1-1-2010

Towards Secure Cloud Data Management

Wenchao Zhou
University of Pennsylvania

William R. Marczak
University of California - Berkeley

Tao Tao
University of Pennsylvania

Zhuoyao Zhang
University of Pennsylvania

Micah Sherr
University of Pennsylvania, msherr@cis.upenn.edu

See next page for additional authors

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-10-10.

Towards Secure Cloud Data Management

Abstract

This paper explores the security challenges posed by data-intensive applications deployed in cloud environments that span administrative and network domains. We propose a *data-centric* view of cloud security and discuss data management challenges in the areas of secure distributed data processing, end-to-end query result verification, and cross-user trust policy management. In addition, we describe our current and future efforts to investigate security challenges in cloud data management using the *Declarative Secure Distributed Systems* (DS^2) platform, a declarative infrastructure for specifying, analyzing, and deploying secure information systems.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-10-10.

Author(s)

Wenchao Zhou, William R. Marczak, Tao Tao, Zhuoyao Zhang, Micah Sherr, Boon Thau Loo, and Insup Lee

Towards Secure Cloud Data Management

Wenchao Zhou* William R. Marczak† Tao Tao* Zhuoyao Zhang* Micah Sherr*
Boon Thau Loo* Insup Lee*
*University of Pennsylvania †University of California at Berkeley

ABSTRACT

This paper explores the security challenges posed by data-intensive applications deployed in cloud environments that span administrative and network domains. We propose a *data-centric* view of cloud security and discuss data management challenges in the areas of secure distributed data processing, end-to-end query result verification, and cross-user trust policy management. In addition, we describe our current and future efforts to investigate security challenges in cloud data management using the *Declarative Secure Distributed Systems* (DS^2) platform, a declarative infrastructure for specifying, analyzing, and deploying secure information systems.

1. INTRODUCTION

Cloud computing is a popular computing paradigm in which computation is moved away from a personal computer or an individual application server to a "cloud" of computers in the network. Cloud computing holds the promise of revolutionizing the manner in which enterprises manage, distribute, and share content. By outsourcing web hosting and other IT capabilities to one or more cloud providers, enterprises may achieve significant cost savings.

In this paper, we propose a *data-centric* view of cloud security – one that focuses on the needs and challenges posed by data-intensive applications deployed in cloud environments than span administrative and network domains. Cloud security issues have recently gained traction in the research community, where the focus has primarily been on securing the low-level operating systems or virtual machine implementations (e.g., [22]) on which cloud computing services are deployed.

Our work complements such existing research and further introduces a *data-driven* approach towards cloud security. Our work is motivated by the observation that the majority of applications deployed on cloud infrastructures will involve some form of distributed management, typically in the form of database backends that are partitioned and replicated. A data-centric view is therefore well-suited for analyzing the security properties of such data-driven cloud computing applications. Further, our data-driven approach enables us to adopt from the ground up well-explored techniques from the database community, such as recent work on query results verification [19, 20, 9, 17] of outsourced databases.

Moreover, as the Internet continues to evolve, it is increasingly dominated by applications that require integration, sharing, and interoperable communication among various services and users on the Internet. Such applications include traditional retail portals that integrate content for comparison shopping and social-network systems that incorporate mashups, blogging, social messaging, and more recently, collaborative social analysis. A comprehensive cloud security solution has to provide mechanisms for securely sharing data among users and content providers, even across administrative and enterprise boundaries. A data-centric approach enables us to build upon well-studied database techniques, ranging from database access control [21] to recent work at using query engines to enforce and implement extensible trust management policies [16, 26].

This paper makes the following contributions: (i) We categorize potential threats posed by malicious entities within the cloud infrastructure; (ii) we then discuss data management challenges in the areas of secure distributed data processing, end-to-end query results verification, and trust policy management; (iii) we propose solutions to many of the identified challenges using *Declarative Secure Distributed Systems* (DS^2) [26], a declarative platform for specifying, analyzing, and deploying secure information systems; and finally, (iv) we outline our current and future efforts towards resolving the remaining cloud data management security challenges.

2. THREATS AND CHALLENGES

We categorize the threats and challenges of secure cloud data management in terms of the three components of most cloud computing systems: *infrastructure*, *communication network*, and *users*.

Infrastructure: The cloud infrastructure – i.e., the hosts that provide cloud computing and data-store services – can be compromised by a malicious adversary. An attacker who controls nodes on the cloud can launch a variety of attacks, including generating incorrect query results, conducting denial-of-service (DoS) attacks, and gaining unauthorized access to sensitive information.

In addition to securing the nodes that constitute the cloud, the distributed data processing that makes the cloud useful must also be secured. For instance, in an untrusted setting, an operation that spans across multiple nodes should be authenticated (and possibly encrypted). All forms of access control and authorizations of machines and users should be seamlessly integrated with the data processing layer of the cloud. Ideally, users of the cloud should be able to validate the completeness and integrity of any query processing that the cloud has performed.

Communication Network: Cloud infrastructures can span across multiple network domains (e.g., across geographically distributed data centers), or in the case of cloud federations [4], the cloud itself may consist of multiple providers that cross administrative boundaries. Communication between cloud nodes must be properly secured to protect the confidentiality and integrity of data.

Users: Security mechanisms must ensure that users of the cloud may access only data that they are authorized. In our context, *users* refer to any customer of the cloud provider (for example, an enterprise that outsources its database-intensive operations to the cloud).

We argue that the existing model of isolating users within virtual machines or *slices* is too restrictive. In several cloud applications (e.g., mashups, collaborative analysis, social networks), data sharing is an integral aspect, and any cloud data management middleware should provide a framework for securely sharing data between users. Since users can dynamically enter and leave the cloud, access control policies should be extensible and easily modifiable.

Given the above security challenges, we next describe DS^2 in terms of its declarative language and capabilities. In Sections 4 through 6, we describe how DS^2 makes significant strides towards secure cloud data management.

3. DS^2 PLATFORM

DS^2 is a declarative platform for specifying, implementing, and deploying networked information systems. In prior work [26], we demonstrate the flexibility and compactness of the *Secure Network Datalog* (*SeNDlog*) language via secure specifications of the path-vector routing protocol, Chord distributed hash table (DHT) [24], and the PIER [10] distributed query processor. DS^2 enhances *SeNDlog* and its distributed execution engine by providing additional support for dynamically layering multiple overlays at runtime (e.g. PIER over Chord) [15] and enabling security policies to be enforced and integrated across various layers.

In DS^2, network protocol and security policies are specified in *SeNDlog* [26], a declarative language primarily rooted in DataLog that unifies declarative networking [14, 13] and logic-based access control [2] specifications. *SeNDlog* programs are disseminated and compiled at each node into individual execution plans. When executed, these execution plans both implement the specified network protocol as well as enforce its desired security policies. As a core feature of declarative networking, *SeNDlog* requires orders of magnitude less code than imperative languages [14].

SeNDlog further extends the basic declarative networking language by adding support for authenticated communication. *SeNDlog* integrates two commonly used constructs in distributed trust management languages (e.g., Binder [8]): (1) the notion of *context* to represent a principal in a distributed environment and (2) a distinguished operator `says` that abstracts away the details of authentication [8, 12].

To demonstrate the key language features of *SeNDlog*, we present the specification of a simple example of a distributed program that computes all pairs of reachable nodes:

```
At S:
  r1 reachable(S,D) :- link(S,D).
  r2 reachable(D,Z)@D :- link(S,D), W says reachable(S,Z).
```

The program consists of two rules (`r1` and `r2`) that are executed in the context of node `S`. The program computes the set of reachable nodes (`reachable`) given the set of network links (`links`) between `S` and `D`. Rule `r1` takes `link(S,D)` tuples, and computes single-hop reachability `reachable(S,D)`. In rule `r2`, multi-hop reachability is computed. Informally, rule `r2` means that "if there is a link between `S` and `D`[1], and there exists a node `W` asserting that `Z` is reachable from `S`, then `D` can also reach `Z` by using `S` as the intermediate node." The `says` primitive in rule `r2` specifies that the authenticity of the received `reachable` tuple should be checked, ensuring it indeed comes from `W` as it claims.

By modifying the above *SeNDlog* rules, one can easily develop more complex routing protocols and policies (e.g., secure BGP). Since DS^2's dataflow framework captures information flow as distributed queries, *data provenance* [3] may be utilized to securely "explain" the existence of any network state (analogous to the use of *proof-trees* [11, 6] in security audits).

We describe two language features of *SeNDlog* of relevance to security, with additional details in [26]:

Communication context: Due to the distributed nature of network queries, a principal does not have control over rule execution at other nodes. *SeNDlog* achieves secure distributed query processing by allowing programs to interoperate correctly and securely via the export and import of rules and derived tuples across contexts.

In the above example, the rules are in the context of `S`, where `S` is a variable that is assigned upon rule installation. In a distributed environment, `S` represents the network address of a node: either a physical address (e.g., IP addresses) or a logical address (e.g., overlay identifier). In a multi-user multi-layered network environment where multiple users and overlay networks may reside on the same physical node, `S` can include a username and network identifier.

Import/export predicates: The *SeNDlog* language allows different principals or contexts to communicate by importing and exporting tuples. The communication serves two purposes: (1) to disseminate update and maintenance messages and (2) to distribute the derivation of security decisions.

During the evaluation of *SeNDlog* rules, derived tuples can be communicated among contexts via the use of *import predicates* and *export predicates*. An **import predicate** is of the form "`N says p`" in a rule body, indicating that principal `N` asserts the predicate `p`.

The use of export predicates ensures confidentiality and prevents information leakage by exporting tuples only to specified principals. An **export predicate** is of the form "`N says p@X`" in a rule head, where principal `N` exports the predicate `p` to the context of principal `X`. Here, `X` can be a constant or a variable. If `X` is a variable, the variable `X` is required to occur in the rule body. In rule `r2` of the above example, node `S` will export `reachable` tuples to node `D`. As a shorthand, "`N says`" can be omitted if `N` is where the rule resides.

The `says` operator implements an authentication scheme [12] that allows the receiver of the tuple to verify its authenticity. The implementation of `says` may depend on the system and its context. In an untrusted environment, `says` may require digital signatures. In a trusted environment, `says` may simply append a cleartext principal header to a message (a less computationally expensive operation than using digital signatures). Somewhere in between, cryptographic signatures may be applied only to certain important messages or when communicating with specific principals. DS^2 provides sufficient flexibility to support a variety of cryptographic authentication primitives.

In addition to authentication, DS^2 also provides mechanisms to ensure the *confidentiality* of transmitted tuples. DS^2 may optionally encrypt transmitted messages to ensure that they cannot be read by adversaries or other unauthorized parties.

4. SECURE QUERY PROCESSING

A significant challenge in cloud data management is ensuring that all query processing is carried out securely within a cloud infrastructure. To be secure, query processing must (1) authenticate users and machines involved in query processing, (2) secure the transfer of data across machines in the cloud, and (3) ensure the integrity of all query results. All three requirements can be directly applied to mitigate potential threats (see Section 2) at the infrastructure, network, and user levels.

Along an orthogonal axis, there is also the issue of *where* the security guarantees are enforced. The obvious solution is to enforce security within the cloud infrastructure. This places security burdens entirely on the cloud provider, consequently requiring all clients to trust their providers. Alternatively, end-to-end verification approaches allow users to verify results without requiring the implicit trust of any given cloud provider, but do so at the cost of more complex application design.

In this section, we focus on the former *in-cloud* mechanisms and defer our discussion of the latter approach to Section 6. We concentrate our discussion on an example based on an *authenticated* implementation of MapReduce [7].

4.1 Example: Authenticated MapReduce

MapReduce [7] is a paradigm that has been extensively used in cloud applications for efficiently utilizing parallel resources to solve

[1]Bidirectional links are assumed, where `D` has a link to `S` as well.

certain classes of distributed problems. Generally, a MapReduce job consists of two steps: *map* and *reduce*. At the map step, a master node splits the job into small sub-problems and distributes them to map workers. The results produced by the map workers are then collected at the reduce step, and are combined into a solution to the original problem.

In the MapReduce paradigm, users supply the `Map` and `Reduce` functions; the middleware transparently handles distribution and fault tolerance. In an untrusted environment, all participating nodes in a MapReduce job must be authenticated. Once authenticated, additional access policies can be enforce (see Section 5).

As an example, we will consider the *WordCounting* program in which MapReduce is used to count the occurrences of words in webpages. The following rules (`m1-m2` for map steps and `r1-r2` for reduce steps) demonstrate an authenticated implementation of MapReduce written in *SeNDlog* and executed using DS^2:

```
At MW:
 m1 map(ID,Content) :- file(MW,ID,Content).
 m2 emits(MW,Word,Num,Offset)@RW :- word(Word,Num,Offset),
        reduceWorker(RID,RW), RID==f_SHA1(Word).

At RW:
 r1 reduceTuple(Word,a_LIST<Num>) :-
        MW says emits(MW,Word,Num,Offset).
 r2 reduce(Word,List) :- reduceTuple(Word,List),
        Master says rBegin(RW).
```

In the program shown above, rules `m1` and `m2` are within the context of a map worker `MW`, and rules `r1` and `r2` are in the context of a reduce worker `RW`. As input to the MapReduce operation, the user supplies the `Map` and `Reduce` functions which encode the detailed operations in the form of imperative programs. We provide a brief description of these functions:

Map operation: Rule `m1` takes as input the `file` predicates which contain the identifier and content of each document. The rule generates the (`ID`, `Content`) pairs – i.e., the local `map` tuples – which are then passed to instances of the user-defined `Map` function. Upon receiving an (`ID`, `Content`) pair, each `Map` instance operates as follows. It splits the contents of a document (stored in the `Content` field) into separate words. For each word, a (`WORD`,1) pair is generated (stored in `word` tuples), denoting that the occurrence count of the word should be increased by one. To prevent miscounting the occurrence of a word that appears at multiple places within the same document, the `Offset` of each word in the document is tagged with the corresponding `word` tuple, which is then sent back to `MW` as the result of the `map` function.

Rule `m2` takes as input `word` tuples and distributes them (in the form of `emits` tuples) to reduce workers. `word` tuples are assigned to reduce workers based on SHA-1 hashes of the words. Additionally, to leverage access control in a multi-user setting, a signature is included within each `emits` tuple using the `says` primitive.

Reduce operation: Reduce workers receive and authenticate (e.g., via digital signatures) `emits` tuples from map workers. The tuples are then grouped by the key field `word`, as shown in rule `r1`. The `a_LIST` aggregate operator maintains the occurrences of each word in a list structure.

After the map workers complete their job, a master node sends a `rBegin` tuple to each reduce worker, signaling that they should start the computation of the reduce job. `reduce` tuples are sent to the user-defined `Reduce` instances, each of which takes as arguments a word and the list of its occurrences. Based on these lists, the `Reduce` instances generate and emit the final results – i.e., the total occurrence counts of words.

Node- vs. user-level authentication: The above program enforces authentication at the *node* level: map workers on the same node (`MW`) share a common key to sign tuples. Reduce workers verify tuples as having been produced by the node `MW`.

Alternatively, authentication can occur at the *user* level. User-level authentication is useful when intermediate and/or final computation results are shared among users. For instance, a user `alice` that runs applications on the cloud may only trust input data from a particular user `bob`. Here, the owner of a MapReduce job (e.g. `bob` in the above example) signs each tuple with its own signature. If the number of workers is large, key management is nontrivial. DS^2 supports public key infrastructures (e.g., certificate authorities) to enable workers to more easily generate and verify signatures.

4.2 Evaluation

This section presents an evaluation of DS^2. The goal of our evaluation is two-fold: (1) to experimentally validate DS^2's ability to implement secure cloud applications using the MapReduce paradigm, and (2) to study the additional overhead incurred by adopting authentication in the system.

As a workload, we adapt the *WordCounting* example from the previous section, implemented as a two-stage MapReduce job:

NormalizeStage: In the first stage, the master node distributes a set of webpages to the map workers. Map workers filter out all HTML tags. The cleansed webpages are further split into small chunks and submitted to the reduce workers. Each reducer distributes the webpage chunks to the map workers of the next stage (CountStage) based on SHA-1 digests of the chunks.

CountStage: The MapReduce operation at the second stage performs the WordCounting operation. Map workers take as input the small chunks produced from the previous stage and emit (`WORD`,1) pairs to the reduce workers. The reduce workers aggregate such pairs and generate the result: the occurrence counts of each word in the input webpages.

As the input of the MapReduce job, we randomly selected 6,400 webpages from a crawled dataset belonging to the Stanford WebBase project [1]. The average size of a webpage is 6KB.

We perform the experiments within a local cluster of 16 quad-core machines with Intel Xeon 2.33GHz CPUs and 4GB RAM running Linux 2.6, interconnected by Gigabit Ethernet. We deploy 16 map workers and 16 reduce workers on the *NormalizeStage* and 32 map workers and 128 reduce workers on the *CountStage*. Each physical machine runs a total of 12 MapReduce worker instances. To avoid packet-drops due to congestion, we rate-limit the number of packets sent per second.

To evaluate the overhead incurred by performing authentications, we constructed three versions of the *WordCounting* job: *NoAuth*, *RSA-1024* and *HMAC*. In *NoAuth*, MapReduce workers transmit tuples without including the sender's signature, whereas in *RSA-1024* and *HMAC* the communication between different map and reduce workers are authenticated using 1024-bit RSA signatures and 160-bit SHA-1 HMACs, respectively.

Figure 1 shows the per-node bandwidth usage over time achieved by the three versions of the *WordCounting* job. All three versions incur spikes in their bandwidth utilization in the first 30s. The spikes are mainly attributed to the cross-node communication at the `NormalizeStage`. The MapReduce operation at this stage is computationally cheap but network intensive, as the tuples transmitted in this phase consist of relatively large chunks of documents (as compared to (`WORD`,1) pairs in the `CountStage`).

We observe that *NoAuth* finishes the computation in 350 seconds, whereas *HMAC* and *RSA-1024* incur an additional 17.4% (60s) and 78.3% (270s) overhead in query completion latency, respectively. The increase in query completion time is due primarily to the com-

putation incurred by signature generation and verification.

Given that a dominant portion of the communication is small-sized (WORD,1) pairs, the relative overhead incurred by tagging signatures with transmitted tuples is significant. We observe that the respective aggregated communication overheads of *HMAC* and *RSA-1024* are 18.4% (7.5MB) and 53.3% (21.8MB) higher than *NoAuth*. However, due to the use of network throttles in our evaluation, the per-node bandwidth utilization for the three versions are similar.

4.3 Layered Encryption and Authentication

The above example illustrates authenticated computations at the data processing layer of a cloud infrastructure. Given that cloud infrastructures can span multiple network or administrative domains, communication must also be properly secured. *SeNDlog* provides additional security constructs for encrypting communication.

Authentication and encryption can be enforced either at the data processing layer or at the network layer. Since DS^2 is based on declarative networking, it is a natural platform for implementing secure communication substrates (e.g., implementing secure routing protocols on the control plane and secure forwarding on the data plane). For instance, when the map and reduce workers transmit tuples, these tuples may traverse untrustworthy channels. DS^2 provides a mechanism for performing secure forwarding of tuples, effectively constructing a transient virtual private network between workers. For example, the following *SeNDlog* program performs authenticated packet forwarding given the `forwarding` table at every node in the network:

```
At Router,
 f1 packet(Pid,Dest,Data)@NextHop :-
         forwarding(Dest,NextHop), Router != Dest,
         P says packet(Pid,Dest,Data).
```

The `forwarding` table maintains the `nextHop` router along the shortest path to any given destination (`Dest`). Rule `f1` forwards a packet `Pid` received from the upstream router `P` along the best path to `Dest` via the `nextHop` router. This payload is recursively routed by rule `f1` to the destination, with authentication occurring at every hop. Similar declarative techniques can be used to compactly specify encrypted communication using the *SeNDlog* language.

Most importantly, DS^2's ability to unify different layers of protocols (i.e., data processing and network routing) make it an interesting framework for performing cross-layer protocol analysis. For example, DS^2 may be well-suited for constructing provenance information [18] that spans system and network layers and crosses administrative domains.

5. SECURE DATA SHARING

Data sharing and integration are integral to many classes of applications that operate in a multi-user cloud environment. If the cloud spans administrative domains, a domain may wish to define policies to share data with another domain for the purposes of further distributing or outsourcing of computation. Moreover, applications deployed in the cloud may be isolated within their own virtual machines. Such isolation may be relaxed by enabling principals to securely interact using access control policies.

The enforcement of access control policies can become complicated by the fact that users may frequently enter or leave the cloud, requiring policies to be rapidly updated. Real-world trust policies are complex and may require functionality beyond that offered by traditional tuple-level access control policies. Devising a scheme to respond to the dynamics and complexity of cloud environments represents a significant challenge.

In this section, we propose methods in which DS^2 can be employed to readily support complex and dynamic policies through query rewriting and other techniques. Our approach builds upon traditional database-based and logic-based access control mechanisms for reasoning about trust policies.

5.1 View-based Access Control and Rewrites

Traditional databases typically utilize *view-based* access control in which views are expressed in SQL, with access controls to these views enforced via explicit permissions granted to authorized users. Such views are easily expressible in DS^2. For example, if a user `alice` owns an `employees` predicate that stores the names, departments, and salaries of employees, then she may create a security view for some other user, `Bob`:

```
At alice:
 sv1: employee_sv_bob(Name,Dept) :-
         employee(Name,Dept,Salary), Salary < 5000.
 sv2: predsecview("employee","employee_sv_bob",bob).
```

In this example, `alice` allows `bob` to view only those employees in any department who have a salary less than $5000. The `employee_sv_bob` predicate represents both a horizontal (salary<5000) and vertical (salary column omitted) partition of the `employee` predicate. Note that `employee_sv_bob` (defined by rule `sv1`) is dynamically applied as the predicates in the rule body update, and hence its contents change with the contents of the `employee` predicate. In rule `sv2`, an additional predicate `predsecview` is maintained to associate security views with the protected predicate.

DS^2 can easily express security views that apply to entities other than single users. For example, `alice` may delegate access to the users whom a certificate authority believes are `good` by replacing rule `sv2` with the following rule:

```
sv2': predsecview("employee","employee_sv_good",U) :-
         cert_authority says good(U).
```

In addition to supporting simple horizontal and vertical slicing, DS^2 also permits more complex partitioning of access control. For example, access rights may be based on provenance information (e.g., Alice allows Bob to view only those employee tuples derived using information from Bob). In general, *SeNDlog*'s flexibility enables DS^2 to support a wide variety of access control policies.

Enforcement. We have thus far left unspecified how security views are enforced. Standard Datalog cannot prevent users from submitting a query that references predicates directly, nor can unmodified Datalog dynamically rewrite queries to refer to the security views. To enable these capabilities, we extend Datalog to represent the currently executing queries in tables accessible to the program – a concept we call the *DS^2 meta-model*. Our approach is similar to the recently proposed Evita-Raced meta-compiler catalog [5].

Preventing queries from directly reading predicates (e.g., `employee`) is made possible by adding schema constraints called *meta-constraints* [16] to the meta-model. Meta-constraints restrict the set of allowable queries. In particular, DS^2 uses meta-constraints to express that users can only insert queries that refer to security views:

```
says(U,R), body(R,A), functor(A,P) -> predsecview(_,P,U).
```

The above example introduces the schema constraint format. Note that instead of `:-`, the head and body are separated by a right-arrow (`->`). The logical meaning of this format is that if, for any assignment of the variables, the left hand side (LHS) is true, then the right hand side (RHS) must also be true. If, on the other hand, the LHS is true but the RHS is false, then evaluation of the query terminates with an error.

In the above constraint, `says` associates a user `U` with a query `R`, and `body` and `functor` are meta-model predicates that examine the

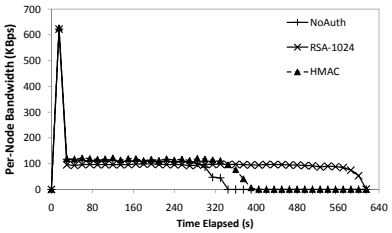

Figure 1: Bandwidth utilization (KBps)

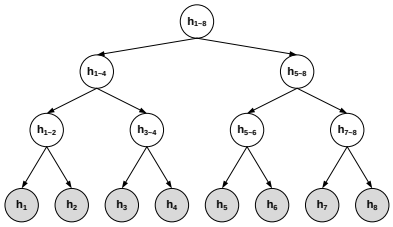

Figure 2: MHT for table X

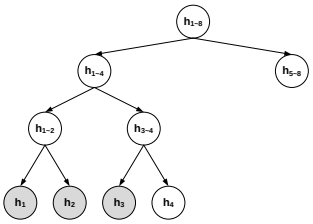

Figure 3: P-MHT for table X_1

structure of the rule. The constraint expresses that every predicate mentioned in the body must be the security view of some other predicate (since the variable for this other predicate is unimportant in the constraint, we represent it with an underscore).

Rewriting queries is equally simple using meta-rules:

```
-rulebody(R,B), +rulebody(R,C), +atom(C), +atompred(C,S),
+atomargs(C,N,V) :-
  P says R, rulebody(R,B), atompred(B,P),
  predsecurityview(P,S), predargs(S,N,V).
```

This rule leverages DS^2's support for *code generation* (also called updating rules). Code generation changes the current state of the database by asserting and retracting specific tuples. In this case, the rule updates the body of the query by *retracting* (-) the body atom representing the original predicate and *asserting* (+) the body atom representing the security view. Note that the logical meaning of a conjunction of atoms in the head of a rule `q1,q2,...,qm :- p1,p2,...,pn` is the set of m rules: `qi :- p1,p2,...,pn`.

5.2 Multi-user Multi-stage MapReduce

In Section 4, we introduced an authenticated version of MapReduce. Given that both access control policies and the MapReduce program are specified within *SeNDlog*, it is easy to integrate access control policies as additional rules in the program. The programmer needs only to supply a set of policy rules; DS^2 automatically performs the query rewrites to generate the appropriate rules for executing the MapReduce program.

In practice, given that MapReduce operations can span multiple stages (i.e., compose several MapReduce operations), access control can be enforced across stages. For instance, if `alice` starts a MapReduce operation and the resulting output is used by `bob` for his subsequent MapReduce operation, one needs to ensure that `bob` only accesses data that he is authorized to view. One naïve solution is to enforce access control only at the boundaries of each MapReduce operation – i.e., on the resulting output between jobs. A more sophisticated solution that we are currently exploring is to "push-down" selection predicates dynamically into each MapReduce execution plan. This ensures that proper filtering is applied early so that only authorized data flows downstream to later stages of the MapReduce operations. In this example, the trust relationships between `alice` and `bob` may result in filtering of input data as early as `alice`'s MapReduce operation, causing only the data that `bob` is authorized to view to be processed. We are currently investigating methods to adapt the dynamic rewrite capabilities presented in [16] to handle more complex sharing, particularly across users and multiple stages of MapReduce computations.

6. END-TO-END VERIFICATION

The above discussion assumes that security is enforced within the cloud infrastructure by the cloud provider. That is, the provider correctly applies the security mechanisms described in Section 4 to support authenticated and confidential queries.

Alternatively, queries may be protected using *end-to-end verification*. These approaches are motivated by clients who outsource their database operations to third parties (e.g., cloud operators).

End-to-end verification works by requiring nodes to transmit *verification objects* (VOs) along with query results. Clients use VOs to verify the correctness of the tuples in the returned resultsets.

Below, we briefly review existing end-to-end verification techniques and discuss some of their limitations. Additionally, we describe our proposed adaptations of these techniques for the cloud environment. For ease of exposition, we consider the case in which a content provider outsources its database operations to a third-party server. The provider (the client) wishes the verify the correctness of the query results.

6.1 Existing Techniques

A simple approach for creating a VO is via per-tuple signatures [19]. Here, the data owner digitally signs each tuple before storing them on the server's site. In response to a query result, the server sends the matching tuples and their corresponding signatures to prove the integrity and correctness (but not necessarily the completeness) of the results.

A more robust approach that ensures completeness of query results relies on *signature-chaining* [20] in which a signature is generated from each tuple and its two adjacent tuples (*neighbors*) in the table. The signature chain guarantees completeness since it is not possible to omit a result tuple while still satisfying the neighbors' signatures. The order of tuples in the table can be determined based on tuple ID or a sort on one of the attributes. Upon computing the query results, the server returns the result tuples, their neighbors, and the corresponding signatures. The approach may be customized for range queries to ensure that a malicious server cannot omit a result tuple within the queried range.

Unfortunately, the signature-chaining scheme is inefficient in terms of storage, communication, and computation costs. An alternative authentication approach for range queries uses a *Merkle Hash Tree* (MHT) [9, 17]. The MHT is constructed as a binary tree in a bottom-up fashion, in which the leaves correspond to the hashes of the tuples in a sorted table. Non-leaf nodes correspond to the hash of the concatenation of the children, and the tree construction process is carried out recursively to the root of the tree. The content provider maintains the MHT along with the *master database*. When it outsources the database to servers in the cloud, it sends both the database and the MHT to the server. In addition, it also authenticates the MHT root with its signature. A VO consists of the signed root and the collection of $\log(n)$ internal tree nodes that enable the verifier to recompute the root of the MHT. In the verification process, given a set of query results and the corresponding VO returned by the server, a client verifies the query results by computing the MHT based on the query results. If the root of its computed MHT matches that of the signed root contained in the VO, then the query results are guaranteed to be valid and complete.

Built upon the techniques presented above for verifying range queries, recent work [25] proposes three techniques – AISM, AIM, and ASM – to perform join operations with the capability to verify the completeness and integrity of the computation results. While

AIM utilizes the MHTs of the both joining tables, AISM and ASM relax the requirement on the availability of MHTs by sacrificing performance.

6.2 End-to-End Verification in the Cloud

Several characteristics of the previously described existing approaches make them unsuitable for the cloud environment: First, the requirement of a master database at the content provider limits the possibility of queries that integrate content from multiple clients on the cloud. Second, the MHT-based approach is not amenable to a database in which the contents are dynamic, since the MHT needs to be recomputed whenever the base data changes. Third, in the cloud environment, data can potentially be distributed across several nodes, complicating the process of MHT construction.[2]

We are currently exploring techniques that address all of the above limitations by developing new capabilities using the DS^2 platform. One of our ongoing research approaches is to add completeness and integrity verification capabilities to *SeNDlog*'s query results. Additionally, we are exploring the application of data provenance for checking the completeness and integrity of data processing. Along another line, researchers have been exploring challenge-based approaches (see, for example, [23]) to verify the compliance of program executions to their expected behaviors. In general, we plan to support distributed verification with the ability to handle dynamism in both data and nodes within the DS^2 system.

Due to space constraints, we focus below on the challenge of constructing MHT for distributed verification. Consider the case in which a table is horizontally partitioned across a set of machines, where each machine maintains a range of tuples in a sorted table. We propose an approach called *partitioned MHT* (P-MHT) that may be applied to data partitions, thus allowing efficient data maintenance and verification across cloud providers.

To demonstrate how P-MHT can be applied, we consider the table $X = \{x_1, x_2, x_3, ..., x_8\}$. The MHT of the table is shown in Figure 2, where $h(x_i)$ is the digest of the tuple x_i. Suppose X is partitioned into three sub-tables, i.e., $X_1 = \{x_1, x_2, x_3\}$, $X_2 = \{x_4, x_5, x_6\}$, and $X_3 = \{x_7, x_8\}$. In addition to storing the tuples, each sub-table maintains a P-MHT that maintains sufficient information for the local generation of the VO for any tuple in the sub-table. For instance, Figure 3 shows the P-MHT for X_1. Although the P-MHT is a partial MHT, it may still generate the VO for any tuple maintained in X_1. For example, the VO for x_3 is $h(x_{1-2}), h(x_4), h(x_{5-8})$, which, when combined with x_3, can be used to compute the MHT root and thus validate the integrity of the query result by comparing with the root signed by the table owner.

Our approach works for tables that are ordered and range partitioned across nodes. Verifying query results across hash-based partitions is still an open area of future work.

7. CONCLUSION

This paper outlines the challenges of secure cloud data management. We propose security mechanisms based on the DS^2 system that address these challenges. Our initial results indicate that DS^2 provides query authenticity with little overhead. We plan to deploy our prototype on PlanetLab to validate its performance in more realistic settings.

The *SeNDlog* language and interpreter are available for download at http://netdb.cis.upenn.edu/rapidnet/downloads.html.

Although we have presented our examples as compact *SeNDlog* programs, other policy specification languages may also be applicable. As an alternative that achieves backward compatibility and legacy support, we are exploring utilizing DS^2 as a separate extensible trust management policy engine that may be incorporated into existing distributed computing software (e.g. Hadoop).

[2]The authenticated join processing techniques presented in the previous section are also also inapplicable when tables are partitioned across nodes: AISM and ASM require on-the-fly sort on at least one joining table. Such a requirement lacks an efficient distributed solution and seriously jeopardizes the performance and even feasibility of these two approaches. On the other hand, the AIM algorithm is still applicable even when tables are distributed across nodes; however, it cannot handle more complex multi-way or multi-level joins. Additionally, one may have to build multiple MHTs for a table (each for a different join key) when using AIM on joins that are conducted with different join keys.

8. REFERENCES

[1] Stanford WebBase. http://diglib.stanford.edu:8091/ testbed/doc2/WebBase/.

[2] ABADI, M. Logic in Access Control. In *Symposium on Logic in Computer Science* (June 2003).

[3] BUNEMAN, P., KHANNA, S., AND TAN, W. C. Why and where: A characterization of data provenance. In *ICDT* (2001).

[4] CAMPBELL, R., GUPTA, I., HEATH, M., KO, S. Y., KOZUCH, M., KUNZE, M., KWAN, T., LAI, K., LEE, H. Y., LYONS, M., MILOJICIC, D., O'HALLARON, D., AND SOH, Y. C. Open cirrus cloud computing testbed: Federated data centers for open source systems and services research. In *HotCloud* (2009).

[5] CONDIE, T., CHU, D., HELLERSTEIN, J. M., AND MANIATIS, P. Evita raced: Metacompilation for declarative networks. In *VLDB* (2008).

[6] CROSBY, S. A., AND WALLACH, D. S. Efficient Data Structures for Tamper-Evident Logging. In *18th USENIX Security Symposium* (August 2009).

[7] DEAN, J., AND GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In *Proceedings of Usenix Symposium on Operating Systems Design and Implementation* (December 2004).

[8] DETREVILLE, J. Binder: A logic-based security language. In *IEEE Symposium on Security and Privacy* (2002).

[9] FEIFEI LI, M. H., AND KOLLIOS, G. Dynamic authenticated index structures for outsourced databases. In *SIGMOD* (2006).

[10] HUEBSCH, R., HELLERSTEIN, J. M., LANHAM, N., LOO, B. T., SHENKER, S., AND STOICA, I. Querying the Internet with PIER. In *VLDB* (2003).

[11] JIM, T. SD3: A Trust Management System With Certified Evaluation. In *IEEE Symposium on Security and Privacy* (May 2001).

[12] LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. Authentication in Distributed Systems: Theory and Practice. *ACM TOCS* (1992).

[13] LOO, B. T., CONDIE, T., HELLERSTEIN, J. M., MANIATIS, P., ROSCOE, T., AND STOICA, I. Implementing Declarative Overlays. In *ACM SOSP* (2005).

[14] LOO, B. T., HELLERSTEIN, J. M., STOICA, I., AND RAMAKRISHNAN, R. Declarative Routing: Extensible Routing with Declarative Queries. In *SIGCOMM* (2005).

[15] MAO, Y., LOO, B. T., IVES, Z., AND SMITH, J. M. MOSAIC: Unified Declarative Platform for Dynamic Overlay Composition. In *ACM CoNEXT 2008*.

[16] MARCZAK, W. R., ZOOK, D., ZHOU, W., AREF, M., AND LOO, B. T. Declarative reconfigurable trust management. In *4th Biennial Conference on Innovative Data Systems Research (CIDR)* (2009).

[17] MOURATIDIS, K., SACHARIDIS, D., AND PANG, H. Partially materialized digest scheme: an efficient verification method for outsourced databases. *VLDB Journal)* (2009).

[18] MUNISWAMY-REDDY, K., BRAUN, U., HOLLAND, D. A., MACKO, P., MACLEAN, D., MARGO, D., SELTZER, M., AND SMOGOR, R. Layering in provenance systems. In *USENIX Annual Technical Conference* (2009).

[19] MYKLETUN, E., NARASIMHA, M., AND TSUDIK, G. Authentication and integrity in outsourced databases. In *Symposium on Network and Distributed Systems Security (NDSS)* (2004).

[20] PANG, H., AND TAN, K.-L. Verifying Completeness of Relational Query Answers from Online Servers. *ACM Transactions on Information and Systems Security)* (2008).

[21] RIZVI, S., MENDELZON, A. O., SUDARSHAN, S., AND ROY, P. Extending query rewriting techniques for fine-grained access control. In *SIGMOD* (2004).

[22] SANTOS, N., GUMMADI, K. P., AND RODRIGUES, R. Towards Trusted Cloud Computing. In *HotCloud* (2000).

[23] SION, R. Query execution assurance for outsourced databases. In *VLDB* (2005).

[24] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *SIGCOMM* (2001).

[25] YANG, Y., PAPADIAS, D., PAPADOPOULOS, S., AND KALNIS, P. Authenticated join processing in outsourced databases. In *SIGMOD* (2009).

[26] ZHOU, W., MAO, Y., LOO, B. T., AND ABADI, M. Unified Declarative Platform for Secure Networked Information Systems. In *ICDE* (2009). *See also http://netdb.cis.upenn.edu/ds2/*.