                        $HOME/.ssh/authorized_keys
```

Installation Hadoop We have to download Hadoop from the Apache Download Mirrors and ex-tract the contents of the Hadoop package to a location of our choice. (eg. /usr/local/hadoop)

```
$ cd /usr /local
$ sudo tar xzf hadoop-0.20.2.tar.gz
$ sudo mv hadoop-0.20.2 hadoop
```

Now, we have to add the following lines to the end of \$HOME/.bashrc file.

```
JAVA_HOME=/usr/lib /jvm/java-6-sun
export JAVA_HOME
PATH=$PATH:$JAVA_HOME/bin
export PATH
HADOOP_HOME=/home/hduser/hadoop
export HADOOP_HOME
```

```
PATH=$PATH:$HADOOP_HOME/bin
export PATH
```

The only required environment variable we have to configure for Hadoop is `JAVA_HOME`. Open `/conf/hadoop-env.sh` in the editor and set the `JAVA_HOME` environment variable to the Sun JDK JRE 6 directory.

```
#The java implementation to use. Required.
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

conf/*-site.xml In this section, we will configure the directory where Hadoop will store its data files, the network ports it listens to, etc. Our setup will use Hadoop's Distributed File System, HDFS.

We have to change `hadoop.tmp.dir` variable. We will use the directory `/app/hadoop/tmp` in this tutorial. Hadoop's default configurations use `hadoop.tmp.dir` as the base temporary directory both for the local system and HDFS. In `conf/core-site.xml`.

```
<!-- In : conf/core-site.xml >
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary
directories.</description>
</property>
```

In `conf/mapred-site.xml`

```
<!-- In: conf /mapred-site.xml >
<property>
<name>mapred.job.tracker</name>
<value>localhost:54311</ v a l u e >
<description>The host and port that the MapReduce
job tracker runs at. If "local", then jobs
are run in process as a single map
and reduce task.
</description>
</property>
```

In `conf/hdfs-site.xml`

```
<!-- In: conf/hdfs-site.xml>
<property>
<name>dfs.replication</name>
<value>1</value>
<description>Default block replication.
The actual number of replications can be
specified when the file is created.
The default is used if replication is
not specified in create time.
</description>
</property>
```

The first step to starting up our Hadoop installation is formatting the Hadoop filesystem which is implemented on top of the local filesystem of our cluster (which includes only our local machine). We need to do this the first time we set up a Hadoop cluster.

```
user@ubuntu :~$ /usr/local/hadoop/bin/hadoop namenode -format
```

To start Hadoop

```
user@ubuntu :~$ /usr/local/hadoop/bin/start-all.sh
```

This will startup a Namenode, Datanode, Jobtracker and a Tasktracker on machine.

D.2 Running Hadoop on Multi Node Cluster

We will build a multi-node cluster using two Ubuntu boxes. The best way to do this is to install, configure and test a local Hadoop setup for each of the Ubuntu boxes, and in a second step to merge these single-node clusters into one multi-node cluster in which one Ubuntu box will become the designated master (but also act as a slave with regard to data storage and processing), and the other box will become only a slave.

D.2.1 Networking

Both machines must be able to reach each other over the network. The easiest is to put both machines in the same network (eg. 192.168.56.x/24). To make it simple, we will assign the IP address 192.168.56.2 to the master machine and 192.168.56.3 to the slave machine. Update /etc/hosts on both machines with the following lines:

```
192.168.56.2 master
```

```
192.168.53.3 slave1
```

Figure D.1 and D.2 show a simple multi node cluster architecture. The master node will

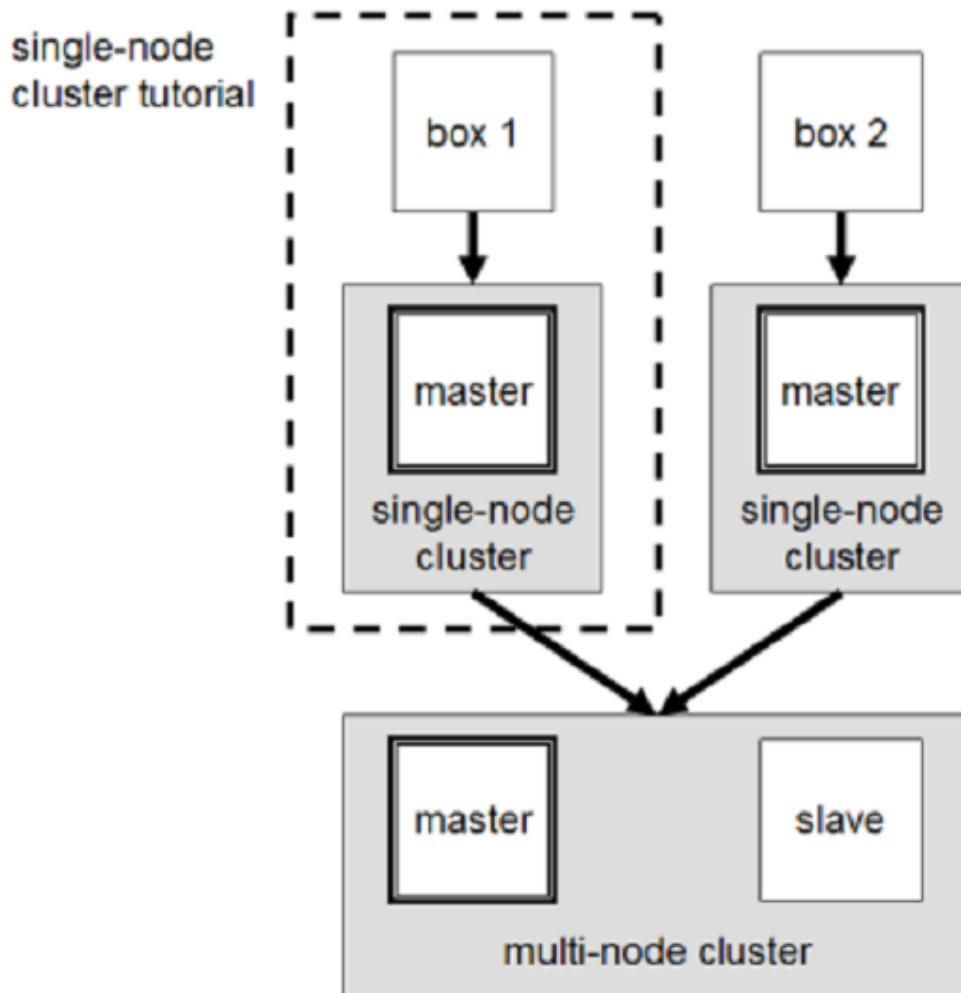


Figure D.1: Multi Node Cluster Architecture

also act as a slave because we only have two machines available in our cluster but still want to spread data storage and processing to multiple machines. The master node will run the master daemons for each layer: namenode for the HDFS storage layer, and jobtracker for the MapReduce processing layer. Both machines will run the slave daemons: datanode for the HDFS layer, and tasktracker for MapReduce processing layer. Basically, the master daemons are responsible for coordination and management of the slave daemons while the latter will do the actual data storage and data processing work. Typically one machine in the cluster is designated as the NameNode and another machine the as JobTracker, exclusively. These are the masters. The rest of the machines in the cluster act as both DataNode and TaskTracker.

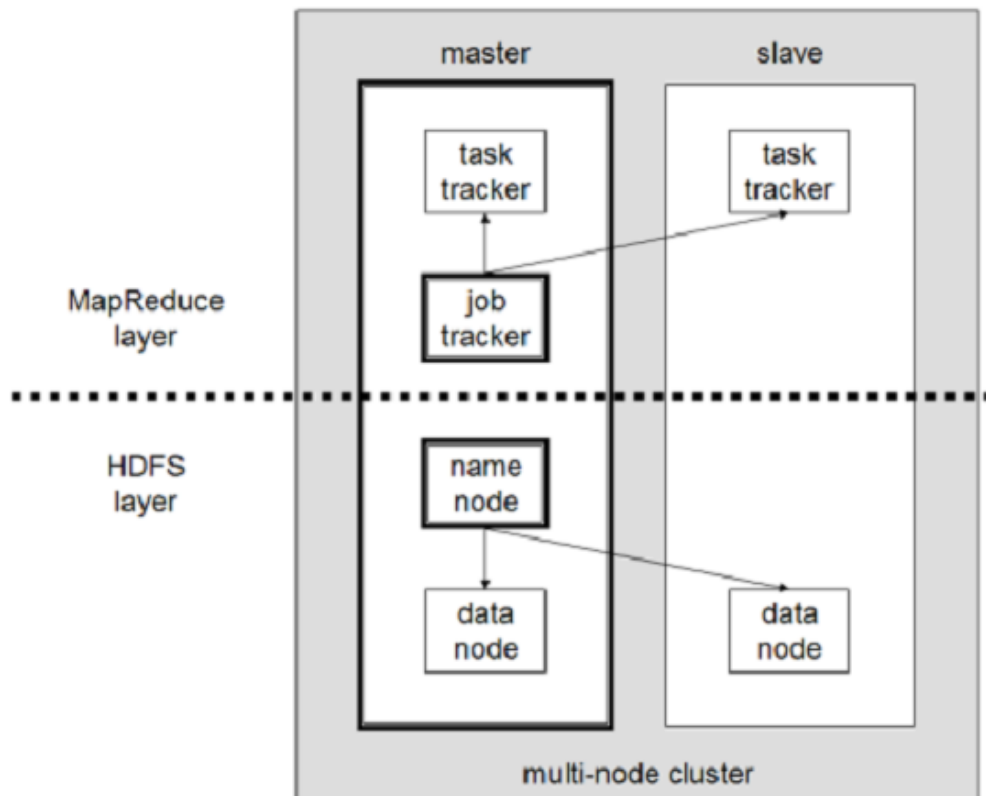


Figure D.2: Multi Node Cluster Architecture

These are the slaves.

On master, update `/conf/masters` that it looks like this:

```
master
```

This `conf/slaves` file lists the hosts, one per line, where the Hadoop slave daemons (datanodes and tasktrackers) will be run. We want both the master box and the slave box to act as Hadoop slaves because we want both of them to store and process data. On master (only), update `/conf/slaves` that it looks like this:

```
master
```

```
slave
```

We have to change some parameters in `conf/*-site.xml` (on all machines). First, we have to change the `fs.default.name` variable (in `conf/core-site.xml`) which specifies the NameNode (the HDFS master) host and port. In our case, this is the master machine.

```
<!-- In: conf/core-site.xml >
<property>
<name>fs.default.name</name>
<value>hdfs://master:54310</value>
```

```
<description>The name of the default file system .  
A URI whose scheme and authority determine  
the File System implementation. The uri's  
scheme determines the config property(fs .SCHEME.impl)  
naming the File System implementation class.  
The uri's authority is used to determine the  
host, port, etc. for a filesystem.</description>  
</property>
```

Second, we have to change the *mapred.job.tracker* variable (in *conf/mapred-site.xml*) which specifies the JobTracker (MapReduce master) host and port. Again, this is the master in our case

```
<!-- In: conf/mapred-site.xml >  
<property>  
<name>mapred.job.tracker</name>  
<value>master:54311</value>  
<description>The host and port that  
the MapReduce job tracker runs at.  
If "local", then jobs are run in process  
as a single map and reduce task.  
</description>  
</property>
```

Third, we change the *dfs.replication* variable (in *conf/hdfs-site.xml*) which specifies the default block replication. It defines how many machines a single file should be replicated to before it becomes available

```
<!-- In: conf/hdfs-site.xml >  
<property>  
<name>dfs.replication</name>  
<value>N</value>  
<description>Default block replication.  
The actual number of replications can be  
specified when the file is created.  
The default is used if replication is  
not specified in create time.  
</description>  
</property>
```

Where N is the number of cluster elements.

Appendix E

Java Code

```
package it.fimiani.main;
import java.util.ArrayList;
import java.util.Random;
import net.sourceforge.jpcap.net.IPAddress;
import it.fimiani.persistence.write;
import it.fimiani.utils.utils;
/**
 * @author Pasquale Fimiani
 *
 */
public class genlog {
    static public String source_ip = "n/a";
    static public String destination_ip;
    static public String start_date;
    static public String end_date;
    static public String operation;
    static public String url;
    static public String uid;
    static public boolean ip_yet_inserted = true;
    /**
     * args[0] = num log to generate
     * args[1] = num row for every log
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("start");
    }
}
```

```
ArrayList<String> ips = new ArrayList<String>();
for(int j = 0; j < Integer.parseInt(args[0]); j++){
  for (int i = 0; i < Integer.parseInt(args[1]); i++){
    destination_ip = (j+1)+"."+j+1)+"."+j+1)+"."+j+1);
    String[] date = utils.dataCasuale();
    start_date = date[0];
    end_date = date[1];
    source_ip = utils.getRandomIPAddress();
    while(ip_yet_inserted){
      ip_yet_inserted = false;
      if(ips.contains(source_ip)){
        int rand= utils.getRandom(1000);
        if(rand <= 2){
          ip_yet_inserted = false;
        }else{
          source_ip = utils.getRandomIPAddressv2();
          ip_yet_inserted = true;
        }
      }
    }
    ip_yet_inserted = true;
    ips.add(source_ip);
    uid = utils.getUID(source_ip, start_date, end_date);
    operation = utils.getOperazione("login");
    url = utils.getURL(destination_ip,"login");
    write.write_log(source_ip,destination_ip,start_date,end_date,operation,
      url,uid,j+1);
    operation = utils.getOperazione("logout");
    url = utils.getURL(destination_ip,"logout");
    write.write_log(source_ip,destination_ip,date[2],date[3],operation,url,
      uid,j+1);
  }
}
System.out.println("end");
}
```

Listing E.1: genlog.java - Log generator

```
/**
 * @author Fimiani
 *
 */
public class Server extends Thread
{
    private ServerSocket Server;
    private SSLServerSocket sslserversocket;
    private SSLServerSocketFactory sslserversocketfactory;
    /**
     * @param argv
     * @throws Exception
     */
    public static void main(String argv[]) throws Exception
    {
        new Server();
    }
    /**
     * @throws Exception
     */
    public Server() throws Exception
    {
        sslserversocketfactory = (SSLServerSocketFactory) SSLServerSocketFactory.
            getDefault();
        sslserversocket =
            (SSLServerSocket) sslserversocketfactory.createServerSocket(4000);
        System.out.println("Server is listening on port 4000.");
        this.start();
    }
    public void run()
    {
        while(true)
        {
            try {
                SSLSocket client = (SSLSocket) sslserversocket.accept();
                System.out.println("Connection accepted from: "+
```

```
        client.getInetAddress());
    Connect c = new Connect(client,true);
}
catch(Exception e) {
    e.printStackTrace();
}
}
}
}
}
/**
 * @author Fimiani
 *
 */
class Connect extends Thread
{
    static int privacy_level = 1024;
    private SSLSocket client = null;
    ObjectInputStream in = null;
    ObjectOutputStream out = null;
    private boolean pkenabled;
    public Connect() {}
    /**
     * @param clientSocket
     * @param mode
     */
    public Connect(SSLSocket clientSocket, boolean mode)
    {
        client = clientSocket;
        pkenabled = mode;
        try
        {
            //open object stream
            out = new ObjectOutputStream(new BufferedOutputStream(client.
                getOutputStream()));
            out.flush();
            in=new ObjectInputStream(new BufferedInputStream(client.getInputStream())
                );
        }
    }
}
```

```
}
catch(Exception e1)
{
    try { client.close(); }
    catch(Exception e) { System.out.println(e.getMessage());}
    return;
}
this.start();
}
public void run()
{
    String todo="";
    String message = "";
    String operationid = null;
    Object o = null;
    byte[] ipreceived = new byte[32];
    while(true){
        if(client!=null){
            try
            { todo = (String)in.readObject();
              if(todo.equalsIgnoreCase("000")){
                while(!(message=(String) in.readObject()).equalsIgnoreCase("000-END")){
                    System.out.println("message = "+(String)message);
                    operationid = message;
                }
                System.out.println("Starting RSA key generation for operationid= "+
                    operationid);
                if(pkenabled){
                    //start generation of RSA value
                    RSA rsa = new RSA();
                    KeyPair pq;
                    pq = rsa.generateRSAkey(privacy_level,operationid);
                    out.writeObject(pq.getPublic());
                    out.flush();
                    System.out.println("Ending generation of RSA data for operationid= "+
                        operationid);
                    client.close();
                }
            }
        }
    }
}
```

```
        client = null;
    }
}
}
catch(Exception e) {e.printStackTrace();}
}
}
}
```

Listing E.2: Proxy main class

```
/**
 * @author Fimiani
 *
 */
class Encrypt extends Thread
{
    static int privacy_level = 1024;
    static String enctype = null;
    secure secure;
    int encnum = 0;
    static int opid;
    PublicKey publickey = null;
    static boolean ready[] = new boolean[4];
    public Encrypt(String type, secure s, int encnumber, int oid, PublicKey pk)
    {
        this.enctype = type;
        this.encnum = encnumber;
        this.opid = oid;
        this.publickey = pk;
        if(enctype.equalsIgnoreCase("symmetric")){
            this.secure = s;
            this.start();
        }else{
            this.start();
        }
    }
}
```

```
public void run()
{
    System.out.println("Thread.encrypt starting encryption - enctype "+enctype
        );
    if(enctype.equalsIgnoreCase("symmetric")){
        ready[0]=false;
        ready[1]=false;
        ready[2]=false;
        ready[3]=false;
        if(ready[0]==false){
            readwrite.encryptfiledata("bank1.txt", "bank1-bank1e-"+String.valueOf(
                opid)+".txt", secure);
            ready[0]=true;
        }
    }
    else{
        switch (encnum) {
        case 1:
            try{
                while(ready[0]==false){
                    sleep(5000);
                }
                System.out.println("start RSA encryption with proxy 1 publickey");
                RSA rsa = new RSA();
                rsa.encryptRSA(opid, 1, publickey, privacy_level);
                System.out.println("end RSA encryption with proxy 1 publickey");
            }catch(Exception e){
                e.printStackTrace();
            }
            ready[1]=true;
            break;
        case 2:
            try{
                while(ready[1]==false){
                    System.out.println("waiting RSA encryption with proxy 1 publickey");
                    sleep(5000);
                }
            }
        }
    }
}
```

```
        System.out.println("start RSA encryption with proxy 2 publickey");
        RSA rsa = new RSA();
        rsa.encryptRSA(opid, 2, publickey, privacy_level);
        System.out.println("end RSA encryption with proxy 2 publickey");
    }catch(Exception e){
        e.printStackTrace();
    }
    ready[2]=true;
    break;
case 3:
    try{
        while(ready[2]==false){
            System.out.println("waiting RSA encryption with proxy 2 publickey");
            sleep(5000);
        }
        System.out.println("start RSA encryption with proxy 3 publickey");
        RSA rsa = new RSA();
        rsa.encryptRSA(opid, 3,publickey, privacy_level );
        System.out.println("end RSA encryption with proxy 3 publickey");
    }catch(Exception e){
        e.printStackTrace();
    }
    ready[3]=true;
    break;
default:
    break;
}
}
System.out.println("Thread.encrypt ending encryption");
}
}
/**
 * @author Fimiani
 *
 */
public class Client extends Thread
{
```

```
ObjectInputStream in = null;
ObjectOutputStream out = null;
SSLSocket socket = null;
static int operationid = 1569877455;
boolean ready[] = new boolean[3];
int nserver = 0;
static secure secure;
String tdo = "";
public static void main(String argv[])
{
    try {
        secure = new secure();
        secure.keygen("AES", "gateways_secret_seed");
        new Encrypt("symmetric", secure, 0, operationid, null).join();
    } catch (Exception e) {
        e.printStackTrace();
    }
    new Client("000",0, 4000);
    new Client("000",1, 5000);
    new Client("000",2, 6000);
}

public Client(String todo, int nserv, int port){
    try
    {
        // open a socket connection
        SSLSocketFactory sslsocketfactory = (SSLSocketFactory) SSLSocketFactory.
            getDefault();
        socket = (SSLSocket) sslsocketfactory.createSocket("localhost", port);
        System.out.println("Open socket to "+socket.getPort());
        this.tdo = todo;
        this.nserver = nserv;
        this.start();
    }
    catch(Exception e) { System.out.println(e.getMessage());}
}

/**
```

```
* @return
*/
public boolean checkready(){
    for(int i = 0; i<4;i++){
        if(ready[i]==false) return false;
        else{System.out.println("true");}
    }
    return true;
}
/**
 * @param operation
 */
public void dosomething(int operation){
    switch (operation) {
    case 1:
        try{
            // 1 - ASK FOR PUBLIC KEY
            out.writeObject(new String("000"));
            out.writeObject(new String(String.valueOf(operationid)));
            out.writeObject(new String (String.valueOf("000-END")));
            out.flush();
            PublicKey pk = null;
            pk = (PublicKey)in.readObject();
            out.close();
            in.close();
            socket.close();
            System.out.println("public key received");
            if(nserver==0){
                new Encrypt("asymmetric", null, 1, operationid, pk);
            }else if(nserver==1){
                new Encrypt("asymmetric", null, 2, operationid, pk);
            }
            else if(nserver==2){
                new Encrypt("asymmetric", null, 3, operationid, pk);
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

```

    }
    break;
default:
    break;
}
}
public void run()
{
    String todo = tdo;
    try {
        out = new ObjectOutputStream(new BufferedOutputStream(socket.
            getOutputStream()));
        out.flush();
        in=new ObjectInputStream(new BufferedInputStream(socket.getInputStream())
            );
        if(todo.equalsIgnoreCase("000")){
            dosomething(1);
            todo = "";
        }
    }
    catch(Exception e) {e.printStackTrace();}
}
}

```

Listing E.3: Gateway main class

```

/**
 * @author Pasquale Fimiani
 * This program provides the following cryptographic functionalities
 * 1. Encryption using AES
 * 2. Decryption using AES
 *
 * High Level Algorithm :
 * 1. Generate a AES key (specify the Key size during this phase)
 * 2. Create the Cipher
 * 3. To Encrypt : Initialize the Cipher for Encryption
 * 4. To Decrypt : Initialize the Cipher for Decryption
 *

```

```
*
*/

public class secure {
    public static String stringToEncrypt = new String();
    public static String stringChiperText = new String();
    public static String stringDecryptedText = new String();
    public static Cipher aesCipher = null;
    public static SecretKey secretKey = null;
    public static byte[] byteChiperText = null;
    public static byte[] byteDataToEncrypt = null;
    public static KeyGenerator keyGen = null;
    /**
     * @param s
     * @return
     */
    public String encrypt(String s){
        try{
            secretKey = readwrite.getKey();
            createCipher();
            return doencryption(s);
        }catch(Exception e){
            System.out.println("secure.encrypt exception");
            System.out.println(e);
        }
        return "encryption_failed";
    }
    /**
     * @param s
     * @return
     */
    public String decrypt(String s){
        try{
            createCipher();
            SecretKey key = readwrite.getKey();
            return dodecryption(s,key);
        }
    }
}
```

```
}catch(Exception e){
    System.out.println(e);
}
return "dencryption_failed";
}
/**
 * @param alg
 * @param seed
 * @throws NoSuchAlgorithmException
 */
public void keygen(String alg,String seed) throws NoSuchAlgorithmException{
    keyGen = KeyGenerator.getInstance("AES");
    byte[] seedb = seed.getBytes();
    SecureRandom random = new SecureRandom(seedb);
    keyGen.init(128,random);
    secretKey = keyGen.generateKey();
    readwrite.setKey(secretKey);
}
/** 1 STEP
 * @throws NoSuchAlgorithmException
 * @throws NoSuchPaddingException
 * @throws InvalidKeyException
 */
private void createCipher() throws NoSuchAlgorithmException,
    NoSuchPaddingException, InvalidKeyException{
    aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.ENCRYPT_MODE,secretKey);
}
/**
 * @param toencrypt
 * @return
 * @throws IllegalBlockSizeException
 * @throws BadPaddingException
 */
private String doencryption(String toencrypt) throws
    IllegalBlockSizeException, BadPaddingException{
    stringToEncrypt = toencrypt;
```

```

byteDataToEncrypt = stringToEncrypt.getBytes();
byteCipherText = aesCipher.doFinal(byteDataToEncrypt);
stringChiperText = Base64.encodeBytes(byteCipherText);
return stringChiperText;
}
/**
 * @param todecrypt
 * @return
 * @throws InvalidKeyException
 * @throws InvalidAlgorithmParameterException
 * @throws IllegalBlockSizeException
 * @throws BadPaddingException
 * @throws IOException
 */
private String dodecryption(String todecrypt,SecretKey key) throws
    InvalidKeyException, InvalidAlgorithmParameterException,
    IllegalBlockSizeException, BadPaddingException, IOException{
secretKey = key;
aesCipher.init(Cipher.DECRYPT_MODE,secretKey,aesCipher.getParameters());
//byteCipherText = todecrypt.getBytes();
byteCipherText = Base64.decode(todecrypt);
byte[] byteDecryptedText = aesCipher.doFinal(byteCipherText);
stringDecryptedText = new String(byteDecryptedText);
System.out.println(" Decrypted Text message is " +stringDecryptedText);
return stringDecryptedText;
}
}

```

Listing E.4: AES java class

```

public class RSA {
static KeyPair keypair;
static Cipher cipher;
/**
 * @param operationid
 * @param encnum
 * @param pk
 */

```

```
public void encryptRSA(int operationid, int encnum, PublicKey pk, int
    keysize){
    KeyPairGenerator kpg;
    try{
        Security.addProvider(new BouncyCastleProvider());
        kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(keysize);
        keypair = new KeyPair(pk, null);
        readwrite.setRSAKey(keypair.getPublic(), "publickey"+encnum+1);
        cipher = Cipher.getInstance("RSA/NONE/NoPadding");
        System.out.println("encrypt #"+encnum);
        System.out.println(keypair.getPublic());
        FileInputStream fstream = new FileInputStream("bank1-bank"+(encnum)+"e
            -1569877455.txt");
        DataInputStream in = new DataInputStream(fstream);
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        FileWriter fstreamout = new FileWriter("bank1-bank"+(encnum+1)+"e
            -1569877455.txt");
        BufferedWriter out = new BufferedWriter(fstreamout);
        String string;
        while((string = br.readLine()) != null){
            StringTokenizer t = new StringTokenizer(string, "\t");
            int i = 1;
            while(t.hasMoreElements()){
                if((i==1) || (i==8)){
                    String toencrypt = t.nextToken().toString().trim();
                    RSA a = new RSA();
                    String crypted = a.encrypt(toencrypt);
                    out.write(crypted+"\t");
                }else{
                    String toencrypt = t.nextToken().toString().trim();
                    out.write(toencrypt+"\t");
                }
                i++;
            }
            out.write("\n");
        }
    }
```

```
    in.close();
    out.close();
}
catch(Exception e){
    e.printStackTrace();
}
}
/**
 * @param plaintext
 * @return
 * @throws Exception
 */
public String encrypt(String plaintext) throws Exception{
    this.cipher.init(Cipher.ENCRYPT_MODE, this.keypair.getPublic());
    byte[] bytes = plaintext.getBytes("UTF-8");
    byte[] encrypted = blockCipher(bytes,Cipher.ENCRYPT_MODE);
    char[] encryptedTranspherable = Hex.encodeHex(encrypted);
    return new String(encryptedTranspherable);
}
/**
 * @param encrypted
 * @return
 * @throws Exception
 */
public String decrypt(String encrypted) throws Exception{
    this.cipher.init(Cipher.DECRYPT_MODE, this.keypair.getPrivate());
    byte[] bts = Hex.decodeHex(encrypted.toCharArray());
    byte[] decrypted = blockCipher(bts,Cipher.DECRYPT_MODE);
    return new String(decrypted,"UTF-8");
}
/**
 * @param bytes
 * @param mode
 * @return
 * @throws IllegalBlockSizeException
 * @throws BadPaddingException
 */
```

```
private byte[] blockCipher(byte[] bytes, int mode) throws
    IllegalBlockSizeException, BadPaddingException{
    byte[] scrambled = new byte[0];
    byte[] toReturn = new byte[0];
    // if we encrypt we use 100 byte long blocks. Decryption requires 128
        byte long blocks (because of RSA)
    int length = (mode == Cipher.ENCRYPT_MODE)? 100 : 128;
    byte[] buffer = new byte[length];
    for (int i=0; i< bytes.length; i++){
        if ((i > 0) && (i % length == 0)){
            scrambled = cipher.doFinal(buffer);
            toReturn = append(toReturn,scrambled);
            int newlength = length;
            if (i + length > bytes.length) {
                newlength = bytes.length - i;
            }
            buffer = new byte[newlength];
        }
        buffer[i%length] = bytes[i];
    }
    scrambled = cipher.doFinal(buffer);
    toReturn = append(toReturn,scrambled);
    return toReturn;
}
/**
 * @param prefix
 * @param suffix
 * @return
 */
private byte[] append(byte[] prefix, byte[] suffix){
    byte[] toReturn = new byte[prefix.length + suffix.length];
    for (int i=0; i< prefix.length; i++){
        toReturn[i] = prefix[i];
    }
    for (int i=0; i< suffix.length; i++){
        toReturn[i+prefix.length] = suffix[i];
    }
}
```

```
    return toReturn;
}
}
```

Listing E.5: RSA java class

```
/**
 * @author fimiani
 *
 */
public class database {
    private static String driverName =
        "org.apache.hadoop.hive.jdbc.HiveDriver";
    /**
     * @param dbname
     * @param port
     * @param username
     * @param passw
     * @return
     * @throws SQLException
     */
    public static Connection getDBConnection
        (String dbname,String port,String username,String passw)
        throws SQLException{
        Connection con = null;
        try {
            Class.forName(driverName);
            String connect =
                "jdbc:hive://localhost:"+port+"/"+dbname;
            con = DriverManager.getConnection
                (connect, username, passw);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return con;
    }
}
```

```
//getConnection
Connection con =
database.getDBConnection("hive","2000", "user", "pwd");
```

Listing E.6: Setup Hive Thrift Server connection

```
/**
 * @author fimiani
 *
 * @param table
 * @param con
 * @param threshold_type2
 */
private static void execQuerytype2v2privacy (String table,
      Connection con, int threshold_type2) {
    String sql = "select a.iban,count(a.iban) from evaluation a
    WHERE a.bank='"+table+"' GROUP BY a.bank,a.iban";
    Statement stmt;
    try {stmt = con.createStatement();
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
        while(res.next()){
            String output = res.getString(1)+
            "\t"+res.getString(2)+"\t"+table;
            write.write_output("output", output);
        }
    }catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Listing E.7: Submit a query to Hive

```
/**
 * @author Fimiani
 *
 */
class Encrypt extends Thread
{
```

```
static int privacy_level = 1024;
static String enctype = null;
secure secure;
int encnum = 0;
static int opid;
PublicKey publickey = null;
static boolean ready[] = new boolean[4];
static PublicKey[] pks = {null,null,null};
public Encrypt(String type, secure s, int encnumber, int oid, PublicKey pk)
{
    this.enctype = type;
    this.encnum = encnumber;
    this.opid = oid;
    this.publickey = pk;
    if(enctype.equalsIgnoreCase("symmetric")){
        this.secure = s;
        this.start();
    }else{
        //store the retrieved publickey
        pks[encnumber-1] = pk;
        if(encnumber == 3){
            //now I have all publickey - protocols can start
            this.start();
        }
    }
}
public void run()
{
    PublicKey proxy1 = pks[0];
    PublicKey proxy2 = pks[1];
    PublicKey proxy3 = pks[2];
    String filename = "";
    String efilename = "";
    String source_ip = null;//1
        String destination_ip = null;//2
        String start_date = null;//3
        String end_date = null;//4
```

```

String http_operation = null;//5
String url = null;//6
String user_session_id = null;//7
String iban = null;//8
secure = new secure();
try {
Socket requestSocket = new Socket("10.79.61.54", 9001);
ObjectOutputStream outt = new ObjectOutputStream(requestSocket.
    getOutputStream());
secure.keygen("AES", "gateways_secret_seed");
//input stream from clear file simulate real time data
FileInputStream clearStream = new FileInputStream("combined_3months_plain
    .txt");
DataInputStream in = new DataInputStream(clearStream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
FileWriter fstreamout = new FileWriter("bank1-bank4e-"+String.valueOf(
    opid)+".txt");
BufferedWriter out = new BufferedWriter(fstreamout);
//read data line by line - every line do 4 encryption and send to
    EsperEngine for processing
String string;
RSA rsa = new RSA();
int numline = 1;
while((string = br.readLine()) != null){
    System.out.print("Input EVENT("+numline+"): ");
    System.out.printf ("Corrected time: %1$ta, %1$td %1$tb %1$tY, %1$tI:%1
        $tm:%1$tS.%1$tL %1$tp %1$tZ%n", SntpClient.getTime());
    StringTokenizer t = new StringTokenizer(string, "\t");
    int i = 1;
    int encnum = 1;
    while(t.hasMoreElements()){
        String element = (String)t.nextToken();
        if((i==1)|| (i==8) || (i==7) || (i==2)){
            if(!element.equalsIgnoreCase("n/a")){
                //STEP 1 - ENCRYPTION SYMMETRIC
                String cripted = secure.encrypt(element);
                // STEP 2 - ENCRYPTION 1ST PUBLICKEY

```

```
cripted = rsa.singleRSAencryption(pks[0], encnum, privacy_level,
    cripted);
//STEP 3 - ENCRYPTION 2st PUBLICKEY
cripted = rsa.singleRSAencryption(pks[1], encnum, privacy_level,
    cripted);
//STEP 4 - ENCRYPTION 3st PUBLICKEY
cripted = rsa.singleRSAencryption(pks[2], encnum, privacy_level,
    cripted);
if(i==1){
    source_ip = cripted;
}
else if(i==2){
    destination_ip = cripted;
}
else if(i == 8){
    iban = cripted;
}
else if(i == 7){
    user_session_id = cripted;
}
out.write(cripted+"\t");
}
else{
    iban = "n/a";
    out.write(element+"\t");
}
}else{
    //no crypt needed
    if(i == 3){
        start_date = element;
    }
    else if (i == 4){
        end_date = element;
    }
    else if (i == 5){
        http_operation = element;
    }
}
```

```

        else if (i == 6){
            url = element;
        }
        out.write(element+"\t");
    }
    i++;
    encnum++;
}
out.write("\n");
//SEND EVENT TO ESPER
outt.writeObject(new LogEvent(source_ip,destination_ip,start_date,
    end_date,http_operation,url,user_session_id,iban));
outt.flush();
numline++;
Thread.sleep(100);
}
in.close();
out.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Listing E.8: Gateway java main class

```

package eg;
/**
 * (source ip,destination ip,start_date,end_date,HTTP operation,url,user
 * session id,iban[optional])
 * @author Fimiani Pasquale
 */
public class LogEvent {
    public String source_ip;
        public String destination_ip;
        public String start_date;
        public String end_date;
        public String http_operation;

```

```
public String url;
public String user_session_id;
public String iban;

public LogEvent(String source_ip, String destination_ip, String
    start_date, String end_date, String http_operation, String url,
    String user_session_id, String iban) {
    this.source_ip = source_ip;
    this.destination_ip = destination_ip;
    this.start_date = start_date;
    this.end_date = end_date;
    this.http_operation = http_operation;
    this.url = url;
    this.user_session_id = user_session_id;
    this.iban = iban;
}
}
```

Listing E.9: LogEvent java class

```
public static long getTime(){
    try {
        final String serverName = "server.ntp.address";
        final DatagramSocket socket = new DatagramSocket();
        final InetAddress address = InetAddress.getByName(serverName);
        final byte[] buffer = new NtpMessage().toByteArray();
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address
            , PORT);
        socket.send(packet);
        packet = new DatagramPacket(buffer, buffer.length);
        socket.receive(packet);
        // Immediately record the incoming timestamp
        final double destinationTimestamp = NtpMessage.now ();
        final NtpMessage msg = new NtpMessage(packet.getData());
        socket.close();
        /* Presumably, msg.orginateTimestamp unchanged by server. */
        // Formula for delay according to the RFC2030 errata
        final double roundTripDelay =
```

```
(destinationTimestamp - msg.originateTimestamp) -
(msg.transmitTimestamp - msg.receiveTimestamp);
// The amount the server is ahead of the client
final double localClockOffset =
((msg.receiveTimestamp - msg.originateTimestamp) +
(msg.transmitTimestamp - destinationTimestamp)) / 2;
    final long now = System.currentTimeMillis(); // milliseconds 1
        x10e-3 seconds
final long cor = now + Math.round (1000.0*localClockOffset);
return cor;
} catch (Exception e) {
    e.printStackTrace();
}
return -1;
}
```

Listing E.10: Retrieve time from NTP server

List of Figures

- 2.1 Trusted Third Party (TTP) Model 4
- 2.2 Secure Multi-Party Computation Model 6
- 2.3 Airavat: Security and Privacy for MapReduce 8
- 2.4 Deployment diagram of Sharemind 9
- 2.5 Deployment scenario for SEPIA 9

- 3.1 Map-Reduce Execution Overview 14
- 3.2 A Hadoop cluster has many parallel machines that store and process large data sets.
Client computers send jobs into this computer cloud and obtain results. 15
- 3.3 HDFS clients 18
- 3.4 Hive System Architecture 23
- 3.5 Esper System Architecture 26

- 4.1 Three dimensional privacy 30
- 4.2 Proxy Based (Data Spread) - Overview 31
- 4.3 Matrix Proxy Based - Overview 33
- 4.4 Asymmetric Proxy Based - Overview 35

- 5.1 Screenshot of trojan’s admin panel 42
- 5.2 Configuration Options for the Attacker 43
- 5.3 Transaction Balance 44
- 5.4 Attacker trojan’s log 45

- 6.1 Log 50
- 6.2 Class Diagram 53
- 6.3 Detection Accuracy with or without Privacy (daily) 55
- 6.4 Error Level with or without Privacy (daily) 55
- 6.5 Detection Accuracy with or without Privacy (monthly) 56
- 6.6 Error Level with or without Privacy (monthly) 56
- 6.7 Detection Accuracy with or without Privacy (3 months) 56
- 6.8 Error Level with or without Privacy (3 months) 57
- 6.9 Query Execution Time (using 3 months logs no privacy) 57

6.10 Error Level Trend	57
6.11 Total Execution Time	58
6.12 Total Time Elapsed (3 months with privacy)	58
6.13 Encryption/Decryption impact on total time	59
6.14 MySQL vs Hadoop/Hive Loading Time	60
6.15 MySQL vs Hadoo/Hive Processing Time	60
6.16 Logs' size before/after Encryption	61
6.17 Detection Latency Esper	64
D.1 Multi Node Cluster Architecture	77
D.2 Multi Node Cluster Architecture	78

List of Tables

4.1	RSA encryption and decryption execution timings	36
6.1	Indicatore Sintentico di Costo (ICS)	50

Bibliography

- [1] D. Agrawal A. Evfimievski, R. Srikant and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of 8th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 217–228, 2002.
- [2] J. Gehrke A. Evfimievski and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Prof. of 22nd ACM SIGMOD-SIGACTSIGART Symposium on Principles of Database Systems*, pages 211–222, 2003.
- [3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Prof. of 6th ACM SIGMOD Int. Conf. on Management of Data*, pages 439–450, 2000.
- [4] Single Euro Payments Area. Sepa. http://en.wikipedia.org/wiki/Single_Euro_Payments_Area.
- [5] De Nederlandsche Bank. Internetbankieren nu en in de toekomst, dnb / kwartaalbericht. http://www.dnb.nl/dnb/home/file/Kwartaalbericht%20compleet_tcm46-156016.pdf, 2007.
- [6] Crypto++ Benchmarks. <http://www.cryptopp.com/benchmarks.html>.
- [7] Narasimham Challa and Jayaram Pradhan. Performance analysis of public key cryptographic systems rsa and ntru. *IJCSNS International Journal of Computer Science and Network Security*, 7:4, 2007.
- [8] Microsoft Corporation. Microsoft security intelligence report (sir) volume 10. <http://www.microsoft.com/security/sir/default.aspx>.
- [9] Daemen and Rijmen. The rijndael block cipher. <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf>.
- [10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04 Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*, 2004.
- [11] Istat.it Dati definitivi del censimento dell'industria e dei servizi. <http://archivio.rassegna.it/2004/lavoro/articoli/istat/02.htm>.
- [12] O. Goldreich. *The Foundations of Cryptography*. Cambridge University Press, 2004.

- [13] Q. Wang H. Kargupta, S. Datta and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proc. of 3rd ICDM IEEE Conf. on Data Mining*, pages 99–106, 2003.
- [14] Apache HIVE. <http://hive.apache.org/>.
- [15] George Kurtz. Google attack is tip of iceberg. <http://siblog.mcafee.com/cto/google-attack-is-tip-of-iceberg/>, January 2010.
- [16] Giorgia Lodi Roberto Baldoni Leonardo Aniello, Giuseppe Antonio Di Luna. A collaborative event processing system for protection of critical infrastructures from cyber attacks. *SAFE-COMP 2011, The International Conference on Computer Safety, Reliability and Security*, 1:5–7, 2011.
- [17] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Journal of Cryptology*, pages 177–206, 2002.
- [18] Mysql list partitioning at. <http://dev.mysql.com/doc/refman/5.1/en/partitioning-list.html>.
- [19] S. Quinlan M. K. McKusick. Gfs: Evolution on fast-forward. In *ACM Queue*, 2009.
- [20] B. McCarty. *SELinux: NSAs Open Source Security Enhanced Linux*. O'Really Media, 2004.
- [21] M. C. McChesney. Banking in cyberspace: an investment in itself. In *IEEE Spectrum*, February, 1997.
- [22] M.Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of 31st ACM Symposium on Theory of Computing*, pages 245–254, 1999.
- [23] S. Micali O. Goldreich and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. *Proceedings of 19th ACM Conf. on Theory of computing*, 1:218–229, 1987.
- [24] University of Texas at Austin. <http://www.cs.utexas.edu/~indrajit/airavat.html>.
- [25] R. B. Ross P. H. Carns, W. B. Ligon III and R. Thakur. Pvfs: A parallel file system for linux clusters. In *Proc. of 4th Annual Linux Showcase and Conference*, pages 317–327, 2000.
- [26] A. Evfimicvski R. Agrawal and R. Srikant. Information sharing across private databases. In *Proc. of 22nd ACM SIGMOD Int. Conference on Management of Data*, pages 86–97, 2003.
- [27] A. Shamir Rivest R and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 21, 2:120–126, 1978.
- [28] Gregory Chockler Roberto Baldoni. *Collaborative Financial Infrastructure Protection*. Springer, 2012.

- [29] S. Leung S. Ghemawat, H. Gobioff. The google file system. In *Proc. of ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.
- [30] Gregg Schudel and Bradley Wood. *Modeling the behaviour of the cyber-terrorist.*, 2000.
- [31] SEPIA. Privacy-preserving aggregation of multi-domain network events and statistics. <http://sepia.ee.ethz.ch/>.
- [32] A. Shamir. How to share a secret. In *Communications of the ACM 22, 11*, 1979.
- [33] Federal Information Processing Standard. Fips 197. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
- [34] tuttitalia.it. <http://www.tuttitalia.it/banche/>.
- [35] Banca Unicredit. Indicatore sintetico di costo (ics). <https://www.unicredit.it/library/it/common/docs/guidaisc.pdf>.
- [36] Centraal Bureau voor de Statistiek. De digitale economie. <http://www.cbs.nl/NR/rdonlyres/D2E9E66D-D6A1-4438-83C4-541DC6D92B30/0/2006p34pub.pdf>, 2006.
- [37] Z. Zhan W. Du. Using randomized response techniques for privacy-preserving data mining. In *Proc. of 9th ACM SIGKDD Conf. on Knowledge Discovery and Data mining*, pages 505–510, 2003.
- [38] G. Gibson W. Tantisiroj, S. Patil. Data-intensive file systems for internet services: A rose by any other name. In *Technical Report CMU- PDL-08-114, Parallel Data Laboratory, Carnegie Mellon University*, 2008.
- [39] Wikipedia. Set cover. http://en.wikipedia.org/wiki/Set_cover_problem.
- [40] Bradley Wood. *An insider threat model for adversary simulation*, 2000.
- [41] A. C. Yao. How to generate and exchange secrets. In *Proc. of 27th Annual Symposium on Foundations of computer science*, pages 162–167, 1986.
- [42] A.C. Yao. Protocols for secure computations. *Proceedings of 23rd Symposium on Foundations of Computer Science*, 1:160–164, 182.