TPM-based Authentication Mechanism for Apache Hadoop

Issa Khalil[1], Zuochao Dou[2], and Abdallah Khreishah[2]

[1] Qatar Computing Research Institute, Qatar Foundation, Doha, Qatar,
ikhalil@qf.org.qa
[2] Electrical Computer Engineering Department, New Jersey Institute of Technology, Newark, USA,
{zd36,abdallah}@njit.edu

Abstract. Hadoop is an open source distributed system for data storage and parallel computations that is widely used. It is essential to ensure the security, authenticity, and integrity of all Hadoop's entities. The current secure implementations of Hadoop rely on Kerberos, which suffers from many security and performance issues including single point of failure, online availability requirement, and concentration of authentication credentials. Most importantly, these solutions do not guard against malicious and privileged insiders. In this paper, we design and implement an authentication framework for Hadoop systems based on Trusted Platform Module (TPM) technologies. The proposed protocol not only overcomes the shortcomings of the state-of-the-art protocols, but also provides additional significant security guarantees that guard against insider threats. We analyze and compare the security features and overhead of our protocol with the state-of-the-art protocols, and show that our protocol provides better security guarantees with lower optimized overhead.

Key words: Hadoop, Kerberos, Trusted Platform Module (TPM), authentication, platform attestation, insider threats

1 Introduction and Related Work

Apache Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm [1] [2]. The basic architecture of Hadoop is shown in Fig. 1. The core components are Hadoop Distributed File System (HDFS) and Hadoop MapReduce. HDFS provides distributed file system in a Master/Slave manner. The master is the NameNode, which maintains the namespace tree and the mapping of data blocks to DataNodes. The slaves are the DataNodes which store the actual data blocks. A client splits his data into standardized data blocks and stores them in different DataNodes with a default replication factor of 3. The MapReduce is a software framework for processing large data sets in a parallel and distributed fashion among many DataNodes. MapReduce contains two sub-components: JobTracker and TaskTracker. The JobTracker, together with the NameNode, receives the

MapReduce jobs submitted by the clients and splits them into smaller tasks to be sent later to TaskTrackers for processing. Each DataNode has a corresponding TaskTracker, which handles the MapReduce tasks.

There are 5 types of communication protocols in HDFS: DataNodeProtocol (between a DataNode and the NameNode); InterDataNodeProtocol (among different DataNodes); ClientDataNodeProtocol (between client and DataNodes); ClientProtocol (between a client and the NameNode); NameNodeProtocol (between the NameNode and the Secondary NameNode). On the other hand, there are 3 types of communication protocols in MapReduce: InterTrackerProtocol (between the JobTracker and a TaskTracker); JobSubmissionProtocol (between a client and the JobTracker); and TaskUmbilicalProtocol (between task child process and the Tasktracker). In addtion, there is a DataTransferProtocol for data flow of Hadoop.

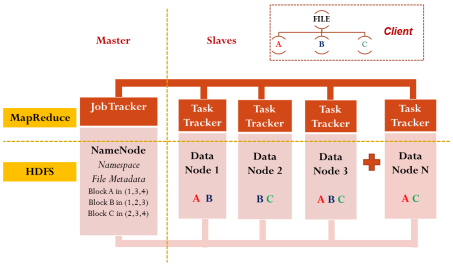

Fig. 1. Basic architecture of Hadoop.

Hadoop clients access services via Hadoop's remote procedure call (RPC) library. All RPC connections between Hadoop entities that require authentication use the Simple Authentication and Security Layer (SASL) protocol. On the top of SASL, Hadoop supports different types of sub-protocols for authentication, such as generic security service application program interface (GSSAPI, e.g., Kerberos [3] [4]) or digest access authentication (i.e., DIGEST-MD5) [5]. In practice, Hadoop uses Kerberos as the primary/initial authentication method and uses security tokens (DIGEST-MD5 as protocol) to supplement the primary Kerberos authentication process within the various components of Hadoop (NameNode, DataNodes, JobTracker, TaskTracker, etc.). This Kerberos based authentication mechanism is first implemented in 2009 by a team at Yahoo [5].

However, there are many limitations and security issues in using Kerberos for Hadoop authentication. The first weakness of Kerberos lies in its dependency on passwords. The session key for data encryption during the initial communication phase to key distribution center (KDC) is derived from the user's password. It has been shown in many situations that passwords are relatively easy to break (e.g., password guessing, hardware key-loggers, shoulder surfing etc.) mainly due to bad or lazy selections of passwords. For example, in 2013, almost 150 million people have been affected by a breach into Adobe's database [6]. The breach is due to mistakes made by Adobe in handling clients' passwords. All passwords in the affected database were encrypted with the same key. Additionally, the encryption algorithm used did not handle identical plaintexts, which results in similar passwords being encrypted into similar ciphers. Disclosure of KDC passwords allows attackers to capture users' credentials, which turns all Hadoop's security to be useless (at least for the owners of the disclosed passwords). The second issue of Kerberos lies in having a single point of failure. Kerberos re-

quires continuous availability of the KDC. When the KDC is down, the system will suffer from the single point of failure problem. Although Hadoop security design deploys delegation tokens to overcome this bottleneck of Kerberos, it introduces a more complex authentication mechanism. The introduced tokens add extra data flows to enable access to Hadoop services. Moreover, many token types have been introduced including delegation tokens, block tokens, and job tokens for different subsequent authentications, which complicate the configuration and management of these tokens [7]. The third issue in Kerberos lies in its dependence on a third-party online database of keys. If anyone other than the proper user has access to the key distribution center (KDC), the entire Kerberos authentication infrastructure is compromised and the attacker will be capable of impersonating any user [8]. This issue highlights the insider threat problems in Kerberos. Kerberos cannot provide any protection against an administrator who has the privilege to install hardware/software key loggers or any other malware to steal users' credentials and other sensitive data (passwords, tokens, session keys, and data).

In early 2013, Intel launched an open source effort called Project Rhino to improve the security capabilities of Hadoop. They propose Task HADOOP-9392 (Token-Based Authentication and Single Sign-On) which is planned to support tokens for many authentication mechanisms such as Lightweight Directory Access Protocol (LDAP), Kerberos, X.509 Certificate authentication, SQL authentication, and Security Assertion Markup Language (SAML) [9]. They mainly focus on how to extend the current authentication framework to a standard interface for supporting different types of authentication protocols. Nevertheless, all these authentication protocols, including Kerberos, are software-based methods that are vulnerable to privileged user manipulations. A privileged insider may be able indirectly collect users' credentials through, for example, the installation of malware/spyware tools on the machines they have access to in a way that is transparent to the victims. Furthermore, Rhino trades off flexibility with complexity. It enhances the flexibility of the authentication mechanisms at the cost of increasing the complexity of the overall system. Project Rinho did not provide overhead analysis or performance evaluation which makes it hard to compare with other protocols and raises questions about its practicality.

In this work, we propose an TPM-based authentication protocol for Hadoop that overcomes the shortcomings of the current state-of-the-art authentication protocols. To date, more than 500 million PCs have been shipped with TPMs, an embedded crypto capability that supports user, application, and machine authentication with a single solution [10]. TPM offers facilities for the secure generation of cryptographic keys, and limitation of their use, in addition to a random number generator. TPM supports three main services, namely: (1) *Remote Attestation* which creates a nearly un-forgeable hash-key summary of the hardware and software configuration. The program encrypting the data determines the extent of the summary of the software. This allows a third party to verify that the software has not been changed or tampered with. (2) *Binding* which encrypts data using the TPM endorsement key, a unique RSA key burned

into the chip during its production, or another trusted key descended from it. (3) *Sealing* which encrypts data in similar manner to binding, but in addition specifies a state in which the TPM must be in order for the data to be decrypted (unsealed). Since each TPM chip has a unique secret RSA key burned in as it is produced, it is capable of performing platform authentication [11].

In addition to providing the regular authentication services supported by Hadoop, our protocol ensures additional security services that cannot be achieved by the current state-of-the-art Hadoop authentication protocols. In addition to eliminating the aforementioned security weakness of Kerberos, our protocol guards against any tamper in the target machines (the machine in the cloud that is supposed to store users' encrypted data and process it) hardware or software. In public cloud environments, the user does not need to trust the system administrators on the cloud. Malicious cloud system administrators pose great threats to users' data (even though it may be encrypted) and computations. Those administrators, even though, may not have direct access to the user's data, they may be able to install malicious software (malware, spyware, etc.) and hardware (key loggers, side channels, etc.) tools that can ex-filtrate users data and sensitive credentials.

In [12], the author proposes a TPM-based Kerberos protocol. By integrating Private Certification Authority (PCA) functionality into the Kerberos authentication server (AS) and remote attestation is done by the (Ticket-Granting Server) TGS, the proposed protocol is able to issue tickets bound to the client platform. However, the present mechanism does not provide any attestation for Hadoop internal components. Nothing can prevent malicious Hadoop insiders from tampering with internal Hadoop components. In this paper, we use TPM functionalities to perform authentication directly inside Hadoop to completely get rid of the trusted-third-party.

In [13], the authors propose a Trusted MapReduce (TMR) framework to integrate MapReduce systems with the Trusted Computing Infrastructure (TCG). They present an attestation protocol between the JobTracker and the TaskTracker to ensure the integrity of each party in the MapReduce framework. However, they mainly focus on the integrity verification of the Hadoop MapReduce framework, and did not address the authentication issues of Hadoop's HDFS and Clients. The work does not provide a general authentication framework for the whole Hadoop system.

In [14], the authors present a design of a trusted cloud computing platform (TCCP)based on TPM techniques, which guarantees confidential execution of guest VMs, and allows users to attest to the IaaS provider to determine if the service is secure before they launch their VMs. Nevertheless, they do not provide much details about how this work will be implemented and no performance evaluation is provided. Also, this work does not focus on a general authentication framework specific for the Hadoop system.

In this paper, we design and implement a TPM-based authentication protocol for Hadoop that provides strong mutual authentication between any internally interacting Hadoop entities, in addition to mutually authenticate with exter-

nal clients. Each entity in Hadoop is equipped with a TPM (or vTPM) that locks-in the root keys to be used for authenticating that entity to the outside world. In addition to locally hiding the authentication keys and the authentication operations, the TPM captures the current software and hardware configurations of the machine hosting it in an internal set of registers (PCRs). Using the authentication keys and the PCRs, the TPM-enabled communicating entities establish session keys that can be sealed (decrypted only inside the TPM) and bound to specific trusted PCRs value. The bind and seal operations protect against malicious insiders since insiders will not be able to change the state of the machine without affecting the PCR values. Additionally, our protocol provides remote platform attestation services to clients of third party, possibly not trusted, Hadoop providers. Moreover, the seal of the session key protects against the ability to disclose the encrypted data in any platform other than the one that matches the trusted configurations specified by the communicating entities. Finally, our protocol eliminates the trusted third party requirement (such as Kerberos KDC) with all its associated issues including single point of failure, online availability, and concentration of trust and credentials. Fig. 2 shows the high level overview of our protocol.

We summarize our contributions in this work as follows: (1) Propose a TPM-based authentication protocol for Hadoop that overcomes the shortcomings of Kerberos. Our protocol utilizes the binding and sealing functions of TPM to secure the authentication credentials (e.g., Session keys) in Hadoop communications. (2) Propose and implement a periodic platform remote attestation mechanism to guard against insider malicious tampering with Hadoop entities. (3) Perform performance and security evaluation of our protocol and show the significant security benefits together with the acceptable overhead of our new authentication protocol over the current state-of-the-art protocols (Kerberos). (4) Implement our protocol within Hadoop to make it practically available for vetting by Hadoop community.

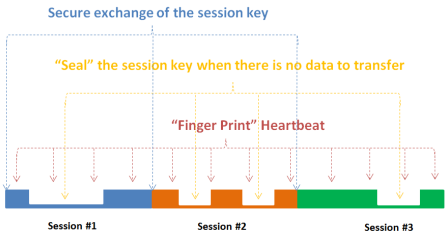

Fig. 2. High level overview of the authentication framework.

The rest of this paper is organized as follows. In Section 2, in addition to providing a background on the state-of-the-art Hadoop security design and the TPMs, we lay out our attack model. In Section 3, we describe our proposed TPM-based authentication protocol in details. In Section 4, we present the system design and implementation method. In Section 5, we conduct the performance evaluation of our proposed authentication protocol.

2 Background

2.1 Hadoop Security Design

Apache Hadoop uses Kerberos to support the primary authentication in Hadoop communications. It introduces three types of security tokens as Supplementary Mechanisms. The first token is the *Delegation Token (DT)*. After the initial authentication to the NameNode using Kerberos credentials, a user obtains a delegation token, which will be used to support subsequent authentications of user's jobs. The second token is *Block Access Token (BAT)*. The BAT is generated by the NameNode and is delivered to the client to access the required DataNodes. The third token is the *Job Token (JT)*. When a job is submitted, the JobTracker creates a secret key that is only used by the tasks of the job to request new tasks or report status [5].

The complete authentication process in Hadoop using Kerberos is shown in Fig. 3. The client obtains a delegation token through initial Kerberos authentication (step 1). When the client uses the delegation token to authenticate, she first sends the ID of the DT to the NameNode (step 2). The NameNode checks if the DT is valid. If the DT is valid, the client and NameNode try to mutually authenticate using their own Token Authenticators (which is contained in the delegation token) as the secret key and DIGEST-MD5 as the protocol (step 3, 4, 5 and 6) [15]. This represents the main authentication process in secure Hadoop system, although there are other slightly different authentication procedures such as the Shuffle in the MapReduce process.

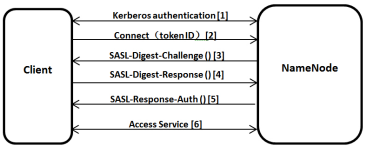

Fig. 3. Authentication Process of Hadoop Security Design developed by Yahoo.

2.2 Trusted Platform Module

The Trusted Platform Module (TPM) is a secure crypto-processor, which is designed to secure hardware platforms by integrating cryptographic keys into devices [11]. It is specifically designed to enhance platform security which is beyond the capabilities of today's software-based protections [16]. Fig. 4 shows the components of a Trusted Platform Module.

The TPM has a random number generator, a RSA key generator, an SHA-1 hash generator and an encryption-decryption-signature-engine. In the persistent memory, there is an Endorsement Key (EK). It is an encryption key that is permanently embedded in the Trusted Platform Module (TPM) security hardware at the time of manufacture. The private portion of the EK is never released outside of the TPM. The public portion of the EK helps to recognize a genuine TPM. The storage root key (SRK) is also embedded in persistent memory and is used to protect TPM keys created by applications. Specifically, SRK is used to encrypt other keys stored outside the TPM to prevent these keys from being usable in any platform other than the trusted one [17].

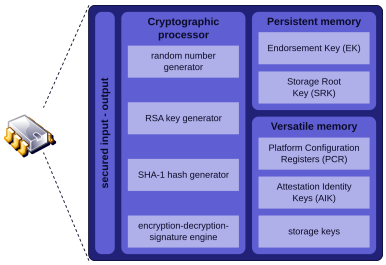

Fig. 4. Components of a Trusted Platform Module [18].

In the versatile memory, the Platform Configuration Register (PCR) is a 160 bit storage location for integrity measurements (24 PCRs in total). The integrity measurements includes: (1) BIOS, ROM, Memory Block Register [PCR index 0-4]; (2) OS loaders [PCR index 5-7]; (3) Operating System (OS) [PCR index 8-15]; (4) Debug [PCR index 16]; (5) Localities, Trusted OS [PCR index 17-22]; and (6) Applications specific measurements [PCR index 23] [19].

The TPM is able to create an unlimited number of Attestation Identity Keys (AIK). The AIK is an asymmetric key pair used for signing, and is never used for encryption and it is only used to sign information generated internally by the TPM, e.g., PCR values [20]. For signing the external data, storage keys are required. A storage key is derived from the Storage Root Key (SRK) which is embedded in the persistent memory of the TPM during manufacture. Using the generated storage key along with PCR values, one could perform sealing operation to bind data into a certain platform state. The encrypted data could only be unsealed/decrypted under the same PCR values (i.e., the same platform state).

2.3 Attack Model

In addition to the traditional external threats, we believe that clouds are more susceptible to internal security threats especially from untrusted privileged users such as system administrators.

Many enterprises are likely to deploy their data and computations among different cloud providers for many reasons including load balancing, high availability, fault tolerance, and security, in addition to avoiding single-point of failure and provider locking [21][22][23]. For example, an enterprise may choose to deploy the NameNode in their home machine to provide high security by only allowing local access to local managers, and deploy the DataNodes among different cloud platforms to distribute the storage and computational load. Obviously, this increases the probability of compromise of the DataNodes. If one of the DataNodes is injected with some malwares, Hadoop becomes vulnerable.

In the public cloud deployments of Hadoop, a privileged user could maliciously operate on behalf of the user by installing or executing malicious software to steal sensitive data or authentication credentials. For example, a malicious system administrator in one of the DataNodes on the public cloud may be able to steal users' private data (e.g., insurance information etc.) that is stored in the compromised DataNode. With the appropriate privileges, the administrator can install a malware/spyware that ex-filtrates the stored sensitive data. Kerberos based Hadoop authentication cannot protect against such insider attackers and thus systems running Kerberos are vulnerable to this attack. In Kerberos-based

secure Hadoop, the DataNode authenticates with other parties using delegation tokens, and the action of installing malware on the DataNode machine will not be detected. On the other hand, the Trusted Platform Module (TPM) is capable of detecting the changes of hardware and software configurations, which will help in mitigating such attacks.

We assume attackers are capable of performing replay attacks. The attacker could record the message during the communication and try to use it to forge a future communication message. Such replay attacks may cause serious problems, such as denial of service (keep sending the message to overload the server), or repeated valid transaction threats (e.g., the attacker capture the message of a final confirmation for a transaction, then he can repeatedly send it message to the server and result in repeated valid transactions if there is no proper protection).

3 TPM-based Hadoop Authentication Protocol

In this section, we present the details of our proposed Hadoop authentication protocol. The key idea of the protocol lies in the utilization of TPM binding keys to securely exchange and manage the session keys between any two parties of Hadoop (NameNode/JobTracker, DataNodes/TaskTracker and Client).

To achieve this, we assume every party in Hadoop, namely, the DataNode, the NameNode, and the client, has a TPM. Fig. 5 depicts the high level processes of the protocol which are explained in detail in the following sub sections. The protocol consists of two processes, the certification process and the authentication process.

Fig. 5. The high level processes of our TPM-based Hadoop authentication protocol (Client to NameNode in this example).

3.1 The Certification Process

The certification process (which is similar to that presented in [12]) is triggered by the client and is depicted in Fig. 6. The client's TPM creates a RSA key using the SRK as a parent. This key will be used as the client's Attestation Identity Keys (AIK[client]). The AIK [client] is then certified by a PCA. This process only takes place once during the initialization of the TPM (a one-time pre-configuration operation). The client's TPM then creates a binding key that is bound to a certain platform (i.e., the private portion of the binding key is inside the TPM and could only be used in this platform), then we seal the private part of the binding key to a certain PCR configuration. Finally, the client uses the AIK[client] which is certified by the PCA to certify

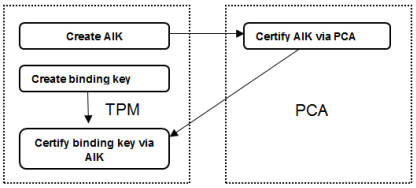

Fig. 6. Certification process of the TPM binding key.

the public part of the binding key. The AIK[client] is not used directly for authentication in order to maintain higher security guarantees by minimizing the chances of successful cipher analysis attacks to disclose the key. The AIK[client] is only used to sign PCRs value and other TPM keys. We can certify the binding key directly through the PCA instead of using the certified AIK[client]. However, using the certified AIK[client] is simpler, faster and provides the same security guarantees. Once we certify the AIK[client], we can use it to sign all kinds of keys generated by the clients' TPM without referring back to the PCA, which greatly reduces the communication overhead at the cost of local processing overhead.

3.2 The Authentication Process

In the authentication process (Fig. 7), the client tries to authenticate itself to the NameNode and the NameNode authenticates itself to the client. The Client sends a random number K_1 along with the corresponding IDs (e.g., fully qualified domain name) to the NameNode. This message is encrypted by the public binding key of the NameNode. The NameNode sends a random number K_2 along with corresponding ID to the client. This message is encrypted by the public binding key of the client. Using K_1 and K_2, both the client and the NameNode generate the session key $Key\_session = K_1 \oplus K_2$. Note that only the correct NameNode can obtain K_1 by decrypting the message sent by the client using the NameNode's SK_bind, which is bind to the target NameNode's TPM with a certain software and hardware configuration (sealed binding key). Similarly, only the correct client can obtain K_2 by decrypting the message sent by the NameNode using the client's SK_bind, which is bind to the client's TPM with the appropriate software and hardware configurations. This ensures mutual authentication between the client and the NameNode.

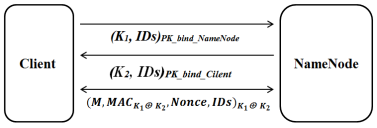

Fig. 7. The authentication process of the TPM-based authentication protocol.

The session key exchanged is then locked into a certain PCRs value in an operation known as seal operation using the TPM command *Seal* that takes the two inputs: the PCRs value and the session key ($Seal(PCRsindexes, Key\_session)$). This ensures that $Key\_session$ can only be decrypted using the hardware secured keys of the TPM in that particular platform state. By sealing the session key to specific acceptable hardware and software configurations (specific PCRs value), we protect against any tamper of the firmware, hardware, or software on the target machine through for example, malware installations or added hardware/software key loggers. Moreover, the session key ($Key\_session$) is made to be valid only for a predefined period of time, after which the session key expires and the authentication process has to be restarted to establish a new session key if needed. The validity period of the session key is an important security parameter in our protocol. Short validity periods provide better security in the case of session key disclosure since fewer communications are exposed by disclosing the key. However, shorter periods incur extra overhead in establishing more session keys.

Additionally, a *nonce* is added to every message (for example, $Nonce = K_2++$) to prevent replay attacks. Finally, message authentication codes (MAC) are included with each message to ensure data integrity. The communication message format is as follows: $(Message, MAC, Nonce = K_2++, IDs)key\_session$.

3.3 Periodic "Fingerprint" Checking (Cross Platform Authentication)

In a non-virtualized environment, the Trusted Platform Module (TPM) specification assumes a one to one relationship between the operating system (OS) and the TPM. On the other hand, virtualized scenarios assume one to one relationship between a virtual platform (virtual machine) and a virtual TPM [24]. However, Hadoop systems are master/slaves architectures. The NameNode is the master that manages many DataNodes as slaves. If the number of DataNodes grows, the number of session establishment processes that the NameNode is involved in also grows relatively. Each session involves many TPM operations (e.g., Seal and unseal). For large systems, the TPM may become a bottleneck due to the limitation of one TPM/vTPM per each NameNode according to current implementations of TPM/vTPM.

To address this issue and alleviate the potential performance penalty of TPM interactions, we introduce the concept of periodic "Fingerprint" platform checking mechanism based on the Heartbeat protocol in Hadoop (Fig. 8). The idea is to offload most of the work from the TPM of the NameNode to the NameNode itself. However, this requires us to loosen our security guarantees and change the attack model by assuming that the NameNode is "partially" trusted. Partially here, means that an untrusted (compromised) NameNode will only have transient damage on the security of Hadoop system. A nameNode that gets compromised will only stay unnoticed for a short time since other interacting parties (such as DataNodes) may randomly request attestation of the authenticity of the NameNode. In this on-demand attestation request, an interacting entity with the NameNode (e.g., DataNode, client, etc.) asks the name node to send a TPM-sealed value of its current software and hardware configuration. If the requesting entity receives the right values for the PCRs of the NameNode within a predefined time, then the NameNode is trusted, otherwise a suspicious alert is raised about the healthiness of the NameNode. The response time to receive the sealed PCRs value from the NameNode is set to account for the communication time, the load on the NameNodes (size of Hadoop System), and the seal operations assuming that the perpetrator controlling the untrusted NameNode will not be able to roll back the configurations of the NameNode to the trusted one within this time.

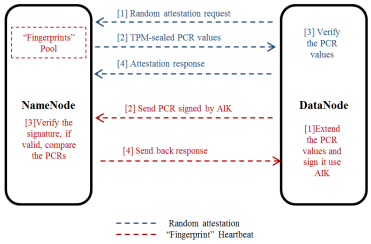

Fig. 8. Random attestation and Periodic "Fingerprint" attestation illustration.

As mentioned earlier, the PCR values inside the TPM captures the software and hardware configurations of the system hosting the TPM. Therefore, a particular PCR value can be considered as a "Fingerprint" of the corresponding platform. We collect the "Fingerprint" of each entity that needs to interact with the NameNode (e.g., DataNode) a priori and store it on the NameNode (This can be achieved during the registration process of the entity to the NameNode). The Heartbeat protocol in Hadoop periodically sends alive information from one entity to another (e.g., from DataNode to NameNode). Therefore, we configure each entity interacting with the NameNode (e.g., DataNode) to periodically (or can be configured to be on-demand) send the new PCR values (achieve by PCR extension operation) to the NameNode to check the consistency of the stored PCRs and the new PCRs. The TPM in the interacting entity signs its current PCR values using its AIK key and sends the message to the NameNode. When the NameNode receives the signed PCR values, it verifies the signature, and if valid, it compares the received values with the trusted pre-stored values. If a match is found, the authentication will succeed and the session will continue. Otherwise, the authentication will fail and penalty will apply (e.g., clear up the session key, shut down the corresponding DataNode, etc.). By doing so, the number of NameNode side TPM operations decrease significantly as we replace the TPM seal and unseal operations with the "Fingerprint" verification that are carried out outside the TPM. See Fig.8.

3.4 Security Features

In this section, we elaborate on the security services provided by our protocol. The security services can be broadly classified into the Common security services and the New security services. The common security services are supported by both our protocol and other Hadoop authentication protocols, while the new security services are novel and supported only by our protocol. The common security services include: (1) *Replay attack prevention.* A $nonce = K_2 + +$ is included with each communicated message to prevent replay attacks. (2) *Data Integrity.* A MAC is included in the message to ensure data integrity. The MAC is computed as $Hash(SessionKey||Message)$. Digital signature is another way to achieve data integrity as well as authenticity. However, digital signatures are more computationally involved, as they rely on asymmetric keys, compared to hash functions that use symmetric keys. The *New* security services include: (1) *Session key binding.* The session key is generated by $XORing$ a local and an external random numbers (K_1 and K_2). The local one is generated locally and the external is received from the other party. The local random number is encrypted using the public portion of the binding key of the other party before sending it to that party. This ensures that only the party that has the appropriate private portion of the binding key will be able to decrypt the message and get the external random number. Furthermore, the decryption keys exist only inside the TPM chip and are sealed under a certain hardware/software configuration. This protects against even malicious insiders as they will not be able to know anything about the session key. (2) *Session key sealing.* The session key is sealed

with TPM functions. The sealed session key can be decrypted only under the same platform conditions (as specified by the PCRs value) using the associated EK that resides inside the TPM. If the attacker installs malware/spyware to steal the session key, he will not be able to successfully decrypt and obtain the key as the decryption will fail due to the change in the system configuration which will be reflected in the PCRs. (3) *Periodic "Fingerprint" attestation mechanism.* This enables one way attestation of DataNode while reducing the load on a partially trusted NameNode. This disables any privileged malicious user controlling a DataNode from being able to install or inject malware/spyware without affecting the internal view of the TPM about the system. (4) *Disk Encryption.* In addition to the traditional disk encryption, we could choose using TPM keys to protect the HDFS data from directly steal on the disk.

4 System Design and Implementation

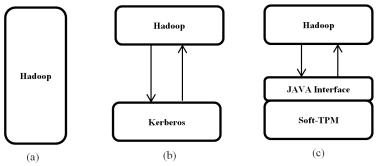

Fig. 9. The three Hadoop architectures implemented: (a) Hadoop without Security; (b) Hadoop with Kerberos Security; (c) Hadoop with TPM-based Security.

To evaluate the security guarantees and the performance overhead of our authentication protocol, we compared three different Hadoop implementations, namely, Hadoop without Security (Baseline), Hadoop with Kerberos (Kerberos) and Hadoop with our protocol (TPM). We use Hadoop version 0.20.2-cdh3u6 [25] since it is the most stable version of the classic first generation of HDFS and MapReduce. We modify the source code of Hadoop using ant on Eclipse [26]. For Kerberos, we use krb5.x86_64 [27]. For TPM, we use TPM Emulator since it is the best choice for debugging and testing purposes. To incorporate TPM with Hadoop project, we use IAIK jTSS (TCG Software Stack for the Java (tm) Platform [28]) as the Java interface between Hadoop and TPM. The three Hadoop architectures are shown in Fig. 9.

4.1 Implementation Details

A. Hadoop Deployment. We configure Hadoop-0.20.2-cdh3 in a distributed manner. The implementation involves two virtual machines with CentOS 6.5 operating system, 1GB memory, 20 GB hard disk, Java version = jdk-7u51-linux-x64. One of the machines is installed with one NameNode, one JobTracker, one DataNode and one TaskTracker, the other one installed with one DataNode and one TaskTracker.

B. Hadoop Deployment with Kerberos. For Hadoop security design with Kerberos, we had to configure Hadoop and Kerberos separately. Table. 1 shows the summary of the corresponding configurations.

C. Hadoop Deployment with TPM. Hadoop deployment here involves two parts: (1) TPM emulator configuration and jTSS java interface configuration;

Table 1. Summary of Kerberos secured Hadoop configuration.

Hadoop	**Install** hadoop-0.20-native; hadoop-0.20-sbin. **Configure** core-site.xml; hdfs-site.xml; mapred-site-xml. **Add** taskcontroller.cfg.
Kerberos	**Install** krb5-server.x86_64; krb5-workstation.x86_64; krb5-devel.x86_64. **Configure** krb5.conf; kdc.conf; kadm5.acl. **Create** "EXAMPLE.COM" database. **Add** hdfs/fully.qualified.domain.name@EXAMPLE.COM; mapred/fully.qualified.domain.name@EXAMPLE.COM; host/fully.qualified.domain.name@EXAMPLE.COM. **Generate** hdfs.keytab; mapred.keytab.

(2) Hadoop source codes modification environment setup. We use software based TPM emulator [29], which provides researchers and engineers of trusted systems with a powerful testing and debugging tool.

We use IAIK jTSS 0.7.1 (TCG Software Stack for the Java (tm) Platform, Copyright (c) IAIK, Graz University of Technology) as the java interface between TPM and Hadoop project. There are two ways to configure the IAIK jTSS: (1) local bindings, which is well suited for development, experimenting and debugging. (2) SOAP bindings, which allows any unprivileged application access [28]. We choose SOAP bindings since we want Hadoop to utilize TPM. For integrating TPM functionalities into Hadoop project, we setup a modification environment. Since the Hadoop-0.20.2-chd3u6 is used, we choose Eclipse IDE for Java EE Developers as platform and Apache Ant 1.9.3 as build tool.

5 Performance Evaluation

5.1 Security Analysis and Evaluation

We discuss here the security features of our protocol and compare it with Kerberos based Hadoop. Table. 2 summarizes the results of the comparison.

As we can see from Table. 2, Kerberos-based Hadoop depends on passwords. Loss of passwords means loss of authentication capability and deny of access to any of the resources. Similarly, in TPM-based Hadoop, the ownership of a TPM depends on the password. Loss of password will result in authentication incapability and encrypted data inaccessible.

However, TPM-based Hadoop provides hardware security in addition to the software security provided in Kerberos-based Hadoop. The TPM ensures the security bond to the hardware and software configurations, which protects against tamper in the software or the hardware (including install malware/spyware). On the other hand, Kerberos-based Hadoop solely rely on the security of soft tokens/tickets. Also, the Kerberos based Hadoop requires an on-line KDC, which presents a single point of failure.

In our protocol, the authentication credentials exchanged are not only encrypted with the public keys of the parties but also bound to specific hardware

Table 2. Comparison of TPM-based Hadoop and Kerberos-based Hadoop.

Kerberos-based Hadoop	**TPM-based Hadoop**
Password/ticket/token are required to authenticate users. Weakness on password dependency and ticket/tokens lost	TPM key is used to authenticate users. The key is stored on TPM, and secured by PCRs (a certain platform state). Hardware security.
An online Key Distribution Center (KDC) is required. Kerberos has a single point failure problem (KDC)	Not apply
Lost password: loss of passwords prevents the ability to authentication with KDC	Lost Password: Loss of passwords associated with the TPM will result in encrypted data inaccessible.
Not apply	Only support one to one mode between one VM and one TPM/VTPM, resource insufficient for unsealing operation.

and software configurations in each party. This setup ensures that not only the right party can access the credentials, but also that the credentials can only be accessed under specific hardware and software configurations. This guards against both user and machine masquerading. For example, in the session establishment between the NameNode and a DataNode, the random number K_1 is encrypted with the public key of the NameNode. K_1 can be decrypted only by the NameNode with certain hardware and software configurations, because the decryption key is bonded to the corresponding PCR values.

We setup an experiment to evaluate this security feature in our protocol, using the "Fingerprint" as a sample scenario. We develop pseudo codes to simulate the changes of PCR value (i.e., manually change the PCR values after 5 heartbeats of the DataNodes). In our modified heartbeat protocol, we set the heartbeat interval to 3 seconds (i.e., The DataNode sends the PCRs value to the NameNode every 3 seconds). We set a counter to an initial value of zero and adds one every single heartbeat. When the counter reaches 5, we manually set the PCR to a wrong value. The results show that the authentication fails and the session with this DataNode is shut down by the NameNode. Table. 3 summarizes the parameters and the results of the experiment.

Table 3. Evaluation of the periodic "Fingerprint" checking mechanism (Heartbeat Interval = 3 seconds).

NameNode "Fingerprint"	**DataNode PCR values**	**Expected Result**	**Result**
PCR="0"	PCR="0"(count=1:4). PCR="1"(count=5). Count: # of heartbeats	DataNode shutdown when count=5	DataNode shutdown when count=5

5.2 Overhead Analysis and Evaluation

In our work, we realize that it is not enough to develop a secure authentication protocol, but most importantly, we have to ensure that the algorithm is practical. Therefore, we have to keep the performance penalty and cost of the added security guarantees within acceptable bounds. In this section, we thoroughly analyze the performance overhead of our protocol and compare it with the baseline and the Kerberos-based authentication protocols. The necessary additions to the exchanged messages that to prevent replay attacks, ensure data integrity, ensure data confidentiality (encryption and decryption operations of the exchanged data messages) are the same for both our protocol and Kerberos-based protocol. However, both our protocol and Kerberos-based protocol have extra overhead that does not exist in the other. In our protocol, we have a one-time TPM setup overhead which is introduced when the TPM in each Hadoop component generates the binding keys, AIK, in addition to obtaining certificates for these keys. This is a lightweight overhead and will not impact the day-to-day operation of Hadoop system as it is a pre-configuration one time overhead. Furthermore, at the beginning of each RPC session, our protocol introduces an extra overhead to transfer two random numbers and to generate the session key by Xoring the two random numbers. Additionally, during each RPC session, our protocol involves a recurring overhead to seal and unseal the session keys and the "Fingerprint" heartbeat verifications. The seal operation only reoccurs when a legitimate change in the PCRs value of the other party is acknowledged and approved. In this case, the first party needs to reseal the session key to work with the new PCRs value in the second party. The unseal operation reoccurs with every RPC request for data exchange since we need to retrieve the session key through TPM unseal operation. Finally, the "Fingerprint" heartbeat checking is an overhead that depends on the security parameters configured, i.e., how frequently the first party needs to check the status of the second and whether it is done periodically or on-demand. We evaluated the one time overhead (binding key generation and certification) using jTSS under SAOP binding, which is used for third party application such as Hadoop. Table. 4 shows the average overhead for each step.

Table 4. One time overhead of the proposed system design.

binding key creation	**binding key loading**	**AIK creation**	**AIK loading**	**binding key certification**	**Sum**
˜355.8ms	˜27.1ms	˜108.4ms	˜24.1ms	˜17.0ms	˜532.4ms

We next compare the overhead of the baseline Hadoop (no security), Kerberos-based Hadoop, and TPM-based Hadoop (our protocol). We use a classic MapReduce job: Pi example. Pi example is to calculate the value of π in a distributed way. We set the number of map tasks to 5 and set the number of samples per task to 1000 (#map task=5, #samples per task=10000). This means, we divide the job into five smaller tasks, each task will be handling 1000 samples using quasi-Monte Carlo method.

Since we have not finished the entire authentication framework implementation of the TPM-based Hadoop, we use for this work the number of RPC sessions (i.e., the # of session key generations) and the number of RPC connection requests (correspond to the # of "unseal" operation) during the Pi example excution. There are 3 RPC sessions created during this Pi example. The average overhead to transfer two random numbers (1024 bits each) via RPC connection (i.e., simulation in Java program) and to generate the session key is around 40.8 ms, which indicates there is an total overhead around 122.4 ms for the Pi example. Furthermore, the average overhead of "unseal" operation of the unseal overhead is the # of RPC requests times the average "unsealing" time. In the Pi example, 99 RPC connection requests initiated that result in 99 "unseal" operations to decrypt session keys. The average "unseal" operation overhead was 45.1 ms, which makes the accumulated overhead around 4.46s for the Pi example. Next. We compute the overhead for encryption and decryption. For the Pi example, according to the task report, there are 1205 bytes data read from the HDFS, 215 bytes data written to the HDFS and 140 bytes data for the reduce shuffle phase. By simulation using triple data encryption algorithm (3DES, i.e., the default encryption algorithm for Kerberos based Hadoop), the additionally cryptographic overhead is around 441.9 ms (i.e., 269ms for encryption and 172.9ms for decryption).

Table 5. Overhead comparison of the three Hadoop implementations (No security, Kerberos, and TPM).

No Security	**Kerberos**	**TPM-based Hadoop w/ unseal operation**	**TPM-based Hadoop w/ fingerprint check**
~38.90s	~46.24s	3 RPC sessions × 40.8ms + 99 unseal operations × 45.1ms + 441.9ms =**5.03s**	3 RPC sessions × 40.8ms + 14 heartbeats × (41.1+15.7+0.3) ms + 441.9ms =**1.36s**

On the other hand, for the repeated overhead of the periodic "Fingerprint" attestation, it depends on the length of the heartbeat interval (i.e., integer of seconds) and the PCR extension, AIK signing and signature verification process for each heartbeat. The PCR extension operation takes the old PCR's value and the platform new measurements as input, therefore its overhead depends on the size of the new measurements. The new measurements are conducted by a third party called Integrity Measurement Architecture (IMA)[30]. As a result, the Estimated Overhead for Heartbeat "Fingerprint" checking = # of Heartbeats × (PCR_extension(size of new measurements)+AIK_signing+Signture_verification)). With the jTSS soap binding, for 1KB measurements, the average overhead for each PCR extension is 41.1 ms, for each AIK signing is 15.7 ms, and for each signature verification is 0.3 ms. With heartbeat interval equals to 3 seconds, in the Pi example, there will be

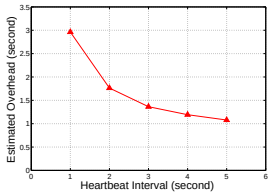

Fig. 10. Estimated overhead of Heartbeat "Fingerprint" checking of different heartbeat interval (1 KB measurements).

about 14 times heartbeats such that we have about 799.4 ms overhead for our system. Table. 5 shows the results.

As mentioned on Section 3.3, the number of NameNode side TPM operations decreases significantly as we replace the TPM seal and unseal operations with the finger print verification that is carried out outside the TPM. Furthermore, in Fig.10, we show that the Heartbeat "Fingerprint" checking interval could be adjusted according to the security requirements for different applications (e.g., longer interval means lower security).

6 Conclusion and Future Work

In this paper, we design and implement a TPM-based authentication protocol for Hadoop that provides strong mutual authentication between any internally interacting Hadoop entities, in addition to mutually authenticate with external clients. The bind and seal operations supported by the TPM protect against malicious insiders since insiders cannot change the machine state without affecting the PCR values. Additionally, our protocol provides remote platform attestation services to clients of third party, possibly not trusted, Hadoop providers. Moreover, the seal of the session key protects against the ability to disclose the encrypted data in any platform other than the one that matches the trusted configurations specified by the communicating entities. Finally, our protocol eliminates the trusted third party requirement (such as Kerberos KDC) with all its associated issues such as single point of failure, online availability, and concentration of trust and credentials.

We analyze the security features of our protocol and evaluate its performance overhead. Moreover, we study and resolve the practical limitations that are imposed by the current Hadoop design (one NameNode) and by the current TPM implementations (one TPM/vTPM per machine). Finally, we compare the security features and overhead of our protocol with the state-of-the-art protocols and show that our protocol provides better security guarantees with acceptable overhead

In the future work, we will tighten the security requirements of the NameNode by removing the assumption of partial trust. Specifically, we plan to explore the use of server-aided cryptography techniques to shift most of the work of sealing and unsealing from inside the TPM chip to off the chip (in the NameNode itself).

References

1. Apache Hadoop. `http://hadoop.apache.org`.
2. K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), IEEE 26th Symposium on*, pages 1–10, 2010.
3. S. Bagchi, N. Shroff, I. Khalil, R. Panta, M. Krasniewski, and J. Krogmeier. Protocol for secure and energy-efficient reprogramming of wireless multi-hop sensor networks, 2012. US Patent 8,107,397.

4. I. Khalil and S. Bagchi. Secos: Key management for scalable and energy efficient crypto on sensors. *Proc. IEEE Dependable Systems and Networks (DSN)*, 2003.
5. O. O'Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell. Hadoop security design. *Yahoo, Inc., Tech. Rep*, 2009.
6. A. Hern. Did your Adobe password leak? `http://www.theguardian.com/technology/2013/nov/07/adobe-password-leak-can-check`.
7. K. Smith. Big Data Security: The Evolution of Hadoop's Security Model. `http://www.infoq.com/articles/HadoopSecurityModel`.
8. Kerberos. `http://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/ch--kerberos.html`.
9. Project Rhino. `https://issues.apache.org/jira/browse/HADOOP-9392`.
10. Trusted Platform Module (TPM): Built-in Authentication. `http://www.trustedcomputinggroup.org/solutions/authentication`.
11. Trusted Platform Module. `http://en.wikipedia.org/wiki/Trusted_Platform_Module`.
12. Andreas Leicher, Nicolai Kuntze, and Andreas U Schmidt. Implementation of a trusted ticket system. In *Emerging Challenges for Security, Privacy and Trust*, pages 152–163. Springer, 2009.
13. A. Ruan and A. Martin. Tmr: Towards a trusted mapreduce infrastructure. In *Services (SERVICES), IEEE Eighth World Congress on*, pages 141–148, 2012.
14. N. Santos, K. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, 2009.
15. Hadoop Security Analysis. `http://www.tuicool.com/articles/NFf6be`.
16. Trusted platform module (tpm) quick reference guide. *Intel Corporation*, 2007.
17. TPM Management. `http://technet.microsoft.com/en-us/library/cc755108.aspx`.
18. TPM architecture. `http://en.wikipedia.org/wiki/File:TPM.svg`.
19. R. Ng. Trusted platform module tpm fundamental. *Infineon Technologies Asia Pacific Pte Ltd*, 2008.
20. Trusted Computing: TCG proposals. `https://www.cs.bham.ac.uk/~mdr/teaching/modules/security/lectures/TrustedComputingTCG.html`.
21. R. Panta, S. Bagchi, and I. Khalil. Efficient wireless reprogramming through reduced bandwidth usage and opportunistic sleeping. *Ad Hoc Networks*, 7(1):42–62, 2009.
22. S. Bouktif, F. Ahmed, I. Khalil, and G. Antoniol. A novel composite model approach to improve software quality prediction. *Information and Software Technology*, 52(12):1298–1311, 2010.
23. J. Shi, M. Taifi, A. Khreishah, and J. Wu. Sustainable gpu computing at scale. In *Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference on*, pages 263–272. IEEE, 2011.
24. The Trusted Computing Group (TCG). Virtualized trusted platform architecture specification, version 1.0, revision 0.26. 2011.
25. CDH3u6 Doc. `http://www.cloudera.com/content/support/en/documentation/cdh3-documentation/cdh3-documentation-v3-latest.html`.
26. Eclipse. `https://www.eclipse.org/`.
27. The KDC and related programs for Kerberos 5. `http://linuxsoft.cern.ch/cern/slc5X/x86_64/yum/updates/repoview/krb5-server.html`.
28. Trusted Computing for the Java(tm) Platform. `http://trustedjava.sourceforge.net/index.php?item=jtss/about`.
29. TPM emulator. `http://tpm-emulator.berlios.de/designdoc.html`.
30. Integrity Measurement Architecture (IMA). `http://sourceforge.net/p/linux-ima/wiki/Home/`.