Robust Insider Attacks Countermeasure for Hadoop: Design & Implementation

Zuochao Dou, *Student Member, IEEE,* Issa Khalil, *Member, IEEE,* Abdallah Khreishah, *Member, IEEE,* and Ala Al-Fuqaha, *Senior Member, IEEE*

***Abstract*—Hadoop is an open source software framework for storage and processing of large-scale data sets. The proliferation of cloud services and its corresponding increasing number of users lead to a larger attack surface, especially for internal threats. Therefore, in corporate data centers, it is essential to ensure the security, authenticity, and integrity of all the entities of Hadoop. The current secure implementations of Hadoop mainly utilize Kerberos, which is known to suffer from many security and performance issues including the concentration of authentication credentials, single point of failure, and online availability. Most importantly, these Kerberos-based implementations do not guard against insider threats. In this paper, we propose an authentication framework for Hadoop that utilizes Trusted Platform Module (TPM) technology. The proposed approach provides significant security guarantees against insider threats, which manipulate the execution environment without the consent of legitimate clients. We have conducted extensive experiments to validate the performance and the security properties of our approach. The results demonstrate that the proposed approach alleviates many of the shortcomings of Kerberos-based state-of-the-art protocols and provides unique security guarantees with acceptable overhead. Moreover, we have formally proved the correctness and the security guarantees of our protocol via Burrows-Abadi-Needham (BAN) logic.**

***Index Terms*—Hadoop, Kerberos, Trusted Platform Module (TPM), authentication, platform attestation, insider threats**

I. INTRODUCTION

APACHE Hadoop provides an open source framework for the storage and parallel processing of large-scale data sets on clusters of commodity computers. As the amount of data maintained by industrial corporations grows over time, big data processing becomes more important to enterprise data centers. However, the threat of data leaks also continues to grow due to the increasing number of entities involved in running and maintaining cloud infrastructure and operations [2]. The recent boost of big data start-ups such as MongoDB, DataStax, MapR Technologies and Skytree leads to an increased number of points of access in the cloud, that is, larger attack surface for intruders. This can be clearly inferred from a recent report of the Health Information Trust Alliance (HITRUST), which reveals that the total cost of health-care data breach incidents has grown to $4.1 billion over the recent years [3].

Authentication is the process of gaining assurance that an entity is performing robustly and precisely as intended [4] [5]. In addition, data confidentiality in the cloud is tightly correlated to the user authentication [2]. Therefore, a secure and robust authentication mechanism of both users and services is imperative for secure and private cloud computing and storage operations [6]. However, the continuous growth and the concentration of data in clouds, combined with the increasing adoption of security solutions such as authentication, access control, and encryption drives intruders to be more persistent and creative in developing sophisticated attack strategies [7]. One way to protect clouds and to successfully combat such sophisticated attacks is to push the bar higher through the combination of hardware and software security solutions. Pushing the security down to the hardware level in conjunction with software techniques provides better protection over software-only solutions [8], which is especially feasible and suitable for entity authentication and platform attestation in the cloud.

Currently, Hadoop leverages Kerberos [9] [10] as the primary authentication method and uses DIGEST-MD5 security tokens [11] to supplement the primary Kerberos authentication process, as detailed in Section II. However, in addition to its limitations and security weaknesses, the use of Kerberos for authentication in Hadoop-based environments raises many security concerns. The most vital weakness of Kerberos lies in its dependency on passwords. The session key for data encryption during the initial communication phase with the Key Distribution Center (KDC) is derived from the user's password. Disclosure of KDC passwords allows attackers to capture users' credentials, which turns all Hadoop's security to be useless. The large number of password disclosure incidents through cracking, social engineering, or even database leakage, clearly indicates that this threat is real and pervasive. For example, in 2011, RSA was a target of a spear Phishing attack [12]. A backdoor was installed due to a mistake by an employee who retrieved the Phishing email from her junk mail box and opened it. The malware successfully harvested credentials as confirmed by RSA FraudAction Research Labs. It threatened the security of many important defense contractors like Northrop Grumman, Lockheed Martin, and L-3.

Another important issue of Kerberos lies in its dependency on the KDC which constitutes a single point of failure and even a single point of attack for persistent and dedicated

Z. Dou is with the Electrical Computer Engineering Department, New Jersey Institute of Technology, Newark, USA, E-mail: zd36@njit.edu.

I. Khalil is with the Qatar Computing Research Institute, Hamad bin Khalifa University, Doha, Qatar. E-mail: ikhalil@qf.org.qa.

A. Khreishah is with the Electrical Computer Engineering Department, New Jersey Institute of Technology, Newark, USA, E-mail: abdallah@njit.edu.

A. Al-Fuqaha is with the NEST Research Lab, College of Engineering & Applied Sciences Computer Science Department, Western Michigan University, Kalamazoo, USA, E-mail: ala.al-fuqaha@wmich.edu.

This work is supported by award No. ECCS-1331018.

Part of this work was presented in proceedings of 10th International Conference on Security and Privacy in Communication Networks(SecureComm 2014), Beijing, China [1].

attackers. Although Hadoop's security design introduces delegation tokens to overcome this bottleneck, they lead to a more complex authentication mechanism due to the extra tokens and data flows that are required to enable access to Hadoop services. Many types of token have been introduced, including delegation tokens, block tokens, and job tokens for different subsequent authentications. This, relatively, large number of tokens, not only complicates the configuration and the management of the tokens, but also expands the attack surface [13]. Kerberos keys are stored in an on-line third-party database. If anyone other than the proper user has access to the KDC, through, for example, a malware installation by an insider, the entire Kerberos authentication infrastructure will be compromised and the attacker will be able to impersonate any user [14]. This highlights the fact that insiders could create havoc in Kerberos infrastructure itself, and consequently affect the security posture of the supported Hadoop. It is clear that Kerberos is not well-equipped against insiders or outsiders who could change the execution environment that the user trusts. For example, attackers may install key loggers or other malware-tools to steal users' credentials and data.

In this paper, we **design** and **implement** a TPM-based authentication protocol for Hadoop that provides strong mutual authentication services among internal Hadoop entities, in addition to mutually authenticating external clients. Each entity in Hadoop is equipped with a TPM (or vTPM in the case of multi-home virtualized environments) that locks-in the root keys for authenticating the 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 Platform Configuration Registers (PCRs), as detailed in Section II-C. Using the authentication keys and the PCRs, the TPM-enabled communicating entities are able to establish session keys that can be sealed (decrypted only inside the TPM) and bound to specific trusted platform software and hardware configurations. The bind and seal operations protect against malicious insiders, because they will not be able to change the state of the machine without affecting the values of the PCRs. Our protocol enables 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. As Fig. 1 shows, the protocol consists of three overlapping phases: (1) Secure exchange of the session key at the beginning of each communication session; (2) Session key management; and (3) "Fingerprint" attestation mechanism. The details are presented in the subsequent sections. Our contributions can be summarized as follows:

- Propose a novel TPM-based authentication protocol among Hadoop entities to reduce the risk of the continuously evolving insider attacks, especially due to the proliferation of cloud services and big data applications.
- Propose a periodic remote mechanism for attesting the hardware and software platform configurations. Such attestation mechanism provides hardware level security that supports other software mechanisms in reducing the risk of internal, as well as, external threats.
- Implement the whole system on real world Hadoop platforms and conduct extensive sets of experiments to evaluate the performance overhead and the security merits of our mechanism. The performance and security evaluation clearly shows that our framework provides better security guarantees with acceptable performance overhead compared to the state-of-the-art Kerberos-based implementations.
- Provide a thorough theoretical analysis using the BAN logic to rigorously prove the correctness and the trustworthiness properties of our authentication protocol.

The rest of this paper is organized as follows. In Section II, we provide the background knowledge about Hadoop architecture, the state-of-the-art security design of Hadoop, and the overview of Trusted Platform Module technology (TPM). In Section III, we first lay out our attack model and then present the design of our TPM-based authentication protocol. In Section IV, we provide the formal proof of the proposed authentication protocol. In Section V, we perform the comparative security analysis between Kerberos-based Hadoop and the proposed TPM-based Hadoop. In Section VI, we present the detailed implementation methodology of the proposed protocols. In Section VII, we introduce the experimental test-bed configurations and present the performance evaluation of our protocol. Section VIII presents the related work. In Section IX, we conclude this study and discuss potential future research extensions.

II. BACKGROUND

In this section, we introduce necessary information about Hadoop architecture, state-of-the-art Hadoop security design and the basis of Trusted Platform Module technology (TPM).

A. Hadoop Structure

As depicted in Fig. 2, Hadoop clusters have three major categories of server roles: (1) Client machines, (2) Master nodes, and (3) Slave nodes. The role of the client machine is to load the data into the cluster, to submit MapReduce jobs describing how data should be processed, and to fetch or view the results of the task when processing finishes. The Master nodes (i.e., NameNode, Secondary NameNode and JobTracker) supervise the two major components of Hadoop, namely, the distributed data storage (HDFS), and the distributed data processing (MapReduce) [15] [16]. The NameNode is responsible for coordinating data storage operations when a client machine requests to load the data into HDFS, while the JobTracker is in charge of supervising parallel MapReduce processing. The slave nodes (i.e., DataNodes and TaskTrackers) are responsible for storing the data and executing the computational tasks assigned by the Master nodes, respectively.

B. Hadoop Security Design

Hadoop uses Kerberos for its authentication operations [11]. The complete authentication process is illustrated in Fig. 3. The client obtains a Delegation Token (DT) through initial Kerberos authentication (step 1). When the client uses the DT to authenticate, she first sends the ID of the DT to the

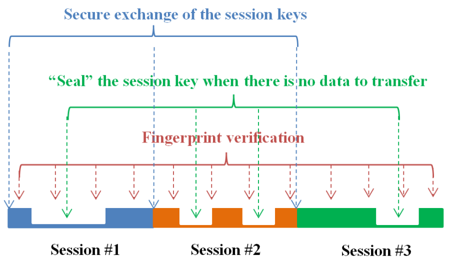

Secure exchange of the session keys
"Seal" the session key when there is no data to transfer
Fingerprint verification
Session #1
Session #2
Session #3

Fig. 1: The overlapping phases of the TPM-based authentication protocol

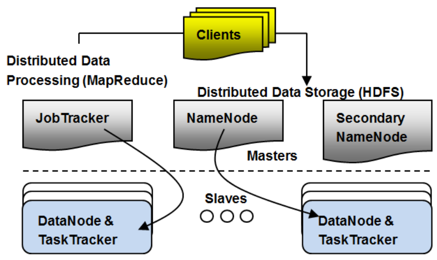


Fig. 2: Hadoop Architecture based on Server Roles

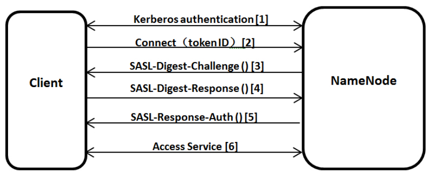

Client
NameNode
Kerberos authentication [1]
Connect（token ID）[2]
SASL-Digest-Challenge () [3]
SASL-Digest-Response () [4]
SASL-Response-Auth () [5]
Access Service [6]

Fig. 3: Authentication process in Kerberos-based Hadoop

NameNode (step 2). Then, 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 are contained in the delegation tokens, as the secret key using DIGEST-MD5 protocol (steps 3-6) [17].

C. Trusted Platform Module (TPM)

TPM is a secure crypto-processor, which is designed to protect hardware platforms by integrating cryptographic keys into devices [18]. It has been designed with the goal to enhance platform security through mechanisms that are beyond the capabilities of today's software-based systems [19].

TPM supports three main services: (1) The remote attestation service creates a nearly un-forgeable hash-key summary of the hardware and software configurations in a way that allows other parties to verify the integrity of the software and the hardware. (2) The binding service 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) The sealing service encrypts data in a similar manner to binding, but it additionally specifies the state that the TPM must be in for the data to be unsealed (decrypted) [18].

The platform configuration register (PCR) of the TPM is a 160-bit storage location that is used to record aggregate software and hardware measurements of the platform, which include: (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] [20].

The TPM is capable of creating an unlimited number of Attestation Identity Keys (AIK). Each AIK is an asymmetric key pair that is used for signing data that is internally generated by the TPM, such as the storage key. A storage key is derived from the Storage Root Key (SRK), which is embedded in the TPM chip during the manufacturing process. Using the generated storage key along with the PCR values, one could perform sealing to bind the data to a certain platform state (i.e., a specific platform software and hardware configuration). The encrypted data could only be unsealed (decrypted) under the same PCR values (i.e., the same platform state).

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 [8]. Additionally, many virtual TPM implementations exist for virtualized environments [21] [22].

D. Integrity Measurement Architecture

Co-operating with the hardware TPM, the integrity measurement architecture (IMA) (proposed by IBM [23]) provides an efficient measurement system for dynamic executable content. IMA provides real time measurements of the platform (user applications, OS libraries, etc.) during the post-boot period, while the TPM enables the pre-boot measurements. In this work, we assume the IMA is pre-configured and installed on each platform.

III. TPM-BASED HADOOP AUTHENTICATION FRAMEWORK

In this section, we first introduce the attack model, then, we present the design of the proposed TPM-based Hadoop authentication protocol.

The protocol uses the key binding of the TPM to secure the exchange and management of the session keys between any two Hadoop communicating entities (NameNode/JobTracker, DataNode/TaskTracker and Client). To achieve this, we assume that every Hadoop entity has a TPM. Fig. 4 (a) depicts the high-level processes of the protocol which are detailed in the following subsections. The protocol consists of two processes, the certification process and the authentication process.

Note that the public key infrastructure (PKI) is only used to secure the exchange of the symmetric session keys. The expensive certification and management process is only used during the certification process, where the cost amortized through the use of TPM AIK functionality as explained in Section III-B.

A. Attack Model

In addition to the traditional external threats, we believe that clouds are more susceptible to internal security threats, especially from other untrusted users [24] [25]. Many enterprises are likely to deploy their data and computations across different cloud providers for many reasons including load balancing, high availability, fault tolerance, and security, in addition to avoiding single-point of failure and vendor lock-in [26] [27] [28]. However, such behavior increases the attack surface and the probability of compromise of Hadoop entities. For example, a DataNode may get infected with a malware that makes it unsafe to operate on sensitive data. Therefore, it is imperative for any security solution for Hadoop to enable the detection of any unauthorized change in the software and hardware configurations of its entities. Our entity authentication protocol is specifically designed to counter such adversarial actions, assuming an attack model with the following attacker capabilities and possible attack scenarios:

- **Attacks against TPM**: We do not address the attacks against TPM (e.g., side channel timing attack [29]).Thus, we assume attackers can not compromise the TPM chip or its functions.
- **Attacks against slave nodes or client machines**: We assume that attackers are capable of installing (directly or remotely) malware or making malicious hardware changes in the compromised slave nodes (e.g., the DataNodes) or client machines.
- **Attacks against master nodes**: We assume that attackers are capable of installing (physically or remotely) malware or making malicious hardware changes in the master nodes (e.g., the NameNode). However, this capability could be revoked if the NameNode is deployed in a trustworthy environment (e.g., a private cloud with strict security policy enforced), as detailed in Section III-D.
- **Denial of Service attacks (DoS attacks)**: We do not address DoS attacks.

B. TPM-Based Hadoop Certification Process

The certification process (similar to that presented in [30]) is triggered by the client or the NameNode and is depicted in Fig. 4 (b). The client in this paper refers to any entity that interacts with the NameNode such as a user submitting a job or a DataNode. The TPM of the client/NameNode creates a RSA key using the SRK as a parent. This key is used as the AIK. The AIK is then certified by a PCA. This process is a onetime pre-configuration operation that takes place once during the initialization of the TPM. The client's/NameNode's TPM then creates a binding key that is bound to a certain platform. Then the TPM seals the private part of the binding key with a certain PCR configuration. Finally, the client/NameNode uses the AIK to certify the public part of the binding key. Once the AIK is certified by the PCA, it can be used to sign all types of keys generated by the TPM without referring back to the PCA, which greatly reduces the communication overhead.

C. The Authentication Process

The authentication process (cf. Fig. 4 (c)) implements the mutual authentication between the NameNode and the client. The Client sends a random number K_1 along with the corresponding IDs (e.g., $remoteID$ in Hadoop codes) 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 the 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 bound 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 bound to the client's TPM with the corresponding software and hardware configurations. This ensures mutual authentication between the client and the NameNode.

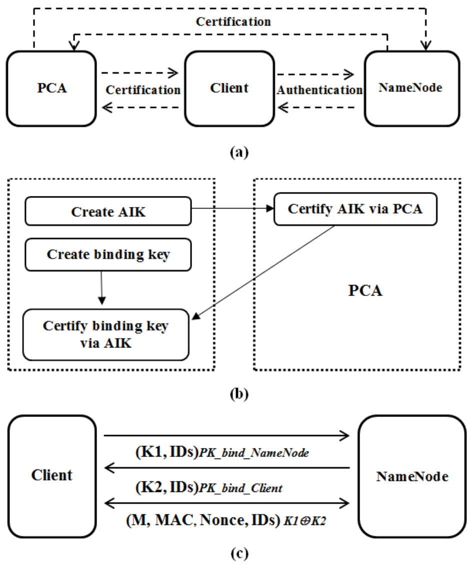


Fig. 4: (a)The high level processes of our TPM-based Hadoop authentication protocol (Client to NameNode in this example); (b)TPM-based Hadoop certification process; and (c)TPM-based Hadoop authentication process

The newly exchanged session key is then locked into a certain PCR value in an operation known as seal operation using the TPM's command "Seal". Seal takes two inputs, the PCR value and the session key ($Seal(PCRindexes, Key\_session)$). 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 (i.e., specific PCRs value), we protect against potential tampering with the firmware, hardware, or software on the target machine (e.g., through 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 as 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 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: $(Message, MAC, Nonce = Nonce{+}{+}, IDs)key\_session$.

D. Periodic Fingerprint Checking (Cross-Platform Authentication)

In a non-virtualized environment, the TPM specification assumes a one-to-one relationship between the 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 [31]. However, Hadoop systems employ a master/slaves architecture. 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. 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 NameNode according to current implementations of TPM/vTPM.

To address this practical issue and alleviate the potential performance penalty of TPM operations, we introduce the concept of periodic platform-Fingerprint checking mechanism based on the heartbeat protocol in Hadoop (cf. Fig. 5). 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. We argue that this assumption is reasonable for the following reasons: (i) Hadoop deployment usually involves one or few NameNodes [32], and hence, it is plausible and affordable to secure them in a trusted environment. For example, an enterprise can secure its NameNode by deploying it in the enterprise's local data center or in a high-reputation cloud platform with strict security standards. While the DataNodes can be deployed in environments with less strict security requirements. (ii) Our protocol is designed to limit the potential security damage of untrusted NameNode. A NameNode that gets compromised (that is, its software and/or hardware configuration is changed illegally) will only stay unnoticed for a short time, because other parties (such as DataNodes and clients) are designed to randomly request attestation of the authenticity of the NameNode. In on-demand attestation, an interacting entity with the NameNode asks the NameNode to send a TPM-sealed value of its current software and hardware configurations. If the requesting entity receives the right values for the PCR of the NameNode within a predefined time, then the NameNode is trusted; otherwise, a suspicious alert is raised about the trustworthiness of the NameNode.

The platform Fingerprints (i.e., PCR values) of each Hadoop entity that interacts with the NameNode (e.g., DataNode) are collected a priori and stored in the NameNode. This can be achieved during the registration process of the entity with the NameNode. The heartbeat protocol in Hadoop periodically sends alive information from one entity to another (e.g., from DataNode to NameNode). We leverage this native Hadoop feature by configuring each Hadoop entity to periodically (or on-demand) send the new PCR values (modified by PCR extension operations) to the NameNode for consistency checking with the stored PCR values. 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 succeeds and the session continues. Otherwise, the authentication fails and penalty may apply (e.g., clear up the session key, shut down the corresponding DataNode, etc.). By doing so, the number of NameNode TPM operations decreases significantly as we replace the TPM seal and unseal operations with the Fingerprint verification that is performed outside the TPM (cf. Fig.5).

However, there is a tradeoff between the security guarantees and the interval of the Fingerprint verification process, which reflects the extra overhead of the system. In other words, the interval value depends on the user's security requirements and is application dependent.

So far we have assumed that the NameNode is partially trusted and argue in favor of that. Nevertheless, in systems that require higher security guarantees and that can afford redundancy, we can eliminate the assumption of partial trusted NameNode. Untrusted NameNode can be neutralized by borrowing concepts from the fault tolerance domain. Multiple redundant NameNodes can be deployed and the NameNode attestation can be achieved through majority voting among the responses of the different NameNodes [33], [34], [35]. In fact, multiple NameNodes have been used in Hadoop implementations to scale up services such as directory, file and block management [32]. We can leverage such deployment (or deploy new NameNodes if necessary) to implement majority voting on the process of periodic platform-Fingerprint checking mechanism. Correct NameNode operations can be achieved as long as the total number of NameNodes is more than $2n$, where n is the number of compromised NameNodes (assuming no faulty nodes). For example, given 5 NameNodes that are independently running on different clouds, the attestation decision can be made based on the majority voting among them. Under this scenario, even if two of the NameNodes are compromised, the attestation decision will still be the correct one.

E. Security Features

The security features of our design include: (1) *Session key binding*. The session key is generated by $XORing$ a local and an external random numbers (K_1 and K_2). 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 to specific hardware and software configurations. This would protect against potential 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 PCR values) using the associated sealing key that resides inside the TPM. (3) *Periodic Fingerprint attestation mechanism*. This guards against malicious users who attempt to change the execution environment in a DataNode (by, for example, installing malware/spyware) in a bid to compromise data confidentiality.

IV. Formal Security Analysis

We use Burrows-Abadi-Needham (BAN) logic ([30]) to formally prove the following two properties of the proposed authentication protocol:

- **Correctness**: Implies that the protocol performs as intended, that is, two legitimate parties should always correctly authenticate.
- **Trustworthiness**: Implies that the protocol is secure under the attack model defined earlier, that is, only legitimate parties can successfully authenticate.

The proof sketch is shown as follows:

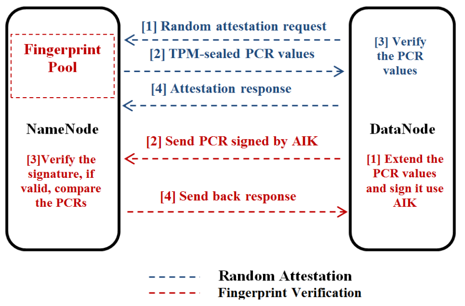

Fingerprint Pool
NameNode
[3]Verify the signature, if valid, compare the PCRs
[1] Random attestation request
[2] TPM-sealed PCR values
[4] Attestation response
[2] Send PCR signed by AIK
[4] Send back response
[3] Verify the PCR values
DataNode
[1] Extend the PCR values and sign it use AIK
Random Attestation
Fingerprint Verification

Fig. 5: Illustration of the random attestation and the periodic Fingerprint verification

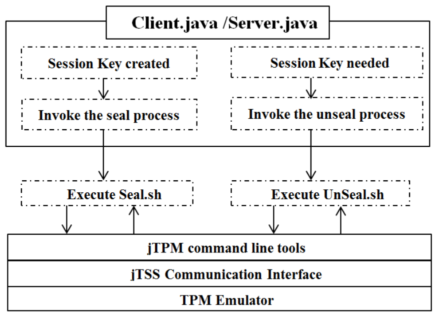


Fig. 6: Flow of seal and unseal operations within the various TPM-based Hadoop implementation layers

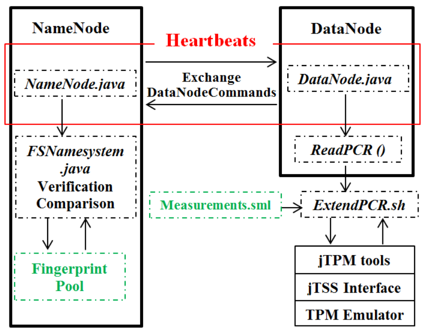


Fig. 7: Illustration of the detailed process of periodic Fingerprint checking mechanism

1) Present the notations and postulates/rules of BAN logical.
2) Generate a simplified version of the original protocol.
3) Idealize the simplified protocol.
4) Symbolize the assumptions.
5) Formalize the goals.
6) Present the formal prove of the goals.

Recall that the proposed protocol (cf. Fig. 4) consists of two main phases: (1) the certification phase and (2) authentication phases. The simplified objective of the protocol is to correctly and securely exchange the necessary keys among legitimate entities in the presence of the defined attackers. Therefore, we first prove that the public keys distributed by the PCA correspond to the claimed owners, that is, the mutual authentication parties believe in the public keys of each other, and that the random numbers (K_1 and K_2) are securely exchanged. Second, we need to prove that only legitimate entities can compute the session keys, that is, an entity can get the session key if and only if it maintains the same hardware and software state bound to that key.

A. *Notations and postulates/rules of BAN logical*

Following is a list of the BAN logical notations that we use in our proof steps:

1) Basic notations:

- C, N, P denote the client, the NameNode and the PCA.
- K_1 and K_2 are random numbers.
- K_{cn} is the computed symmetric session key, which is only known by C and N, where, $K_{cn} = K_1 \oplus K_2$.
- K_c, K_n, K_p are the public keys of C, N, and P.
- $K_c^{-1}, K_n^{-1}, K_p^{-1}$ are the private keys of C, N, and P.
- X or Y : represents a formula (e.g., a message sent).

In addition to the traditional basic notations, we introduce two new basic notations that represent the hardware and software configurations (recorded in the PCR values of the TPM):

- U and U' represent the TPM combined hardware and software initial state and the state when authenticating, respectively.

2) Logical notations:

- $C \mid\equiv X$: C believes/would be entitled to believe X;
- $C \lhd X$: C sees X. Someone has sent a message containing X to C, who can read and repeat X;
- $C \mid\sim X$: C said X. The principal C at some time sent a message including the statement X;
- $C \mid\Longrightarrow X$: C controls X. The principal C is an authority on X and should be trusted on this matter;
- $C \overset{X}{\rightleftharpoons} N$: X is only known by C and N;
- $C \overset{K_{cn}}{\leftrightarrow} N$: K_{cn} is the key shared by C and N;
- $\#(X)$: X is fresh. X has not been sent in a message at any time before the current run of the protocol;
- $\{X\}_{K_{cn}}$: X is encrypted by K_{cn};
- $\overset{K_c}{\mapsto} C$: K_c is a public key of C;
- $\frac{X}{Y}$: if X is true, then Y is true.

3) Ban logical postulates: BAN logic consists of many logical rules. These pre-agreed rules are used as theorems in the deduction. For the purpose of the current proof, we use 4 of the existing rules, in addition to a new TPM unsealing rule. Note that every C in the rules could be replaced by N.

– The *Message meaning rule* for the interpretation of messages. Here, we only use the rule for public keys.

$$\frac{C \mid\equiv \overset{K}{\mapsto} P, P \lhd \{X\}_{K^{-1}}}{C \mid\equiv P \mid\sim X}$$

That is, if C believes that K is P's public key, and C receives a message encoded with P's secret key, then C believes P once said X.

– The *nonce-verification* rule expresses the check that a message is recent, and hence, that the sender still believes in it:

$$\frac{C \mid\equiv \#(X),\, C \mid\equiv P \mid\sim X}{C \mid\equiv P \mid\equiv X}$$

That is, if C believes that X could have been uttered only recently and that P once said X, then C believes that P believes X.

– The *jurisdiction* rule states that if C believes that P has jurisdiction over X, then C trusts P on the truth of X:

$$\frac{C \mid\equiv P \mid\Longrightarrow X,\, C \mid\equiv P \mid\equiv X}{P \mid\equiv X}$$

– If a principal sees a formula, then he also sees its components, provided that he knows the necessary keys:

$$\frac{C \mid\equiv \overset{K^{-1}}{\mapsto} N,\, C \lhd \{X\}_K}{P \lhd X}$$

– If a principle's hardware and software state equals to the initial state, then he is able to unseal the content he sealed before.

$$\frac{C(U') == C(U), C \lhd \{X\}_{seal}}{C \lhd X}$$

B. The simplified version of the original protocol

The original authentication protocol (ref. fig 5) can be simplified as follows:
Step 1: C and N get each other's public key from P:

$$P \rightarrow C : \{K_n, N\}_{K_p^{-1}}; \quad P \rightarrow N : \{K_c, C\}_{K_p^{-1}}$$

Step 2: C and N exchange two random numbers K_1 and K_2 to generate the session key.

$$C \rightarrow N : \{K_1, C\}_{K_n}; \quad N \rightarrow C : \{K_2, N\}_{K_c}$$

Step 3: C and N unseal K_c^{-1} and K_n^{-1} to extract K_1 and K_2 if the software and hardware configuration do not change, then the shared session key $K_cn = K1 \oplus K2$ can be generated and used for later communication.

$$C \text{ gets } K_2; \quad N \text{ gets } K_1$$

C. The idealized protocol

Now we idealize the original protocol to the standard form of BAN logic: **Step 1**: $P \rightarrow C : \{\overset{K_n}{\mapsto} N\}_{K_p^{-1}}$; **Step 2**: $P \rightarrow N : \{\overset{K_c}{\mapsto} C\}_{K_p^{-1}}$; **Step 3**: $C \rightarrow N : \{K_1\}_{K_n}$; **Step 4**: $N \rightarrow C : \{K_2\}_{K_c}$; **Step 5**: $C \triangleleft K_2, N \triangleleft K_1$.

D. Symbolizing the assumptions

We formalize the assumptions of the proposed protocol as follows:

$$C \mid\equiv \overset{K}{\mapsto} C,\ N \mid\equiv \overset{K}{\mapsto} N,\ C \mid\equiv \overset{K_p}{\mapsto} P,\ N \mid\equiv \overset{K_p}{\mapsto} P$$

$$P \mid\equiv \overset{K_c}{\mapsto} C,\ P \mid\equiv \overset{K_n}{\mapsto} N,\ P \mid\equiv \overset{K}{\mapsto} P$$

$$C \mid\equiv (P \mid\Longrightarrow \overset{K_n}{\mapsto} N),\ N \mid\equiv (P \mid\Longrightarrow \overset{K_c}{\mapsto} C)$$

$$C \mid\equiv \#(K_1),\ N \mid\equiv \#(K_2)$$

$$C \mid\equiv C \overset{K_1}{\rightleftharpoons} N,\ C \mid\equiv C \overset{K_2}{\rightleftharpoons} N$$

$$C \mid\equiv \#(\overset{K_n}{\mapsto} N),\ N \mid\equiv \#(\overset{K_c}{\mapsto} N)$$

C and N know the public key of P, as well as their own keys. In addition, P knows his own keys, and the public keys of C and N. C and N trust P to correctly sign certificates giving the public keys of them. Also, C and N believe the random numbers that they generate are fresh, secure. Last but not least, C and N assume that the message containing the public key of each other is fresh.

E. Formalization of the security goals

The security goals of the proposed protocol can be formalized as:
Goal 1: C and N trust each other's public keys.

$$C \mid\equiv \overset{K_n}{\mapsto} N,\ N \mid\equiv \overset{K_c}{\mapsto} C$$

Only if C and N believe in the public keys of each other, then the two random numbers K_1 and K_2 can be securely exchanged.
Goal 2: C and N both get the session key if they maintain the same hardware and software state.

$$\frac{C(U') == C(U) \text{ and } N(U') == N(U)}{C \overset{K_{cn}}{\rightleftharpoons} N}$$

F. Formal proof

1) Proof of goal 1: $\because$ the assumption: $C \mid\equiv \overset{K_p}{\mapsto} P$, and the protocol step 1: $C \triangleleft \{\overset{K_n}{\mapsto} N\}_{K_p^{-1}}$, according to rule (1):

$$\therefore \quad C \mid\equiv P \mid\sim \overset{K_n}{\mapsto} N \tag{1}$$

$\because$ the assumption: $C \mid\equiv \#(\overset{K_n}{\mapsto} N)$, and equation (1), according to rule (2):

$$\therefore \quad C \mid\equiv P \mid\equiv \overset{K_n}{\mapsto} N \tag{2}$$

$\because$ the assumption: $C \mid\equiv (P \mid\Longrightarrow \overset{K_n}{\mapsto} N)$, and equation (2), according to rule (3):

$$\therefore \qquad C \mid\equiv \overset{K_n}{\mapsto} N$$

Similarly, we can prove that: $N \mid\equiv \overset{K_c}{\mapsto} C$ Therefore, **Goal 1** is verified.

2) Proof of Goal 2: $\because$ C sealed its private key K_c^{-1} himself (i.e., $C \triangleleft \{K_c^{-1}\}_{seal}$), according to rule (5):

$$\therefore \quad \frac{C(U') == C(U)}{C \triangleleft K_c^{-1}} \tag{3}$$

$\because$ The assumption: $C \mid\equiv \overset{K}{\mapsto} C$, the protocol step 4: $C \triangleleft \{K_2\}_{K_c}$ and equation(3), according to rule (5):

$$\therefore \quad \frac{C(U') == C(U)}{C \triangleleft K_2} \tag{4}$$

$\because$ C generates K_1 and equation (4).

$$\therefore \quad \frac{C(U') == C(U)}{C \triangleleft (K_{cn} = K_1 \oplus K_2)} \tag{5}$$

Similarly, we can prove that:

$$\frac{N(U') == N(U)}{N \triangleleft (K_{cn} = K_1 \oplus K_2)} \tag{6}$$

Consequently, based on equation (5) and equation (6), we can conclude that:

$$\frac{C(U') == C(U) \text{ and } N(U') == N(U)}{C \overset{K_{cn}}{\rightleftharpoons} N}$$

Therefore, **Goal 2** is verified.

V. Comparative Security Analysis

We discuss here the security features of our protocol and compare them to those of the Kerberos-based Hadoop. Table I summarizes the results of the comparison.

As we can see from Table I, Kerberos-based Hadoop relies on passwords to authenticate various entities. This makes Kerberos-based Hadoop vulnerable to all password weaknesses. For example, loss of password leads to the inability to authenticate and hence denial of access to resources. In TPM-based Hadoop, only the ownership of a TPM relies on passwords, a process that is done only few times.

In Kerberos-based Hadoop, the default lifetime of the initial authentication credentials (i.e., Kerberos related tickets) is 24 hours. The three types of tokens for subsequent authentications must be renewed once every 24 hours for up to 7 days, and they can also be cancelled explicitly (e.g., a job token could be cancelled after the completion of the corresponding job). If these authentication credentials are lost, the attacker will be able to access the system/services for a long time. While

TABLE I: Comparison of TPM-based Hadoop and Kerberos-based Hadoop

Kerberos-based Hadoop	TPM-based Hadoop
Password is required to authenticate users. Lost of passwords leads to denial of access to resources.	Ownership of TPM relies on passwords. Lost of passwords will result in sealed/encrypted data inaccessible.
Authentication credentials lifetime: Kerberos TGT = 24 hours. Tokens must be renewed once every 24 hours for up to 7 days or could be cancelled explicitly	Session keys' lifetime could be set to a long period of time by integrating the Fingerprint verification mechanism. The attestation interval could be adjusted according to the user's specific security requirements.
Kerberos lacks of machine/platform integrity verification.	TPM enables hardware/software configuration integrity verification. It provides protection against injections of malware/spyware.
An online Key Distribution Center (KDC) is required. And Kerberos has a single point failure problem.	TPM only requires one-time certification of its AIK, then uses AIK to certify all TPM keys. This reduces the usage of trust third party.

for TPM-based Hadoop, by introducing periodical Fingerprint verification mechanism, the session keys' lifetime could be set to a long period of time and we could adjust the attestation interval to ensure the security requirements of the user.

TPM-based Hadoop provides integrity verification of the platform on each Hadoop entity through remote attestation, a feature that is not provided by Kerberos-based Hadoop. The TPM authenticates the hardware and the software configurations. This feature enables the detection of any tamper in hardware or software configurations, including malware/spyware installations. The authentication credentials exchanged are not only encrypted with the public keys of the parties, but also bound to specific hardware 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, 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 bound to the corresponding PCR values.

Finally, Kerberos-based Hadoop requires a *frequently* queried on-line KDC, which presents a run time single point of failure. The entities (e.g., DataNodes) in Hadoop have to request/renew the security tokens via the KDC every 24 hours or every completion of a job. In contrast, the PCA in our approach is *rarely* used. The PCA is only needed to certify the AIKs of each entity. Once the AIKs are certified, they can be used to sign all kinds of keys generated by the clients' TPM without referring back to the PCA.

VI. Implementation

A. Implementation of the Authentication Protocol

To efficiently manage the complex communication flows, Hadoop utilizes RPC-Dynamic Proxy that creates a simple interface between one client and one server [36].

We divide the implementation of our authentication protocol into three sub-tasks.

Task 1: Includes the exchange of two random numbers K_1 and K_2 between the client and the server. In general, multiple "calls" (i.e., different HDFS/MapReduce commands or operations) may occur within one RPC connection, and we use the first call in each RPC connection to exchange K_1 and K_2. For sending K_1 from the client to the server, we create a new variable $Call.connection.K_1$ in the RPC connection header field, and then use $WriteRPCHeader()$ function in $Client.Java$ to forward it to the server. The server then reads K_1 in $ReadAndProcess()$ function. For sending K_2 from the server to client, we also create a new variable $Call.connection.K_2$ and conditionally (i.e., if K_2 has never been sent) forward it to the client via $SetupResponse()$ function in $Server.java$. The client then decodes K_2 in $ReceiveResponse()$ function.

Note that both K_1 and K_2 are encrypted using the corresponding receiver's public binding key and decrypted via their sealed private binding key which is bound to a certain platform configuration.

Task 2: Includes the dynamic protection of the $Session\_Key = K_1 \oplus K_2$ using TPM seal and unseal operations. After securely exchanging the two random numbers K_1 and K_2, each of the client and the server generates its own copy of the $Session\_Key$. Right after the generation of the $Session\_Key$, a java runtime process is invoked to execute a shell script (i.e., $seal.sh$) that immediately seals the $Session\_Key$ using jTPM commands via jTSS java interface. Whenever there is a need to use the $Session\_Key$, the client/server invokes another java runtime process to conduct a shell script (i.e., $unseal.sh$) to unseal the $Session\_Key$ for encryption or decryption of the data. Fig. 6 illustrates the flow of seal and unseal operations within the various TPM-based Hadoop implementation layers.

Task 3: Includes the management and synchronization of the security credentials (e.g., $Session\_Key$, $Sealed\_Session\_Key$ and $Unsealed\_Session\_Key$). In order to efficiently and securely manage the communication credentials, we build a management and synchronization mechanism for the TPM-based Hadoop. Firstly, we distinguish the $Session\_Key$ by the users (e.g., hdfs, mapred) in the same platform. Since Hadoop utilizes RPC dynamic proxy to simplify the communication, the RPC connections of the same user could share the same $Session\_Key$. This mechanism greatly reduces the number of seal/unseal operations while maintaining the same security level. Secondly, since many RPC connections share the same $Session\_Key$, synchronization issues arise (i.e., simultaneous accesses to the same $Session\_Key$). To handle such issues, we create a file called $session\_key\_list$ that records the IDs of all the $Session\_Keys$ that currently exist. The client/server checks the list before creating or using the $Session\_Key$, and locks the corresponding $Session\_Key$ while using it. Thirdly, we define different access control policy and lifetime for different security credentials. The $Session\_Key$ has the shortest lifetime (i.e., deleted right after sealing it) and could only be accessed by the user who created it and the TPM owner who seals it. The $Sealed\_Session\_Key$ holds the longest lifetime (its lifetime could be adjusted according to the user's security

TABLE II: Access control policy and lifetime for different security credentials

	Session Key	Sealed Session Key	Unsealed Session Key	TPM Sealing Key	Session Key List
Lifetime	Shortest	Longest	Medium	Permanent	Permanent
Access Control	All Users	TPM Owner	User	TPM Owner	All Users

requirements) and could only be accessed by the owner of the TPM (i.e., the one who knows the password of the TPM). The $Unsealed\_Session\_Key$ keeps the medium lifetime (depends on the user's security requirements) and could only be accessed by the user of the corresponding $Session\_Key$. Furthermore, the sealing key (i.e., used for seal/unseal operations) is well protected by the TPM and can only be accessed by the owner of TPM. In addition, the $session\_key\_list$ only contains the IDs of the $Session\_Keys$, thus knowing the contents of the $session\_key\_list$ will not help the attacker to obtain the $Session\_Key$. Table II shows the access control and lifetime for different security credentials.

B. Heartbeat Protocol Implementation

As mentioned in section 3.3, in order to offload the work of the TPM on the NameNode, we introduce the periodic Fingerprint checking mechanism based on the heartbeat protocol in Hadoop.

We introduce a new variable PCR_signed (the attestation PCR value signed by the AIK) in the $DataNodeCommand$ array of the DataNode.java and the NameNode.java. The DataNode will periodically exchange the $DataNodeCommand$ array with the NameNode via heartbeat protocol. After receiving the attestation data, the $FSNamesystem.java$ running on the NameNode verifies the signature and the PCR values' using the Fingerprint pool (which contains all the clients' PCR values). Finally, a response is generated. For example, if the attestation failed, a penalty may be applied (e.g., shut down the corresponding DataNode).

The PCR values are periodically updated through $ReadPCR()$ function in $DataNode.java$ via a runtime process (i.e., $ExtendPCR.sh$). The shell script extends the PCR value using previous PCR value and the new measurements produced by $measurements.sml$, as shown in Fig. 7.

VII. PERFORMANCE EVALUATION

A. Test-bed Design and Configuration

To evaluate the security guarantees and the runtime overhead of our authentication protocol, we compare three different Hadoop implementations, namely, baseline Hadoop (no security), Kerberos-based Hadoop and TPM-based Hadoop (our protocol). For Kerberos, we use krb5.x86_64 [37]. For TPM, we use Software-based TPM Emulator because we target the public cloud which relies on virtualized environment. Additionally, according to IBM's introduction of software TPM, an application that can be developed using the software TPM will run using a hardware TPM without changes[38]. On the other hand, using hardware-based TPMs is easier and provides better performance, therefore, the performance results obtained here will be better if hardware-based TPMs were used. To incorporate TPM with Hadoop project, we modify the source code of Hadoop using ant within eclipse [39] and use IAIK jTSS (TCG Software Stack for the Java (tm) Platform [40]) and IAIK jTpmTools (jTT) as the communication interface between Hadoop and TPM. The detailed test-bed configuration is listed below:

TABLE III: 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

*1) **Hadoop Deployment**:* We configure Hadoop in a test-bed environment that involves Ubuntu 10.04 LTE operating system, 2GB memory, 60 GB hard disk, Java version = Oracle Java-1.7.0_67. There are two DataNodes, two TaskTrackers, one NameNode and one JobTracker in the Hadoop system.

*2) **Hadoop Deployment with Kerberos**:* For Hadoop security design with Kerberos, we choose krb5-server.x86_64, krb5-workstation.x86_64, and krb5-devel.x86_64.

*3) **Hadoop Deployment with TPM**:* Hadoop deployment here involves two parts: (1) virtual TPM configuration and jTSS communication interface configuration; (2)Hadoop source code modification environment setup. We use software-based ETH Zurich virtual TPM [41]. The virtual TPM provides all the functionalities of the hardware TPM. However, the virtual TPM is slower than the hardware TPM and, hence, the overhead results presented for our protocol are upper bounds. The runtime overhead of our protocol is expected to be lower when hardware TPM is used.

B. Runtime Overhead Analysis

It is important to emphasize that developing a new secure authentication protocol, even though is important, is not enough unless it is feasible and practical. Therefore, we have to keep the performance penalty and cost of the added security features within acceptable bounds. In this section, we thoroughly analyze the runtime overhead of our protocol and compare it with the baseline and the Kerberos-based authentication protocols.

The cost of cryptographic processes that prevent replay attacks and ensure data integrity and confidentiality over various communication paths is the same for both TPM-based protocol and Kerberos-based protocol. On the other hand, each protocol has a different overhead that does not exist in the other. The TPM-based protocol incurs a one-time TPM setup overhead that takes place when the TPM in each entity of Hadoop generates the binding keys, AIK, in addition to obtaining certificates for these keys. This is a lightweight overhead and does not create big impact on the day-to-day operations of Hadoop as it is a pre-configuration one time overhead. Table III shows the pre-configuration runtimes for TPM-based Hadoop under our system configurations. On the other hand, Kerberos-based Hadoop has pre-configuration overhead to perform KDC registration (e.g., adding principal to the KDC database) and to acquire and renew TGT (Ticket Granting Ticket).

The runtime overhead of TPM-based Hadoop is mainly due to: (1) the seal operation to encrypt the $Session\_Key$, (2) the unseal operation to retrieve the $Session\_Key$ whenever required, and (3) the extend and read PCR operation of the heartbeat protocol. The number of seal/unseal operations depends on the life cycle of the $Unsealed\_Session\_Key$ (identical to $Session\_Key$ which is deleted right after the seal operation). The life cycle of the $Unsealed\_Session\_Key$ depends on the verification interval value of the Fingerprint process.

The overall runtime overhead of TPM-based Hadoop (T) can modelled as:

$$T = function(Ns \times Ts, Nu \times Tu, Np \times Tp, To)$$

Here, Ns is the number of seal operations; Ts is the time cost for one seal operation; Nu is the number of unseal operations; Tu is the time cost for one unseal operation; Np is the number of extend and read PCR operations; Tp is the time cost for one extend and read PCR operation; To is the other extra time costs (e.g., verification of PCR signature or exchange of two random numbers, which incurs negligible time cost compared to the above TPM operations).

*1) **Overall Runtime Overhead for Different MapReduce Applications**:* In this experiment, we measure the overall runtime of five different MapReduce applications to compare the overhead of the two secure Hadoop implementations relative to the baseline (none secure) implementation of Hadoop. The first two applications are HDFS write and HDFS read, both of which belong to Hadoop TestDFSIO benchmark. These two test cases focus on testing Hadoop IO performance. The third test case is PI example, which calculates the value of π in a distributed way. It has moderate computation workload and low communication traffic. Another test case is TeraSort benchmark, a sorting application that collectively tests the HDFS and MapReduce layers. The last test case is WordCount example, which has moderate communication traffic and small computation load.

Fig. 8 presents the results of this experiment. For Kerberos-based Hadoop, the runtime overhead is about 20% relative to the baseline Hadoop. For the TPM-based Hadoop, the runtime overhead is about 42% for the HDFS write/read and about 25% for the other 3 applications. The relatively high overhead in HDFS write/read is due to the heavy load created by these applications on DataNodes.

*2) **Runtime Overhead under Different Workloads**:* In this experiment, we use the matrix multiplication MapReduce job produced by John Norstad, Northwestern University [42] to test the impact of varying workloads on the overall runtime overhead of the three Hadoop implementations under investigation. We vary the matrix size to generate different workloads. We select three dense matrices with different sizes (i.e., 250x250, 500x500 and 1000x1000). We run the experiment 10 times for each matrix size and compute the average runtime over all the runs. Fig. 9 shows the results. The figure shows that the TPM-based Hadoop has an extra overhead of about 6.1% over that of Kerberos-base Hadoop due to the TPM operations. The runtime overhead decreases as the workload increases for both Kerberos and TPM-based Hadoop. This trend in the overhead is mainly because of the quadruple increase of the original computational time which makes the approximately linear TPM and Kerberos operations relatively smaller.

*3) **Communication Overhead**:* To test the communication overhead, we choose the Sleep example in Hadoop. In the Sleep example, all tasks do nothing but wait for an assigned amount of time. The purpose of this test scenario is to eliminate the computational overhead by making it the same among all implementations. According to previous non-HDFS applications' experiments, the TPM-based Hadoop has an average of 6.7% extra overhead than Kerberos-based Hadoop. After eliminating this difference, the estimated communication overhead (TPM-based Hadoop compared to the Kerberos-based Hadoop) is summarized in Fig. 10. The communication overhead increases as the number of Map tasks increases. This is due to the increase in the number of TPM operations for the additional RPC sessions that deliver Map tasks. TPM has an average of 13.4% communication overhead over Kerberos. This extra communication overhead includes the exchange of the random numbers for session key establishment and the Fingerprint verification operations.

*4) **Variable Session Key Lifetime and Heartbeat Interval Values**:* As mentioned in Section III-D, the number of NameNode side TPM operations decreases significantly as we replace the TPM seal and unseal operations with the Fingerprint verification that is carried out outside the TPM.

Fig. 11 presents the runtime overhead for different session key lifetimes without the Fingerprint verification mechanism. Shorter session key lifetime achieves better security guarantees at the cost of higher runtime overhead. The overhead is mainly due to the unseal operations to retrieve the session key. On the other hand, the Fingerprint verification helps to offload the NameNode's TPM while maintaining the same security guarantees by carefully adjusting the attestation interval value which is based on the heartbeat rate. Fig. 12 shows the runtime overhead relative to the baseline Hadoop for the PI example for various attestation intervals. Intuitively, the figure shows that the higher the attestation interval, the lower the overhead. Also, the higher the lifetime of the session key, the lower the overhead. The former trend is due to the lower number of Fingerprint operations with high attestation intervals; while the latter trend is due to lower number of seal/unseal operations with high session key lifetime. Therefore, by turning the session key lifetime and the attestation interval, we can control the tradeoff between the overhead and security in TPM-based Hadoop.

VIII. Related Work

In early 2013, Project Rhino was launched by Intel as an open source project with a goal to improve the security capabilities of Hadoop. The group proposes Task HADOOP-9392 (Token-Based Authentication and Single Sign-On) which intends 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) [43]. The project mainly focuses on how to extend the current authentication framework to a standard interface for supporting different types of authentication protocols. Nevertheless, all these

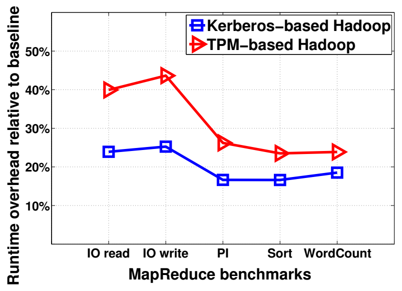


Fig. 8: The runtime overhead relative to the baseline Hadoop on 5 different types of MapReduce applications.

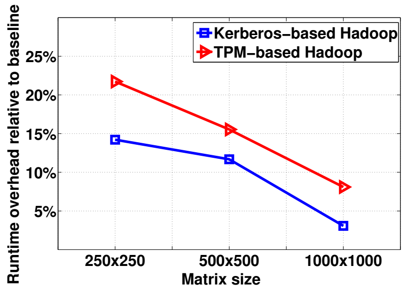


Fig. 9: The runtime overhead relative to the baseline Hadoop under different workloads.

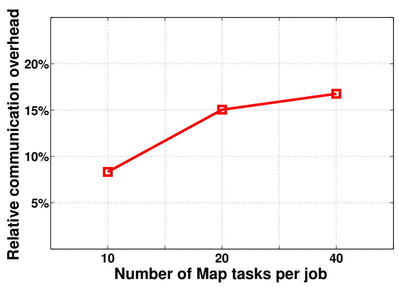


Fig. 10: Estimated communication overhead of TPM-based Hadoop relative to Kerberos-based Hadoop.

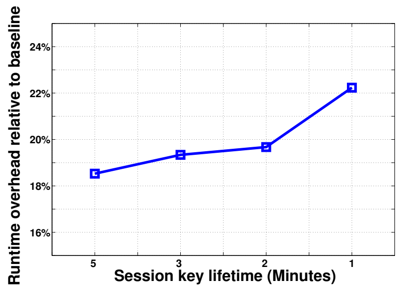


Fig. 11: Runtime overhead of TPM-based Hadoop relative to baseline with variable session key lifetime.

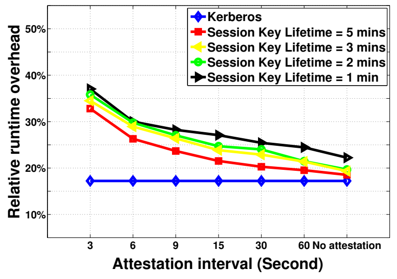


Fig. 12: Runtime overhead of TPM-based Hadoop relative to the baseline with variable Fingerprint attestation intervals for various session key lifetimes

authentication protocols, including Kerberos, are software-based methods that are vulnerable to privileged user manipulations. An insider or possibly an outsider could 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. Rhino design trades off flexibility with complexity. Overall, it enhances the flexibility of the authentication mechanisms at the cost of increasing the complexity of the system.

In [30], the author proposes a TPM-based Kerberos protocol. The proposed protocol is able to issue tickets bound to the client platform through integrating PCA functionality into the Kerberos authentication server (AS) and remote attestation functionality into the Ticket-Granting Server (TGS). However, the proposed mechanism does not provide any attestation for Hadoop's 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 and eliminate the need for any trusted-third-party.

In [44], the authors propose a Trusted MapReduce (TMR) framework that integrates MapReduce systems with the TCG (i.e., Trusted Computing Group) trusted computing infrastructure. 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 without addressing the authentication issues of Hadoop's HDFS and Clients. Therefore, the authors do not provide a general authentication framework for the whole Hadoop ecosystem.

In [45], the authors present a design of a trusted cloud computing platform (TCCP) based on TPM technologies. The proposed design 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 details about how their design will be implemented and no performance evaluations are provided. Also, they fail to provide a complete authentication framework among all the components of Hadoop.

IX. CONCLUSION

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 authenticating 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. Furthermore, our protocol alleviates the use of the trusted third party by using the AIK certification. Moreover, 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.

REFERENCES

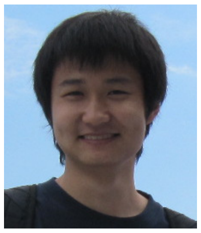
[1] I. Khalil, Z. Dou, and A. Khreishah, "TPM-based authentication mechanism for apache hadoop," in *In 10th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2014.

[2] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation computer systems*, vol. 28, no. 3, pp. 583–592, 2012.

[3] Health Information Trust Alliance (HITRUST), "U.S. Healthcare Data Breach Trends," https://hitrustalliance.net/breach-reports/.

[4] G. White, "Trends in Hardware Authentication," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2015.

[5] S. A. Chaudhry, M. S. Farash, H. Naqvi, S. Kumari, and M. K. Khan, "An enhanced privacy preserving remote user authentication scheme with provable security," *Security and Communication Networks*, 2015.
[6] H. Li, Y. Dai, L. Tian, and H. Yang, "Identity-based authentication for cloud computing," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 157–166.
[7] C. Technology, "Advanced Authentication Methods: Software vs.Hardware," http://www3.ca.com/ /media/Files/whitepapers/ebook-advanced-authenticaiton-methods.PDF.
[8] Trusted Platform Module (TPM): Built-in Authentication, http://www.trustedcomputinggroup.org/solutions/authentication.
[9] 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.
[10] I. Khalil and S. Bagchi, "SECOS: Key management for scalable and energy efficient crypto on sensors," *Proc. IEEE Dependable Systems and Networks (DSN)*, 2003.
[11] O. O'Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell, "Hadoop security design," *Yahoo, Inc., Tech. Rep*, 2009.
[12] Spear Phishing: Real Life Examples, http://resources.infosecinstitute.com/spear-phishing-real-life-examples/.
[13] K. Smith, "Big Data Security: The Evolution of Hadoop's Security Model," http://www.infoq.com/articles/HadoopSecurityModel.
[14] Kerberos, http://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment _Guide-en-US/ch–kerberos.html.
[15] Apache Hadoop, http://hadoop.apache.org.
[16] 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*, 2010, pp. 1–10.
[17] Hadoop Security Analysis, http://www.tuicool.com/articles/ NFf6be.
[18] Trusted Platform Module, http://en.wikipedia.org/wiki/Trusted_Platform_Mo
[19] I. Corporation, "Trusted platform module quick reference guide," 2007.
[20] R. Ng, "Trusted platform module tpm fundamental," *Infineon Technologies Asia Pacific Pte Ltd*, 2008.
[21] R. Perez, R. Sailer, L. van Doorn *et al.*, "vTPM: virtualizing the trusted platform module," in *USENIX*, 2006.
[22] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun, "Enabling secure vm-vtpm migration in private clouds," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 187–196.
[23] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a tcg-based integrity measurement architecture." in *USENIX Security Symposium*, vol. 13, 2004, pp. 223–238.
[24] I. Khalil, A. Khreishah, S. Bouktif, and A. Ahmad, "Security concerns in cloud computing," in *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on*. IEEE, 2013, pp. 411–416.
[25] I. Khalil, A. Khreishah, and M. Azeem, "Cloud computing security: a survey," *Computers*, vol. 3, no. 1, pp. 1–35, 2014.
[26] R. Panta, S. Bagchi, and I. Khalil, "Efficient wireless reprogramming through reduced bandwidth usage and opportunistic sleeping," *Ad Hoc Networks*, vol. 7, no. 1, pp. 42–62, 2009.
[27] S. Bouktif, F. Ahmed, I. Khalil, and G. Antoniol, "A novel composite model approach to improve software quality prediction," *Information and Software Technology*, vol. 52, no. 12, pp. 1298–1311, 2010.
[28] 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*. IEEE, 2011, pp. 263–272.
[29] E. R. Sparks and E. R. Sparks, "A security assessment of trusted platform modules computer science technical report," Tech. Rep., 2007.
[30] A. Leicher, N. Kuntze, and A. U. Schmidt, "Implementation of a trusted ticket system," in *Emerging Challenges for Security, Privacy and Trust*. Springer, 2009, pp. 152–163.
[31] T. T. C. Group, "Virtualized trusted platform architecture specification, version 1.0, revision 0.26," 2011.
[32] S. Sriniwas, "An introduction to hdfs federation," 2011.
[33] M. S. Ridout, "An improved threshold approximation for local vote decision fusion," *IEEE Transactions on Signal Processing*, 2013.
[34] N. Katenka, E. Levina, and G. Michailidis, "Local vote decision fusion for target detection in wireless sensor networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 1, pp. 329–338, 2008.
[35] L. A. Klein, "A boolean algebra approach to multiple sensor voting fusion," *IEEE transactions on Aerospace and Electronic systems*, vol. 29, no. 2, pp. 317–327, 1993.
[36] HadoopRPC, https://wiki.apache.org/hadoop/HadoopRpc.
[37] The KDC and related programs for Kerberos 5, http://linuxsoft.cern.ch/cern/slc5X/x86_64/yum/updates/ repoview/krb5-server.html.
[38] Software TPM Introduction, http://ibmswtpm.sourceforge.net/.
[39] Eclipse, https://www.eclipse.org/.
[40] Trusted Computing for the Java(tm) Platform, http://trustedjava.sourceforge.net/index.php?item=jtss/about.
[41] TPM emulator, http://tpm-emulator.berlios.de/designdoc.html.
[42] A MapReduce Algorithm for Matrix Multiplication, http://www.norstad.org/matrix-multiply/.
[43] Project Rhino, https://issues.apache.org/jira/browse/HADO OP-9392.
[44] A. Ruan and A. Martin, "TMR: Towards a trusted mapreduce infrastructure," in *Services, IEEE Eighth World Congress on*, 2012.
[45] N. Santos, K. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proceedings of the 2009 conference on Hot topics in cloud computing*, 2009.



Zuochao Dou received his B.S. degree in Electronics in 2009 at Beijing University of Technology. From 2009 to 2011, he studied at University of Southern Denmark, concentrating on embedded control systems for his M.S degree. Then, he received his second M.S. degree at University of Rochester in 2013 majoring in communications and signal processing. He is currently working towards his Ph.D. degree in the area of cloud computing security and network security with the guidance of Dr. Abdallah Khreishah and Dr. Issa Khalil.

Issa Khalil received the B.Sc. and the M.Sc. degrees from Jordan University of Science and Technology in 1994 and 1996 and the PhD degree from Purdue University, USA in 2006, all in Computer Engineering. Immediately thereafter he worked as a post-doctoral researcher in the Dependable Computing Systems Lab of Purdue University until he joined the College of Information Technology (CIT) of the United Arab Emirates University (UAEU) in August 2007. In September 2011 Khalil was promoted to associate professor and served as the department chair of the Information Security Department in CIT. In June 2013, Khalil joined the Cyber Security Group in Qatar Computing Research Institute (QCRI), Hamad bin Khalifa University, a member of Qatar Foundation, as a Senior Scientist. Khalil's research interests span the areas of wireless and wireline network security and privacy. He is especially interested in cloud security, botnet detection and takedown, and security data analytics. Dr. Khalil served as the technical program co-chair of the 6th International Conference on Innovations in Information Technology and was appointed as a Technical Program Committee member and reviewer for many international conferences and journals. In June 2011, Khalil was granted the CIT outstanding professor award for outstanding performance in research, teaching, and service.

Abdallah Khreishah received his Ph.D and M.S. degrees in Electrical and Computer Engineering from Purdue University in 2010 and 2006, respectively. Prior to that, he received his B.S. degree with honors from Jordan University of Science & Technology in 2004. During 2009-2010, he worked with NEESCOM. In Fall 2012, he joined the ECE department of New Jersey Institute of Technology as an Assistant Professor. His research spans the areas of network coding, wireless networks, congestion control, cloud computing, and network security.

Ala Al-Fuqaha (S'00-M'04-SM'09) received his M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Missouri, in 1999 and 2004, respectively. Currently, he is an Associate Professor and director of NEST Research Lab at the Computer Science Department of Western Michigan University. His research interests include intelligent network management and planning, QoS routing and performance analysis and evaluation of software-defined networks and VANETs. He is currently serving on the editorial board for John Wiley's Security and Communication Networks Journal, John Wiley's Wireless Communications and Mobile Computing Journal, Industrial Networks and Intelligent Systems Journal, and International Journal of Computing and Digital Systems. He is a senior member of the IEEE and has served as Technical Program Committee member and reviewer of many international conferences and journals.