Communication Pattern based Data Authentication (CPDA) Designed for Big Data Processing in a Multiple Public Cloud Environment

DOI: 10.1109/ACCESS.2020.3000989

Document Version
Final published version

Citation for published version (APA):

Published in:
IEEE Access

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester’s Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.
Communication Pattern Based Data Authentication (CPDA) Designed for Big Data Processing in a Multiple Public Cloud Environment

SOONTORN SIRAPAISAN1, NING ZHANG1, AND QIAN HE2

1Department of Computer Science, The University of Manchester, Manchester M13 9PL, U.K.
2School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China

Corresponding author: Soontorn Sirapaisan (soontorn.sirapaisan@postgrad.manchester.ac.uk)

This work was supported by the University of Manchester under Dean’s Doctoral Scholar Awards.

ABSTRACT With the development of cloud computing, there is a growing trend of multi-cloud Collaborative Big Data Computation (CBDC). In this environment, threats from authorized insiders are of particular concerns. Based on an extreme case of distributed computation where multiple collaborators jointly perform CBDC on shared datasets using an example distributed computing framework, MapReduce (MR), deployed in a Multiple Public Cloud (MPC) environment, this paper investigates how to protect the authenticity of data used during the computation in an efficient and scalable manner by proposing and evaluating a novel data authentication solution. The solution, called a Communication Pattern based Data Authentication (CPDA) framework, ensures data authenticity and non-repudiation of origin at the finest granularity without compromising efficiency and scalability. This is achieved by using an idea of communication pattern based authentication data aggregation. The framework has been comprehensively evaluated both theoretically and experimentally. The evaluation results show that the CPDA framework offers the strongest level of data authenticity protection (equivalent to that provided by digitally signing each data object individually) but introduces much lower overhead cost than the digital signature based solution. The results demonstrate that the idea of communication pattern based authentication data aggregation brings much benefit in terms of supporting efficient and scalable data authentication in a large-scale distributed system.

INDEX TERMS Big data, cloud, data authentication, distributed computing, MapReduce.

I. INTRODUCTION

The emergence of Big Data [1], [2], characterized by high volume, velocity, and variety, and their analyses have led to the advancement of many fields, such as healthcare [3], agriculture [4], and environmental sustainability [5], making innovative applications possible [6]. As part of this, there is a growing trend for inter-organizational data sharing and collaborative data processing on their joint datasets [7], [8]. For efficiency reasons, Big Data processing jobs are preferably carried out by using distributed computing services, and, in many cases, these services are provided by third-party service providers [9].

Such Collaborative Big Data Computation (CBDC) raises some serious security concerns, particularly if the datasets used are mission critical. Examples where CBDC is used include cyber threat analysis and attack detections [10], [11] as well as healthcare and medical condition diagnosis [12]. In the first example, data integrity violations, such as deleting traces of unauthorized access to systems from security logs, may make security breaches go undetected, allowing attackers to proceed with further attacks and causing more harm to the systems. In the second example, contaminated data, e.g., medical or clinical data that are tampered with, may lead to misdiagnoses, causing serious consequences or even loss of life.

Carrying out CBDC in a Multiple Public Cloud (MPC) environment can further complicate the issue. The datasets to be processed as well as the computing and storage components used may be physically located in different geographical locations, managed in different administrative domains, or both. In such cases, the datasets, the components, and the underlying infrastructures are likely to be connected via Wide Area Networks (WANs) or the Internet, which are vulnerable...
to a wide range of security threats and attacks. The lack of national boundaries and the anonymous nature of the Internet make the prevention and detection of threats and attacks much more difficult, if not impossible.

In the CBDC-MPC (CBDC executed on an MPC platform) context, threats imposed by authorized insiders are also a major concern [13], [14]. Unlike external entities, insiders usually have certain privileges to access data and the systems used to process the data, so they have more opportunities to tamper with the data and systems. In addition, for Big Data processing, the requirements of efficiency and scalability are more stringent. Solutions that are not designed for such a context may not satisfy these requirements. As a distributed system is usually optimized to support concurrent data processing, a slight increase in one or both of computational and communication overheads may significantly deteriorate the performance of the entire system.

Data authentication is an essential security requirement for CBDC-MPC. It encompasses data origin authentication and data integrity protection. Data origin authentication ensures that data are indeed from a claimed source, whereas data integrity protection ensures that data have not been tampered with since their origination. Furthermore, owing to insider threats, a data authentication solution developed for the CBDC-MPC context should also provide accountability, a capability to hold every entity accountable for the data it generates. Accountability in this case can be assured by non-repudiation of origin, which is a security property to protect against false denial of data generation.

Most existing data authentication solutions are not specifically designed for the CBDC-MPC context. They may not provide the required security protections or may negatively affect the performance of the underlying data processing system. This paper investigates how to effectively and efficiently achieve data authentication in this context by proposing and evaluating a communication pattern based data authentication solution. For proof of concept and without losing generality, we have chosen MapReduce (MR) [15] as the underlying distributed computing framework to carry out the design of the solution. MR is selected because it captures some vital characteristics of distributed computing, i.e., data are stored in, transferred among, and processed by, multiple distributed machines that are connected via networks. It involves multiple data processing stages and each stage exhibits a different communication pattern. In addition, it is a prominent distributed computing framework widely used in many Big Data processing systems, such as Apache Hadoop [16], Hortonworks Data Platform [17], and MapR Converged Data Platform [18]. MR has also been widely used for research and production purposes. Examples include High Energy Physics group at Caltech [19], Facebook [20], and Twitter [21].

Our novel Communication Pattern based Data Authentication (CPDA) framework, proposed in this paper, is intended to provide the strongest level of data authenticity protection (i.e., assuring data origin and integrity authentication, as well as non-repudiation of origin) at the finest granularity (protecting each data object individually), but with as less overhead cost as possible. The design of CPDA has exploited the use of multiple ideas: (1) communication pattern based authentication data and communication aggregation in which multiple aggregation methods are used, each for a distinctive communication pattern; (2) a hybrid use of different message authentication mechanisms, namely hash functions, Message Authentication Codes (MACs), and digital signatures, such that computationally less expensive mechanisms (i.e., hash functions and MACs) are used to protect individual objects that are transferred between an untrustworthy and a trustworthy entities whereas computationally more expensive mechanisms (digital signatures) are used to protect a group of objects that are produced and consumed by multiple untrustworthy entities; and (3) authentication data for each object are tailored in such a manner that different objects can be verified independently. In this way, we can maximize the level of protection while minimizing overhead cost, regardless of which object is produced by which producer or consumed by which consumer.

To evaluate the effectiveness (the strength of security protections), efficiency (the cost incurred in providing the protections), and scalability (the increase in cost in relation to the scale of the underlying MR service) of the approach, the CPDA framework has been analyzed and evaluated extensively by using both theoretical analysis and real system experiments. The theoretical analysis is conducted by using both qualitative and quantitative methods. The experiments are carried out on a testbed consisted of 5 networked machines hosting up to 400 worker processes with a real-world dataset.

In the remaining part of this paper, we examine the requirements needed to support secure MR based CBDC in an MPC environment. We identify what is missing in existing solutions before describing the design and evaluation of CPDA. In detail, Section II presents a threat analysis in the context and specifies the requirements. Based on the requirements, Section III critically analyzes existing solutions and related work, highlighting areas of improvements. Sections IV, V, VI, and VII respectively, describe high-level ideas, design preliminaries, the low-level design, and a working example of the CPDA framework. Sections VIII, IX, and X, respectively, give the security analysis, theoretical and experimental performance evaluations, of CPDA and compare it with the most relevant solutions. Finally, Section XI concludes the paper and highlights future work.

II. THREAT ANALYSIS AND REQUIREMENT SPECIFICATIONS

To investigate threats and attacks that violate the authenticity of data used in a data processing job, we formulate a generic MR based CBDC-MPC model and carry out a threat analysis based on the model. We then specify a set of requirements for an effective, efficient, and scalable data authentication solution in the context.
A. GENERIC MR BASED CBDC-MPC MODEL

MR, as a generic distributed computing framework, has been adopted in many applications thus architectures. In this model, we have chosen YARN [22], one of the most popular architectures for resource scheduling in distributed computing frameworks, including MR. YARN is more scalable than other architectures for MR such as the one reported in [15]. It can support large scale job executions involving tens of thousands of tasks running on thousands of machines [23]. In addition, it is used in Hadoop [16], one of the most prominent Big Data systems.

The components used by an MR service are machines and containers of computation resources (e.g., application processes) that are hosted on the machines. Based on their functions, the MR components can be largely classified into three groups: client components, data processing components, and data storage components. For generality, each group is assumed to be hosted in a separate cluster, thus leading to three clusters, i.e., a Client cluster, a Distributed Processing System (DPS) cluster, and a Distributed File System (DFS) cluster.

The Client cluster hosts multiple ClientNodes. Each ClientNode hosts one ClientApp container. ClientApp allows a user to submit jobs and input of the jobs, as well as to retrieve the output of the jobs.

The DPS cluster consists of one MasterNode and multiple WorkerNodes. MasterNode hosts one ResourceManager container. ResourceManager manages the resources of the MR service and schedules job executions. Each WorkerNode hosts one WorkerManager container and multiple JobManager, Mapper, and Reducer container sets; each such set serves a particular job execution. WorkerManager manages the resources of the WorkerNode. JobManager schedules and manages the execution of the tasks carried out by Mappers and Reducers assigned to the job. Each Mapper and Reducer, respectively, carry out a map and a reduce tasks. Mappers and Reducers are also collectively referred to as Workers.

The DFS cluster consists of one NameNode and multiple DataNodes. NameNode hosts one NameManager container. NameManager maintains a file system and the metadata of all the files (the input and output of the jobs) and directories, which are stored in the DFS cluster. Each DataNode hosts one DataStore container. DataStore keeps the data blocks (contents) of the files.

An overview of the MR components is given in Fig. 1. In this generic MR based CBDC-MPC model, we assume that each of the three clusters is hosted in a different cloud, different clusters are connected via WANs or the Internet, and machines hosted in the same cluster are connected via Local Area Networks (LANs).

The process of a job execution consists of three phases: the job submission phase, the map phase, and the reduce phase. In the job submission phase, a user uses ClientApp to submit a job execution request to ResourceManager. If the request is accepted, the user divides the input data into multiple instances, called InputSplits, and uploads them onto the DFS cluster. The number of InputSplits is equal to the number of Mappers assigned to the job. In other words, each InputSplit will be processed (i.e., consumed) by a different Mapper. After the user finishes uploading the InputSplits, ResourceManager launches JobManager to manage and orchestrate the execution of this job. JobManager starts Workers (Mappers and Reducers) and monitors the execution of the tasks on the Workers.

In the map phase, each Mapper retrieves the assigned InputSplit from the DFS cluster, executes a map task, and produces an output file, called IntermediateResult. Each IntermediateResult contains multiple data items, called PartitionSegments. The maximum number of PartitionSegments contained in an IntermediateResult is equal to the number of Reducers assigned to the job. Each PartitionSegment will be retrieved and consumed by a different Reducer. The PartitionSegments produced by a Mapper are stored in the local storage of the WorkerNode hosting the Mapper.

In the reduce phase, each Reducer retrieves the assigned PartitionSegments (one from a different Mapper) from the corresponding WorkerNodes, executes a reduce task, and produces an output file, called FinalResult. The FinalResults produced by all the Reducers are uploaded onto the DFS cluster. When the job execution finishes, ClientApp retrieves the FinalResults from the DFS cluster and notifies the user.

The data flows during a job execution are depicted in Fig. 2. From the figure, it can be seen that a different phase of the execution is characterized by a different communication pattern, i.e., the job submission phase is characterized by the one-to-many (O2M) pattern, the map phase by the many-to-many (M2M) pattern, and the reduce phase by the many-to-one (M2O) pattern.

B. THREAT ANALYSIS AND CLASSIFICATIONS

For ease of discussion, data that are used, processed, and generated during a job execution are collectively referred to as JobData. Threats and attacks on JobData authenticity could
be mounted at any of the three levels: the job level, the MR service level, and the cloud (or infrastructure) level.

At the job level, JobData (InputSplits, IntermediateResults, and FinalResults), as indicated in Fig. 2, are exposed to threats and attacks at multiple points. These points include when ClientApp writes InputSplits to the DFS cluster, when a Mapper reads an InputSplit from the DFS cluster and writes an IntermediateResult (PartitionSegments) to the local storage, when a Reducer reads PartitionSegments from different WorkerNodes and writes a FinalResult back to the DFS cluster, and when the ClientApp reads the FinalResults from the DFS cluster. The threats include the injection of fraudulent JobData and tampering of JobData while they are in transit and at rest. In a distributed context such as the one addressed here, a producer, which is a component that produces JobData, delegates the management and storage of the JobData it produces to another component. In other words, the producer is different from a custodian, a component that manages and stores the JobData. Similarly, the custodian is different from a consumer, a component that consumes the JobData. As a result, the producer does not have any control over the JobData it produces and the consumer may not have any, or a sufficient level of, assurance that the JobData it consumes are authentic. If any of these JobData is forged or tampered with at any point during the job execution, the FinalResults will be contaminated. In other words, JobData should not be increased. In other words, JobData should not be revealed to any other components than those that are involved in the processing of the JobData.

At the MR service level, multiple jobs executed on the same MR service hosted on shared machines are submitted by different users. Curious or malicious users may attempt to gain unauthorized access to, or tamper with, other users’ JobData. They may do so directly or via compromising service components. For example, a Worker could be compromised and used to gain access to the local storage and tamper with other users’ JobData.

At the cloud level, attacks could be mounted against JobData-in-transit and JobData-at-rest due to insecure communication channels and the resource sharing nature of clouds. JobData transferred from one cloud to another (i.e., inter-cluster data transfer) are through open and insecure WANs or the Internet. These data are susceptible to interceptions and unauthorized alterations. These attacks could be mounted on the input and output data that are being written to, or read from, the DFS cluster. In addition, resource sharing in clouds allows multiple tenants to access shared resources. By exploiting vulnerabilities or misconfiguration, tenants of the same cloud but external to the MR service may also gain access to JobData used by the MR service.

To summarize, the threats and attacks discussed above can be classified into the following three categories:

T1) Data injection attacks: These attacks refer to unauthorized attempts to inject new instances of fraudulent JobData at any points of the data flow.
T2) Data tampering attacks: These attacks refer to unauthorized alterations to JobData, such as adding, modifying, deleting some portions of JobData. T1 and T2 are external attacks. These attacks can lead to the contamination of the results of a job execution.
T3) Repudiation attacks: These attacks refer to any false denials of the generation of JobData. Repudiation is commonly used to evade responsibility or accountability. T3 are insider attacks. Addressing these attacks is necessary in a collaborative environment, such as the CBDC-MPC context, where multiple organizations are involved and datasets from multiple organizations are used.

C. REQUIREMENT SPECIFICATIONS

Based on our generic CBDC-MPC model and threat analysis, we here specify a set of requirements for an effective, efficient, and scalable data authentication solution. The requirements are used as the criteria for related work analysis and to guide the design of our CPDA framework. The requirements are classified into functional, security and performance requirements.

1) FUNCTIONAL REQUIREMENTS
FR1) Full-cycle protection: All JobData should be protected throughout the entire job execution cycle, from when the input is being submitted to the MR service to when the output of the job is retrieved by the user who submitted the job.
FR2) Fine-grained verifiability: JobData objects should be individually verifiable. This is necessary as objects produced by a producer are consumed by different consumers, objects assigned to a consumer are produced by different producers, and different consumers may consume the assigned objects at different times.
FR3) Limited JobData exposure: In providing data authentication, the exposure of JobData should not increase. In other words, JobData should not be revealed to any other components than those that are involved in the processing of the JobData.
2) SECURITY REQUIREMENTS

SR1) Data origin authentication: The origin of each object should be verifiable to ensure that the object is indeed produced by the claimed producer. This requirement is used to counter data injection attacks (T1).

SR2) Data integrity protection: The integrity of each object should be verifiable to ensure that the object has not been tampered with since its generation. This requirement is used to counter data tampering attacks (T2).

SR3) Non-repudiation of origin: The generation of each object should be bound to its producer so that any false denial of its generation can be detected. This requirement is used to counter repudiation attacks (T3).

3) PERFORMANCE REQUIREMENTS

PR1) High efficiency: The overheads imposed on a job execution as a result of achieving data authentication should be as low as possible. The overheads are considered in two aspects: (1) computational overhead, i.e., computational cost of generating and verifying authentication data; and (2) communication overhead, i.e., the amount of authentication data transmitted over networks.

PR2) High scalability: When the number of components and the volume of JobData increase, the rate of increase in the overheads should be no more than linear.

III. RELATED WORK OVERVIEW

Existing data authentication solutions designed for networked and distributed systems can be largely classified into two groups: those that are designed for non-MR based systems (referred to as non-MR specific solutions) and those that are specifically designed for MR based systems (referred to as MR specific solutions).

A. NON-MR SPECIFIC SOLUTIONS

Depending on the cryptographic schemes used, non-MR specific solutions can be further classified into three groups: secret-share based, symmetric-key based, and asymmetric-key based.

In a secret-share based solution, authentication data are generated with a secret but verified with a secret-share derived from the secret. The secret is divided into $N$ secret-shares. Any $k$ or more (out of $N$) secret-shares can be used to reconstruct the secret, but $k - 1$ or fewer secret-shares cannot. Desmedt et al. [24] proposed such a scheme for multicast services where one producer sends an object to multiple consumers. In this scheme, for each object, the producer uses two polynomials of degree $k - 1$ (known only to the producer) to generate authentication data and $N$ secret-shares. Each secret-share is distributed to a different consumer so that each consumer can independently verify the authentication data. Safavi-Naini and Wang [25][26] improved on the Desmedt’s scheme by reducing the number of polynomials required to authenticate multiple objects. To authenticate $m$ objects, the producer uses only $m + 1$ polynomials, as opposed to $2*m$ polynomials as required by the Desmedt’s scheme. This cuts the costs in generating and storing authentication data by half. Nonetheless, the secret-share based solutions incur a high level of computational overhead due to the cost of computing polynomials of degree $k - 1$, especially when $k$ is large.

Symmetric-key based data authentication solutions, such as MACs, are designed to counter external attacks. They do not provide non-repudiation protections, making them vulnerable to threats, e.g. tag forgeries, imposed by authorized insiders. To address this issue, the idea of asymmetry is used. There are two forms of asymmetry: information asymmetry and time asymmetry. With an information-asymmetry based scheme [27], a producer has a full view of a secret (a set of secret keys) whereas each consumer has only a partial view of the secret (a subset of the secret keys). The entire set of the secret keys is used to generate authentication data (tags), whereas a subset of the secret keys is used to verify the tags. A subset of the secret keys is made available for each consumer and these subsets are different from each other. Tag forgeries are countered by limiting the number of secret keys revealed to each consumer. This approach incurs a high level of computational as well as storage overheads, as multiple tags are processed (generated and verified) and multiple secret keys are required for the processing of such tags.

With a time-asymmetry based scheme, tag forgeries are countered by controlling when a secret key is used for generating tags and when the key is being made available for verifying the tags. In other words, the secret key is used to generate tags in one time period, and it is released for the verification of the tags in another time period. Examples of such schemes include Chained Stream Authentication (CSA) [28], Timed Efficient Stream Loss-tolerant Authentication (TESLA) [29]–[31], and µ TESLA [32]. Although delaying the release of the keys does not introduce additional computational overhead, it increases the data processing time and offsets the benefit of parallel computations provided by distributed computing frameworks such as MR.

To ensure data authenticity and provide non-repudiation of origin, digital signatures are frequently used. With a digital signature based solution, two asymmetric keys (a private key and a public key) are used, respectively, for generating and verifying authentication data (signatures). As long as the public key is certified and the private key is kept secret, it is computationally infeasible for another entity, rather than the owner of the private key, to forge signatures. However, signature operations (generation and verification) are computationally expensive, much more expensive than MAC operations [33]. In addition, for the same security level, the lengths of asymmetric keys and signatures are
usually much longer than those of symmetric keys and tags, respectively [34]–[36]. Therefore, using digital signatures to secure individual objects in Big Data applications are neither efficient nor scalable.

A number of data authentication schemes have been proposed with an intention to reduce the number of signatures used. These schemes employ a signature amortization technique. Such a technique builds a chain of authentication data in a way that the authentication data of one object are linked to those of other objects. In this way, only a subset of the objects is signed but the protection is provided to the whole set of the objects. Related work in this category has been focusing on how to construct authentication data chains so that the dependency among the objects and the amount of authentication data embedded in the objects can be reduced. In the method proposed by Gennaro and Rohatgi [37], a chain of authentication data is constructed by embedding the authentication data of one object in the preceding object and the first object is signed with a digital signature scheme. This method is not designed for applications where data are sent over unreliable networks as the loss of one object would make the succeeding objects unverifiable. A number of schemes have been proposed to address this limitation, and these are Efficient Multi-chained Stream Signature (EMSS) [29], p-Random Authentication [38], the piggybacking scheme [38], Golle and Modadugu’s scheme [39], and Adaptive source Authentication protocol for multiCAST streams (A2Cast) [40]. The essence of these schemes is to embed the authentication data of one object in a number of other objects. In this way, the remaining objects will still be verifiable even if some of the objects are lost. Nonetheless, this is achieved by using redundancy and at the cost of increased communication and storage overheads.

To reduce the redundancy thus the overheads, a number of schemes employing error correction codes are proposed. These schemes are Signature Amortization using IDA (SAIDA)[41], [42] and Data Authentication Model based on Reed-Solomon Error-correcting Code (DAM-RSEC) [43]. In these schemes, the hashes of the whole set of objects and the signature of the aggregated hashes are encoded with an error correcting code and the resulting code is split and embedded in the objects. In this way, the authentication data can be reconstructed from a subset of the objects and the amount of authentication data carried by each object is reduced.

All the above schemes are designed for multicast and broadcast services where there is only one data producer but multiple consumers. A major limitation of these schemes when being applied to the CBDC-MPC context is that they do not allow each of the objects to be independently verifiable. An exception is the scheme proposed by Kei Wong and Lam [44]. This scheme allows the verification of individual objects while reducing the number of objects to be signed and verified. This is done by constructing a hash tree of the objects and signing only the hash of the root node of the tree. The verification of a particular object is done by using the hash of the object, the hashes along the path leading to the root node, and the signature.

As can be seen from the above discussions that none of the existing solutions is readily applicable to the CBDC-MPC context which is characterized by the following characteristics: (1) each job execution involves multiple data processing stages and (2) at each stage, a different communication pattern is implied. In addition, there are stringent requirements for timeliness and high scalability due to the nature of Big Data computation.

B. MR SPECIFIC SOLUTIONS

Data authentication solutions specifically designed for MR applications can be largely classified into two groups, task-replication based and non-task-replication based.

As indicated by the name, task-replication based solutions use task replication to ensure the correctness of JobData that are generated during a job execution, thus providing data integrity protection. With a task-replication based solution, each data processing task (a map or reduce task) is assigned to multiple Workers and the outputs produced by these Workers are compared to detect inconsistencies. This approach has been used by a number of schemes published in literature, and these are Verification-based Integrity Assurance Framework for MR (VIAF) [45], Cross Cloud MapReduce (CCMR) [46], IntegrityMR [47], Verification-based Anti-collusive Worker Scheduling (VAWS) [48], and Hadoop Trust MANager (Hatman) [49]. However, the above schemes do not protect against repudiation of origin attacks. AssureMR [50] and CorrectMR [51] improve on this by making use of a Pedersen-Merkle-R-Tree based authenticated data structure and a digital signature scheme in addition to task replication. Task replication imposes a high level of resource requirements; it multiplies the computational resource required to process each task. In addition, the approach also depletes scalability. To lower the resource requirements, TrustMR [52], Trusted Sampling-based Third-party Result Verification (TS-TRV) [53], and Accountable MR [54], [55] are proposed. These schemes replicate only a subset of the tasks, thus reducing the resource consumption. Nonetheless, they still introduce a high level of overhead cost and do not protect against repudiation of origin attacks.

SecureMR [56] counters such attacks by employing task replication in conjunction with other measures, namely a commitment protocol, a verification protocol, and a digital signature scheme. However, like the earlier mentioned task-replication based schemes, the approach is still costly. More importantly, the task-replication based schemes mostly apply protections at the task level; they do not provide fine-grained, or object-level, protections.

Non-task-replication based solutions make use of cryptographic primitives and security protocols to protect the authenticity of JobData. The most notable solution is the one employed by Hadoop [16]. In this solution, a number of security measures are taken [57]. To protect data-in-transit, it uses the Simple Authentication and Security Layer (SASL)
framework, encryption schemes (e.g., AES), and Hyper-text Transfer Protocol Secure (HTTPS) to, respectively, protect messages transmitted over Remote Procedure Call (RPC), Transmission Control Protocol over Internet Protocol (TCP/IP), and HTTP protocols. However, these security measures are intended for countering external attacks. They do not provide data authenticity protection to data-at-rest, and they are intended for an MR service deployed in a single domain. In [58], Zhou et al. proposed a secure data processing system for distributed computing services, called Declarative Secure Distributed System (DS2). As a proof of concept, the system is used to implement an MR service with a data authentication facility, called Authenticated MapReduce. In this system, each JobData object produced by a Mapper is signed with a data authentication scheme, such as HMAC-SHA1 (MAC) or RSA-1024 (digital signature). Their experimental results show that, with the respective use of HMAC-SHA1 and RSA-1024, the query completion latency of a job execution is increased by 17.4% and 78.3%, in comparison with the case where no data authentication measure is used. This indicates that, when using a MAC scheme (HMAC-SHA1), the protection level is insufficient as non-repudiation of origin is not provided, but when applying the digital signature scheme to secure each individual object (in order to provide non-repudiation), a significant level of delay is added onto a job execution process and is highly inefficient.

In summary, none of the existing data authentication solutions discussed above satisfies all the requirements specified for the CBDC-MPC context (as set in Section II-C). In particular, none of the solutions provides a full-cycle and object-level JobData protection (FR1 and FR2). The novel solution to be described in the remaining part of this paper is intended to address this knowledge gap; it offers security protections to JobData in terms of data authenticity (SR1 and SR2) and non-repudiation of origin (SR3) while being highly efficient (PR1) and scalable (PR2).

IV. HIGH-LEVEL IDEAS

This section gives high-level ideas used for the design of CPDA. For ease of discussion, data that are used for providing the specified security protections are collectively referred to as Authentication Data (AuthData).

The overhead imposed on data processing components (producers and consumers) in providing the specified security protections (SR1, SR2, and SR3) can be reduced if we could reduce the costs incurred in generating and verifying AuthData. These reductions should be done without compromising the functional requirements (FR1, FR2, and FR3) and the performance requirements (PR1 and PR2). We achieve this objective by using the following two ideas. Firstly, we apply and maximize the use of aggregation to AuthData and communications. AuthData aggregation means combining, or producing a digest of, multiple AuthData instances, such that the aggregated (combined or digested) AuthData can be used by multiple components. If we only apply digital signature operations to aggregated AuthData, we can reduce the number of costly operations imposed on each data processing component. Communication aggregation means combining or grouping multiple interactions taking place among the data processing components into a smaller set of interactions. This can reduce the number of communications (interactions) among the components, thus reducing processing cost imposed on each component. This can also reduce bandwidth cost in the network. Communication aggregation can be done by introducing a third-party aggregator (referred to as Aggregator). Depending on the communication pattern used, AuthData aggregation and communication aggregation can be applied separately, or in a hybrid manner, to maximize the benefits they both bring. We have thus adopted a communication pattern based approach, i.e., we identify and classify different communication patterns among the data processing components during different phases of a job execution and apply one or both of AuthData aggregation and communication aggregation accordingly.

The second idea employed is a hybrid use of MAC and digital signature schemes, in conjunction with segregation of credentials. As mentioned earlier, a MAC scheme is computationally more efficient but does not provide non-repudiation of data origin, whereas a digital signature scheme provides the non-repudiation protection but is computationally expensive. To provide all of these protections at the finest granularity but with minimal overhead, we apply the MAC scheme to AuthData instances that are pairwise transmitted (between one producer and Aggregator) but apply the signature scheme to aggregated AuthData that are used by multiple consumers. This hybrid use of cryptographic scheme can ensure the accountability of producers. With regard to credential segregation, pairwise keys used by different producers are segregated. In other words, each producer uses a different key to generate AuthData. This narrows the scope of accountability to the two entities sharing a key.

In the following, we describe, at a high level, how the two ideas described above are implemented. With regard to the communication pattern based AuthData aggregation, as there are three communication patterns (identified in Section II-A), i.e., the one-to-many (O2M) pattern taking place in the job submission phase, the many-to-many (M2M) pattern in the map phase, and the many-to-one (M2O) pattern in the reduce phase, three AuthData aggregation methods are designed, one for each pattern. The three methods are, respectively, called Tree based AuthData Aggregation (TreeAgg) for O2M, Hybrid AuthData Aggregation (HybridAgg) for M2M, and Flat AuthData Aggregation (FlatAgg) for M2O. Before describing HybridAgg, we explain TreeAgg and FlatAgg as these two methods are used as building blocks for the design of HybridAgg.

A. TREEAGG METHOD

In the O2M pattern, there is one producer but multiple consumers. The producer produces multiple, typically dissimilar, objects, one for each consumer. To minimize the cost in
protecting these objects, we should require the producer to perform only one signature signing operation, but the resulting AuthData should allow each consumer to verify the object assigned to it independently.

The TreeAgg method is designed to accomplish this function. With this method, a binary tree containing aggregated AuthData for the whole set of \( N \) objects is constructed. The tree consists of \( N \) leaf nodes and \( N - 1 \) internal nodes layered at multiple levels. Each leaf node represents the AuthData of a different object. Each internal node at the next level up in the hierarchy represents aggregated AuthData derived from its children (child nodes). The internal node at the top level is called the root node. The root node is the aggregated AuthData (referred to as Root-AuthData) for the entire set of the objects. The signature is then signed on the Root-AuthData. To minimize the amount of AuthData needed to verify each individual object, we make this tree a balanced full binary tree.\(^2\)\(^\[44\]\). An example of an AuthData tree for 8 objects is shown in Fig. 3.

![FIGURE 3. An AuthData tree for 8 objects.](image)

AuthData needed for the verification of each object are object dependent, i.e., for different objects, their respective AuthData are different. This is because, as mentioned earlier, objects consumed by different consumers are typically different, and paths connecting each object to the root (Root-AuthData) of the tree are different. Furthermore, to minimize the size of each such AuthData thus minimizing the computational and communication overheads, any redundant item in AuthData should be excluded. For these reasons, the AuthData associated to a particular object are constructed as the signature of the Root-AuthData (this token is the same for all of the objects) along with a set of object-specific AuthData tokens (these tokens are specifically tailored for each consumer). The object-specific AuthData tokens (referred to as Sibling-AuthData) are the sibling nodes along the path from the leaf node (associated to the object) to the root node. The Sibling-AuthData for an object is illustrated in Fig. 4. From the figure, we can see that the Sibling-AuthData for \( d_3 \) consists of three tokens: the AuthData of \( d_4 \), the AuthData of \( d_1 \) to \( d_2 \), and the AuthData of \( d_5 \) to \( d_8 \). Comparing with using all the leaf nodes to reconstruct the whole tree (thus the other seven nodes have to be transmitted along with the signature of Root-AuthData), our approach yields a reduction of 50% in communication overhead in terms of the number of tokens transmitted.\(^3\)

![FIGURE 4. The Sibling-AuthData for \( d_3 \) in an AuthData tree for 8 objects.](image)

Fig. 5 contrasts the process and AuthData tokens sent by a producer to \( Q \) consumers with and without applying the TreeAgg method. As shown in the figure, when TreeAgg is not applied, the producer would need to sign the AuthData of each of the \( Q \) objects, respectively, before dispatching them to \( Q \) consumers with and without applying the TreeAgg method. As shown in the figure, when TreeAgg is not applied, the producer would need to sign the AuthData of each of the \( Q \) objects, respectively, before dispatching them to

\(^2\)A balanced full binary tree is a tree in which every internal node has exactly two child nodes and the left and the right subtrees of every node differ in height by no more than one. The height of a tree for \( N \) objects is \( \lceil \log N \rceil \).

\(^3\)The tree-reconstruction approach requires 8 tokens (1 signature of the Root-AuthData and 7 Sibling-AuthData tokens) to be transmitted whereas our approach requires only 4 tokens (1 signature of the Root-AuthData and 3 Sibling-AuthData tokens) to be transmitted, thus the reduction of \( 4/8 = 50\% \).
the consumers. This means that the producer needs to perform $Q$ signature signing operations. In contrast, when TreeAgg is applied, the producer only needs to perform one AuthData aggregation operation and one signing operation.

**B. FLATAGG METHOD**

In the M2O pattern, there are multiple producers but one consumer. Objects produced by different producers are typically different and each object is signed by a respective producer. To minimize computational overhead incurred in verifying the objects, we should require the consumer to perform only one signature verification operation. To achieve this, we have introduced an idea of a third-party based aggregation method. The third party, called Aggregator, off-loads computational overhead away from the consumer as much as possible. It obtains and verifies AuthData generated and signed (with a MAC scheme) by different producers, then generates aggregated AuthData and signs (with a digital signature scheme) the aggregated AuthData before dispatching both the aggregated AuthData and the signature to the consumer. The size of the AuthData has also been reduced as much as possible to minimize bandwidth consumptions.

These measures have been captured in the FlatAgg method. Fig. 6 illustrates the AuthData exchanged among the producers and the consumer with and without the use of the FlatAgg method.

![FIGURE 6. AuthData transmitted among components in the M2O pattern. (a) Without the use of the FlatAgg method. (b) With the use of the FlatAgg method.](image)

**C. HYBRIDAGG METHOD**

The M2M pattern can be viewed as the integration of the O2M and M2O patterns. In this pattern, there are multiple ($P$) producers and multiple ($Q$) consumers. Each producer produces up to $Q$ typically dissimilar objects, one for a different consumer. Each consumer consumes up to $P$ objects, one from a different producer. Without any additional measures, each producer will need to sign $Q$ objects, and each consumer will need to verify $P$ objects. As mentioned earlier, to minimize overheads, each producer should only perform one MAC signing operation, and each consumer should only perform one signature verification operation. To accomplish this, we apply two levels of AuthData aggregation, i.e., the intra-producer level aggregation and the inter-producer level aggregation. The intra-producer level aggregation is performed by each producer by using the TreeAgg method to aggregate the AuthData for the objects it produces, but only signing the Root-AuthData with a MAC scheme. The inter-producer level aggregation is performed by the Aggregator; as described in the FlatAgg method, it verifies and aggregates AuthData generated by different producers, and then signs the aggregated AuthData using a digital signature scheme.

This idea has been implemented in the HybridAgg method. Fig. 7 shows the flows of AuthData exchanged among components with and without the use of the HybridAgg method.

The second idea, i.e., a hybrid use of MAC and digital signature schemes in conjunction with the segregation of credentials, is implemented in the FlatAgg and HybridAgg methods. As explained in these methods, a MAC scheme is used to protect AuthData transferred between each producer and Aggregator and a digital signature scheme is used to protect aggregated AuthData dispatched by the Aggregator to consumers. For pairwise transmitted AuthData, MAC can provide a sufficient level of protection. This is because the key used between a producer and the Aggregator is a pairwise key, the AuthData from each producer will be further aggregated and digitally signed by the Aggregator, and the Aggregator is trustworthy (to be explained Section V-A). If any fraudulent AuthData is detected, its origin can be traced via the verification of the signature signed by the Aggregator and the pairwise key used by the originator of the AuthData. The use of different pairwise keys captures the segregation of credentials. This, together with the use of the Aggregator, which is delegated with the task of inter-producer AuthData aggregation, allows us to achieve the specified security goal, while, at the same time, off-loading some computational burden away from producers and consumers.

**V. DESIGN PRELIMINARIES AND BUILDING BLOCKS**

**A. DESIGN PRELIMINARIES**

Design preliminaries cover the threat model and assumptions used in the design, as well as the notations used in describing the design.

1) **THREAT MODEL**

This work focuses on countering insider threats imposed on the authenticity of data (JobData) used during the computation of Big Data. The threat model used and the solution designed take into account of the characteristics of the
underlying Big Data computing platform, which is MR in this case. Existing threat models (or standard threat models) do not take into account of the characteristics of MR; they do not differentiate management and data-handling components, therefore not suited to our problem context. The threat model defines the trust boundary of the MR components. These components are classified into two groups: management components and data-handling components. The management components are components whose functions are to manage computation resources or supervise task executions. These components are ResourceManager, NameManager, Worker-Managers, and JobManager. The data-handling components are components whose functions are one or a combination of producing, consuming, and storing JobData. These components are ClientApps, Mappers, Reducers, and DataStores.

As the functions of management components are important to job executions and to the system, there should be security measures put in place to protect these components. There are a number of existing solutions which can be used to strengthen the protections. An example of such solutions is Trusted Computing [59]. Data-handling components, on the other hand, may not have the same (or adequate) level of protections or security assurance. This is because, firstly, these components are in large quantities, so providing strong security protections to all of them may not be practical due to efficiency reasons. Secondly, unlike management components which interact with only other software components, data-handling components also interact with users at large or executing user-supplied codes (map and reduce functions) which may contain vulnerable or malicious codes. Thirdly, data are assets, hence, data-handling components may be more attractive to attackers. Owing to these reasons, data-handling components are more vulnerable to threats and attacks. They could be compromised for attacking the data or be used as a springboard for further attacks against the system.

Based on the above considerations, our threat model is defined as follows:

**TM1)** The management components are trustworthy; they will perform their functions faithfully.

**TM2)** The data-handling components are untrustworthy; they may be malicious and attack the authenticity of JobData using any of the attack methods highlighted in Section II-B.

In addition, entities that are external to an MR job or the MR service, including those on the Internet, are untrustworthy; they may gain access to the shared resources and launch attacks on the authenticity of JobData.

2) **ASSUMPTIONS**

The following assumptions are used in our solution design.

A1) Users are already authenticated prior to accessing the MR service.

A2) The MR components serving a particular job form an isolated virtual domain called JobDomain [60] and the components in a JobDomain are already authenticated prior to executing the job.

A3) All the cryptographic keys that are used in data authentication are established when the MR components are authenticated; the public keys are certified and known to their respective users.

3) **NOTATIONS**

The notations used in the description of CPDA are shown in Table 1.

**B. BUILDING BLOCKS**

In designing CPDA, we have used a number of schemes as its building blocks, and these are hash functions,
MAC schemes, digital signature schemes, and hash trees. For ease of understanding of CPDA to be described in the next section, in this section, we list the algorithms of these schemes along with their interfaces, including their input and output. Any implementation of each of these schemes can be used interchangeably as long as it supports the defined interfaces.

1) HASH FUNCTIONS
A hash function (also known as a cryptographic hash function) is used to generate a digest of an object, and the digest is called a hash. Some examples are SHA-2 [61], SHA-3 [62], and BLAKE2 [63]. A hash function contains the hash generation algorithm. This algorithm takes a variable-length object \( d \) as input and returns a fixed-length hash \( h \) as output, denoted as \( h = H(d) \).

2) MACS
A MAC scheme is a symmetric-key based data authentication scheme which provides two security properties: data origin authentication and data integrity protection. Some examples are HMAC [64], OMAC [65], and UMAC [66]. It contains two algorithms, signing and verification.

MA1) MAC Signing (MAC-Signing): MAC-Signing is an algorithm that accepts a secret key \( k \) and an object \( d \) as input and returns a tag \( \tau \) as output, denoted as \( \tau = MS(d, k) \).

MA2) MAC Verification (MAC-Verification): MAC-Verification is an algorithm that accepts a secret key \( k \), an object \( d \), and a tag \( \tau \) as input and returns a verification result \( mv \) as output, denoted as \( mv = MV(k, d, \tau) \). The output is either positive or negative.

3) DIGITAL SIGNATURES
A digital signature scheme is an asymmetric-key based data authentication scheme which provides three security properties: data origin authentication, data integrity protection, and non-repudiation of origin. Some examples are RSA [67], DSA [68], and ECDSA [68]. It contains two algorithms, signing and verification.

S11) Digital Signature Signing (SIG-Signing): SIG-Signing is an algorithm that accepts a private key \( sk \) and an object \( d \) as input and returns a signature \( \sigma \) as output, denoted as \( \sigma = SS(sk, d) \).

S22) Digital Signature Verification (SIG-Verification): SIG-Verification is an algorithm that accepts a public key \( pk \), an object \( d \), and a signature \( \sigma \) as input and returns a verification result \( sv \) as output, denoted as \( sv = SV(pk, d, \sigma) \). The output is either positive or negative.

4) HASH TREES
A hash tree, also called a Merkle tree [44], [69], [70], is a tree containing aggregated hashes for a set of objects. In a hash tree, each leaf node is the hash of a respective object and each internal node is the hash of the concatenation of its child nodes. The root node (also called the root hash) is the aggregated hash of all the objects. The hash tree used in CPDA contains three algorithms, Hash Tree Construction, Sibling-AuthData Extraction, and Root-AuthData Recovery.

HTA1) Hash Tree Construction (HT-Construction): HT-Construction takes a set of hashes \( h_1, h_2, \ldots, h_N \) as input and returns a hash tree \( ht \) as output, denoted as \( ht = HTC(h_1, h_2, \ldots, h_N) \).

HTA2) Sibling-AuthData Extraction (SA-Extraction): SA-Extraction takes a hash tree \( ht \) and a hash \( h_i \) (of an object \( d_i \)) as input and returns a Sibling-AuthData token containing a set of hashes and their positions (i.e., left or right) along the path from \( h_i \) to the root node as output, denoted as \( sa_i = SAE(ht, h_i) \). The number of the hashes contained in each \( sa_i \) is at most the height of the tree, i.e., \( \lceil \log N \rceil \) (for a binary hash tree).

HTA3) Root-AuthData Recovery (RA-Recovery): RA-Recovery takes a hash \( h_i \) and a Sibling-AuthData token \( sa_i \) as input and returns a root hash \( rh \) as output, denoted as \( rh = RAR(h_i, sa_i) \).

VI. CPDA IN DETAIL
This section describes our novel data authentication solution, the CPDA framework. We give an overview, and then the detailed description, of the framework. In the last subsection, we put together all the methods and protocols of this framework and show how the framework is used to protect JobData throughout the whole cycle of a job execution. The algorithms implementing the methods used in the framework are formally described in Appendix.
A. AN OVERVIEW OF THE CPDA ARCHITECTURE

As mentioned in the generic model (Section II-A), an MR job execution comprises three phases, and each phase is characterized by a different communication pattern. Hence, the CPDA architecture consists of three modularized functional blocks, one for each job execution phase. These functional blocks are, respectively, the O2M block for the job submission phase, the M2M block for the map phase, and the M2O block for the reduce phase. An overview of the CPDA architecture is depicted in Fig. 8.

FIGURE 8. An overview of the CPDA architecture.

As shown in the figure, each functional block consists of two AuthData generation algorithms (with an exception of the O2M block which has only one algorithm), one AuthData verification algorithm, and one AuthData delivery protocol. The AuthData generation algorithms are used to generate AuthData for JobData objects. They each utilize one of the three AuthData aggregation methods explained in Section IV. The AuthData verification algorithms are used to verify the authenticity of objects with the generated AuthData. The AuthData delivery protocols are used to deliver the AuthData to consumers. The delivery of AuthData is decoupled from the built-in JobData delivery mechanisms so that CPDA is not tightly bound to a specific MR implementation. In this way, it can be applied as an add-on and the modifications made to the underlying MR service are minimal.

Before describing the three functional blocks in detail, we first explain two AuthData aggregation algorithms (collectively implementing the three AuthData aggregation methods) and generic protocol message structure.

B. AUTHDATA AGGREGATION ALGORITHMS

The TreeAgg, FlatAgg, and HybridAgg methods can be realized by two AuthData aggregation algorithms, namely Hash-Tree based AuthData-Aggregation (HT-AuthData-Aggregation) and Hash-Concatenation based AuthData-Aggregation (HC-AuthData-Aggregation). TreeAgg is implemented by HT-AuthData-Aggregation, FlatAgg by HC-AuthData-Aggregation, and HybridAgg by both of the algorithms.

1) HT-AUTHDATA-AGGREGATION ALGORITHM

The HT-AuthData-Aggregation algorithm uses a balanced full binary hash tree to aggregate AuthData for a set of C objects produced by a producer x. It takes the hashes $h_{x,1}, h_{x,2}, \ldots, h_{x,C}$ of the objects as input and returns a root hash $rh_x$ and a set of Sibling-AuthData tokens $SA_x = \{sa_{x,1}, sa_{x,2}, \ldots, sa_{x,C}\}$ as output. The algorithm constructs a hash tree $ht$ by invoking the HT-Construction algorithm with the hashes and assigns the root hash to $rh_x$. Then, it iteratively invokes the SA-Extraction algorithm with each of $h_{x,1}, h_{x,2}, \ldots, h_{x,C}$ and appends the result to $SA_x$. Lastly, it returns $rh_x$ and $SA_x$ as output. The algorithm is detailed in Algorithm 1 (given in the Appendix).

2) HC-AUTHDATA-AGGREGATION ALGORITHM

The HC-AuthData-Aggregation algorithm is used by Aggregator $\alpha$ to generate aggregated AuthData for a set of $P$ objects. These objects are consumed by a consumer $y$. It takes the hashes $h_{1,y}, h_{2,y}, \ldots, h_{p,y}$ of the objects as input and returns a concatenated hash $ch_x$ as output. This is done by concatenating all of the hashes and returning the resulting concatenated hash $ch_x$ as output. The algorithm is detailed in Algorithm 2.

C. PROTOCOL MESSAGE STRUCTURE AND FORMAT

The three AuthData-Delivery protocols, respectively, used in each of the job execution phases share a common transaction flow and message structure. For each instance of AuthData delivery, there are two protocol messages, namely an AuthData Delivery (ADD) message and an Acknowledgement (ACK) message. The ADD message is sent from an initiator to a respondent to transmit AuthData. The ACK message is conversely sent from the respondent back to the initiator to confirm the receipt of the ADD message. The exchange of these messages is depicted in Fig. 9.

FIGURE 9. A generic message transaction flow used in the AuthData-Delivery protocols.

The message structure is shown in Fig. 10. It consists of a header and a payload. The header further consists of eight fields, which is described in Table 2. For ease of discussion, the VER and RSV fields will be omitted in the remaining of the message description.
The payloads of different ADD messages have variable lengths. These will be further explained later on. For each ACK message, on the other hand, the payload contains only one item, i.e., the MID of the preceding ADD message, and has a fixed length. In other words, ACK messages of different protocols have the same format. Assuming that $mid_1$ is the MID of a preceding ADD message and $mid_2$ is the MID of the ACK message acknowledging the receipt of $mid_1$; ACK is the type of the ACK message; $t_2$ is the current timestamp; and $id_x$ and $id_y$ are the IDs of the sender (the respondent) and receiver (the initiator), the ACK message can be expressed as: $msg$-ACK : $\{mid_2, ACK, S(MID), t_2, id_x, id_y, mid_1\}$.

D. O2M FUNCTIONAL BLOCK

The O2M functional block consists of the InputSplit AuthData-Generation (ISAuthData-Generation) algorithm, the InputSplit AuthData-Verification (ISAuthData-Verification) algorithm, and the InputSplit AuthData-Delivery (ISAuthData-Delivery) protocol. The ISAuthData-Generation algorithm is used by ClientApp to generate AuthData for $M$ InputSplits. These AuthData are referred to as ISAuthData. The ISAuthData-Verification algorithm is used by each of the $M$ Mappers for the verification of an InputSplit assigned to the Mapper. The ISAuthData-Delivery protocol is used to deliver the AuthData from ClientApp to JobManager and from JobManager to each of the Mappers. A high-level view of the O2M functional block is shown in Fig. 11.

1) ISAUTHDATA-GENERATION ALGORITHM

The ISAuthData-Generation algorithm uses the HT-AuthData-Aggregation algorithm (implementing TreeAgg) and a digital signature scheme to, respectively, generate and sign ISAuthData. It takes InputSplits $d_{c,m_1}, d_{c,m_2}, \ldots, d_{c,m_M}$ (submitted by ClientApp $c$) and the private key $sk_c$ as input and generates a signature $\sigma_{rh_c}$ (of a root hash $rh_c$) and a set of Sibling-AuthData tokens $SA_{i} = \{sa_{c,m_1}, sa_{c,m_2}, \ldots, sa_{c,m_M}\}$ as output. Firstly, it iteratively invokes the hash generation algorithm with each of the InputSplits to generate the hashes of the InputSplits and invokes the HT-AuthData-Aggregation algorithm with the hashes to obtain $rh_c$ and $SA_c$. It then invokes the SIG-Signing algorithm with $sk_c$ and $rh_c$ to generate $\sigma_{rh_c}$. Lastly, it returns $\sigma_{rh_c}$ and $SA_c$ as output. The algorithm is detailed in Algorithm 3.

2) ISAUTHDATA-VERIFICATION ALGORITHM

The ISAuthData-Verification algorithm verifies the authenticity of an InputSplit with ISAuthData, i.e., the signature of the root hash and the corresponding Sibling-AuthData token. It takes an InputSplit $d_{c,m_i}$ (submitted by ClientApp $c$ and assigned to Mapper $m_i$), the signature $\sigma_{rh_c}$ (of the root hash $rh_c$), the Sibling-AuthData token $sa_{c,m_i}$, and the public key $pk_c$ as input and returns the verification result $sv$ as output. Firstly, it invokes the hash generation algorithm with $d_{c,m_i}$ to generate the hash $h'_{c,m_i}$ of the InputSplit and invokes the RA-Recovery algorithm with $h'_{c,m_i}$ and $sa_{c,m_i}$ to obtain $rh'_{c}$. It then invokes the SIG-Verification algorithm with $pk_c$, $rh'_{c}$, and $\sigma_{rh_c}$ and returns the verification result $sv$ as output. The algorithm is detailed in Algorithm 4.

3) ISAUTHDATA-DELIVERY PROTOCOL

The ISAuthData-Delivery protocol uses two ADD messages, ISADD1 and ISADD2, and one ACK message, ISACK. The ISADD1 message is used to deliver the signature $\sigma_{rh_c}$ and a set of Sibling-AuthData tokens $sa_{c,m_1}, \ldots, sa_{c,m_M}$ from ClientApp $c$ to JobManager $jm$. The ISADD1 message and the subsequent ISACK message are respectively expressed as: $msg$-ISADD1: $\{mid_1, ISADD1, S(\sigma_{rh_c}) + S(sa_{c,m_1}, \ldots, sa_{c,m_M}\}, t_1, id_c, id_{jm}, \sigma_{rh_c}, sa_{c,m_1}, \ldots, sa_{c,m_M}$ and $msg$-ISACK1: $\{mid_2, ISACK, S(MID), t_2, id_c, id_{jm}, \mid mid_1\}$.

The ISADD2 message is used to deliver the signature $\sigma_{rh_c}$ and the respective Sibling-AuthData token $sa_{c,m_i}$ from JobManager $jm$ to each Mapper $m_i$. The ISADD2 message and the subsequent ISACK message are respectively expressed as: $msg$-ISADD2: $\{mid_1, ISADD2, S(\sigma_{rh_c}) + S(sa_{c,m_i}), t_1, id_{jm}, id_{m_i}, \sigma_{rh_c}, sa_{c,m_i}\}$ and $msg$-ISACK2: $\{mid_2, ISACK, S(MID), t_2, id_{m_i}, id_{jm}, \mid mid_1\}$.

E. M2M FUNCTIONAL BLOCK

The M2M functional block consists of the Producer-Generated PartitionSegment AuthData-Generation (PGen-PSAuthData-Generation) algorithm, the Aggregator-Generated PartitionSegment AuthData-Generation (AGen-PSAuthData-Generation) algorithm, the PartitionSegment...
AuthData-Verification (PSAuthData-Verification) algorithm, and the PartitionSegment AuthData-Delivery (PSAuthData-Delivery) protocol. The PGen-PSAuthData-Generation algorithm is used by each of the M Mappers to generate AuthData for a set of R PartitionSegments that are produced by the Mapper. These AuthData are referred to as PGen-PSAuthData. The AGen-PSAuthData-Generation algorithm is used by JobManager to generate aggregated AuthData for the PartitionSegments produced by all the Mappers. These AuthData are referred to as AGen-PSAuthData. The PSAuthData-Verification algorithm is used by each of the R Reducers to verify a set of M PartitionSegments that are assigned to the Reducer. The PSAuthData-Delivery protocol is used to deliver PGen-PSAuthData from each Mapper to JobManager and to deliver AGen-PSAuthData from JobManager to each Reducer. A high-level view of the M2M functional block is shown in Fig. 12.

FIGURE 12. The components involved, and the algorithms and the protocol used, in the M2M functional block.

1) PGEN-PSAUTHDATA-GENERATION ALGORITHM
The PGen-PSAuthData-Generation algorithm uses the HT-AuthData-Aggregation algorithm (which implements the intra-producer level AuthData aggregation of HybridAgg) and a MAC scheme to, respectively, generate and sign PGen-PSAuthData. It takes the PartitionSegments $d_m, 1 \leq m \leq M$, (produced by a Mapper $m$) and the pairwise key $k_{m,jm}$ as input and generates a root hash $r_{hm}$, a tag $\tau_{rhm}$ (of the root hash), and a set of Sibling-AuthData tokens $SA_m = \{sa_{m,r_1}, sa_{m,r_2}, \ldots, sa_{m,r_r}\}$ as output. Firstly, it iteratively invokes the hash generation algorithm with each of the PartitionSegments to generate the hashes of the PartitionSegments and invokes the HT-AuthData-Aggregation algorithm with the hashes to obtain $r_{hm}$ and $SA_m$. It then invokes the MAC-Signing algorithm with $k_{m,jm}$ and $r_{hm}$ to generate $\tau_{rhm}$. Lastly, it returns $r_{hm}, \tau_{rhm}$, and $SA_m$ as output. The algorithm is detailed in Algorithm 5.

2) AGEN-PSAUTHDATA-GENERATION ALGORITHM
The AGen-PSAuthData-Generation algorithm (used by JobManager) verifies PGen-PSAuthData generated by different producers and uses the HC-AuthData-Aggregation algorithm (which implements the inter-producer level AuthData aggregation of HybridAgg) and a digital signature scheme to, respectively, generate and sign AGen-PSAuthData. It takes the root hashes $r_{hm}, r_{h2}, \ldots, r_{hmR}$ (generated by different Mappers), the tags $\tau_{rh_{m1}}, \tau_{rh_{m2}}, \ldots, \tau_{rh_{mM}}$ (of the root hashes), the pairwise keys $k_{m_{r,m}}, k_{m_{r,m2}}, \ldots, k_{m_{r,mM}}$, and the private key $sk_{jm}$ as input and generates a concatenated hash $ch_{jm}$ and the signature $\sigma_{ch_{jm}}$ (of the concatenated hash) as output. Firstly, it iteratively invokes the MAC-Verify algorithm with each set of $k_{m_{r,m}}, r_{hm}$, and $\tau_{rhm}$ to verify the authenticity of $r_{hm}$, where $1 \leq i \leq M$. If all of the root hashes are authentic, then it invokes the HC-AuthData-Aggregation algorithm with the root hashes to generate $ch_{jm}$. Subsequently, it invokes the SIG-Signing algorithm with $sk_{jm}$ and $ch_{jm}$ to generate $\sigma_{ch_{jm}}$. It returns $ch_{jm}$ and $\sigma_{ch_{jm}}$ as output. The algorithm is detailed in Algorithm 6.

3) PSAUTHDATA-VERIFICATION ALGORITHM
The verification process consists of two phases. In the first phase, the authenticity of the concatenated hash $ch_{jm}$ is verified against the signature $\sigma_{ch_{jm}}$. If the verification result is positive, then the process continues to the second phase. In the second phase, each PartitionSegment is verified against the respective Sibling-AuthData token and the respective root hash contained in $ch_{jm}$. The PSAuthData-Verification algorithm takes the PartitionSegments $d_{m_1, r_1}, d_{m_2, r_2}, \ldots, d_{m_M, r_M}$ (assigned to a Reducer $r_j$), a set of Sibling-AuthData tokens $\{sa_{mh_1, r_1}, sa_{mh_2, r_2}, \ldots, sa_{mh_M, r_M}\}$, the concatenated hash $ch_{jm}$, the signature $\sigma_{ch_{jm}}$ (of the concatenated hash), and the public key $pk_{jm}$ (of JobManager) as input and returns the verification result as output. It invokes the SIG-Verification algorithm with $pk_{jm}, ch_{jm}$, and $\sigma_{ch_{jm}}$ to verify the authenticity of $ch_{jm}$. If the result is negative, return negative; otherwise, proceed to the next step. It iteratively invokes the hash generation algorithm with each of the PartitionSegments to generate the hashes of the PartitionSegments and invokes the RA-Recovery algorithm with each of the hashes and the respective SiblingAuthData token to generate root hashes $r_{hm}^{'}, r_{hm2}^{'}, \ldots, r_{hmM}^{'}. It compares each $r_{hm}^{'}, r_{hmM}$ (extracted from $ch_{jm}$) and returns the comparison result. The algorithm is detailed in Algorithm 7.

4) PSAUTHDATA-DELIVERY PROTOCOL
Like the ISAuthData-Delivery protocol, the PSAuthData-Delivery protocol also uses two ADD messages, PSADD1 and PSADD2, and one ACK message, PSACK. The PSADD1 message is used to deliver a root hash $r_{hm}$, a tag $\tau_{rhm}$, and a set of Sibling-AuthData tokens $\{sa_{mh_1, r_1}, \ldots, sa_{mh_M, r_M}\}$ from each Mapper $m_j$ to JobManager $jm$. The PSADD1 message and the corresponding PSACK message are respectively expressed as: msg-PSADD1: $mid_1, \text{PSADD1}, S(r_{hm}) + S(\tau_{rhm}) + S\{sa_{mh_1, r_1}, \ldots, sa_{mh_M, r_M}\}; t_j, id_{jm}, id_{jm}, r_{hm}, \tau_{rhm}, sa_{mh_1, r_1}, \ldots, sa_{mh_M, r_M}$ and msg-PSACK1: $mid_2, \text{PSACK}, S(MID), t_j, id_{jm}, id_{jm}, mid_1$.

The PSADD2 message is used to deliver the concatenated hash $ch_{jm}$, the signature $\sigma_{ch_{jm}}$ and a respective set of Sibling-AuthData tokens $\{sa_{mh_1, r_1}, \ldots, sa_{mh_M, r_M}\}$ from JobManager $jm$ to each Reducer $r_j$. The PSADD2 message and the corresponding PSACK message are respectively expressed as: msg-PSADD2: $mid_2, \text{PSADD2}, S(ch_{jm}) + S(\tau_{rh_{jm}}) + S\{sa_{mh_1, r_1}, \ldots, sa_{mh_M, r_M}\}; t_j, id_{jm}, id_{jm}, ch_{jm}, \tau_{rh_{jm}}, sa_{mh_1, r_1}, \ldots, sa_{mh_M, r_M}$.
as: msg-PSADD2: \{mid_1, PSADD2, S(ch_{jm}) + S(σ_{ch_{jm}}) + S\{s_{m_1, r_1}, \ldots, s_{m_y, r_y}\}, t_1, id_{jm}, id_{r_1}, ch_{jm}, σ_{ch_{jm}}, s_{m_1, r_1}, \ldots, s_{m_y, r_y}\} and msg-PSACK2: \{mid_2, PSACK, S(MID), t_2, id_{r_1}, id_{jm}, mid_{1}\}.

F. M2O FUNCTIONAL BLOCK

The M2O functional block consists of the Producer-Generated FinalResult AuthData-Generation (PGen-FRAuthData-Generation) algorithm, the Aggregator-Generated FinalResult AuthData-Generation (AGen-FRAuthData-Generation) algorithm, the FinalResult AuthData-Verification (FRAuthData-Verification) algorithm, and the FinalResult AuthData-Delivery (FRAuthData-Delivery) protocol. The PGen-FRAuthData-Generation algorithm is used by each of the R Reducers to generate AuthData for the FinalResult produced by the Reducer. These AuthData are referred to as PGen-FRAuthData. The AGen-FRAuthData-Generation algorithm is used by JobManager to generate aggregated AuthData for FinalResults produced by all the Reducers. These AuthData are referred to as AGen-FRAuthData. The FRAuthData-Verification algorithm is used by ClientApp for the verification of the entire set of the R FinalResults. The FRAuthData-Delivery protocol is used to deliver PGen-FRAuthData from each Reducer to JobManager and to deliver AGen-FRAuthData from JobManager to ClientApp. A high-level view of the M2O functional block is shown in Fig. 13.

![Figure 13](image)

FIGURE 13. The components involved, and the algorithms and the protocol, used in the M2O functional block.

1) PGEN-FRAUTHDATA-GENERATION ALGORITHM

The PGen-FRAuthData-Generation algorithm uses a hash function and a MAC scheme to, respectively, generate and sign AuthData for a FinalResult produced by a Reducer. It takes a FinalResult \(d_{r_1,c}\) (produced by a Reducer \(r_j\)) and the pairwise key \(k_{r_j,jm}\) as input and generates the hash \(h_{r_j,c}\) (of the FinalResult) and the tag \(τ_{h_{r_j,c}}\) (of the hash) as output. Firstly, it invokes the hash generation algorithm with \(d_{r_1,c}\) to obtain \(h_{r_1,c}\). It then invokes the MAC-Signing function with \(k_{r_j,jm}\) and \(h_{r_j,c}\) to obtain \(τ_{h_{r_j,c}}\). Lastly, it returns \(h_{r_1,c}\) and \(τ_{h_{r_j,c}}\) as output. The algorithm is detailed in Algorithm 8.

2) AGENT-FRAUTHDATA-GENERATION ALGORITHM

The AGen-FRAuthData-Generation algorithm verifies PGen-FRAuthData generated by different Reducers and uses the HC-AuthData-Aggregation algorithm (which implements FlatAgg) and a digital signature scheme to, respectively, generate and sign AGen-FRAuthData. It takes the hashes \(h_{r_1,c}, h_{r_2,c}, \ldots, h_{r_R,c}\) (of the FinalResults produced by all the Reducers), the tags \(τ_{h_{r_1,c}}, τ_{h_{r_2,c}}, \ldots, τ_{h_{r_R,c}}\) (of the hashes), the pairwise keys \(k_{r_1,jm}, k_{r_2,jm}, \ldots, k_{r_R,jm}\), and the private key \(sk_{jm}\) as input and generates a concatenated hash \(ch_{jm}\) and the signature \(σ_{ch_{jm}}\) (of the concatenated hash) as output. Firstly, it iteratively invokes the MAC-Verify algorithm with each set of \(k_{r_j,jm}, h_{r_j,c}\), and \(τ_{h_{r_j,c}}\) to verify the authenticity of \(h_{r_j,c}\), where \(1 ≤ j ≤ R\). If all hashes are authentic, then it invokes the HC-AuthData-Aggregation algorithm with the hashes to generate \(ch_{jm}\). Subsequently, it invokes the SIG-Signing algorithm with \(sk_{jm}\) and \(ch_{jm}\) to generate \(σ_{ch_{jm}}\). It returns \(ch_{jm}\) and \(σ_{ch_{jm}}\) as output. The algorithm is detailed in Algorithm 9.

3) FRAUTHDATA-VERIFICATION ALGORITHM

Similar to PSAuthData-Verification, the verification process here also consists of two phases. In the first phase, the authenticity of the concatenated hash is verified, and, in the second phase, the hashes of the FinalResults are compared against the hashes contained in the concatenated hash. The FRAuthData-Verification algorithm takes the FinalResults \(d_{r_1,c}, d_{r_2,c}, \ldots, d_{r_R,c}\) (consumed by ClientApp c), the concatenated hash \(ch_{jm}\), the signature \(σ_{ch_{jm}}\) (of the concatenated hash), and the public key \(pk_{jm}\) as input and returns the verification result as output. Firstly, it invokes the SIG-Verification function with \(pk_{jm}, ch_{jm}\), and \(σ_{ch_{jm}}\) to verify the authenticity of \(ch_{jm}\). If the result is negative, return negative; otherwise, proceed to the next step. It iteratively invokes the hash generation algorithm with each of \(d_{r_1,c}, d_{r_2,c}, \ldots, d_{r_R,c}\) to generate the hashes \(h'_{r_1,c}, h'_{r_2,c}, \ldots, h'_{r_R,c}\). It compares each \(h'_{r_j,c}\) with the respective \(h_{r_j,c}\) extracted from \(ch_{jm}\), where \(1 ≤ j ≤ R\). It returns the comparison result as output. The algorithm is detailed in Algorithm 10.

4) FRAUTHDATA-DELIVERY PROTOCOL

Similar to the two AuthData delivery protocol explained earlier, the FRAuthData-Delivery protocol uses two ADD messages, FRADD1 and FRADD2, and one ACK message, FRACK. The FRADD1 message is used to deliver a hash \(h_{r_j,c}\) and a tag \(τ_{h_{r_j,c}}\) from each Reducer \(r_j\) to JobManager jm. The FRADD1 message and the corresponding FRACK message are respectively expressed as: msg-FRADD1: \{mid_1, FRADD1, S(h_{r_j,c}) + S(τ_{h_{r_j,c}}), t_1, id_{r_j}, id_{jm}, h_{r_j,c}, τ_{h_{r_j,c}}\} and msg-FRACK1: \{mid_2, FRACK, S(MID), t_2, id_{r_j}, id_{jm}, mid_{1}\}.

The FRADD2 message is used to deliver the concatenated hash \(ch_{jm}\) and the signature \(σ_{ch_{jm}}\) from JobManager jm to ClientApp c. The FRADD2 message and the corresponding FRACK message are respectively expressed as: msg-FRADD2: \{mid_1, FRADD2, S(ch_{jm}) + S(σ_{ch_{jm}}), t_1, id_{jm}, id_{r_j}, ch_{jm}, σ_{ch_{jm}}\} and msg-FRACK2: \{mid_2, FRACK, S(MID), t_2, id_{r_j}, id_{jm}, mid_{1}\}.
G. PUTTING EVERYTHING TOGETHER: THE CPDA FRAMEWORK IN ACTION

The operation flow of CPDA when applied to a job execution is depicted as a sequence diagram in Fig. 14.

The sequence diagram highlights what and when the algorithms and the protocols are used by which components. The detailed textual description is omitted. Up on a successful execution of CPDA, (1) AuthData for all objects are generated and delivered to the respective consumers, (2) the authenticity (SR1 and SR2) of each object can be verified against the related AuthData, and (3) producers cannot falsely deny producing their objects (SR3).
VII. WORKING EXAMPLE

In this section, we present a working example to demonstrate how CPDA can be applied to an MR based CBDC-MPC application and, by taking into account of data flows, how the overhead introduced is minimized. There have been many use cases of MR based applications reported in literature. Examples include collaborative spam detection [71], biological image processing [72], and weather data analysis [73], [74]. In these use cases, the datasets used are typically large and are required to be processed in a timely manner. For example, one of case studies described in [73] is the visualization of temperature distributions in the UK, the data used were the weather data collected on February 7, 2014, and the size of the data was 200 GB (gigabyte).

Due to data accessibility, we have chosen a weather data analysis application as our working example. This application is to find the highest temperature in 10 years observed by weather stations around the world. The application is run in an environment where a user of an organization is in charge of the data analysis task, multiple datasets are used, and all the datasets are geographically distributed and managed by different organizations. With regard to the size of a weather dataset, according to Global Historical Climatology Network Daily (GHCN-Daily) version 3.25, provided by National Centers for Environmental Information (NCEI) [75], [76], the size of one-year data is approximately 1.2 GB, hence the size of ten-year data (the total size of all the datasets used) is approximately 12 GB. It is assumed that the datasets shared with the user are signed by the respective organizations and verified by the user before initiating a data processing job. The size of each InputSplit is set to 128 MB (megabyte), which is the default block size used in HDFS of Hadoop [16]. The number of Mappers used is 12, 000/128 = 94 (the total size of the datasets divided by the size of each InputSplit) and the number of Reducers used is 20 (a number set by the user).

In the job submission phase, the O2M functional block is used. ClientApp, instructed by the user, divides the datasets into 94 InputSplits (each containing multiple entries of station IDs and temperature values). ClientApp signs these InputSplits using the ISAuthData-Generation algorithm, generating ISAuthData containing the signature of Root-AuthData (thus one signature signing operation) and Sibling-AuthData tokens. ClientApp then uses the ISAuthData-Delivery protocol to deliver the ISAuthData to JobManager. JobManager uses the protocol to deliver the respective ISAuthData to each of the Mappers. Each Mapper verifies the assigned InputSplit using the ISAuthData-Verification algorithm with the received ISAuthData (thus performing one signature verification operation) before performing its map task.

In the map phase, the M2O functional block is used. Each Mapper performs its map task (i.e., outputting the station IDs and the temperature values observed by the stations) on the InputSplit and produces up to 20 PartitionSegments, one for a different Reducer. Each PartitionSegment contains station IDs and the observed temperature values for a different Reducer. Each Mapper signs the PartitionSegments using the PGen-PSAuthData-Generation algorithm, generating PGen-PSAuthData (the tag of Root-AuthData and Sibling-AuthData tokens). It then uses the PSAuthData-Delivery protocol to deliver the PGen-PSAuthData to JobManager. JobManager verifies the PGen-PSAuthData generated by all of the Mappers and generates AGen-PSAuthData containing a concatenated hash and the signature of the concatenated hash (thus performing one signature signing operation) using the AGen-PSAuthData-Generation algorithm. It then uses the PSAuthData-Delivery protocol to deliver the AGen-PSAuthData along with the respective PGen-PSAuthData (Sibling-AuthData tokens) to each Reducer. Each Reducer verifies up to 94 PartitionSegments that are assigned to it using the PSAuthData-Verification algorithm with the received AGen-PSAuthData and PGen-PSAuthData (thus performing one signature verification operation) before performing its reduce task.

In the reduce phase, the M2O functional block is used. PartitionSegments generated by different Mappers but containing the same station IDs are assigned to a respective Reducer. Each Reducer performs its reduce task (i.e., selecting the highest temperature value observed by each station) based on the assigned PartitionSegments and produces a FinalResult. It signs the FinalResult using the PGen-FRAuthData-Generation algorithm, generating PGen-FRAuthData (the hash of the FinalResult and the tag of the hash). It then uses the FRAuthData-Delivery protocol to deliver the PGen-FRAuthData to JobManager. JobManager verifies the PGen-FRAuthData generated by all of the Reducers. Upon successful verifications, JobManager generates AGen-FRAuthData containing a concatenated hash and the signature of the concatenated hash (thus performing one signature signing operation) using the AGen-FRAuthData-Generation algorithm. It then uses the FRAuthData-Delivery protocol to deliver the AGen-FRAuthData along with the respective PGen-FRAuthData to ClientApp. When ClientApp retrieves the FinalResults, ClientApp verifies the 20 FinalResults using the FRAuthData-Verification algorithm with the AGen-FRAuthData (thus performing one signature verification operation) before showing the result of the job execution to the user.

It can be seen that CPDA protects all of the data objects generated and processed during the job execution, and, in each job execution phase, only one signature is generated and verified by each data processing component regardless of the numbers of objects to be protected and data processing components involved.

VIII. SECURITY ANALYSIS

The security of CPDA is analyzed using both an informal (property) and a formal (complexity) analysis methods. With the informal analysis method, we analyze CPDA against the security requirements (SR1, SR2, and SR3) set in Section II-C. The complexity analysis shows how much effort is required to successfully mount any of the attacks...
JobManager. As JobManager is a trustworthy component and pairwise keys uniquely shared between each Mapper and are generated by the respective Mappers using their respective keys. The authenticity of the root hashes is ensured by the tags that protected concatenated hash containing authentic root hashes.

In the map phase, each PartitionSegment can be verified against the respective Sibling-AuthData token generated by the respective Mapper and the respective root hash contained in the concatenated hash which, in turn, is generated by JobManager. The authenticity of the concatenated hash is ensured by the signature generated by JobManager.

In the reduce phase, each FinalResult can be verified against the respective hash contained in the concatenated hash generated by JobManager. The authenticity of the concatenated hash, similarly, is ensured by the signature generated by JobManager.

1) DATA ORIGIN AUTHENTICATION
Entities external to a job should not be able to inject a fraudulent object into the job. No entities should be able to falsify the origin of an object. This is because it is computationally difficult to find an object that is different from an authentic one, but produces the same hash value, or to forge a new AuthData token (e.g., a tag or signature) for a fraudulent object. Hence, the CPDA framework satisfies the requirement of data origin authentication (SR1).

2) DATA INTEGRITY PROTECTION
Any modifications made to any of the objects would change the hashes of the objects, thus different from when the objects are generated. When the tampered objects are verified against the respective AuthData, the result will be negative and such attempts will be detected. Therefore, the CPDA framework meets the requirement of data integrity protection (SR2).

3) NON-REPUDIATION OF ORIGIN
In the job submission phase, non-repudiation is achieved by ClientApp signing the root hash of the hash tree. As only ClientApp knows the signature signing key and the signature verification key has been certified by a trusted entity (e.g., a certificate authority), any signature that has been positively verified must be from ClientApp.

In the map phase, JobManager provides a signature-protected concatenated hash containing authentic root hashes. The authenticity of the root hashes is ensured by the tags that are generated by the respective Mappers using their respective pairwise keys uniquely shared between each Mapper and JobManager. As JobManager is a trustworthy component and each pairwise key is only known by JobManager and the corresponding Mapper, it is hard for the Mapper to falsely deny that it has produced the PartitionSegments.

Similarly, in the reduce phase, JobManager provides a signature-protected concatenated hash containing authentic hashes. The authenticity of the hashes is, in turn, protected by using pairwise keys that are known only by JobManager and the respective Reducers. It is hard for each of the Reducers to falsely deny having produced the respective FinalResult. Therefore, the CPDA framework satisfies the requirement of non-repudiation of origin (SR3).

4) SECURITY PROPERTIES: A COMPARISON
There are some differences in the security properties offered by the MAC based and signature based schemes. Here, in Table 3, we provide a summary of the security properties provided by these two schemes along with the use of JobManager, a trusted third party (TTP). The result shows that CPDA satisfies all of the specified security requirements, and it provides the same level of security protection as that provided by digitally signing all the data objects individually.

| TABLE 3. The comparisons of security properties achieved by CPDA, the MAC based scheme, and the signature based scheme. |
| MAC | MAC with TTP | Signature | CPDA |
| (SR1) Data origin authentication | ✓ | ✓ | ✓ | ✓ |
| (SR2) Data integrity protection | ✓ | ✓ | ✓ | ✓ |
| (SR3) Non-repudiation of origin | ✓ | ✓ | ✓ | ✓ |

B. COMPLEXITY ANALYSIS
We analyze the strengths of the security protections provided by CPDA in terms of computational complexities required to successfully mount a data injection attack (T1) and a data tampering attack (T2), respectively. In the following, we first give a list of notations used in the analysis, then the security strengths of three cryptographic schemes (a hash function, a MAC, and a digital signature), before comparing the strength of CPDA with those of the MAC based and signature based schemes.

1) NOTATIONS
Table 4 shows the notations used in this analysis; all of the lengths are expressed in bits.

| TABLE 4. Notations used in the computational complexity analysis. |
| L_k | Hash length |
| L_e | Tag length |
| L_q | Signature length |

2) THE STRENGTH OF CRYPTOGRAPHIC SCHEMES
The strength of a cryptographic scheme is measured as the upper bound of computational complexity needed to...
The adversary has to perform one of preimage, second preimage, and collusion attacks, which requires a minimum computational complexity of $2^L_{\text{sign}}/2$. For (3), forging a new tag and a new signature, respectively, requires computational complexities of $2^\min(L_k, L_{\sigma})$ and $2^l$. Therefore, the complexity of successfully launching a data injection attack is $\min(2^L_{\text{sign}}/2, 2^\min(L_k, L_{\sigma}), 2^l)$.

4) RESISTANCE AGAINST A DATA TAMPERING ATTACK
To tamper with an existing object without being detected, an adversary may modify an existing object in a way that the modified object yields the same hash as an existing object, or generate fraudulent AuthData (e.g., Sibling-AuthData tokens, tags, and signatures) for the modified object. A successful data tampering attack requires compromising a hash, a tag, or a signature. Therefore, the complexity of successfully launching a data tampering attack is also $\min(2^L_{\text{sign}}/2, 2^\min(L_k, L_{\sigma}), 2^l)$.

5) SECURITY STRENGTHS: A COMPARISON
The strengths of the MAC based and signature based schemes are equal to the strengths of the underlying cryptographic schemes, i.e., $2^\min(L_k, L_{\sigma})$ and $2^l$, respectively. The strengths of CPDA and these two schemes are summarized in Table 6.

### TABLE 6. Security strengths of CPDA, the MAC based scheme, and the signature based scheme.

<table>
<thead>
<tr>
<th>Attack</th>
<th>MAC</th>
<th>Signature</th>
<th>CPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Data injection attacks</td>
<td>$2^\min(L_k, L_{\sigma})$</td>
<td>$2^l$</td>
<td>$\min(2^L_{\text{sign}}/2, 2^\min(L_k, L_{\sigma}), 2^l)$</td>
</tr>
<tr>
<td>(2) Data tampering attacks</td>
<td>$2^\min(L_k, L_{\sigma})$</td>
<td>$2^l$</td>
<td>$\min(2^L_{\text{sign}}/2, 2^\min(L_k, L_{\sigma}), 2^l)$</td>
</tr>
</tbody>
</table>

### IX. PERFORMANCE EVALUATION
The overheads introduced by CPDA are theoretically evaluated in two aspects, computational overhead and communication overhead. The results are then compared with the overheads introduced by the MAC based and signature based schemes.

### A. NOTATIONS
Table 7 shows the notations used in performance evaluation.

### TABLE 7. Notations used in performance evaluation.

<table>
<thead>
<tr>
<th>$M, R$</th>
<th>The numbers of Mappers, Reducers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$OS_{\text{h}}$, $OL_{\text{h}}$</td>
<td>Hash operation on a small object, a large object</td>
</tr>
<tr>
<td>$OS_{\text{ma}}$, $OL_{\text{ma}}$</td>
<td>MAC-Signing operation on a small object, a large object</td>
</tr>
<tr>
<td>$OS_{\text{mv}}$, $OL_{\text{mv}}$</td>
<td>MAC-Verification operation on a small object, a large object</td>
</tr>
<tr>
<td>$OS_{\text{sig}}$, $OL_{\text{sig}}$</td>
<td>SIG-Signing operation on a small object, a large object</td>
</tr>
<tr>
<td>$OS_{\text{svp}}$, $OL_{\text{svp}}$</td>
<td>SIG-Verification operation on a small object, a large object</td>
</tr>
<tr>
<td>$l_{\text{h}}$</td>
<td>The header length of an ADD message</td>
</tr>
<tr>
<td>$l_{\text{ack}}$</td>
<td>The total length of an ACK message</td>
</tr>
<tr>
<td>$l_{\text{h}}, l_{\text{c}}, l_{\sigma}$</td>
<td>The lengths of a hash, a tag, a signature</td>
</tr>
</tbody>
</table>
B. COMPUTATIONAL OVERHEADS

The computational overheads are evaluated in terms of the number of cryptographic operations performed by each of the CPDA components. Non-cryptographic operations (such as tree traversal and hash concatenation) are omitted as their costs (in terms of execution times) are negligible in comparison with those of cryptographic operations. The cryptographic operations are classified into five groups: hash generation \((OS_h, OL_h)\), MAC-Signing \((OS_{ms}, OL_{ms})\), MAC-Verification \((OS_{mv}, OL_{mv})\), SIG-Signing \((OS_{sv}, OL_{sv})\), and SIG-Verification \((OS_{sv}, OL_{sv})\). As the cost of an operation is also affected by the size of an object, we count the operations performed on small objects \((OS_h, OS_{ms}, OS_{mv}, OS_{sv}, and OS_{sv})\) and on (potentially) large objects \((OL_h, OL_{ms}, OL_{mv}, OL_{ss}, and OL_{sv})\), separately.

1) CPDA FRAMEWORK

In the job submission phase, two data authentication algorithms are used, ISAuthData-Generation and ISAuthData-Verification. ISAuthData-Generation is executed by ClientApp. It contains two sets of operations: one is for constructing a hash tree for \(M\) InputSplits \((M \ast OL_h + (M - 1) \ast OS_h)\), and the other is for signing the root hash with a digital signature scheme \((OS_{sv})\). Hence, the total number of operations is \(M \ast OL_h + (M - 1) \ast OS_h + OS_{sv}\). ISAuthData-Verification is executed by each Mapper. It contains three sets of operations, respectively, for computing the hash of its InputSplit \((OL_{kh})\), for recovering the root hash from the hash \((\log M) \ast OS_h\), and for verifying the root hash against the signature \((OS_{sv})\). Hence, the total number of operations is \(OL_{kh} + \log M \ast OS_h + OS_{sv}\).

In the map phase, three data authentication algorithms are used, PGen-PSAuthData-Generation, AGen-PSAuthData-Generation, and PSAuthData-Verification. PGen-PSAuthData-Generation is executed by each Mapper. It contains two sets of operations, respectively, for constructing a hash tree for \(R\) PartitionSegments \((R \ast OL_h + (R - 1) \ast OS_h)\), and for signing the root hash with a MAC scheme \((OS_{ms})\). Hence, the total number of operations is \(R \ast OL_h + (R - 1) \ast OS_h + OS_{ms}\). AGen-PSAuthData-Generation is executed by JobManager. It contains two sets of operations, respectively, for verifying the authenticity of \(M\) root hashes against the respective tags \((M \ast OS_{mv})\), and for signing the concatenated hash with a digital signature scheme \((OL_{ss})\). Hence, the total number of operations is \(M \ast OL_h + M \ast OS_{mv}\). PSAuthData-Verification is executed by each Reducer. It contains three sets of operations, respectively, for verifying the authenticity of the concatenated hash against the signature \((OL_{sv})\), for computing the hashes of \(R\) PartitionSegments \((M \ast OL_h)\), and for recovering \(M\) root hashes from the hashes \((M \ast \log R) \ast OS_h\). Hence, the total number of operations is \(M \ast OL_h + OL_{sv} + M \ast \log R \ast OS_h\).

In the reduce phase, three data authentication algorithms are used, PGen-FRAuthData-Generation, AGen-FRAuthData-Generation, and FRAuthData-Verification. PGen-FRAuthData-Generation is executed by each Reducer. It contains two sets of operations, respectively, for computing the hash of its FinalResult \((OL_{kh})\) and for signing the resulting hash with a MAC scheme \((OS_{mv})\). Hence, the total number of operations is \(OL_{kh} + OS_{mv}\). AGen-FRAuthData-Generation is executed by JobManager. It contains two sets of operations, respectively, for verifying \(R\) hashes against the respective tags \((R \ast OS_{mv})\) and for signing the concatenated hash with a digital signature scheme \((OL_{ss})\). Hence, the total number of operations is \(OL_{ss} + R \ast OS_{sv}\). FRAuthData-Verification is executed by ClientApp and contains two sets of operations, respectively, for verifying the authenticity of the concatenated hash against the signature \((OS_{sv})\) for computing the hashes of \(R\) FinalResults \((R \ast OL_{kh})\). Hence, the total number of operations is \(R \ast OL_{kh} + OS_{sv}\).

2) MAC BASED AND SIGNATURE BASED SCHEMES

With the MAC based scheme, each object is signed and verified individually using a MAC scheme. Similarly, with the signature based scheme, each object is individually protected by using a digital signature scheme. Hence, the number of operations performed by an individual component is equal to the number of objects to be protected.

With the MAC based scheme, in the job submission phase, ClientApp signs \(M\) InputSplits (ISAuthData-Generation). Each Mapper verifies one InputSplit (ISAuthData-Verification). Therefore, the numbers of operations performed by ClientApp and each Mapper are \(M \ast OL_{ms}\) and \(OL_{mv}\), respectively.

In the map phase, each Mapper signs \(R\) PartitionSegments (PGen-PSAuthData-Generation). Each Reducer verifies \(M\) PartitionSegments (PSAuthData-Verification). Therefore, the numbers of operations performed by each Mapper and each Reducer are \(R \ast OL_{ms}\) and \(M \ast OL_{mv}\), respectively.

In the reduce phase, each Reducer signs one FinalResult (PGen-FRAuthData-Generation). ClientApp verifies \(R\) FinalResults (FRAuthData-Verification). Therefore, the numbers of operations performed by each Reducer and ClientApp are \(OL_{ss}\) and \(R \ast OL_{mv}\), respectively.

Using the same method with the signature based scheme, the numbers of operations performed by each of the components are the same as those by using the MAC based scheme. The only difference lies in the cost of each signing and verification operation; here the operation is a signature operation, rather than a MAC operation.

3) THE COMPARISONS OF THE COMPUTATIONAL OVERHEADS

The computational overheads when different data authentication solutions are applied are summarized in Table 8. The operations performed on large objects are highlighted in red. The result shows that CPDA reduces the number of expensive signature signing and verifying operations performed by each data processing component to one and these operations are performed on aggregated AuthData (root hashes and concatenated hashes) which are usually smaller than
non-aggregated ones. This is achieved at a cost of additional operations imposed on JobManager. We anticipate that the level of reduction by CPDA should increase as the number of objects increases owing to a more significant level of decrease in expensive operations performed on large objects.

C. COMMUNICATION OVERHEADS

The communication overheads are evaluated in terms of the number and the sizes of messages exchanged between CPDA components. As explained in Section VI-C, each AuthData delivery instance consists of two message transactions, one ADD message and one ACK message. The total size of an ADD message is equal to the sum of the size of the header ($h_d$) and the size of the payload. The size of the payload is dependent on the number and sizes of AuthData tokens ($L_d$, $L_r$, and $L_a$) contained in the payload. The size of an ACK message is $L_{ack}$. For the purpose of comparison, it is assumed that, for the cases where the MAC based and signature based schemes are used, AuthData sent from producers to consumers are also through JobManager, in the same way as the case for CPDA.

1) CPDA FRAMEWORK

In the job submission phase, ClientApp sends one ADD message containing one signature and $M$ instances of Sibling-AuthData (each containing up to $\lceil \log M \rceil$ hashes) to JobManager. Hence, the size of the message is $L_{hd} + L_s + M \ast \lceil \log M \rceil \ast L_h$. JobManager replies with an ACK message with the size of $L_{ack}$ to ClientApp. It then sends one ADD message containing one signature and one instance of Sibling-AuthData to each Mapper. The size of the message is $L_{hd} + L_s + \lceil \log M \rceil \ast L_h$. Each Mapper replies with an ACK message with the size of $L_{ack}$ to JobManager.

In the map phase, each Mapper sends one ADD message containing one tag, one root hash, and $R$ instances of Sibling-AuthData (each containing up to $\lceil \log R \rceil$ hashes) to JobManager. The size of the message is $L_{hd} + L_t + L_h + R \ast \lceil \log R \rceil \ast L_h$. JobManager replies with an ACK message with the size of $L_{ack}$ to each Mapper. After it generates AGen-PSAuthData, it sends one ADD message containing one signature, one concatenated hash (containing $M$ root hashes), and $M$ instances of Sibling-AuthData to each Reducer. The size of the message is $L_{hd} + L_s + M \ast L_h + M \ast \lceil \log R \rceil \ast L_h$. Each Reducer replies with an ACK message with the size of $L_{ack}$ to JobManager.

In the reduce phase, each Reducer sends one ADD message containing one tag and one hash to JobManager. The size of the message is $L_{hd} + L_t + L_h$. JobManager replies with an ACK message with the size of $L_{ack}$ to each Reducer. After it generates AGen-FRAuthData, it sends one ADD message containing one signature and one concatenated hash (containing $R$ hashes) to ClientApp. The size of the message is $L_{hd} + L_s + R \ast L_h$. ClientApp replies with an ACK message with the size of $L_{ack}$ to JobManager.

2) MAC BASED AND SIGNATURE BASED SCHEMES

In the MAC based and signature based schemes, the size of the payload of each ADD message is dependent on the number and the size of authentication tokens to be delivered. With the MAC based scheme, in the job submission phase, ClientApp sends one ADD message containing $M$ tags to JobManager and JobManager replies with an ACK message to ClientApp. The sizes of these messages are respectively $L_{hd} + M \ast L_t$ and $L_{ack}$. JobManager sends one ADD message containing one tag to each Mapper and each Mapper replies with an ACK message to JobManager. The sizes of these messages are respectively $L_{hd} + L_t$ and $L_{ack}$.

In the map phase, each Mapper sends one ADD message containing $R$ tags to JobManager and JobManager replies with an ACK message to each Mapper. The sizes of these messages are respectively $L_{hd} + R \ast L_t$ and $L_{ack}$. JobManager sends one ADD message containing $M$ tags to each Reducer and each Reducer replies with an ACK message to JobManager. The sizes of these messages are respectively $L_{hd} + M \ast L_t$ and $L_{ack}$.

In the reduce phase, each Reducer sends one ADD message containing one tag to JobManager and JobManager replies with an ACK message to each Reducer. The sizes of these messages are respectively $L_{hd} + L_t$ and $L_{ack}$.
JobManager sends one ADD message containing $R$ tags to ClientApp and ClientApp replies with an ACK message to JobManager. The sizes of these messages are respectively $L_{\text{hd}} + R \times L_r$ and $L_{\text{ack}}$.

Similarly, with the digital signature based scheme, the numbers of messages exchanged between components are the same as those by using the MAC based scheme. The only difference is that the sizes of authentication tokens contained in the payloads of the messages used in these two schemes are different (i.e., $L_{\sigma}$ rather than $L_{\tau}$).

### 3) THE COMPARISONS OF THE COMMUNICATION OVERHEADS

The communication overheads when different data authentication solutions are applied are shown in Table 9. The result shows that, the three solutions introduce the same numbers of messages. However, among the three solutions, the sizes of the ADD messages used in CPDA are the largest. We argue that the impact of the increased payload size to the underlying networks is insignificant as an ADD message is much smaller than a JobData object. For example, in the job submission phase when CPDA is applied, assuming that $M = 1000$, $L_h = 256$ bits, and $L_{\sigma} = 3072$ bits, the payload size of an ADD message that JobManager sends to each Mapper is equal to $3072 + \lceil \log 1000 \rceil \times 256 = 5632$ bits = 704 B which is much smaller than the size of a 128-MiB InputSplit. Moreover, with CPDA, it is possible to reduce the communication overhead introduced by using signature caching. One copy of the same signature can be sent and cached on each WorkerNode rather than sending multiple copies to different Workers hosted on the same WorkerNode. The signature caching technique is not applicable to the MAC based and signature based schemes as AuthData tokens for different objects are different.

### X. EXPERIMENTAL EVALUATION

We have conducted experiments to investigate how well CPDA performs when it is applied to MR job executions on a real-system testbed. For benchmarking, we compare the results with those of the MAC based and signature based schemes. In the following, we first explain methodology and evaluation metrics, then describe testbed setup and parameters used, before reporting our experimental results.

#### A. METHODOLOGY AND EVALUATION METRICS

The performance of CPDA is affected by computational (operational costs imposed on components) as well as communication overheads (volume of traffics transmitted via networks for AuthData delivery). To evaluate the overhead costs and the performance, we have implemented three data authentication services (CPDA, the MAC based scheme, and the signature based scheme) and applied them to an MR service deployed on a cluster of machines. The evaluation has been conducted in three experiments, Exp1, Exp2, and Exp3. Exp1 evaluates the costs of the cryptographic algorithms used, i.e., hash generation, MAC-Signing, MAC-Verification, SIG-Signing, and SIG-Verification. Exp2 evaluates the costs of data authentication algorithms, i.e., AuthData-Generation and AuthData-Verification algorithms, imposed on individual CPDA components. Exp3 evaluates the performance of the data authentication services when applied to job executions.

The costs of the cryptographic algorithms, the data authentication algorithms, and performance of the data authentication services are measured in terms of the execution times of the algorithms and jobs, respectively. For each particular set of parameter values, we collect multiple samples of execution times to calculate statistical values (i.e., mean values and standard error of the mean).

---

**Table 9. The comparisons of the communication overheads introduced by different data authentication solutions.**

<table>
<thead>
<tr>
<th>Interacting Components</th>
<th>MAC</th>
<th>Signature</th>
<th>CPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between ClientApp and JobManager</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + M \times L_r$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + M \times L_\sigma$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + M \times \lceil \log M \rceil \times L_h$</td>
</tr>
<tr>
<td></td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
</tr>
<tr>
<td>Between JobManager and each Mapper</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_r$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_\sigma$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_\sigma + \lceil \log M \rceil \times L_h$</td>
</tr>
<tr>
<td></td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
</tr>
<tr>
<td>The Map Phase</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interacting Components</th>
<th>MAC</th>
<th>Signature</th>
<th>CPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between each Mapper and JobManager</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + R \times L_r$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + R \times L_\sigma$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_r + L_h + R \times \lceil \log R \rceil \times L_h$</td>
</tr>
<tr>
<td></td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
</tr>
<tr>
<td>Between JobManager and each Reducer</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + M \times L_r$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + M \times L_\sigma$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_h + M \times \lceil \log R \rceil \times L_h$</td>
</tr>
<tr>
<td></td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
</tr>
<tr>
<td>The Reduce Phase</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interacting Components</th>
<th>MAC</th>
<th>Signature</th>
<th>CPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between each Reducer and JobManager</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_r$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_\sigma$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_r + L_h$</td>
</tr>
<tr>
<td></td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
</tr>
<tr>
<td>Between JobManager and ClientApp</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + R \times L_r$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + R \times L_\sigma$</td>
<td>$1^*\text{ADD}: L_{\text{hd}} + L_r + R \times L_h$</td>
</tr>
<tr>
<td></td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
<td>$1^*\text{ACK}: L_{\text{ack}}$</td>
</tr>
</tbody>
</table>
B. TESTBED SETUP

Our testbed consists of an MR service and the three data authentication services deployed on five networked machines. In the following, we describe the software and hardware of the testbed.

1) SOFTWARE

Fig. 15 depicts the software architecture of our testbed. It consists of a (simplified) MR service and three data authentication services. The interactions between MR components are shown as solid arrowed lines. The invocations of the three data authentication services are shown as dashed arrowed lines.

The MR service is implemented using MapReduce Lite [82] which is developed by Tencent. It provides job submission, task scheduling, and task execution functions. It has two types of components, Scheduler and Worker, which are written in Python. Scheduler performs the functions of both ClientApp and JobManager whereas each Worker performs the function of either a Mapper or a Reducer. To execute a task, Worker calls external data processing functions, i.e., map and reduce functions, and these functions are written in C++ and supplied by users. Workers are executed as application processes and they can be run on a single machine or multiple distributed machines. To enable data authentication for MapReduce Lite, we have made a number of improvements to allow the invocations of the data authentication services and the transmission of AuthData.

The three data authentication services are implemented as a single executable file, called DataAuthTools, which is written in C++. The cryptographic functions are implemented using the Botan cryptographic library [83]. In these implementations, we have selected (1) SHA-256 for the hash scheme; (2) HMAC with SHA-256 and 128-bit keys (referred to as HMAC-128) for the MAC scheme; and (3) RSA with SHA-256, 3072-bit keys, and the PSS padding scheme (referred to as RSA-3072) for the digital signature scheme. These schemes are chosen as they are widely accepted by academia and in industries. Examples where these schemes are used include the Transport Layer Security (TLS) protocol (RFC 5246), the Internet Protocol Security (IPSec) protocol suite (RFC 4368), and the Secure Shell (SSH) protocol (RFC 6594). The key and token sizes are set to achieve a sufficient level of security protection; at the time of this writing, a security level of 128 bits or higher is recommended by NIST [34].

The specifications of the underlying operating system, the C/C++ compiler, the Python interpreter, and the cryptographic library used are given in Table 10.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Linux Manjaro 18.0.0 Illyria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>4.14.81-1-MANJARO x86 64</td>
</tr>
<tr>
<td>C/C++ compiler</td>
<td>gcc 8.2.1</td>
</tr>
<tr>
<td>Python interpreter</td>
<td>python 3.7.1</td>
</tr>
<tr>
<td>Cryptographic library</td>
<td>botan 2.8.0</td>
</tr>
</tbody>
</table>

2) HARDWARE

The testbed consists of five machines, labelled as PC1 through to PC5. PC1 is used to conduct Exp1 and Exp2 whereas all the PCs are used to conduct Exp3. The same set of software is installed on all machines. The hardware specifications of the machines are summarized in Table 11.

<table>
<thead>
<tr>
<th>PC1</th>
<th>CPU: Quad Core Intel Core i7-6700, 64-bit, max 4.0 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAM: 16 GB</td>
</tr>
<tr>
<td></td>
<td>HDD: 1 TB</td>
</tr>
<tr>
<td>PC2</td>
<td>CPU: Quad Core Intel Core i5-3470, 64-bit, max 3.6 GHz</td>
</tr>
<tr>
<td></td>
<td>RAM: 8 GB</td>
</tr>
<tr>
<td></td>
<td>HDD: 500 GB</td>
</tr>
<tr>
<td>PC3</td>
<td>CPU: Quad Core Intel Core i5-3470, 64-bit, max 3.6 GHz</td>
</tr>
<tr>
<td></td>
<td>RAM: 8 GB</td>
</tr>
<tr>
<td></td>
<td>HDD: 500 GB</td>
</tr>
<tr>
<td>PC4</td>
<td>CPU: Quad Core Intel Core i7-2600, 64-bit, max 3.8 GHz</td>
</tr>
<tr>
<td></td>
<td>RAM: 8 GB</td>
</tr>
<tr>
<td></td>
<td>HDD: 500 GB</td>
</tr>
<tr>
<td>PC5</td>
<td>CPU: Dual Core Intel Core i3-12100, 64-bit, max 3.1 GHz</td>
</tr>
<tr>
<td></td>
<td>RAM: 4 GB</td>
</tr>
<tr>
<td></td>
<td>HDD: 250 GB</td>
</tr>
</tbody>
</table>

All the machines (PC1 to PC5) are connected to a LAN via a 100-Mbps switch, as shown in Fig. 16. ClientApp and JobManager are hosted on PC1, whereas Workers (Mappers and Reducers) are hosted on all of the machines and the distribution of the Workers is 25%, 20%, 20%, 20%, and 15%, respectively.

C. PARAMETERS AND CONFIGURATIONS

The sizes of data objects used are expressed in bytes (B) and a unit prefix (for multiples of units) is a binary prefix (indicating multiplication by a power of 2), i.e., 1 KiB (kilobyte) refers to $2^{10}$ B = 1024 B and 1 MiB (mebibyte) refers to $2^{10}$ KiB = 1048576 B.

In Exp1, we measure the execution times of cryptographic algorithms performed on objects of different sizes. Each mean execution time is obtained from 1,000 samples. The sizes of the objects range from 32 B (the size of a hash) to 128 MiB (the size of an InputSplit) with an increment of
FIGURE 16. Network topology and the deployment of MR components on five machines.

twofold. The input data used are randomly generated binary data.4

In Exp2, we measure the execution times of the algorithms used in implementing the three data authentication services with varying object sizes and varying numbers of Mappers and Reducers. Each mean execution time is obtained from 100 samples. The sizes of input objects are, respectively, 1 MiB, 16 MiB, and 128 MiB for ISAuthData-Generation and ISAuthData-Verification; 128 KiB, 1 MiB, and 16 MiB for PGen-PSAuthData-Generation, PSAuthData-Verification, PGen-FRAuthData-Generation, and FRAuthData-Verification; and 32 B for AGen-PSAuthData-Generation and AGen-FRAuthData-Generation. The object sizes are set based on the following considerations: (1) InputSplits are usually large (e.g., 128 MiB); (2) PartitionSegments and FinalResults are usually smaller than the InputSplits; and (3) the sizes of hashes and root hashes are fixed (32 B). The respective numbers of Mappers and Reducers are set to 1, 10, 20, ..., 100. Like Exp1, the input data used in Exp2 are also randomly generated binary data.

In Exp3, we measure the execution times of jobs without data authentication and with each of the three data authentication services, respectively, given varying numbers of Mappers and Reducers. The time is measured from when ClientApp starts performing ISAuthData-Generation to when ClientApp finishes performing FRAuthData-Verification. Each mean execution time is obtained from 25 samples. The numbers of Mappers and Reducers used are set to 5, 50, 100, 200 and 5, 40, 80, 120, 160, 200, respectively. We use the MR job described in Section VII for the experiment. The input data used are GHCN-Daily version 3.25 provided by NCEI.5 The input data are divided into multiple 128-MiB InputSplits. The InputSplits are stored on all of the machines prior to a job submission and the FinalResults are stored on the machine hosting ClientApp (i.e., PC1).

All of the input data for all the experiments are stored in RAM to minimize I/O overhead. In Exp3, due to the large size of weather data, 10-year data (approximately 12 GiB) cannot fit into RAMs of PC2 to PC5. As the content of input data should not affect how jobs are executed, we use a symbolic link approach to create a set of 10-year data from a smaller set of data. With this approach, we divide 2 years (2016 and 2017) of data into 10 of 128-MiB InputSplits. The remaining InputSplits are symbolic links pointing to the 10 InputSplits in a round-robin fashion. In this way, all InputSplits (including symbolic links) can be stored in RAMs of all the machines.

We use standard error of the mean to measure the dispersion of sample means around the population mean. According to our experiments, the results (the execution times) sampled in Exp1 are more dispersed than those in Exp2 and Exp3, respectively. Therefore, to get more accurate results, Exp1 requires more samples than those of Exp2 and Exp3, respectively. The justification for the chosen sample sizes (1,000 for Exp1, 100 for Exp2, and 25 for Exp3) is that, with these sample sizes, the uncertainties of the mean execution times in terms of the relative standard error of the mean are lower than 1.5%. In addition, a slight increase in the sample sizes will considerably increase experimental times; this does not justify a marginal gain of accuracy.

Each point plotted in a graph is a mean execution time obtained under a particular set of parameters.

D. EXPERIMENTAL RESULTS

This section reports the experimental results and discuss our findings.

1) EXP1: COSTS OF CRYPTOGRAPHIC ALGORITHMS

The execution times of SHA-256, HMAC-128, and RSA-3072 on objects with varying sizes are depicted in Fig. 17.

![Graph showing execution times of cryptographic algorithms](image)

FIGURE 17. The comparisons of the execution times of SHA-256, HMAC-128, and RSA-3072 on objects with varying sizes.

4The cryptographic schemes used in the experiments operate on binary data.

5In particular, we use files stored in the “by_year” directory on the website.
From the figure, we can make the following observations. The mean execution times for SHA-256, HMAC-128-Signing and HMAC-128-Verification have similar values and they increase almost linearly as the size of the objects increases. For example, when the object size increases from 32B to 128 MiB (an order of $10^6$ increase), the execution time increases from less than 2 microseconds, to about 440,000 microseconds (an order of $10^9$ increase). This is because the larger size of the objects increases the workload of the algorithms thus the execution times.

With regard to RSA-3072-Signing and RSA-3072-Verification, their execution times are of similar values when the size of each object goes beyond 512 KiB, and the values and trend are similar to those of the hash function and MAC based algorithms mentioned above. In other words, the differences in the costs of these algorithms are insignificant. However, when the object sizes are small, the execution time of RSA-3072-Signing is much higher than that of RSA-3072-Verification, and they do not change much when the object size is smaller than 32 KiB. For example, when the object size is 32B, RSA-3072-Signing takes about 1,700 microseconds to execute whereas for RSA-3072-Verification, the value is 65 microseconds. This means that, for small sized objects, RSA-3072-Signing is 26 times more expensive than RSA-3072-Verification. This is because executions of these two algorithms are mostly influenced by two internal operations, (1) hash generation applied on the object and (2) a signature operation (signing or verification) applied on the resulting hash. The execution time of (1) increases when the object size increases but the execution time of (2) is fixed (as the hash size is fixed). When the object size is small (e.g., 32 B), the execution time of (2) is much longer than that of (1) (due to difficulty of computing asymmetric-key algorithms).

In addition, RSA SIG-Signing is much more computationally expensive than SIG-Verification. However, the execution time of (1) surpasses that of (2) when the object size goes beyond a certain threshold (e.g., 512 KiB) and becomes the dominant cost of the signature based algorithms; the larger the object sizes, the closer the execution times of hash function, MAC based, and signature based algorithms thus the smaller difference in the execution times.

These results leads to the following findings: (1) the overhead introduced by data authentication solutions constructed based on these cryptographic algorithms should increase when the object sizes increase; (2) the hybrid approach to data authentication solutions, which minimizes the use of digital signatures combined with the use of hash functions and MACs assisted with pairwise keys shared with a trusted third party, can bring significant reduction in computational overheads in providing data authentication; and (3) this reduction is more significant when the sizes of the objects to be protected are smaller. For example, when the object size is 32B, the hash function and MAC cost approximately 0.1% and 2.6% of RSA based signature signing and verification costs, respectively.

2) EXP2: COSTS OF DATA AUTHENTICATION ALGORITHMS

The execution times of AuthData-Generation and AuthData-Verification algorithms used in the MAC based, signature (SIG) based, and CPDA are compared under different parameter value settings in terms of object sizes and the numbers of Workers used. The experimental results are shown in Fig. 18.

Fig. 18(a) shows the execution times of ISAuthData-Generation (executed by ClientApp) against the size of objects and number of Mappers. From the figure, we can see that the execution times increase as the object size increases, but the differences among the three execution times are disappearing when the object sizes are large. For example, with 100 Mappers and 1-MiB object, the mean execution times of the MAC based, signature based, and CPDA are respectively 0.35 seconds, 0.76 seconds, and 0.35 seconds; whereas, with objects of 128-MiB size, the mean execution times are approximately 44 seconds. This is because the execution times of ISAuthData-Generation are dependent on the execution times of the underlying cryptographic algorithms. The execution times of these algorithms increase as the size of the object increases and the difference among the execution times of different algorithms disappears when the object size is sufficiently large (as explained in Exp1).

In addition, the execution times also increase as the number of Mappers increases. This is because the increase in Mappers increases the number of InputSplits to be signed and verified, hence, the increase in execution times.

The most important observation from these results is that CPDA markedly outperforms the signature based when the size of the objects is small. For example, as reported above, given 100 Mappers, the mean execution times for the MAC based, the signature based, and CPDA are, respectively, 0.35 seconds, 0.76 seconds, and 0.35 seconds. This shows that CPDA gives a similar performance as the MAC based method. It is 53% more efficient than the signature based method.

Fig. 18(b) shows the execution times of ISAuthData-Verification using the three different methods as against different object sizes. This verification operation is performed by a single Mapper. From the results, it can be seen that the three methods introduce a similar level of costs at any given object size and the costs only increase as the object size increases. For example, with 1-MiB sized objects, the mean execution times are 0.004 seconds, whereas, with 128-MiB sized objects, the mean execution times are 0.4 seconds. The costs of the algorithms used in three different solutions are at the same level because: (1) the costs of the algorithms are dependent on the numbers, and costs, of the underlying cryptographic operations used (shown in Table 8); (2) all the three solutions each perform one cryptographic operation (but different class of cryptographic operation) on the assigned InputSplit (which is large), with an exception of CPDA which introduces additional operations (i.e., root hash recovery and SIG-Verification on the root hash) on small objects; (3) as shown in Fig. 17, when the object size is large, the three
FIGURE 18. The comparisons of the execution times of the AuthData-Generation and AuthData-Verification algorithms used in the three data authentication solutions: MAC based, signature (SIG) based and CPDA. (a) ISAuthData-Generation. (b) ISAuthData-Verification with $M = 1$. (c) ISAuthData-Verification of CPDA. (d) PGen-PSAuthData-Generation with $M = 1$. (e) AGen-PSAuthData-Generation. (f) PSAuthData-Verification with $R = 1$. 
classes of cryptographic operations introduce the same level of costs, i.e., the differences in costs among different classes are very small; and (4) the costs of operations on small objects are negligible in comparison with those of operations on large objects when InputSplits are sufficiently large (e.g., 1 MiB).

Fig. 18(c) shows the execution times of ISAuthData-Verification using CPDA with objects of different sizes and different numbers of Mappers. From the figure, it can be seen that the size of objects has a major effect on the execution times, the larger the size of the objects, the longer the execution time. This result is within our expectation, as the cost of hash generation increases as the object (InputSplit) size increases. However, for any given object size, the increase in execution times caused by the increase in Mappers is negligible. This is consistent with our theoretical analysis result. According to the analysis described in Section IX-B, an increase in the number of Mappers would lead to an increase in the overhead of the method by $\lceil \log M \rceil \times OS_h = \lceil \log 100 \rceil \times 1 \text{ microseconds} = 7 \text{ microseconds}$, which is negligible compared with the values of $OL_h = 440,000 \text{ microseconds}$ and $OS_{sv} = 66 \text{ microseconds}$.

Fig. 18(d) shows the execution times of PGen-PSAuthData-Generation (performed by a Mapper) against different object sizes and different numbers of Reducers. The trend in the results is very similar to the that in Fig. 18(a), with an exception that here, in this figure, there are larger performance gaps between CPDA and the signature based and the gaps expanded to medium (1-MiB) sized objects. This means that CPDA performs better in comparison with the signature based method with regard to this algorithm.

Here, $OS_h$ is a hash operation on 64-byte data (a concatenation of two hashes), $OL_h$ is a hash operation on 128-MiB data, and $OS_{sv}$ is a SIG-Verification operation on 32-byte data (with padding to 3072 bits = 384 B).
reason for this is that, in this algorithm, a MAC scheme is used to sign each root hash and the reduction in costs is larger than the ISAuthData-Generation algorithm.

Fig. 18(e) shows the execution times of AGen-PSAuthData-Generation (performed by JobManager) against different numbers of Mappers. The execution times increase as the number of Mappers increases; they increased from 0.004 seconds to 0.005 seconds (an increase of 25%) when the number of Mappers increased from 1 to 100 (an order of two-magnitude increase). This indicates that, with regard to this algorithm, CPDA is highly scalable as the increase in the execution time is a fraction of the increased number of Mappers. As explained in Section IX-B, the increase in the number of Mappers increases the times needed for verifying the root hashes using MAC-Verify and the size of the concatenated hash to be digitally signed thus the increase in the execution times of the algorithm. However, such increase is small compared to the execution time of SIG-Signing.

Fig. 18(f) shows the execution times of PSAuthData-Verification (performed by a Reducer) against different object sizes and different numbers of Mappers. The trend is similar to that shown in Fig. 18(a) with an exception that the gaps between the results of CPDA and those of the signature based are smaller here. Although CPDA greatly reduce the number of signature verification operations used in this algorithm, the cost incurred in hash generation and MAC-Verify is close to that incurred in RSA SIG-Verification thus small reduction in the execution times.

Fig. 18(g) shows the execution times of PSAuthData-Verification using CPDA (performed by a Reducer) against object sizes with the use of 1 Mapper and varying numbers of Reducers. The results show the similar patterns as those in Fig. 18(c).

Fig. 18(h) shows the execution times of PGen-FRAuthData-Generation (performed by a Reducer) against object sizes. Similar to the results shown in Fig. 18(b), the execution times increase as the object size increases. In addition, as explained in Fig. 18(d), as CPDA uses MAC scheme to sign each root hash, which is much cheaper than the digital signature based method, we get a larger cut in execution times in PGen-FRAuthData-Generation.

Fig. 18(i) shows the execution times of AGen-FRAuthData-Generation (performed by JobManager) against the number of Reducers. The trend is similar to that in Fig. 18(e).

Fig. 18(j) shows the execution times of FRAuthData-Verification (performed by ClientApp) against object sizes with different numbers of Reducers. It exhibits the same trend as that in Fig. 18(f).

The results from Exp2 show that the costs incurred by the CPDA algorithms are very close to those introduced by the MAC based algorithms. CPDA is markedly more efficient than the signature based method, particularly when the data objects to be protected are of smaller sizes and the quantities of the objects are large (i.e., large number of producers and consumers). This is due to the reduction in expensive operations by CPDA and large differences in computational costs among different classes of cryptographic operations when applied to small objects. The cost reduction benefits all the data processing components in the system. The largest reduction occurs in PGen-PSAuthData-Generation; in comparison with the signature based method, this algorithm brings a cost reduction of 90%.

3) EXP3: PERFORMANCE OF DATA AUTHENTICATION SERVICES

The times taken to execute jobs (job execution times) under four different conditions, i.e., without any data authentication (No-Auth) and with each of the three data authentication services (MAC based, signature (SIG) based and CPDA) are investigated against varying numbers of Workers (Mappers and Reducers). The results are depicted in Fig. 19.

Based on the results shown in the figures, we can make the following observations. Firstly, as the numbers of Mappers and Reducers increase, the job execution times in all the four cases increase. For example, when 5 Mappers are used, with 5 Reducers, the job execution times are approximately 9 seconds (No-Auth), 12 seconds (MAC), 13 seconds (SIG), and 14 seconds (CPDA), but with 200 Reducers, the corresponding values are 56 seconds, 60 seconds, 65 seconds, and 64 seconds. The rates of increase for the four cases are respectively, 6.2 (No-Auth), 5 (MAC), 5 (SIG), and 4.5 (CPDA). The reason for the increase in job execution times as the number of Workers (one or both of the Mappers and Reducers) increases is that when the number of these Workers increases, the number of objects to be signed and verified also increases, and this introduces additional overhead costs (e.g., process initialization, memory allocation, and inter-process communication). These overhead costs offset the benefit of task parallelism due to the limited number of tasks that can be executed concurrently.

The second observation is that, among the three data authentication services, the MAC based service adds the smallest amount of delays whereas the signature based service adds the largest. The more the Workers that are used, the smaller the gaps between the CPDA and MAC based service and the larger the gaps between the CPDA and signature based service, which means the bigger the benefit CPDA brings in terms of cutting down execution times. For example, when 5 Mappers and 200 Reducers are used, the differences in job execution times between CPDA and the MAC based service and between CPDA and the signature based service are, respectively, 4.6 and 1 seconds. However, when 200 Mappers and 200 Reducers are used, these values are, respectively, 12 and 24 seconds. This means that, in this setting, CPDA cuts down up to two thirds of the additional overhead cost while still providing the same level of protection as that of the signature based service. This reduction is significant, as, in some application contexts such as security threat analysis or intrusion detection, a minor reduction in a job execution time means sooner production of analysis.
XI. CONCLUSION

This paper has presented our extensive research and investigation regarding how to provide authenticity and non-repudiation protection to Big Data in a large-scale distributed setting effectively and efficiently. We have chosen MR, a prominent distributed computing framework, as the underlying distributed computing framework, and assumed that the data, the data processing services, and the underlying infrastructure are all from different administrative domains and there is minimal trust among the entities involved. Based on the analysis of the characteristics of the data processing system (i.e., MR), we proposed a novel approach, a communication pattern based approach, to authenticity and non-repudiation protection to Big Data in this context. We have implemented the approach by designing a novel data authentication solution, called the CPDA framework. CPDA uses communication pattern based AuthData aggregation along with a hybrid use of multiple cryptographic schemes to provide the strongest level of security protection to data at the finest granularity (equivalent to the protection level offered by digitally signing each data object individually). In addition, the protection applied to each object by CPDA is end-to-end for the entire cycle of a job execution, i.e., from the moment when the input data is being submitted to when the final execution results are ready for collection.

We have evaluated CPDA both theoretically and experimentally. The evaluation results show that the overhead costs introduced by CPDA are closer to the level introduced by a symmetric-key based (MAC) service that is cost efficient but cannot protect data against repudiation attacks or threats.
imposed by authorized insiders. The evaluation results also show that the more distributed components that are used during data process, the more the overhead saving CPDA can offer.

In addition, as we have adopted a modular design method in designing CPDA, the approach, the framework, and findings can also be applied to other distributed computing frameworks or contexts, not just MR based.

For future work, we intend to integrate CPDA with entity authentication facilities and evaluate the effectiveness and the efficiency of the integrated authentication service.

**APPENDIX**
See Algorithms 1–10.

**Algorithm 1** HT-AuthData-Aggregation
1: function HTAA($h_{x,1}, h_{x,2}, \ldots, h_{x,C})$
2: 
3: \[ h_t = HTC(h_{x,1}, h_{x,2}, \ldots, h_{x,C}) \]
4: \[ rh_x = \text{the value contained in the root node of } h_t \]
5: \[ \text{for } i = 1 \text{ to } C \text{ do} \]
6: \[ sa_{i,x} = SAE(h_t, h_{x,i}) \]
7: \[ \text{end for} \]
8: \[ SA_t = \{sa_{x,1}, sa_{x,2}, \ldots, sa_{x,C}\} \]
9: return \[ \{rh_x, SA_t\} \]
10: end function

**Algorithm 2** HC-AuthData-Aggregation
1: function HCAA($h_{1,y}, h_{2,y}, \ldots, h_{p,y})$
2: 
3: \[ ch_y = h_{1,y} || h_{2,y} || \ldots || h_{p,y} \]
4: return \[ ch_y \]
5: end function

**Algorithm 3** ISAuthData-Generation
1: function ISADG($d_{c,m_1}, d_{c,m_2}, \ldots, d_{c,m_M}, sk_c)$
2: \[ \text{for } i = 1 \text{ to } M \text{ do} \]
3: \[ h_{c,m_i} = H(d_{c,m_i}) \]
4: \[ \text{end for} \]
5: \[ \{rh_c, SA_c\} = HTAA(h_{c,m_1}, h_{c,m_2}, \ldots, h_{c,m_M}) \]
6: \[ \sigma_{rh_c} = SS(sk_c, rh_c) \]
7: return \[ \{\sigma_{rh_c}, SA_c\} \]
8: end function

**Algorithm 4** ISAuthData-Verification
1: function ISADV($d_{c,m_i}, \sigma_{rh_c}, sa_{c,m_i}, pk_c)$
2: \[ h'_{c,m_i} = H(d_{c,m_i}) \]
3: \[ rh'_c = RAR(h'_{c,m_i}, sa_{c,m_i}) \]
4: \[ sv = SV(pk_c, rh'_c, \sigma_{rh_c}) \]
5: return \[ sv \]
6: end function

**Algorithm 5** PGen-PSAuthData-Generation
1: algorithm PPSADG($d_{m_i,r_1}, \ldots, d_{m_i,r_R}, k_{m_i,jm})$
2: \[ \text{for } j = 1 \text{ to } R \text{ do} \]
3: \[ h_{m_i,r_j} = H(d_{m_i,r_j}) \]
4: \[ \text{end for} \]
5: \[ \{rh_{m_i}, SA_{jm}\} = HTAA(h_{m_i,r_1}, h_{m_i,r_2}, \ldots, h_{m_i,r_R}) \]
6: \[ \tau_{rh_{m_i}} = MS(k_{m_i,jm}, rh_{m_i}) \]
7: return \[ \{rh_{m_i}, \tau_{rh_{m_i}}, SA_{jm}\} \]
8: end function

**Algorithm 6** AGen-PSAuthData-Generation
1: algorithm APSADG($rh_{m_1}, \ldots, rh_{m_M}, \tau_{rh_{m_1}}, \ldots, \tau_{rh_{m_M}}, k_{m_1,jm}, \ldots, k_{m_M,jm}, sk_{jm})$
2: \[ \text{for } i = 1 \text{ to } M \text{ do} \]
3: \[ mv = MV(k_{m_i,jm}, rh_{m_i}, \tau_{rh_{m_i}}) \]
4: \[ \text{if } mv \text{ is negative then} \]
5: \[ \text{throw Exception(negative, } i) \]
6: \[ \text{end if} \]
7: \[ \text{end for} \]
8: \[ ch_{jm} = HCAA(rh_{m_1}, rh_{m_2}, \ldots, rh_{m_M}) \]
9: \[ \sigma_{ch_{jm}} = SS(sk_{jm}, ch_{jm}) \]
10: return \[ ch_{jm}, \sigma_{ch_{jm}} \]
11: end function

**Algorithm 7** PSAuthData-Verification
1: algorithm PSADV($d_{m_i,r_j}, \ldots, d_{m_M,r_j}, sa_{m_1,r_j}, \ldots, sa_{m_M,r_j}, ch_{jm}, \sigma_{ch_{jm}}, pk_{jm})$
2: \[ sv = SV(pk_{jm}, ch_{jm}, \sigma_{ch_{jm}}) \]
3: \[ \text{if } sv \text{ is negative then} \]
4: \[ \text{return } \{\text{negative, } "ch'_{jm}"\} \]
5: \[ \text{end if} \]
6: Extract \[ \{rh_{m_1}, \ldots, rh_{m_M}\} \text{ from } ch_{jm} \]
7: \[ INDICES = \{\} \]
8: \[ \text{for } i = 1 \text{ to } M \text{ do} \]
9: \[ h'_{m_i} = H(d_{m_i,r_j}) \]
10: \[ rh'_{m_i} = RAR(h'_{m_i}, sa_{m_i,r_j}) \]
11: \[ \text{if } rh'_{m_i} = rh_{m_i} \text{ then} \]
12: \[ \text{add } i \text{ to } INDICES \]
13: \[ \text{end if} \]
14: \[ \text{end for} \]
15: \[ \text{if } INDICES \text{ is empty then} \]
16: return \[ \{\text{positive}\} \]
17: \[ \text{else} \]
18: \[ \text{return } \{\text{negative, } INDICES\} \]
19: \[ \text{end if} \]
20: \[ \text{end function} \]

**Algorithm 8** PGen-FRAuthData-Generation
1: algorithm PFRAADG($d_{r,c}, k_{r,c}$)
2: \[ h_{r,c} = H(d_{r,c}) \]
3: \[ \tau_{h_{r,c}} = MS(k_{r,c,jm}, h_{r,c}) \]
4: return \[ \{h_{r,c}, \tau_{h_{r,c}}\} \]
5: end function
Algorithm 10 AGen-FRAuthData-Generation

1: algorithm AFRADG(h1,c, ..., hr,c, τ1,c, ..., τr,c, k1,j,m, ..., kr,j,m, sk(j,m))
2: for j = 1 to r do
3:   mv = MV(kj,j,m, h1,c, τ1,c)
4:   if mv is negative then
5:     throw Exception(negative, j)
6:     end if
7:     ch jm = HC(h1,c, h2,c, ..., hr,c)
9:     return ch jm, σch jm
11: end function

Algorithm 10 FRAuthData-Verification

1: algorithm FRADV(d1,c, ..., dr,c, ch jm, σch jm, pk jm)
2: sv = SV(pk jm, ch jm, σch jm)
3: if sv is negative then
4:   return negative, "ch jm"
5: end if
6: Extract h1,c, ..., hr,c from ch jm
7: INDICES = {}
8: for j = 1 to r do
9:   hj,j,c = H(dj,j,c)
10: if hj,j,c = "cher c then
11:   Add i to INDICES
12: end if
13: end for
14: if INDICES is empty then
15: return positive
16: else
17: return negative, INDICES
18: end if
19: end function

REFERENCES


SOONTORN SIRAPAISAN received the B.Eng. and M.Sc. degrees in computer science and technology from Tsinghua University, China, in 2011 and 2013, respectively. He is currently pursuing the Ph.D. degree in computer science with The University of Manchester, U.K.

Since 2013, he has been a Research Assistant with the Information Security (SEC) Research Team, National Electronics and Technology Center (NECTEC), Thailand. He is a CompTIA Security+ and CCSK (v4) certified security professional. His research interests include cloud computing, cloud security, network security, and information security.

NING ZHANG received the B.Sc. degree (Hons.) in electronic engineering from Dalian Maritime University, China, and the Ph.D. degree in electronic engineering from the University of Kent, U.K.

Since 2000, she has been with the Department of Computer Science, The University of Manchester, U.K., where she is currently a Senior Lecturer. Her research interests are in security and privacy in networked and distributed systems, such as ubiquitous computing, electronic commerce, wireless sensor networks, and cloud computing with a focus on security protocol designs, risk-based authentication and access control, and trust management.

QIAN HE received the B.Sc. degree from Hunan University, China, in 2001, the M.Sc. degree from the Guilin University of Electronic Technology, China, in 2004, and the Ph.D. degree from the Beijing University of Posts and Telecommunications, China, in 2011.

He is currently a Full Professor with the Guilin University of Electronic Technology. His research interests include network security and distributed computing.

Prof. He is a member of ACM and a Senior Member of CCF.