An Implementation of the Audit Control Environment (ACE) to Support the Long Term Integrity of Digital Archives

Michael Smorul
Institute for Advanced Computer Studies
University of Maryland, College Park
College Park, MD 20742 USA
toaster@umiacs.umd.edu

Sangchul Song
Department of Electrical and Computer Engineering
Institute for Advanced Computer Studies
University of Maryland, College Park
College Park, MD 20742 USA
scsong@umiacs.umd.edu

Joseph JaJa
Department of Electrical and Computer Engineering
Institute for Advanced Computer Studies
University of Maryland, College Park
College Park, MD 20742 USA
joseph@umiacs.umd.edu

ABSTRACT
In this paper, we describe the implementation of the Audit Control Environment (ACE)[1] system that provides a scalable, auditable platform for ensuring the integrity of digital archival holdings. The core of ACE is a small integrity token issued for each monitored item, which is part of a larger, externally auditable cryptographic system. Two components that describe this system, an Audit Manager and Integrity Management Service, have been developed and released. The Audit Manager component is designed to be installed locally at the archive, while the Integrity Management Service is a centralized, publically available service. ACE allows for the monitoring of collections on a variety of disk and grid based storage systems. Each collection in ACE is subject to monitoring based on a customizable policy. The released ACE Version 1.0 has been tested extensively on a wide variety of collections in both centralized and distributed environments.

Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: Systems and Software; H.3.7 [**Information Storage and Retrieval**]: Digital Libraries

Keywords
ACE, Data Integrity, Digital Archiving.

1. INTRODUCTION
In this paper, we introduce a general software environment called ACE (Auditing Control Environment), which is based on a rigorous cryptographic approach and yet quite efficient and can interoperate with any archiving architecture. Using the new framework, we introduce procedures to continually verify the integrity of the archive. Our approach will allow an independent auditor to verify the integrity of every version of an archived digital object as well as link the current version to the original form of the object when it was ingested into the archive.

Specifically, ACE is based on creating a small-size integrity token for each digital object upon its deposit into the archive (or upon registration of the object of an existing archive), to be stored either with the object itself or in a registry at the archive as authenticity metadata. Cryptographic summary information that depends on all the objects registered during a dynamic time period is stored and managed separately. The summary information is very compact and is of size independent of the number or sizes of the objects ingested. Regular audits will be continuously conducted, which will make use of the integrity tokens and the summary integrity information to ensure the integrity of both the objects and the integrity information. In our implementation, audits can also be triggered by an archive manager or by a user upon data access. However we are assuming that the auditing services are not allowed to change the content of the archive even if errors are detected. The responsibility for correcting errors is left to the archive administrator after being alerted by the auditing service.

DigCCurr2009, April 1-3, 2009, Chapel Hill, NC, USA

2. Overview of ACE Integrity Approach
ACE adopts a two-tier approach. The first tier deals with creating a small size Integrity Token (IT) (Figure 1) for each digital object upon its deposit into the archive (or upon registration of the object of an existing archive), to be stored either with the object itself or in a registry at the archive as authenticity metadata. Cryptographic Summary Information (CSI) depending on all the objects registered during a dynamically adjustable time interval is stored and managed independently of and separately of the archive. The ITs and CSIs are used to continually verify the authenticity of the corresponding digital object. The second tier involves the generation of very compact witness values that cryptographically depend on all the objects ingested during the previous day.

2.1 Tier 1
The first tier integrity information types (IT and CSI) are generated in two steps; aggregative registration and hash-linking. The aggregative registration of the objects is typically invoked during ingestion, and composed of aggregation rounds. The

interval of each round is determined dynamically based on the number of registration requests and time passed. This dynamic aggregation period allows us to control both the maximum size of ITs and maximum wait-time for registration. During an aggregation round, the hashes of all the objects submitted for registration as well as random hashes as necessary are aggregated using an authentication tree such as the Merkle's tree [2]. Note that, in practice, the hash of the object is submitted as a part of the registration request (IT Req in Figure 1). The internal node in the authentication tree has the hash value of the concatenated hashes at the children.

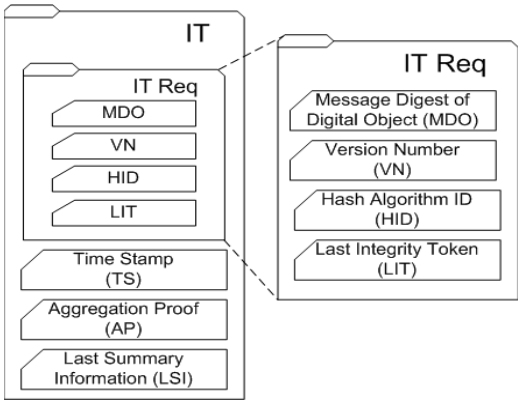

Figure 1: Integrity Token

We insert random hash values into each round to ensure that the tree will always have a certain minimal number of leaves. Figure 2 shows an authentication tree for a round involving eight objects with hash values $h_0, h_1, h_2, ..., h_7$.

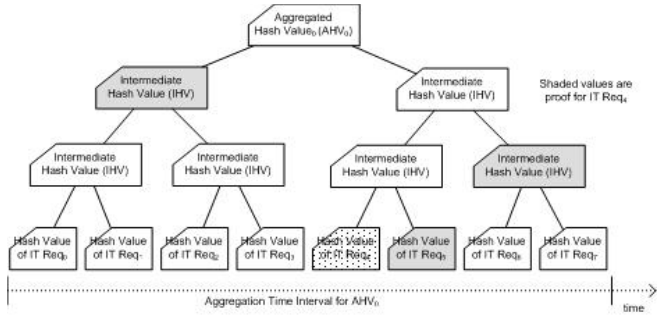

Figure 2: Authentication Tree (IT Reqi contains hi)

Note that the value at the root is a hash value that depends in a cryptographic sense on all the objects processed during a round. For each object, we assemble a short list of hashes from the tree, called an aggregation proof, to enable the derivation of the root value from the hash of the object. We time stamp all the objects participating in each round with the same time stamp.

The second step consists of linking the hash value generated at each round with the hash values generated at the previous rounds using a structure that depends on the linking scheme used. In our prototype, we use a simple binary linking scheme that computes the hash value of the previous Cryptographic Summary Information (CSI) concatenated with the hash value of the current round as illustrated in Figure 3. This is the same scheme as suggested in [3]. In this binary linking scheme, the only two data necessary to construct CSI is the previous CSI and the root value of the authentication tree. The former is included in IT (LSI in Figure 1), whereas the latter can be re-computed using the aggregation proof. In the other words, IT has all the information to re-compute the corresponding CSI at any time.

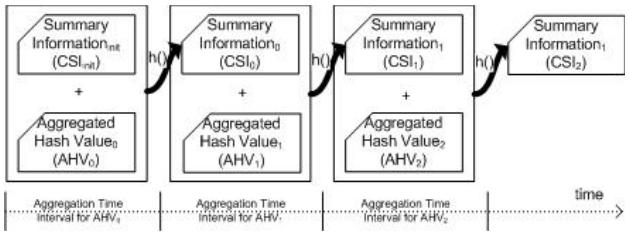

Figure 3: CSI Chain

2.2 Tier 2

As mentioned before, the second tier deals with generating witness values that will ensure the integrity of CSIs which are generated from the first tier operations. A witness value is constructed by aggregating the CSIs that have been created over each day, using an authentication tree whose root value becomes the witness value of the day. These witness values are published over the Internet at well-known public sites offering storage, library, or publication services. Since these witness values are small in size (less than 100KB a year), we also store them on a CD ROM (in fact, on multiple CD-ROMs that are refreshed on a regular basis). Printed versions of this witness are also possible as one line per witness would only require around 30 pages of paper for an entire year! ACE currently uses the Internet newsgroups at Google and UMIACS to publish witness values.

3. ACE Workflow

Two different workflows have been implemented in the first release of ACE. The first is a token registration workflow where new Integrity Tokens are issued from an IMS. The second workflow is the validation workflow where previously issued tokens are used to validate the integrity of files and the token itself.

Registration and validation is performed by an Audit Manager (AM). This audit manager runs physically close to the data that is to be monitored. It is designed to have bit-level access to the data so that it may read all monitored files and generate digests across those files.

The AM requests ITs from an Integrity Management Service (IMS). The IMS performs round aggregation and witness generation as described above. In addition, it also acts as a remote repository for witness values and CSIs. Of course, to fully audit the IMS, a 3rd party would use their own copy of the witness value.

3.1 Token Issuing

Tokens are issued as part the first tier described earlier. The AM generates a SHA-256 digest of the file to be monitored. This generated digest and file name is submitted to the IMS for inclusion in the current round.

The submitted token is aggregated with other requests during the same time interval. The resulting CSI is stored in a database to be later used for witness generation and IT validation. For each request, an IT is generated and returned to the client. This flow is shown in Figure 4.

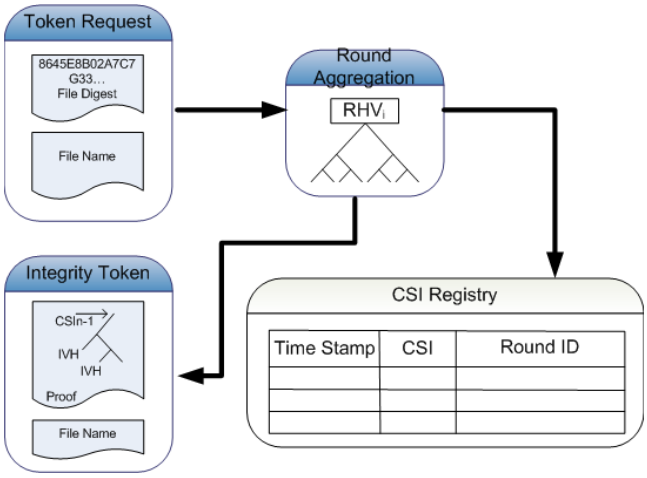

Figure 4: Token Request Workflow

3.2 File and Token Validation

File and token validation occur on the AM subject to a specified policy. This policy may vary between collections. The AM locally stores a copy of digests for each item in a collection. Periodically, each monitored item is read, a digest is calculated and compared with the stored value.

Token validation requires showing the stored digest has not been altered. Validation is done by linking the stored digest to an IMS CSI. This involves calculating the round CSI using the authentication tree stored in the IT and the stored item's digest. The IMS is then queried to retrieve the CSI for the round where the IT was issued. The returned CSI is compared to the calculated CSI and if it matches, the IT and digest are considered trustworthy to a high probability.

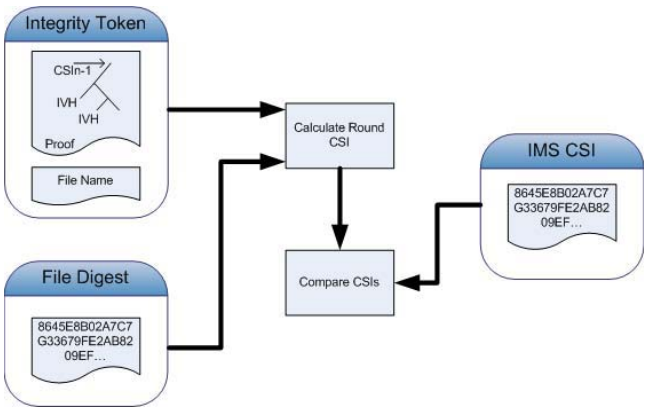

Figure 5: Token and Digest Validation

4. System Components

Version 1.0 of ACE has seen the release of three components. The first is an implementation of the Integrity Management Service which performs round aggregation, token issuing and witness publication. Second is a Java programming interface that allows for high performance access to IMS functionality while being simple to use. Third is a web-based Audit Manager designed to be installed by individual archives to monitor their collections.

Of these components, most archives will only need to concern themselves with managing a local Audit Manager. The IT requirements for installing an AM were designed to be minimal, requiring only MySQL, Java, and Apache Tomcat.

4.1 Integrity Management Service (IMS)

The IMS issues client tokens, stores CSI round data, and publishes a witness value each night. The IMS has been implemented as a Java EE application that provides all IMS functionality as a set of web services. A publically available IMS is running at the University of Maryland at the address ims.umiacs.umd.edu.

The IMS supports two types of aggregation rounds. First is a timed round that will terminate once a timeout or request threshold is reached. The second round is an immediately generated round requested by a client. The immediately requested round will cause the IMS to close any open round, perform aggregation and issue Integrity Tokens.

The IMS offers several web services that allow tokens to be issued and validated. These calls are described below

- *requestTokensAsync* – A non-blocking request for a token or multiple tokens. A receipt is returned to the client that can be used to later retrieve the issued tokens. Requests are added to the current round.
- *retrieveTokens* – Called after requestTokensAsync to retrieve the tokens responses. This must be called after the current round closes or an error will be returned to the client indicating tokens have not yet been calculated.
- *requestTokensImmediate* – A blocking request for a token or multiple tokens. The call requests the tokens be added to the current round and the round be closed. The call will return the requested tokens. The round may include hashes other than ones requested in the call if previous calls to requestTokensAsync were made in the current round.
- *getRoundSummaries* – This call returns a list of CSIs for the requested rounds. This will be used by clients to verify the integrity of its tokens and hashes.

The IMS is backed by a MySQL database that stores unclaimed token responses, round summaries, and nightly witness values.

Nightly witness publication is handled through an API that allows for pluggable publication methods. The WitnessPublisher API provides an abstract class that additional publication methods

must extend. The IMS currently supports an e-mail based publication service.

The IMS installed at umiacs has two e-mail targets configured. First is a listserv hosted at UMIACS called ims-witness available at: http://mailman.umiacs.umd.edu/mailman/listinfo/ims-witness.

Second, a Google group has been created called ace-ims-witness which is available at http://groups.google.com/group/ace-ims-witness. Both mail lists are publically available, allowing anyone to subscribe and receive nightly witness reports. In addition, both lists archive all published witness values.

Archives that wish to challenge the integrity of the IMS in the future should subscribe to one or both of these lists. While public archives of past witness values are available, the values may be considered more trustworthy as an archive can show the provenance of a witness value from publication onward.

4.2 IMS API

A Java API has been written to allow for easy high performance communication with the ACE IMS. The API provides a multithreaded library that allows a client to serially request tokens or token validation while batching those requests and transmitting them in a separate tread. Results are returned to a client through a registered callback. This allows clients to use the bulk request ability of the IMS without having to rework their process to account for the batch processing.

The core of the IMS API is the IMSService class. This class provides a connection to all IMS functionality as well as factory methods for creating token request and validation processes.

The token request and validation processes consist of two parts. First is a queue that clients can serially add requests into. The queue will accumulate requests until either a maximum queue size or timeout is reached. Once either condition occurs, a background thread will be notified. This background thread will copy the work queue and send a request to the IMS. During this process, the client is free to add items to the queue. When the IMS response is complete, the background thread will notify a client supplied callback of the IMS response. Caution must be taken by the client to ensure the callback it supplies is thread-safe with respect to the client thread.

4.3 Audit Manager

The ACE Audit Manager is a Tomcat based Java web application that actively monitors collections on a variety of storage resources. The AM provides a simple web-based dashboard view of all collections that are stored in an archive. After installation of an AM, management of collections is designed to be handled by archivists rather than local IT administrators. A single AM is able to monitor multiple collections on a variety of storage platforms.

An AM handles both registration of new items and monitoring of existing items. The AM is able to request tokens for new items in collections, validate items against their stored digests, and verify those digests using integrity tokens and the IMS. Each collection is able to specify a different audit policy. It also provides complete logging of all actions performed against a collection as well as extensive browsing and reporting capability.

4.3.1 Design

The Audit manager is designed to support multiple types of storage. To do this, it must make a few assumptions about the types of storage it will be operating on. First, this storage is hierarchical. This is generally not a problem, as most filesystems and storage systems are hierarchical in nature. Second, all items are discrete objects. Objects should not be compound objects.

The audit manager stores items in a collection, organized by the parent/child relationship of the items. This allows administrators to browse collections in the same way they would browse files on a hard drive. For each item, the following information is stored

The Audit Manager has a Service Provider Interface (SPI) that allows drivers for other storage mediums to be added. While the AM is able to store all integrity information, it requires the driver to provide a gateway to the underlying storage system and handle connection specific information.

Using the SPI, we have implemented interfaces to the Storage Resource Broker[4], the Integrated Rule-Oriented Data System[5], storage local to the Audit Manager server, and a benchmarking driver to determine maximum performance of a particular AM installation.

4.3.2 Collection Registration and Audit

Collection registration involves gathering all necessary information needed for the AM to communicate with the underlying storage. After a collection has been registered, the AM will perform the first audit of a collection in which it generates tokens for all items in that collection. The following steps illustrate the registration process.

1. Display registration page to client requesting collection root and storage type/driver.
2. Display additional configuration parameters (username, password) as required by the driver. Validate all configuration parameters against the driver.
3. Start a collection audit. Request the driver supply a list of all items in a collection
4. For each item in the collection, register the new item in the database, marking it active, but missing a token.
5. Use the IMS Api to request a token for the new item's hash.
6. Receive notification that a token has been issued.
7. Register new token in a database.
8. Finish Audit.

4.3.3 Collection Metadata

During the auditing process a large amount of metadata is collected for each collection. Each item in the collection has several metadata elements stored to perform audits. In addition, each audit, additional metadata is generated in the form of event logging. These events track every state change of an item while it is under the supervision of ACE.

For each item, the following metadata attributes are collected:

- Item path – Complete path relative to the root of the collection
- First Seen – Date this item was added to the audit manager.
- Last Seen – Last time this item was read and validated. For the initial registration, this will be the first seen date.
- State – Current state of the file. Can be one of the following:
    - A – Active, item is intact, readable and digests match
    - C – Item is present, but its digest doesn't match the stored value
    - M – Item is missing or cannot be fully read.
    - T – Item is registered, but a token has not been received yet.
- Change Date – Date the item's state was last changed. For new items, this will be the date a token was issued.
- Token – Token containing digest and IMS response.

A number of events that change the state of items in a collection will be encountered. These different events have been classified and are recorded in the audit manager. Each event contains the following information

- Event Type – The type of event that occurred. Currently, there are 19 different types of generated events.
- Description – detailed description containing any error message or other information that caused this event
- Session – Audit session this event occurred in
- Date – Date this event occurred.

Events in the AM are grouped by a session identifier. This session identifier connects an event with other events that occurred during the same audit. If only one audit pass has been performed on a collection, then all events associated with that collection will have the same session id. Sessions allow viewing of any events that occurred during a given audit.

Audit messages can be group into two broad categories, normal operating messages and error messages. Normal messages show new files and tokens being added to a collection while error messages show corrupt files, tokens, or storage errors.

Managers are able to filter by item path, collection, event category, and session. This allows for complex queries such as 'show all events for file X in this session' or 'show all errors ever registered for a collection'.

4.3.4 Browsing and Collection Reporting

The Audit Manager provides for collection browsing and reporting. The ACE browser provides a file-system view of items stored in a collection. Items are browsed by expanding folders and clicking on files to view details. From the Browse interface, it's possible to view a token issued to a file, remove a file or directory, download the content of a file, and view the event log for a file.

The Audit Manager is able to generate a report showing items that are corrupt, missing, or otherwise not intact. From this report, managers are able to view log entries associated with flagged items or remove the item from monitoring if the item is later deemed correct. Removal and re-registration of items may be necessary if a new version of an item was added to the archive.

Reports comparing collections can also be generated. For example, a depositor may have a list of digests and filenames they believed were deposited into an archive. Using this list, they can compare the collection in ACE with what they believe was deposited.

In addition to generating web page reports, all reports, status, item details, and event log queries can be exported in JSON (JavaScript Object Notation) format. This allows libraries to include integrity information from ACE in any collection portals they may develop.

4.3.5 Access Control

The Audit Manager provides access controls over to various functions of the web portal. This allows managers to create different usernames and passwords having different roles within the Audit Manager. For example, an account may be created for 'browse' level access which gives read-only access to collections and items in the collection. The browse account may be able to view log files, tokens, file information, but not able to remove items from monitoring or modify collection parameters. The following table shows which access controls are available.

Table 1: Access Permissions

Access	Description
Collection Modify	Modify the connection parameters for a collection.
Browse	Browse files in a collection and display general metadata
Audit	Start a file or token audit on a collection
View Report	View a report showing missing or corrupt items
Remove Item	Remove an item from monitoring
Users	Add or modify users
Download Token	Down integrity tokens attached to files
Download Item	Download the monitored file

5. ACE Use Cases

ACE has been extensively tested, first in the Transcontinental Persistent Archival Prototype (TPAP) and second in the Chronopolis Project. These two testbeds tested ACE against individual collections several terabytes in size and containing several million files. In addition, the TPAP testbed also spanned three different storage types including iRODS[4], SRB[5], and

local file storage. Overviews of these collections are shown in table 2.

Table 2: Collection Overview

Installation	Collections	Items	Log Events
TPAP	32	1,505,392	3,030,152
Chronopolis	4	3,903,922	8,059,007

The TPAP installation of ACE is currently being used to monitor files in a variety of storage architectures at the University of Maryland and San Diego Supercomputing Center. The installation supports collections stored on both the SRB and iRODS. Data is a mix of small text files and large images.

The Chronopolis installation actually involves three independent Audit Monitors installed at the University of Maryland, San Diego Supercomputing Center, and the National Center for Atmospheric Research. Collections are stored in the SRB. Data from the three installations are aggregated into a common portal. This allows depositors to view the overall status of their collections easily without connecting to each site. In addition, the collection comparison functionality is used to ensure that identical copies of each collection exist at all sites.

Collections in Chronopolis vary in both total file count and average file size, allowing us to explore how file size affects collection auditing. The current archive policy is to audit files at UMIACS every 30 days. In Chronopolis, we discovered most of the delay in processing small files was due to SRB overhead, to prevent this from negatively impacting audit speed, ACE audited each collection using 5 threads reading files in parallel. This allowed metadata operations to run in parallel with data retrieval operations. The table below shows the results of these audits. While the SRB can sustain more simultaneous, no more than 5 threads were used to ensure other access to the archive was not impeded.

Table 3: Chronopolis Collections

Installation	Files	Directories	Size	Time(h)
CDL	46,762	28	4,291 Gb	20:32
SIO-GDC	197,718	5,230	815 Gb	6:49
ICPSR	4,830,625	95,580	6,957 Gb	122:48
NC-State	608,424	42,207	5,465 Gb	32:14

A usability test was performed over the summer of 2008 at the SAA Electronic Records Summer Camp. This test involved over 40 archivists and librarians from a variety of non-technical backgrounds. Participants were asked to use the Audit Manager to audit collections stored in the iRODS environment. Most participants were able to successfully audit their collections with less than two hours exposure to the technology.

6. CONCLUSION

In this paper we have described a software system that implements the ACE platform integrity management. We have described an Audit Manager component that is a low maintenance, highly scalable solution for archives and digital libraries to monitor the integrity of their digital assets.

7. REFERENCES

[1] Song, S. and JaJa, J. ACE: A Novel Software Platform to Ensure the Integrity of Long Term Archives. in Archiving 2007. 2007: IS&T.

[2] Ralph Merkle. “Protocols for public key cryptosystems,” In Proceedings of the 1980 Symposium on Security and Privacy, IEEE Computer Society Press, 1980, pp 122–133.

[3] Stuart Haber and W. Scott Stornella, “How to time-stamp a digital document,” Journal of Cryptology, 1991.

[4] A Prototype Rule-based Distributed Data Management System Rajasekar, A., M. Wan, R. Moore, W. Schroeder, HPDC workshop on "Next Generation Distributed Data Management", May 2006, Paris, France.

[5] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In Procs. of CASCON'98, Toronto, Canada, 1998