Mendel: Efficiently Verifying the Lineage of Data Modified in Multiple Trust Domains*

Ashish Gehani
SRI International
Menlo Park, CA
ashish.gehani@sri.com

Minyoung Kim
SRI International
Menlo Park, CA
minyoung.kim@sri.com

ABSTRACT

Data is routinely created, disseminated, and processed in distributed systems that span multiple administrative domains. To maintain accountability while the data is transformed by multiple parties, a consumer must be able to check the lineage of the data and deem it trustworthy. If integrity is not ensured, the consequences can be significant, particularly when the data cannot easily be reproduced. Verifying the provenance of a piece of data generated using inputs from multiple administrative domains is likely to require the use of numerous public keys that originate at external institutions. Current methods for verifying the integrity of such data from other users will not scale for provenance metadata since scores of verifications may be needed to validate a single file's lineage graph. We describe Mendel, a protocol with a three-pronged strategy that combines *eager signature verification*, *lazy trust establishment*, and *cryptographic ordering witnesses* to yield fast lineage verification in distributed multi-domain environments. Further, we show how decisional lineage queries, that is whether one file is the ancestor of the other, can be answered with high probability in constant time.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems; D.4.6 [**Operating Systems**]: Security and Protection—*Verification*; H.3.3 [**Information Systems**]: Information Search and Retrieval

Keywords

provenance, lineage, metadata, certification, verification

*This material is based upon work supported by the National Science Foundation under Grant OCI-0722068. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

HPDC'10 June 20-25, 2010, Chicago, IL, USA.

1. INTRODUCTION

Data is routinely created, disseminated, and processed in distributed systems that span multiple administrative domains. In many instances, the utility of a piece of data depends on the assurances that can be made about it. For example, a legal prosecutor can use forensic evidence only if its chain of custody can be established. Without an estimate of the authenticity and integrity of the data, it cannot be entered as evidence in judicial proceedings. To maintain accountability while the data is transformed by multiple parties, a consumer must be able to check the lineage of the data and deem it trustworthy. If integrity is not ensured, the consequences can be significant, particularly in situations when the data cannot easily be reproduced.

The granularity at which we track the provenance of a data object affects the overhead that will be introduced in the system. The advantage of finer-grain auditing, at the level of assembly instructions or system calls, for example, is that information flow can be traced more precisely, allowing an output's exact antecedents to be ascertained by reconstructing the exercised portion of the control flow graph of the relevant process. The disadvantage is that the system's performance will perceptibly degrade and the monitoring will generate large volumes of provenance metadata. Since persistent data is managed at file granularity, a reasonable compromise on the level of abstraction at which to track data lineage is to define it in terms of files read and written.

Numerous systems have been developed to track the provenance of data. Many of them have been targeted at specific application domains, such as tracking Geographic Information Systems (GIS) data sets [11], establishing the pedigree of safety-critical components [42], and facilitating the repetition of biological experiments [26], while a few, such as Taverna [1], which augments $^{my}Grid$ [31], are more general. The projects either do not provide functionality for certifying the integrity of the provenance metadata or utilize schemes that are designed for single administrative domains [43].

Extant systems [43] certify lineage as follows. Each time an operation occurs, the inputs and output are signed and the resulting digital signature is embedded in the metadata. Since provenance operations can occur frequently, embedding a verification key with each operation would substantially increase the storage overhead. Consequently, verification keys are maintained at a remote public key server from where they are retrieved dynamically during the process of verifying the provenance metadata. Since the network connections to the public key server introduce substantial latency into the verification protocol, such schemes can sup-

port fast verification times only if the lineage is signed with only a few users' keys or if a large subset of the users' keys is locally accessible (through a local replica of the public key server, for example). As the provenance metadata begins to contain signatures from more users from external administrative domains, the likelihood of the corresponding verification keys being available locally (from a replicated public key server or cache) drops.

A number of application domains would substantially benefit if they could obtain low-latency verified responses to lineage queries. Modern scientific applications that utilize large computing infrastructures are frequently assembled from or leverage many software libraries and tools that originate from a range of sources. If the output of such programs is safety critical, then lineage metadata can be used to verify compatibility with the version, runtime environment, and order of invocation of the software that generated each of the inputs. We describe protocol optimizations that allow lineage verification operations to be utilized in environments where low-latency responses are needed, subject to conditions that we detail in Section 4.

We describe our threat model in Section 2 and some challenges in Section 3, and explain our approach to address the problem in Section 4. Section 5 reports the results of our prototype implementation. We describe related work in Section 6 before concluding in Section 7.

1.1 Contributions

The goal of this work is to accelerate the verification of lineage metadata that arises in distributed systems that span multiple administrative domains. The primary concern is that the lineage graph of a single file may consist of many pieces, each signed by a different user, requiring numerous signature verifications. The strategies we employ fall into two categories - those that speedup each signature check:

- An *identity-based signature* [47] is a special type of public key signature scheme where the public key can be computed from a user's identity. We use it to allow verification of lineage elements without requiring high-latency network connections to public key servers.

- *Eager signature verification* is our strategy of optimistically checking signatures without regard to the order in which they occur in the lineage graph. This reduces the latency of verification at the cost of potentially performing extra computation. (The conservative alternative would avoid verifications of descendants when ancestors cannot be validated.)

and those that reduce the number of signatures that need to be checked:

- *Lazy trust establishment* is the idea that pieces of the lineage graph can be verified lazily as they are needed for responding to a query, rather than verifying the entire graph first and then determining the portion needed to construct an answer.

- We introduce the notion of *decisional lineage* which allows two files to be compared to determine if one is the ancestor or descendant of the other without examining their lineage graphs.

- We describe the idea of *ordering witnesses* that can be used to establish a nonrepudiable temporal sequence. Such elements can be used to ascertain the direction of a path in a lineage graph without access to the rest of the graph.

1.2 Background

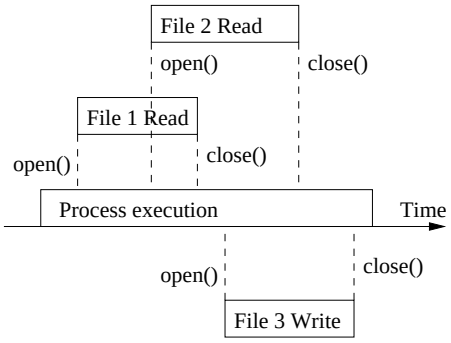

Figure 1: When a file is closed after being written, its provenance includes every file that was read by the process. The provenance of *File 3* is the list {*File 1, File 2*}.

We briefly explain what constitutes the lineage of a piece of data. The semantics of a *primitive operation* are defined to be an output file, the process that generated it, and the set of input files it read in the course of its execution. For example, if a program reads a number of data sets from disk, computes a result and records it in a file, a primitive operation has been performed, as illustrated in Figure 1. If a process modifies a number of files, a separate instance of the representation is used for each output file. Primitive operations are combined into a *compound operation*, as depicted in Figure 2. For instance, if the result of appending together several data sets (by a program such as UNIX *cat*) is then sorted into a particular order (using another program, such as UNIX *sort*, that executes as a separate process), then the combination of appending and sorting is a compound operation. Each vertex shown with a circle represents the execution of a different process, while every vertex shown with a rectangle represents a file. We adopt the convention of identifying a file using both its logical location and its last time of modification to disambiguate different versions of the same file, which avoids cycles in the lineage graph. Thus, every data object is the result of a compound operation that can be represented by a lineage *directed acyclic graph (DAG)*. A description of the types of queries made about the resulting lineage graphs can be seen at the Provenance Challenge Wiki [10].

1.3 Motivation

Some applications, such as the use of provenance metadata to establish a chain of custody, may occur infrequently enough that efficient verification protocols may not seem necessary. However, even in these instances if the data's origins are old enough, that is, it is the result of many composed operations, the speedup from offline verification can be significant. Further, when a large collection of data, such as all the files from a seized computer system, needs to be verified, efficient protocols are critical.

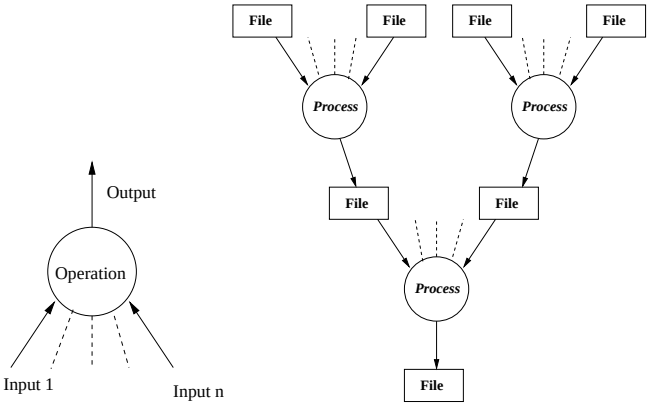

Figure 2: A primitive operation, depicted on the left, takes a set of inputs and produces a single output. An object's lineage, depicted on the right, is a collection of primitive operations assembled into a directed acyclic graph.

When data is the product of employing substantial computational resources for extended periods of time, verifying its antecedents by running the operations again using the same inputs is not an economical strategy. For example, Fermilab's Collider Detector's raw output is too voluminous to be used directly by physicists. Instead, dozens of terabytes of raw data are sent to external Grid nodes at multiple institutions' sites where they may be processed for extended periods. The transformed version is then analyzed by other programs in terabyte chunks over the course of days. Consequently, each piece of information that physicists finally use is the output of a month of processing [50]. Similarly, the cost of analyzing a single protein stored in the Protein Data Bank is $200,000 [44]. The cost of producing such data precludes its availability from an alternate source to cross-check it. If the authenticity of the data is not tracked from the time of creation, through all the operations performed on it, fraudulent modifications may go undetected.

The issue of data being the result of successive computation in numerous administrative domains routinely occurs in diverse settings. One example is the Genome Analysis and Database Update (GADU) system, which is designed to automate the assignment of functions to genes [41]. After a genome analysis workflow completes, entries in the final database can be traced back through a circuitous path. Periodically, queries are made to the National Center for Biotechnology Information (NCBI) [33], Joint Genome Institute (JGI) [27], The Institute for Genomic Research (TIGR) [53], Protein Data Bank (PDB) [36], and Swiss-Prot [51] databases. If any new data is found, it is downloaded to the GADU server. The Pegasus planner [12] dispatches sequence data to hundreds of remote nodes. At each node, reference data is drawn from BLAST [2], PFAM [4], BLOCKS [24], and THMM [29] databases for different types of comparative analyses. The resulting output for each then goes into a database. In this setting, verifying the provenance of the data will require checks of signatures that originate from each of these domains.

Efficient verification protocols are of particular importance when the provenance is utilized as part of an online computation – that is, the output of a provenance query is relied upon to determine successive steps of a workflow. A motivating example for this is the NIGHTINGALE project [19], which aims to let monolingual users query information from newscasts and documents in multiple languages. Input data is transformed multiple times for automatic speech recognition, machine translation between languages, and distillation to extract responses to a query. The pipeline of operations has several steps, each of which can be performed by multiple versions of software being developed in parallel by experts from 15 universities and corporations. Since the functionality of different revisions of the same tool can also differ, the description of the tool that produced a piece of data serves as an input for subsequent tools in the pipeline. This metadata is currently maintained in a file that accompanies the data. If low-latency access to the provenance of data were available, maintenance of the accompanying file would be obviated. The low latency is of significance because the metadata would enable querying to determine which combinations of tools in the pipeline have yielded a better-quality output.

2. THREAT MODEL

The Mendel protocol for verifying multi-domain data lineage can be utilized when we wish to ensure that filesystem lineage metadata is nonrepudiable after it has been certified. In this situation, owners of single-user hosts and administrators of multi-user systems are assumed to be honest at the time that lineage is being generated. The purpose of certifying and verifying the lineage metadata in this case is to prevent a user from making post-fact claims that conflict with that user's original commitments. On the surface this may seem like a weak threat model – that is, one that is easily circumvented and hence of limited utility. However, in practice the ability to discriminate about such claims has significant use, as illustrated by several examples below.

The motivation to alter the lineage of a piece of data may arise after the metadata has been committed. In the first example, consider the case when a company's information infrastructure routinely collects and certifies lineage metadata as part of its auditing process. Subsequently, a failure of one of its products occurs in the field, and tracing the lineage of a piece of data would allow an investigator to find information that exposes the company to severe legal liability. In such a case, there would be a temptation to post-fact alter the audit trail. If the lineage records being generated formed part of the metadata that flows to a remote administrative domain along with output files, post-fact conflicting claims would be detectable using Mendel. As a second example, consider the case when a company is being sued for patent infringement. The company would like to demonstrate prior art in its possession that could negate the patent's validity. In such a circumstance, if the lineage records can be altered to make a set of files appear to be older than they are, a fraudulent defense can be effected. A third example is the case when an individual may make an error when running a computation such as entering faulty input in a safety analysis. The resulting output may be used in a situation where an accident occurs. The individual would then have an interest in altering the lineage records to avoid culpability for the accident. A fourth example would be the case of an individual that made a discovery and wished to claim credit for it. Subsequently, that individual learns of a competitor's earlier results and is tempted to alter the lineage record to

fraudulently demonstrate that the new discovery predates a particular external event.

In general, fraudulent claims can be decomposed into three categories. The first is *input insertion* – that is, claiming that a piece of data was produced using another piece of data when it did not in fact use it. The second is *input deletion* – that is, claiming that a piece of data was produced without using another piece of data when in fact it was used. The third type is *output alteration* – that is, denying making a modification to a specific output after using a particular set of inputs. Mendel's verification prevents each of these attacks after the fact if true claims were committed at the time of the activity.

3. CHALLENGES

As network connectivity becomes ubiquitous, user data in many computing systems is increasingly likely to have originated from a diverse range of sources, some of which are at remote hosts. In particular, in collaborative settings such as universities or research institutes, data is likely to have been processed using external infrastructure such as computational Grids (that allow multiple institutions to pool their resources). If each host certifies the changes it makes, the final output's metadata will include signatures generated using keys from multiple external administrative domains.

Each trust domain will use a different set of identification credentials. They may standardize a single signature algorithm, mode of operation, and key size. However, each domain will still use its own signing key (since otherwise an external entity would be able to alter the membership of an organization). Thus, most of the keys needed to verify the signatures in the provenance metadata will not be available in the public key infrastructure of the user's organization. This issue is particularly problematic for provenance metadata since the integrity of data itself could be assured by verifying a signed cryptographic hash of the data that is produced by its last owner, requiring access to only a single verification key. In practice, data objects are often not even provided this level of end-to-end integrity assurance. Instead data is only protected in transit by securing the communications layer with an SSL-like protocol [15].

Comprehensive lineage provides three types of information. The first is represented at the *process* vertices of a lineage graph (depicted in Figure 2). It denotes which principals transformed the data. Additional provenance, such as the details of the operation may also be included. The second type are the *file* vertices, which record details about a file, including the host where it resides, the location in the host's filesystem namespace, and the time when it was last modified (to disambiguate different versions of the same file). The third type of information is denoted by edges and relates the vertices to each other. In particular, the edges are directed and establish the order in which data flowed through the distributed system. This can have significant ramifications. For instance, IBM traced the provenance of the Linux kernel to BSD source code, thereby arguing that it did not infringe on SCO's license [20]. In general, authenticated edges do not suffice since an adversary could claim that data flowed in the opposite direction from the real one. This would not be an issue if each principal could sign the entire lineage graph of each file it modifies. However, a principal only signs the primitive operations it generates in a decentralized system.

If the vertices at each end of the edge had verifiable timestamps, the direction of the edge could be inferred. However, this does not resolve the problem since the timestamps may occur on different hosts. A significant body of research exists on how to synchronize distributed system clocks in a Byzantine fault model (where any host can fail and exhibit arbitrary behavior) [46]. Synchronized time is a prerequisite for the correct operation of some widely used distributed security protocols. For example, Kerberos [5] uses it to determine when remotely issued authorization tickets should expire. Over 100,000 Network Time Protocol (NTP) daemons operate on the Internet [30] to provide a global notion of time. However, these protocols are not sufficient to prove the authenticity of system timestamps to external parties since a host can change the timestamps on files that it stores to any value of its choosing.

A timestamping service (TSS) is a trusted third party that certifies a document's state at a particular time. It does this by hashing the document, appending the current time, and signing the pair. To prevent the TSS from generating fraudulent timestamps, the content signed also includes the hash of the previous document it signed. This creates a linked chain of signatures that prevents it from subsequently fraudulently constructing a record of having timestamped a document [22]. Numerous other variants have been proposed. However, they all require the document to be sent to the timestamping service for it to verify the document's state (by hashing it). Since the data files can be extremely large, timestamping services do not scale well for our application, where lineage edges need to be constructed and certified each time a file is closed after being modified.

4. MENDEL PROTOCOL

We designed the Mendel protocol to allow us to efficiently verify data lineage in distributed environments where content has been modified in multiple administrative domains. The protocol utilizes a three-pronged strategy that combines eager signature verification, lazy trust establishment, and cryptographic ordering witnesses, each of which is described further below. (The protocol is named after Mendel, the 19th century Augustinian priest whose research underpins our understanding of inheritance, which allows us to reliably resolve questions about ancestor and descendant relationships between individuals.) Mendel is implemented as part of SPADE [49], a system for creating and managing tamper-evident provenance.

Preliminaries

We first describe our framework for defining and certifying lineage metadata.

- A *global namespace identifier n = (h,l)* is assumed to consist of the *host h* where the file is stored and a *local identifier l* for locating the file internally. (Examples of a local identifier are (i) a canonical pathname, or (ii) a disk volume and Unix `inode` number pair.)

- A *file f(n,t)* is identified by both the location in the namespace *n* where the file is and the *time t* at which the data of the file was last modified. (We assume the *last writer wins* semantics of current commodity filesystems, where it is the latest content written by contending threads or processes. We do not consider

storage that provides support for *transactional* semantics.)

- The *data d(f(n,t))* is the version of the content stored in file *f(n,t)* at time *t*.
- An *entity e(h,p)* is the owner of *process p* executing on host *h*.
- The *creation time c(h,p)* is when the process *p* started executing on host *h*.
- The *instantaneous inputs* $i(h,p,\tau) = \{f(n_1,t_1), f(n_2,t_2),\ldots\}$ is the set of files the process *p* reads at time τ. Due to direct memory access (DMA), a single process may read multiple files in a single clock cycle. Also note that $t_1, t_2, \ldots \neq \tau$ since the t_i refer to the modification times whereas τ is the access time.
- The *inputs I(h,p,t)* is the set of all files the process *p* on host *h* has read by time *t*:

$$I(h,p,t) = \bigcup_{c(h,p)\leq\tau\leq t} i(h,p,\tau)$$

- The *parents P(f(n,t))* of file *f(n,t)* is set *I(h,p,t)* since those are the only files from which process *p* has read data by the time it writes content into file *f(n,t)* at time *t*.
- The *ancestors* $V_{ancestors}(v_\alpha)$ of file v_α is the maximal set induced by recursively adding vertices that correspond to files that are the parents of any of the current set of vertices:

$$V_{ancestors}(v_\alpha) = P(v_\alpha) \bigcup_{v_\beta \in P(v_\alpha)} V_{ancestors}(v_\beta)$$

- The *lineage graph G(f(n,t))* of file *f(n,t)* is defined by the set of vertices $V(f(n,t)) = \hat{V}_{ancestors}(f(n,t))$ and the set of edges *E(f(n,t))* where:

$$E(v) = \{(v_\alpha, v_\beta) : v_\alpha \in \hat{V}_{ancestors}(v), v_\beta \in P(v_\alpha)\},$$
and $\hat{V}_{ancestors}(v) = v \cup V_{ancestors}(v)$.

- The *digital signature* $S_{k_{e(h,p)}}(x)$ commits entity *e(h,p)* to the input *x* using the signing key $k_{e(h,p)}$, where *x* is the data $d(f(n,t))$ stored in location *n* at time *t*.
- The *parentage certificate* $\Pi(f(n,t))$ commits *e(h,p)* to the claim that *f(n,t)* was modified at time *t* by a process that had read (all and no other) files in *P(f(n,t))*:

$$\Pi(f(n,t)) = \{e(h,p), f(n,t), P(f(n,t)), S_{k_{e(h,p)}}(e(h,p), f(n,t), P(f(n,t)))\}$$

We require the use of *identity-based signatures* [47] for signing any elements that users insert in the provenance metadata. In practice, users will be oblivious to this since it occurs transparently in the filesystem operations described in Section 5. The reason we use identity-based signatures is that they allow verification to be performed without making a network connection to lookup a public key certificate. The verification key for such signatures can be computed using only the user's identity string (and their domain's root certificate). We also assume that the root certificate for an

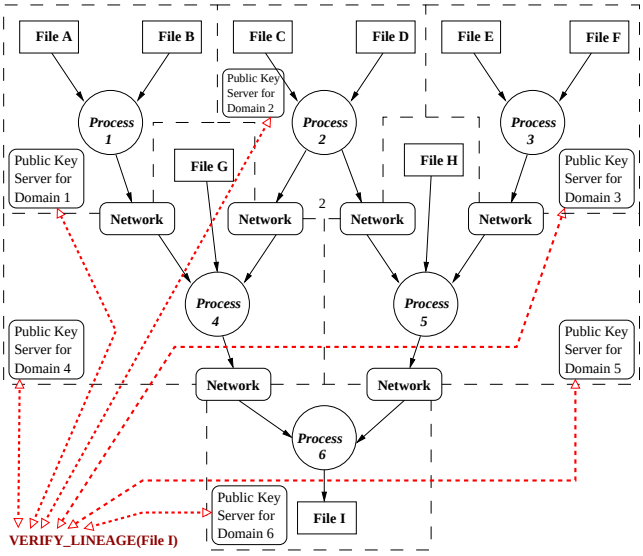

Figure 3: Prior to the Mendel protocol, verifying the lineage of file I would require lookups (shown with dotted arrows) of public keys from servers in domains 1, 2, 3, 4, 5 and 6. This would slow down the verification. Mendel speeds this up with *eager signature verification* using identity-based signatures that can be verified in parallel offline.

administrative domain, C_{AD}, is locally available to the consumer. If it is preferable not to assume this, we could change our protocol to include C_{AD} once in the provenance metadata of each file. Efficient identity-based signature schemes exist that are based on the Discrete Log [47] and RSA [21, 14] problems, although we use one based on Bilinear Maps [7] in our prototype.

4.1 Computational Lineage

Our first concern was that verifying the provenance of a piece of data generated in multiple administrative domains requires the use of numerous public keys that are stored at remote public key servers, as illustrated in Figure 3. The current method for verifying the integrity of file data from another user will not scale for the accompanying lineage metadata, as we explain now. The current approach requires retrieving the external user's public key, the administrative domain's public key, and potentially a certificate to validate the latter key if it is not signed by a previously trusted authority. (In practice, it is reasonable to expect that the root certificate of each administrative domain is locally cached.) Once this certificate chain is validated, the data owner's signature can be verified. However, in the case of the lineage graph, this procedure must be repeated numerous times since many users' signatures may be present. In particular, every vertex may have been signed by a different user. The time to retrieve a strong RSA key from a public key server can be close to 1 second [3]. Clearly, this does not scale for lineage metadata where scores of such verifications may need to be performed to validate a single file's lineage graph.

We note that an RSA verification operation takes less than 0.4 millisecond on commodity hardware [48]. Thus, an alternative approach would be to include the necessary public

keys and certificates in the provenance. Unfortunately, this has two drawbacks. Each provenance edge has two ends – one for the producer and one for the consumer. (Note that the figures depict process vertices to facilitate understanding the workflow. However, in the definitions of Section 4, process vertices are eliminated by directly connecting every input of a process with an edge to every output of the process.) To ensure that the edge cannot be repudiated by either party, signatures from both are needed. When a file is transferred to a remote host, the user at the local host has no knowledge about which user is the actual consumer of the data at the remote host. Thus, the first issue is that the future consumers of the data are not known at the time that the provenance metadata is generated, so it is not possible to predict which certificates to embed. Second, even if the consumer's identity was known in advance, the storage overhead would be substantial since one certificate would need to be embedded for each possible data producer-consumer pair.

Baseline Lineage Verification

Algorithm 4.1: ComputationalLineage(D)

$\{E, S, O, I_1, \ldots, I_n\} \leftarrow$ GetPrimitiveLineage(D)
Output(E)
$K_E \leftarrow$ IbeKeyGenerate(E)
if $I_1, \ldots, I_n = \{\}$
then
  $Result \leftarrow$ Verify(K_E, S, O)
  if $Result =$ False
    then CheckFailed
else
  $Result \leftarrow$ Verify($K_E, S, O|I_1|\ldots|I_n$)
  if $Result =$ True
    then
      for $i \leftarrow 1$ **to** n
        do ComputationalLineage(I_i)
    else CheckFailed

As the provenance of a piece of data increases, verifying its lineage rapidly becomes a nontrivial operation. Therefore, lineage is verified programmatically and only on demand. Our algorithm 4.1 does this by recursively checking each element. The GetPrimitiveLineage function takes a data file D and retrieves the lineage metadata of the primitive operation associated with the last modification of the file. The first element is the identity E of the owner of the process that resulted in the primitive operation defined by the vertex in question. The function IbeKeyGenerate maps the identity to the user's Identity Based Encryption public key, K_E, which is needed to verify signatures the user has generated and can be done locally without the latency of a network connection to a public key server. As the ComputationalLineage function recurses, the output from this step enumerates the identities that have modified any of the data utilized in producing the object. The user's signature S validating the set of inputs $(I_1|\ldots|I_n)$ used to produce the output (O) is verified (even if there are no inputs) with the Verify operation. Note that | refers to the catenation of data, although in practice we use set operations implemented with Bloom filters. Then the lineage of each input (if any exist) is recursively checked. If at any point a signature check fails, the function halts and issues an alert.

Without loss of generality, assume the adversary decides to make a false claim about a primitive operation by principal E that takes inputs $I_1, \ldots, I_n$, and produces output O and signature S. If an adversary inserts a false input I_{n+1}, then Verify($K_E, S, O|I_1|\ldots|I_n|I_{n+1}$) will fail for the relevant primitive operation. Similarly, if they delete an input I_n, then Verify($K_E, S, O|I_1|\ldots|I_{n-1}$) will fail. (The failure will occur even if the input was inserted or deleted at any other location.) Finally, if the adversary claims a different output O', then Verify($K_E, S, O'|I_1|\ldots|I_n$) will fail and cause the change to be flagged.

Optimization 1: Eager Signature Verification

Conservative signature verification would not use a public key until it had been verified using the root certificate of its domain, C_{AD}. This is to avoid expending processing power on checks that may turn out to be unwarranted (if C_{AD}'s authenticity cannot be validated). In our approach, since the public key has been derived from the root certificate, there is no need to wait for such a check to complete. This allows immediate verification of the signatures on elements of the lineage metadata. If C_{AD} is found to be invalid, the signatures dependent on it must be invalidated. However, although the cost of recovery can be high if the transitive closure of dependent signatures in the lineage graph is large, such events are unlikely to occur often since the probability of root domain certificates being revoked is low. As time passes, each node is increasingly likely to have cached valid copies of other domains' certificates, obviating the need for network lookups based on freshness constraints.

We employ a strategy for verifying signatures regardless of whether the corresponding root certificate has already been validated. The approach relies on data flowing through domains with which prior trust relationships exist. We expect that remote nodes will usually act cooperatively much of the time, and malicious activity will be minimal. This allows us to design a protocol with **eager signature verification** of every vertex occurring in parallel rather than serially the way certificate chains are verified - that is, from the sources of the lineage graph to the sink. If an administrative domain certificate C_{AD} is not available, we continue verifying elements of the lineage graph that do not rely on it, flagging failed verifications for subsequent validation. Our scheme produces good average case performance, although it would most likely fail in an adversarial environment where a large fraction of users attempt to abuse it by launching computational denial-of-service attacks or causing timeouts during root certificate retrievals. If such situations occur, the protocol will fail safely – that is, an attacker cannot make the provenance metadata appear authentic when it is not. At worst, the adversary can only cause a data consumer to waste computing resources checking signatures that fail to verify.

Figure 3 depicts the lineage of file I that has been produced as a result of operations in multiple administrative domains. Each process runs in a different administrative domain. In a legacy implementation, the public key of the user who owns each process would be stored at a different public key server. Hence, a data consumer who wished to verify the lineage of file I would need to contact the Public Key Server for Domains 1-6 to get the verification keys $V_1 - V_6$ for the owners of Processes 1-6, respectively, in Figure 3. This is necessary because each primitive operation

Levels / PKI Downtime	2	4	6
5%	0.22	0.98	1
1%	0.04	0.57	0.99
0.5%	0.02	0.34	0.99

Table 1: Probability of verification failure when using traditional PKI for lineage graphs.

(as defined in Section 1.2) in the lineage record of file I is described and signed by the user performing the operation. This approach scales poorly, as illustrated in Table 1 where we estimate the likelihood that a lineage graph's signatures can all be verified in several scenarios: (i) PKI downtime is assumed to be 5%, corresponding to the case when the service is locally administered (as part of the Web service of a department in a university or company, for example). (ii) PKI downtime is assumed to be 1%, as guaranteed by an external managed PKI service provided by Entrust [13]. Finally, the high availability scenario with 0.5% downtime is examined. In all cases, the average fan-in in the graph is assumed to be 4. When public keys are retrieved from locally administered servers, even a lineage graph with just two levels cannot be fully verified 22% of the time. The reliability of the verification process is still acceptable in this case if the keys are hosted by managed PKI services. However, as the number of levels in a lineage graph grows, even in the high availability PKI case, verification is likely to fail with very high probability.

In the Mendel protocol, the administrative domain root certificates $C_{AD1} - C_{AD6}$ allow a data consumer to derive the signers' verification keys $\hat{V}_1 - \hat{V}_6$ from the signers' identities $Id_1 - Id_6$ without any network connections. Identity-based signatures have this property by definition – that is, $\hat{V}_1 = IbeKeyGenerate(Id_1, C_{AD1}), \ldots, \hat{V}_6 = IbeKeyGenerate(Id_6, C_{AD6})$. This information suffices for the signatures in the provenance to be verified, although their authenticity transitively depends on the validity of the administrative domain's root certificates, $C_{AD1} - C_{AD6}$.

CLAIM 1. *Mendel's* eager signature verification *reduces completion time by as much as* $\frac{(f^l-1)(1-u)t}{(f-1)u}$, *where* f *is the average fan-in and* l *the number of levels in the lineage graph,* u *is the average reliability of the PKI service, and* t *is the timeout between retries by the verification client.*

PROOF. A file with a lineage graph with f fan-in and l levels can have content that originated in at most $\frac{f^l-1}{f-1}$ administrative domains, each with its own public key hierarchy. If the corresponding public key servers have an availability of u (in the range 0 to 1), then the probability of being not able to retrieve all the public keys needed to verify the lineage graph is $1-u^{\frac{f^l-1}{f-1}}$. Each time the verification procedure fails to obtain a public key, it will reattempt retrieval after a timeout of t seconds. Upon each subsequent retry, it will fail with probability $1-u$. This process will repeat until all keys have been obtained, resulting in an expected delay of $[\frac{f^l-1}{f-1} \times t \times (1-u)] + [\frac{f^l-1}{f-1} \times t \times (1-u)^2] + \ldots = \frac{f^l-1}{f-1} \times t \times \frac{(1-u)}{u}$. Mendel avoids this delay through the use of identity-based signatures that can be verified offline. □

The large amount of time saved by eager signature verification is illustrated in Table 2. The average fan-in for the lineage graphs is 4. The timeout used is 120 seconds, which is what PGP clients default to [38]. Tables 1 and 2 are constructed by estimating the probability of verification failure and delay in verification, respectively, using the relationship described in Claim 1, and the Entrust [13] and PGP [37] data.

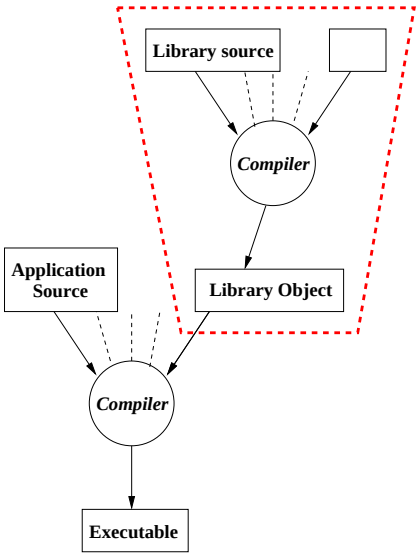

Figure 4: Verifying the lineage of a library used during the compilation and subsequent static linking of an application only requires the signatures of the subgraph with the library as its sink. *Lazy trust establishment* only verifies signatures as needed (rather than checking the entire lineage graph's veracity before performing queries). This saves time by avoiding signature checks on the portion of the lineage graph that are not part of the query (that is, outside the dashed box).

Levels / PKI Downtime	2	4	6
5%	31.6	536.8	8621
1%	6	103	1654.5
0.5%	3	51.3	823.1

Table 2: Delay (in seconds) in verification as a function of the downtime of the PKI service and the number of levels in the lineage graph.

Optimization 2: Lazy Trust Establishment

In our second optimization, when a lineage query for a file is received, we do not verify the signatures on every element of the lineage graph before we return a validated response. Instead, we only verify signatures on vertices as they are used. This is of particular utility when parts of the lineage graph must be retrieved from remote hosts where the lineage was originally generated or is cached. This **lazy trust establishment** allows us to avoid verifying signatures on large subsets of the lineage graph for some queries. For example, if an application executable was statically linked and a

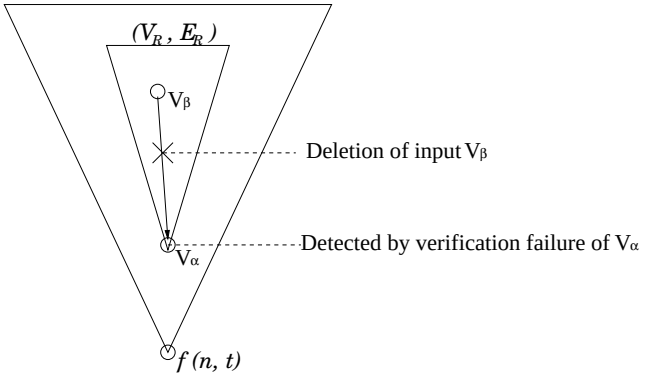

Figure 5: The outer triangle depicts the lineage DAG of file $f(n,t)$ while the inner triangle depicts the response subgraph (V_R, E_R). A fraudulent deletion of input (v_α, v_β) will be detected when Mendel verifies $S_{k_{e(h,p)}}(e(h,p), v_\alpha, P(v_\alpha))$.

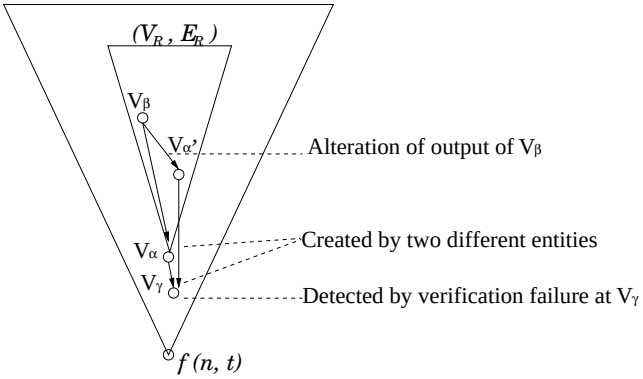

Figure 6: If the adversary replaces output v_α with fraudulent output v'_α, then this will be detected as long as one of the descendants of v_α was created by a trustworthy entity. In the base case, this trustworthy vertex is v_γ and $v_\alpha \in P(v_\gamma)$. Since $v'_\alpha \notin P(v_\gamma$, there will be a verification failure when $S_{k_{e(h,p)}}(e(h,p), v_\gamma, P(v_\gamma))$ is checked.

user only needed to verify the lineage of a particular library used, since there is no separate dynamically linked library to check, the user would ordinarily verify the lineage of the entire executable. However, if the query only asks for the lineage subgraph of the library (embedded in the lineage graph of the application) to be validated, then lazy trust establishment avoids checking the signatures of the other vertices in the lineage graph. This is depicted in Figure 4.

CLAIM 2. *Mendel's* lazy trust establishment *preserves the integrity of responses to computational lineage queries.*

PROOF. A *lineage query Q(f(n,t))* about file *f(n,t)* results in a response subgraph (V_R, E_R). By construction, lazy trust establishment verifies the set of parentage certificates $R = \{\Pi(v) : v \in V_R\}$. To ensure the integrity of the response to the lineage query, we must verify that three properties hold:

- First, if an adversary introduces a spurious input, it must be detected. Such an input would manifest as an edge $(v_\alpha, v_\beta) \in E_R$ where $v_\beta \in P(v_\alpha)$. If v_α is not an element of V_R, no parentage certificate $\Pi(v_\alpha)$ would exist and the fraudulent insertion would thus be detected. If v_α is an element of V_R, then either v_β was fraudulently added to $P(v_\alpha) \in \Pi(v_\alpha)$, in which case the verification of signature $S_{k_{e(h,p)}}(e(h,p), v_\alpha, P(v_\alpha))$ will fail, or $v_\beta \notin P(v_\alpha)$ is detected and (v_α, v_β) is deemed to have been fraudulently inserted.

- Second, the fraudulent deletion of an input must be detected, as depicted in Figure 5. If an adversary deletes input edge (v_α, v_β) where $v_\beta \in V_R$, then since $v_\beta \in V_R$, the verification of $S_{k_{e(h,p)}}(e(h,p), v_\alpha, P(v_\alpha))$ will fail. If $v_\beta \notin V_R$, then $(v_\alpha, v_\beta) \notin E_R$, so the deletion does not affect the integrity of the response. Lazy trust establishment verifies the parentage certificates of all response vertices $v \in V_R$ ensuring that such deletion will be detected. (This can be visualized as verification of all response vertices and accompanying incident edges.) In addition, an adversary may attempt to leave an edge $(v_\alpha, v_\beta) \in E_R$ while removing the corresponding certification to prevent transitive verification along a path through the edge. However, this would be detected when the verification of $S_{k_{e(h,p)}}(e(h,p), v_\alpha, P(v_\alpha))$ failed.

- Third, we must detect any alteration in the intermediate outputs in the lineage graph. Such a modification would manifest as the edge (v_α, v_β) being replaced with edge $(v'_\alpha, v_\beta) \in E_R$ where v'_α was not in the legitimate response to *Q(f(n,t))*. The adversary could fraudulently add v'_α to V_R to prevent this from being detected. The threat model described in Section 2 assumes that vertices are cryptographically committed accurately at the time of generation. Therefore, output modification can be detected as long as at least one vertex in the path between the vertex corresponding to *f(n,t)* and v_β was created by a different entity (from the one that created v'_α). To see this, consider the base case, depicted in Figure 6, where $v_\alpha \in P(v_\gamma)$ and v_α was created by a different entity (from the one that created v_γ). The adversary can change $P(v_\gamma)$ to include v'_α but then the verification for $S_{k_{e(h,p)}}(e(h,p), v_\gamma, P(v_\gamma))$ will fail. Alternatively, the adversary can opt not to change $P(v_\gamma)$, in which case $v'_\alpha \in P(v_\gamma)$ will not hold and the fraudulent output would not manifest in the response. The inductive case follows from exactly the same argument but with v'_α replaced by v_{fraud} and v_γ replaced by $v_{accurate}$, and $(v_{accurate}, v_{fraud})$ being any edge in the path between *f(n,t)* and v_β where the entities that commit $v_{accurate}$ and v_{fraud} differ. Thus, lazy trust establishment provides an integrity guarantee against intermediate output modification in exactly the same case where the legacy approach would – that is, at least one subsequent intermediate vertex is trustworthy.

Since lazy trust establishment provides the same integrity assurance for each response vertex's inputs and output, the integrity of the combined subgraph has the same integrity as would be obtained by computing the entire lineage graph of *f(n,t)*, checking its integrity, and then culling it to the specific query's response. □

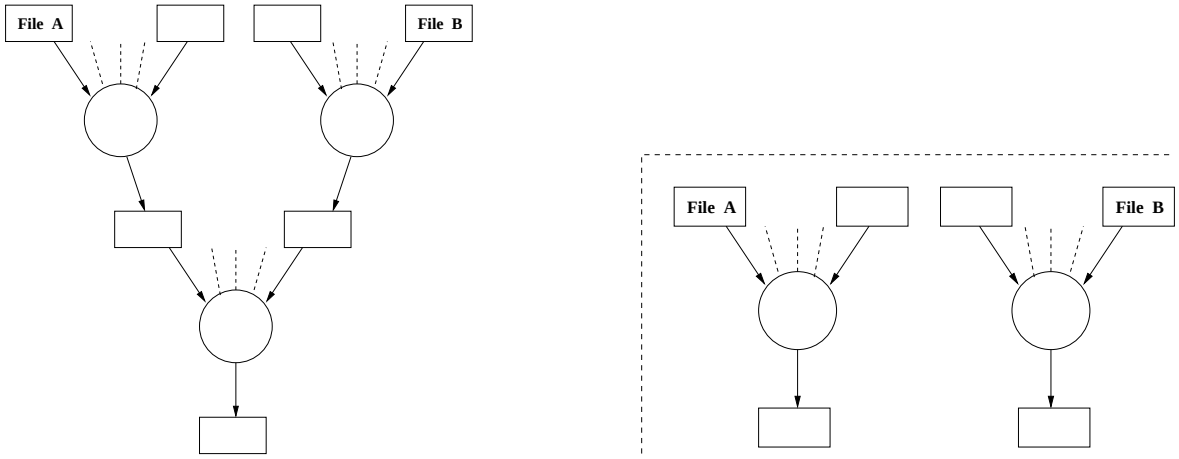

Figure 7: For the purpose of *decisional lineage* queries, we do not distinguish between these two cases: (i) when files A and B have a common ancestor in another file's lineage, and (ii) when files A and B only exist in the lineage of different files.

4.2 Decisional Lineage

In situations when data flows through multiple administrative domains, complete lineage records may not be included in the metadata for a variety of reasons. If the data has a long history, having been derived from many sources that have been repeatedly transformed numerous times, the provenance metadata can grow very large. In the interest of limiting storage and networking costs, lineage records may be retained only at the hosts where they were generated. Independently, such resource constraints may not play a role but the confidentiality or privacy policies may preclude providing the provenance metadata to users external to institutions where the data was processed. In these cases, *decisional lineage* becomes particularly useful. Given two files as input, a decisional lineage algorithm will report whether one is the ancestor or descendant of the other, as illustrated in Figure 8. It will also report if neither relationship holds, which can occur in two cases depicted in Figure 7.

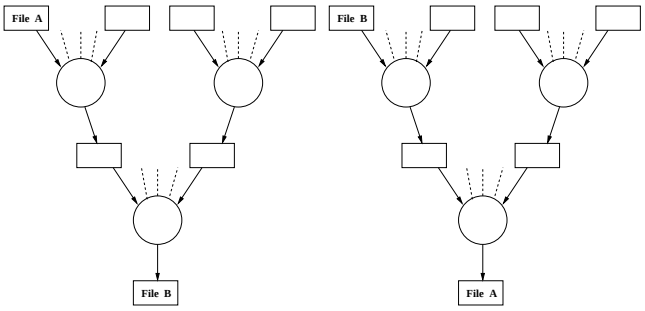

Figure 8: A *decisional lineage* query about two files A and B returns one of three responses: that A is an ancestor of B, that A is a descendant of B, or that neither relationship is true.

The edges in a lineage graph are always directed and point in the direction in which time is flowing. However, locally signed timestamps can be manipulated to alter the apparent direction of edges in a lineage graph. Third-party timestamping services do not scale to the multi-domain setting, as explained in Section 3. We note that the purpose of providing lineage metadata is to assist a subsequent consumer of the data (who may be the same user in a different role). To this end, a principal is free to produce data without adding any provenance information to it. This only decreases the likelihood of the data being found by queries and of being trusted when found. Therefore, our task of disambiguating the temporal ordering of lineage graph edges is more structured than the problem of embedding certified, global, synchronized timestamps.

Optimization 3: Ordering Witnesses

We adapt the idea of document linking from early work on timestamping services [22]. However, instead of an invariant signer, we have invariant objects that progress (as part of the metadata of a derived object) through time. A cryptographically strong scheme to effect this would work as follows. When a piece of data without any provenance is generated, the producer signs a nonce and embeds it as an *ordering witness* in the metadata. Subsequently, when a consumer uses data with embedded provenance, it replaces each ordered witness with another to establish a linked hash chain. Therefore, if an ordered witness $\{\alpha, S_{k_a}(\alpha), \beta, S_{k_b}(\beta)\}$ is extracted, the consumer checks that $\beta \stackrel{?}{=} H(\alpha)$ (where $S()$ is a signature algorithm, $H()$ is a one-way hash, k_i is i's signing key). If so, the consumer replaces the witness with $\{\beta, S_{k_b}(\beta), \gamma, S_{k_c}(\gamma)\}$ where $\gamma = H(\beta)$ and k_c is the consumer's signing key.

Since the above scheme would result in a rapidly growing set of ordering witnesses, we use an alternative which requires substantially lower storage and verification computation at the cost of a weaker security guarantee. When a file without any prior provenance is generated, the producer computes the hash of the file and inserts it into a *Bloom filter* [6], which is a space-efficient data structure that can answer set membership queries in constant time without any false negatives and a false positive probability that can be made arbitrarily small by selecting a large enough size for the filter [9]. The Bloom filter is then signed with the key of the owner of the process that created the file, and the resulting signature and Bloom filter is called the file's *ordering*

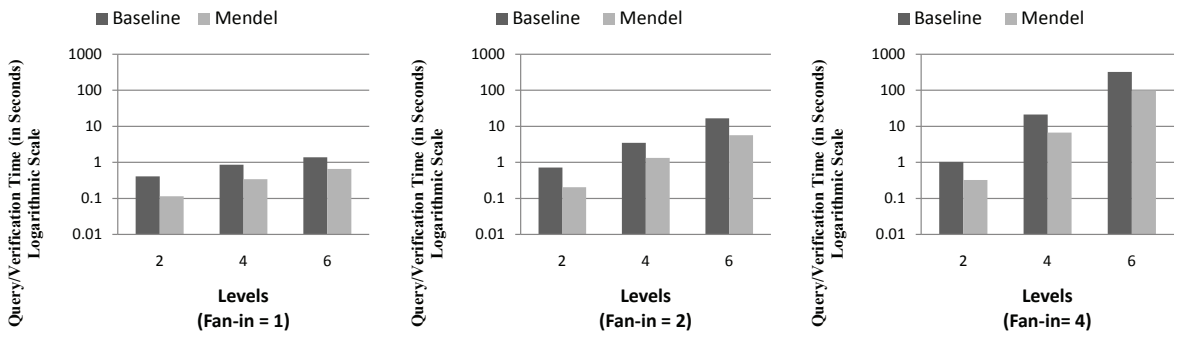

Figure 9: The time to respond to a query and verify all the elements of the lineage graph is compared for the baseline and Mendel schemes. The baseline scheme must retrieve public keys from remote servers in the order that the corresponding signatures are encountered as the lineage graph is traversed from the sink to the sources. Mendel can validate identity-based signatures offline with eager signature verification allowing it to defer validation of root certificates.

witness.

When a file with a provenance record is read, the ordering witness associated with it is extracted from its metadata. At the time a file is written out, the bitwise-OR of all the ordering witnesses is computed. The result is a new Bloom filter having the property that it corresponds to the union of elements in the input Bloom filters [9] – that is, if one of the input Bloom filters answered **true** to a set membership query, the new Bloom filter is guaranteed to answer **true** as well. If all the original filters would have answered **false** to a set membership query, the new Bloom filter will also answer **false** with very high probability. (As mentioned above, by selecting a large enough size for the Bloom filters, this probability can be reduced below any predefined threshold of tolerance.) The hash of the output file is added to the new Bloom filter and it is then signed with the key of the owner of the process writing the file. The result is embedded along with the Bloom filter in the metadata of the output file as its ordering witness.

CLAIM 3. *Mendel answers* decisional lineage *queries with high probability in constant time.*

PROOF. Given two files, say A and B, a decisional lineage query about them can be answered as follows. The hashes h_A and h_B of files A and B, respectively, are computed. The ordering witnesses are extracted from the metadata of files A and B, respectively, and the signatures are verified. Assuming the signatures are valid, the Bloom filters, b_A and b_B, corresponding to files A and B, respectively, are queried with h_A and h_B. If file A was the ancestor of file B, then the query of h_A's membership in b_A and b_B will both return **true**. The query about h_B's membership in b_A will return **false** with high probability, while the query of h_B's membership in b_B will return **true**. Conversely, the case when file A is the descendant of file B corresponds to the same responses with the roles of A and B swapped. If neither relationship were true – that is, file A is not the ancestor or descendant of file B – then with very high probability the query about h_A's membership in b_B will return **false** and the query about h_B's membership in b_A will also return **false**. Note that all these queries take a small constant amount of time as a consequence of the design of Bloom filters. □

5. IMPLEMENTATION

Process Table	
Field	Type
LPID	int(11)
Host	varchar(256)
IP	char(16)
Time	datetime
PID	int(11)
PID_Name	varchar(256)
PPID	int(11)
PPID_Name	varchar(256)
UID	int(11)
UID_Name	char(32)
GID	int(11)
GID_Name	char(32)
CmdLine	varchar(256)
Environ	text

Table 3: A record is added to this table for each process.

File Table	
Field	Type
LFID	int(11)
Host	varchar(256)
IP	char(16)
FileName	varchar(256)
Time	datetime
NewTime	datetime
RdWt	int(11)
LPID	int(11)
Hash	varchar(256)
Signature	varchar(256)
SourceLFID	int(11)
Remote	int(11)

Table 4: When a file is read or written a record is added to this table.

Our initial prototype uses FUSE [18] to intercede on *read()* and *write()* calls. Lineage records are stored in two MySQL [32] tables, shown in Table 3 and Table 4, one for processes and the other for files. When a read or write is intercepted, the calling process's details are extracted using the `/proc` interface. If the process has not read or written a file thus far, a new entry is created in the table of audited processes (shown in Table 3). The record is populated with an identifier that links records in the process and file tables, the hostname and IP address on which the process is running, the time the process was created, the process's name and PID, the parent process's name and PPID, the process owner's effective user name and UID, the process owner's effective group and GID, the command line used to invoke the process, and the environment variables and their values when the process was created.

The first time a file is read or written by a process, a record is added to the table of files (shown in Table 4). The record is populated with a record identifier, the filename, the last modification time of the file when the process first

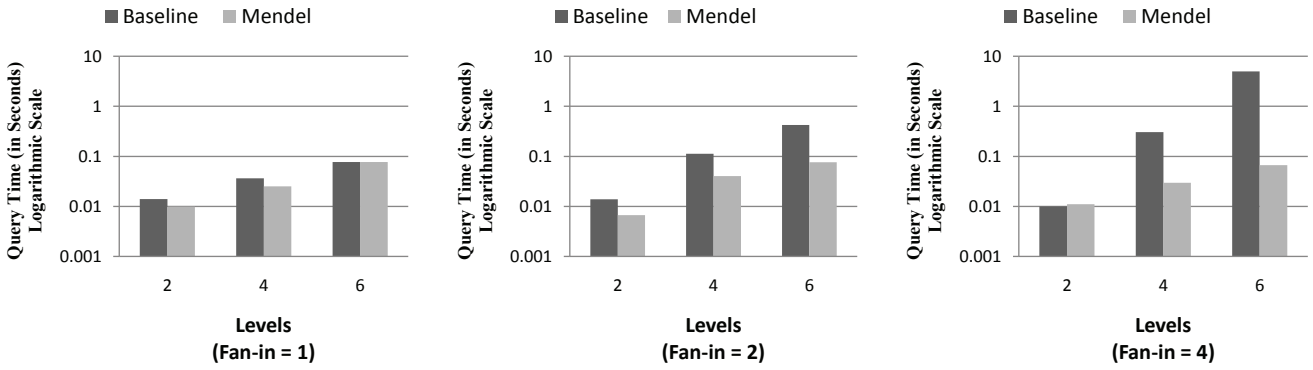

Figure 10: In the previous comparison (in Figure 9), each query required the retrieval of the entire lineage graph. In this comparison, we use queries that only require the retrieval of a path from a source to the sink in the lineage graph. When Mendel uses *decisional lineage* to speed up *lazy trust establishment*, the performance gain over the baseline verification scheme is dramatic.

accesses it, the last time the current process modified the file if it has been opened for writing, whether the file was opened for writing, the identifier that allows linking to the appropriate record in the table of processes, a cryptographic hash to allow verification of the state of the file when it was opened, a digital signature to attest the veracity of the lineage record, and a flag denoting whether the file is local or remote. Though our current implementation does not handle aliasing in the namespace, the issue is tractable. For example, when a symbolic link is created or a file is renamed, we can create an edge between the two corresponding file vertices.

To quantify the performance benefits of Mendel, we used a synthetic workload where the lineage graphs are generated by repeatedly composing operations. The number of *levels* in the resulting lineage graph is one more than the number of pipelined operations used to obtain the final result. The number of inputs each operation reads for the output it produces is the *fan-in*. The baseline protocol in our comparison uses 1024-bit RSA from PolarSSL [39], while Mendel uses identity-based signatures in 512-bit elliptic curve fields implemented with the Pairing-Based Cryptography library [35]. The MIT PGP public key server [37] is used for public key retrieval in our experiments, although in practice the servers would be distinct. Each file read or written is 1 kilobyte in size.

As can be seen in Figure 9, using Mendel to verify a lineage graph is substantially faster than using the baseline public key scheme even when the fan-in is just 2. For example, a lineage graph consisting of a tree with fan-in of 2 and 4 levels takes 3.5 seconds to verify in the baseline scheme, while Mendel only takes 1.3 seconds. Note that the plots are on a logarithmic scale.

When the response to a lineage query is a path through the graph, Mendel leverages decisional lineage to guide the lazy trust establishment process. The verification can proceed without reconstructing the entire lineage graph. Mendel can progress backward from the sink of the query response, at each point checking which parent contains candidates for the previous vertex in the path using the embedded Bloom filters. Since there are few false positives, very few extra paths are reconstructed, with the result that substantially fewer vertices need to be verified than the baseline scheme. To measure the improvement gained, we used queries whose response is a path from a leaf to the root of a lineage tree. The results are shown in Figure 10. Whereas the baseline scheme with fan-in of 4 and 4 levels takes 304 milliseconds, Mendel takes less than a tenth of the time at 30 milliseconds for the same query. The speedup is even more dramatic when the number of levels in the tree increases to 6. The baseline takes 4991 milliseconds, while Mendel takes 66 milliseconds – that is, Mendel is 75 times faster.

6. RELATED WORK

Data provenance has a range of applications. HP SRC's Vesta [25] uses it to make software builds incremental and repeatable. Several distributed systems have been built to help scientists track their data. Chimera [16] allows a user to define a workflow, consisting of data sets and transformation scripts. The system then tracks invocations, annotating the output with information about the runtime environment. $^{my}Grid$ [57] allows users to model their workflows in a Grid environment. It is designed to aid biologists in performing computer-based experiments. The Provenance Aware Service Oriented Architecture (PASOA) project arranges for data transformations to be reported to a central provenance service [52], which can be queried by other users as well. CMCS [34] is a toolkit for chemists to manage experimental data derived from fields like combustion research. It is built atop WebDAV [54], a Web server extension that allows clients to modify data on the server. ESSW [17] is a data storage system for earth scientists. If a script writer uses its libraries and templates, the system tracks lineage so that errors can be tracked back to maintain the quality of data sets. Trio [55] has a data warehouse. It uses the lineage of data to automatically compute its accuracy. Bose and Frew's survey [8] identifies other projects that aid in retrieving the lineage of scientific data. None of these systems focus on ensuring high-performance responses for verified lineage queries when numerous hosts in multiple administrative domains are involved.

Operating system functionality to transparently audit provenance metadata was prototyped in the Lineage File System

(LFS) [45]. LFS modified a Linux kernel to record process creation and destruction; operations to open, close, truncate, and link files; initial reads from and writes to files; and socket and pipe creation. The output was periodically transferred to a local SQL database. The Provenance-Aware Storage System (PASS) [40] audits a superset of the events monitored by LFS, incorporating a record of the software and hardware environments of executed processes. PASS is implemented as a layer in a stackable filesystem [56] and stores its records using an in-kernel port of Berkeley DB [28], providing tight integration between data and metadata. Both LFS and PASS are designed for use on a single host, although their designs can be extended to the file server paradigm by passing the provenance records (and queries about them) from the clients to the server in the same way that other metadata is transmitted (and utilized). However, the use of a central server will not scale when the number of hosts increases if they are all making lineage queries with potentially lengthy responses that must be verified. *Sprov* [23] wraps the Linux C library to record file modifications made by dynamically linked applications. Since it does not track where a process reads data from, its view is limited to a single path in the actual provenance graph - that is, the set of vertices that correspond to the same file after each write operation. They use a hash chain to validate the integrity of the file modifications and introduce the concept of an *integrity spiral* to let the integrity of a particular change be validated with a logarithmic (in the number of changes) sequence of verifications.

7. CONCLUSION

We described Mendel, a protocol for efficient verification of data lineage that is generated in multiple administrative domains. Mendel is designed to be used in automated provenance gathering infrastructures, such as our prototype augmentation of the Linux filesystem. The protocol provides fast computational lineage responses by using lazy trust establishment – that is, only verifying elements of the lineage graph that are needed for answering the respective queries – and eager signature verification, which relies on the use of identity-based signatures to allow parallel offline validation of each element of a lineage graph. Finally, Mendel provides constant time responses to decisional lineage queries about pairs of files using ordering witnesses embedded in Bloom filters inserted in the lineage metadata of each file. With very high probability Mendel can determine whether one file is the ancestor of the other, its descendant, or whether neither relationship holds.

8. REFERENCES

[1] M. Nedim Alpdemir, Arijit Mukherjee, Norman W. Paton, Alvaro A. A. Fernandes, Paul Watson, Kevin Glover, Chris Greenhalgh, Tom Oinn, and Hannah Tipney, Contextualised workflow execution in myGrid, Proceedings of the European Grid Conference, Lecture Notes in Computer Science, Vol. 3470, Springer-Verlag, 2005.

[2] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, Nucleic Acids Research, Vol. 25, 1997.

[3] G. Apostolopoulos, V. Peris, and D. Saha, Transport layer security: How much does it really cost?, Proceedings of the 18th IEEE INFOCOM, 1999.

[4] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S. R. Eddy, S. Griffiths Jones, K. L. Howe, M. Marshall, and E. L. Sonnhammer, The Pfam protein families database, Nucleic Acids Research, Vol. 30, 2002.

[5] Steven M. Bellovin and Michael Merritt, Limitations of the Kerberos Authentication System, Proceedings of the USENIX Conference, 1991.

[6] Burton H. Bloom, Space/Time tradeoffs in hash coding with allowable errors, Communications of the ACM, Vol. 13(7), 1970.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham, Short signatures from the Weil pairing, Proceedings of Asiacrypt, Lecture Notes in Computer Science, Vol. 2248, 2001.

[8] R. Bose and J. Frew, Lineage retrieval for scientific data processing: A survey, ACM Computing Surveys, Vol. 37(1), 2005.

[9] A. Broder and M. Mitzenmacher, Network applications of Bloom filters: A survey, Internet Mathematics, Vol. 1(4), 2005.

[10] Provenance Challenge, `http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge`

[11] D. G. Clarke and D. M. Clark, Lineage, Elements of Spatial Data Quality, Elsevier, 1995.

[12] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, Mapping abstract complex workflows onto Grid environments, Journal of Grid Computing, Vol. 1(1), 2003.

[13] Entrust Managed Services PKI, `http://www.entrust.com/managed_services/key-technical-features.htm`

[14] A. Fiat and A. Shamir, How to prove yourself: Practical solutions to identification and signature problems, Advances in Cryptology, Lecture Notes in Computer Science, Vol. 263, 1987.

[15] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, A security architecture for computational Grids, Proceedings of the 5th ACM Conference on Computer and Communications Security, 1998.

[16] I. T. Foster, J. S. Vockler, M. Wilde, and Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation, SSDBM, 2002.

[17] J. Frew and R. Bose, Earth System Science Workbench: A data management infrastructure for earth science products, SSDBM, 2001.

[18] Filesystem in Userspace, `http://fuse.sourceforge.net`

[19] Novel Information Gathering and Harvesting Techniques for Intelligence in Global Autonomous Language Exploitation, `http://www.speech.sri.com/projects/GALE/`

[20] L. Geppert, Battle of the Xs, IEEE Spectrum, Vol. 40(8), 2003.

[21] L. Guillou and J. J. Quisquater, A paradoxical Identity-Based Signature scheme resulting from

Zero-Knowledge, Advances in Cryptography, Lecture Notes in Computer Science, Vol. 403, Springer-Verlag, 1990.

[22] Stuart Haber and W. Scott Stornetta, How to time-stamp a digital document, Journal of Cryptology, Vol. 3(2), 1991.

[23] Ragib Hasan, Radu Sion, and Marianne Winslett, The case of the fake Picasso: preventing history forgery with secure provenance, 7th USENIX Conference on File and Storage Technologies, 2009.

[24] S. Henikoff, J. G. Henikoff, and S. Pietrokovski, Blocks+: A non-redundant database of protein alignment blocks derived from multiple compilations, Bioinformatics, Vol. 15, 1999.

[25] A. Heydon, R. Levin, T. Mann, and Y. Yu, The Vesta approach to software configuration management, Technical Report 168, Compaq Systems Research Center, 2001.

[26] H. V. Jagadish and F. Olken, Database management for life sciences research, SIGMOD Record, Vol. 33, 2004.

[27] Department of Energy Joint Genome Institute, `http://www.jgi.doe.gov/`

[28] Aditya Kashyap, File system extensibility and reliability using an in-kernel database, Master's Thesis, State University of New York, Stony Brook, 2004.

[29] Anders Krogh, Prediction of transmembrane helices in proteins, `http://www.cbs.dtu.dk/services/TMHMM/`

[30] D. L. Mills, A brief history of NTP time: Confessions of an Internet timekeeper, ACM Computer Communications Review, Vol. 33(2) , 2003.

[31] $^{my}Grid$, `http://www.mygrid.org.uk`

[32] MySQL, `http://www.mysql.com/`

[33] National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov/`

[34] C. Pancerella, J. Hewson, W. Koegler, D. Leahy, M. Lee, L. Rahn, C. Yang, J. D. Myers, B. Didier, R. McCoy, K. Schuchardt, E. Stephan, T. Windus, K. Amin, S. Bittner, C. Lansing, M. Minkoff, S. Nijsure, G. v. Laszewski, R. Pinzon, B. Ruscic, Al Wagner, B. Wang, W. Pitz, Y. L. Ho, D. Montoya, L. Xu, T. C. Allison, W. H. Green, Jr., and M. Frenklach, Metadata in the collaboratory for multi-scale chemical science, Dublin Core Conference, 2003.

[35] Pairing Based Cryptography, `http://crypto.stanford.edu/pbc/`

[36] Protein Data Bank, `http://www.rcsb.org/pdb/`

[37] MIT PGP Public Key Server, `http://pgp.mit.edu/`

[38] PGP Timeout, `https://supportimg.pgp.com/guides/PGP_Command_Line_8.5_man_page.html`

[39] PolarSSL, `http://polarssl.org/`

[40] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer, Provenance-aware storage systems, Proceedings of the USENIX Annual Technical Conference, 2006.

[41] Alex Rodriguez, Dinanath Sulakhe, Elizabeth Marland, Veronika Nefedova, Michael Wilde, and Natalia Maltsev, Grid enabled server for high-throughput analysis of genomes, Workshop on Case Studies on Grid Applications, 2004.

[42] J. L. Romeu, Data quality and pedigree, Material Ease, 1999.

[43] P. Ruth, D. Xu, B. K. Bhargava, and F. Regnier, E-notebook middleware for acccountability and reputation based trust in distributed data sharing communities, Proceedings of the 2nd International Conference on Trust Management, Springer-Verlag Lecture Notes in Computer Science, Vol. 2995, 2004.

[44] Andrej Sali, 100,000 protein structures for the biologist, Nature Structural Biology, Vol. 5, 1998.

[45] Lineage File System, `http://crypto.stanford.edu/~cao/lineage.html`

[46] Fred R. Schneider, Understanding Protocols for Byzantine Clock Synchronization, Technical Report TR-87-859, Cornell University Department of Computer Science, 1987.

[47] A. Shamir, Identity-based cryptosystems and signature schemes, Advances in Cryptology, Lecture Notes in Computer Science 196, 1984.

[48] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon, Performance comparison of security mechanisms for grid services, Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, 2004.

[49] Support for Provenance Auditing in Distributed Environments, `http://spade.csl.sri.com/`

[50] Stefan Stonjek, Morag Burgon-Lyon, Richard St. Denis, Valeria Bartsch, Todd Huffman, Elliot Lipeles, and Frank Wurthwein, Using SAM datahandling in processing large data volumes, UK e-Science All Hands Meeting, 2004.

[51] Swiss-Prot Protein Knowledgebase, `http://us.expasy.org/sprot/`

[52] Martin Szomszor and Luc Moreau, Recording and reasoning over data provenance in Web and Grid services, International Conference on Ontologies, Databases and Applications of Semantics, Lecture Notes in Computer Science, Vol. 2888, Springer-Verlag, 2003.

[53] The Institute for Genomic Research, `http://www.tigr.org/`

[54] `http://www.webdav.org/`

[55] J. Widom, Trio: A system for integrated management of data, accuracy and lineage, Conference on Innovative Data Systems Research, 2005.

[56] Erez Zadok, Ion Badulescu, and Alex Shender, Extending file systems using stackable templates, Usenix Technical Conference, 1999.

[57] J. Zhao, C. A. Goble, R. Stevens, and S. Bechhofer, Semantically linking and browsing provenance logs for E-science, ICSNW, 2004.