

BY RANDAL BURNS AND ZACHARY PETERSON

# Security Constructs for Regulatory-Compliant Storage

IN RESPONSE TO A GROWING BODY OF ELECTRONIC RECORDS legislation, the storage community has enhanced data stores to include privacy, auditability, and a “chain-of-custody” for data. There are currently over 4,000 federal, state, and local regulations that govern the storage, management, and retrieval of electronic records. Most notably, the Sarbanes-Oxley Act of 2002, which regulates corporate financial records. Storage vendors provide “compliance” platforms that store and manage data in accordance with regulations, which aids customers in meeting compliance guidelines. Examples include: EMC Centera Compliance Edition,<sup>TM</sup> NetApp SnapLock,<sup>TM</sup> and IBM Tivoli Security Compliance Manage.<sup>TM</sup>

Many of these platforms add storage management policy to existing systems. Vendors start with systems that manage versions of files or volumes. They add immutability to past versions by preventing writes by policy. They also enforce data retention guidelines by not allowing the deletion of protected files. Enhanced

metadata allows users and auditors to examine the store at any point-in-time and investigate the manner in which data have changed throughout their history.

While these features aid organizations in complying with regulations, they do not provide strong evidence of compliance. By following storage management policies, data are versioned and retained for mandated periods. However, there are many opportunities and motivations to subvert such storage policies. In fact, the file system owner represents the most likely attacker. For example, a corporation might alter or destroy data after the corporation comes under suspicion of malfeasance. The shredding of Enron audit documents at Arthur Anderson in 2001 provides a notable paper analog. Similarly, a hospital or private medical practice might attempt to amend or delete a patient’s medical records to hide evidence of malpractice. In policy-based storage systems, past data may be altered or destroyed by reverse engineering file system formats and editing the file data on disk—a common and well understood data forensics task.

We assert that these features need to be cryptographically strong, providing irrefutable evidence of compliance with regulations. This can be achieved for data retention and chain of custody. A storage system commits to a version history so that, at a later time, an auditor may access past data and gain conclusive evidence that the data have been retained and are unmodified. Further, all data should be bound to the users that modify, create, or delete that data. Such constructs improve the evidentiary value of electronic records within the courts, increase an auditor’s confidence in the veracity of the information on which they report (and for which they are responsible), and enhance an organization’s quality of data management.

To these ends, we review three security constructs for versioning file systems. *Digital audit trails* allow a file system to prove to an independent auditor that it stored data in conformance with regu-

lated retention guidelines. *Fine-grained, secure deletion* allows a system to efficiently delete individual versions of files to meet confidentiality requirements, limit liability, and allow data to be retracted. *Per-block authenticated encryption* adds authenticity guarantees to the confidentiality provided by encryption. We also include a distillation of requirements based on a review of relevant legislation and a brief characterization of the performance impact of these techniques based on their implementation within the ext3cow file system.

### Electronics Records Legislation

An examination of the most significant legislation – HIPAA and SOX – reveals a large number of requirements that affect storage system architecture. There are a multitude of other important regulations that we do not review, and we note that the issue of electronic records management goes beyond health-care and financial records. It covers consumer privacy by the Gramm-Leach-Bliley Act of 1999 and government records by the Federal Information Security Management Act of 2002. Indeed, these issues are international, e.g. the European Union has enacted regulations on the protection of personal data, Regulation (EC) 45/2001, and are actively working on regulations for Corporate Governance.

The Health Insurance Portability and Accountability Act (HIPAA), enacted in 1996, was intended to improve the efficiency and effectiveness of the health care system by allowing an individual's medical information to be transferred easily between insurers. As part of the Act, legislators addressed the privacy and security implications of sharing sensitive patient data, leading to the development of standards for electronic health records. HIPAA includes two provisions that address the security and privacy of "protected health information" (PHI). The *Privacy Rule* addresses the use and disclosure of PHI by requiring "covered entities" (health care providers, insurance companies, and individual physicians) to implement access control and error correction procedures, allowing an individual to manage how their personal information can be used. The *Security Rule* requires a covered entity to ensure the confidentiality, integrity, and avail-

ability of electronic PHI. Additionally, the covered entity must protect against reasonable threats as well as protect against reasonably anticipated misuse or unauthorized disclosure.

The Sarbanes-Oxley Act of 2002 (SOX) was enacted in response to the high-profile Enron and WorldCom financial scandals and, in its own words, is designed to "protect investors by improving the accuracy and reliability of corporate disclosures made pursuant to the securities laws and for other purposes."

SOX contains four sections that critically affect the storage infrastructure of publicly traded companies. *Section 304* requires a company's CFO and CEO to personally certify that their electronic financial records accurately represent the company's financial condition. *Section 404* mandates a third-party auditor perform an annual evaluation of a company's electronic record management procedures. *Section 409* necessitates all reporting of electronic records be done in real-time. Lastly, *Section 802* requires companies to produce and maintain authentic and immutable records for at least five years.

The requirements of electronic records legislation are broad, complex, and often ambiguous. The thrust of major regulations and legislation is to protect the consumer/shareholder/patient from unwanted privacy invasions and financial or other damages resulting from professional misconduct.

Despite their diversity, many of the rules share similar language and goals, allowing them to be distilled into a collection of requirements. In general, electronic records are required to be available, confidential, and authentic. Available means that all versions of all records must be accessible in real-time; recovering tape archives from a warehouse is unacceptable. To meet availability, organizations use on-line, versioning stores. This feature anchors commercial compliance products today. Confidential means protecting data from unauthorized disclosure and use. This includes allowing authorized users to control access to their data, including the ability to redact (delete) sensitive information. Authentic means that data are accurate and modifications to data are immune to repudiation.

To further complicate the issue, legislation fails to name specific technol-

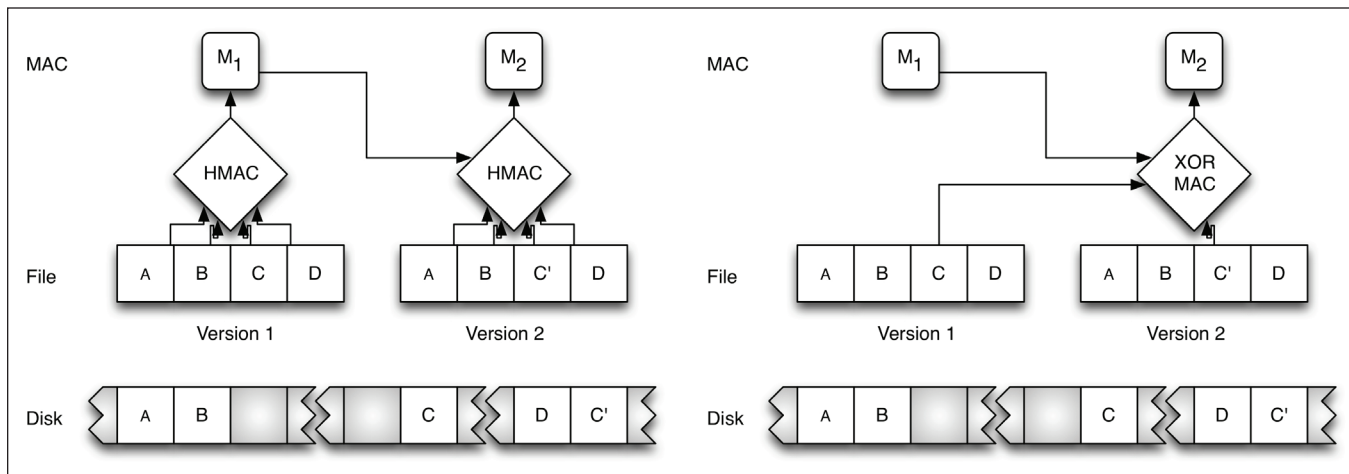
ogies that a company may implement to stay compliant. This is intentional. Binding laws to a particular technology is unnecessarily limiting and may undermine the effectiveness of those laws as technologies change.

### Security Constructs for Compliance

We present three technologies that enhance storage systems to better meet the requirements put forth in compliance legislation. While it may be possible to be "compliant" without using these technologies, our techniques for digital audits, secure deletion, and authenticated encryption allow an organization to make a stronger statement of compliance, providing irrefutable evidence of the same. In the future, were these technologies to become widely deployed as best practices, it may no longer be possible to be compliant without them.

**Secure-Digital Audit Trails.** A digital audit parallels paper audits in process and incentives. The digital audit is a formal assessment of an organization's compliance with data retention regulations. It verifies that data have been retained, have not been modified, and are accessible within the file system. During a failed audit, an auditor can determine the files that have been altered, deleted, or added, but cannot determine their original content, such as the data that were in deleted files. Despite this limitation, the audit process has proven itself in the paper world and offers the same benefits for electronic records. The penalties for failing an audit include fines, imprisonment, and civil liability, as specified by legislation.

To implement a digital audit, a system stores a small piece of cryptographic information at a third party. This information is the output of a keyed cryptographic hash function, or message authentication code (MAC), used to verify the integrity and authenticity of a file's contents. A MAC allows a verifier to detect any changes to the file's content and to identify the user that modified the file. When a file system presents a MAC to the third party, it commits to that version. At a later time, a version may be verified by an auditor; the file system is challenged to produce the data that matches the

**Figure 1: Creating a MAC using serial authentication (HMAC) and incremental authentication (XORMAC).**


stored MAC. In addition to verifying the integrity of a single version's data, the MAC binds that version to all previous versions of the file, authenticating an entire version history (sequence of changes to a file). This is accomplished by combining the result of a previous MAC in the calculation of the current MAC using *hash chaining*.<sup>5</sup> Participating in the audit process reveals nothing about the contents of data. Thus, we consider audit models in which organizations maintain private file systems and publish secure MACs of file data to third parties.

Calculating MACs can be an expensive proposition for a file system with many changing files. This is due to the serial nature of most MAC implementations, such as the keyed-hash message authentication code (HMAC). All file data must be present in memory to calculate an HMAC. Since only small portions of files may be accessed and modified, a serial MAC requires additional I/O to bring all data into cache memory from disk. In our system,<sup>6</sup> we employ *incrementally calculable* MACs based on XOR message authentication codes (XOR MAC).<sup>1</sup> By using the XOR MAC, we construct the MAC for the new version based on the data within the changed blocks only. Figure 1 shows this process when a file has only one block changed ( $C \rightarrow C'$ ). A serial MAC, such as an HMAC, requires the whole file ( $A, B, C, D$ ) as its input and often needs to go to disk to obtain unwritten portions ( $A, B, D$ ): a costly operation exacerbated by the non-contiguity of file blocks on disk. In contrast, the changed blocks ( $C, C'$ ) are always in cache and,

thus, require no additional I/O.

Incremental authentication also provides network bandwidth and storage benefits for the third party by not requiring all MACs for a file to be published. A version history to a file may be audited based on any two published MACs. The auditor starts with the first version and computes MACs for each subsequent version in turn until it computes the last MAC. Due to the properties of hash chaining, when the first and last MAC match, all versions within the history are valid, including those for which no authentication data was stored. This ensures that no versions have been added, removed, or modified.

**Secure Deletion.** The preferred method for non-destructive secure deletion is to repeatedly overwrite data so that the original data may not be recovered.<sup>3</sup> Non-destructive means that the storage media are not damaged and may be reused. The National Institute of Standards and Technology has issued guidelines for overwriting (NIST Special Publication 800-88), making it a best-practice.

Applying secure deletion to a versioning environment is a difficult problem. Repeatedly overwriting the data is intolerably inefficient. Versioning storage relies on *copy-on-write* in which the file system allocates new data blocks only for new or modified data. Unmodified data are preserved and shared across versions. Thus, multiple versions of the same file share much data in common. Securely overwriting a shared block in a past version would erase it from subsequent versions. To address this, a system would need

to detect data sharing dependencies among all versions before committing to a deletion—an expensive operation. Also, in order for secure overwriting to be efficient, the data to be removed should be contiguous on disk. Overwriting non-contiguous data blocks requires many seeks by the disk head. Copy-on-write systems are unable to keep the blocks of a file contiguous in all versions. (If one version is contiguous, all other versions with which it shares data are not).

We have developed techniques that combine encryption with secure overwriting so that a large amount of file data are deleted by overwriting a small stub.<sup>7</sup> In one approach, a stub is the last 128 bits of the output of an all-or-nothing (AON) transform<sup>8</sup> applied to a data block. The AON transform encrypts the data block, producing an output that is slightly larger than the original block, in our case 128 bits larger. The transform has the property that all of the output (stub and data) must be present to decrypt any of the original data block: the all-or-nothing property. Thus, a stub reveals nothing about the contents of data. After a stub has been securely deleted (by overwriting), the corresponding block may no longer be deciphered, even if the adversary later acquires the key. We collect and store stubs contiguously in a file system block so that overwriting a 4K block of stubs deletes the corresponding 1MB of encrypted file data, even if file data are non-contiguous. Figure 2 shows this process. Overwriting a single region of the disk—the stub block—deletes blocks of the file regardless of

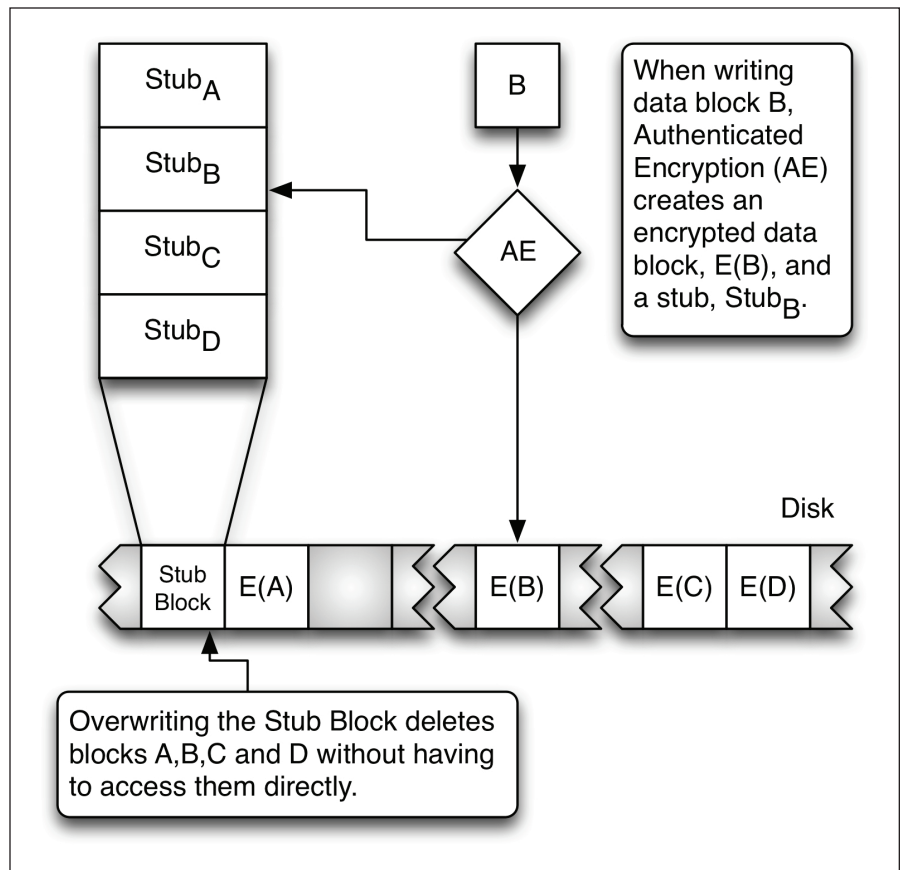
where they are on disk. In practice, this increases deletion throughput by over 200 times when compared with the NIST standard.

**Authenticated Encryption.** Requirements include that personally identifiable and other sensitive data must be kept secure from unauthorized disclosure, be it by accident or malicious attack. A common technique to protect the confidentiality of electronic information is to encrypt it: a transform by which data are readable by only those users that hold the appropriate decryption key. However, encryption alone is insufficient and fails to meet regulatory requirements. An attacker may make undetectable changes to the encrypted information, even absent the encryption key. Thus, a user has no assurance of the integrity or authenticity of decrypted data. Data are required to have integrity, which means that the data as a whole are complete, valid, and free from malicious or accidental alteration. Data are also required to have authenticity, which means that data modifications are bound to an individual user.

To address these requirements, authentication information should be generated on every write and verified on every read. This may be achieved through *authenticated encryption*:<sup>2</sup> a single transform that keeps data both *confidential* and *authentic*. Authenticated encryption binds authentication information to each block of a file, allowing each block to be encrypted and authenticated individually. This benefits read performance, because authentication information is incrementally calculable; when performing small reads, the integrity of data can be verified based on the data read into the cache alone.

Most file systems do not employ authenticated encryption because the encrypted data are larger than the plaintext data. This is an alignment problem, not a capacity problem. File systems use block sizes integral in the size of a disk sector, so that a file system block fits exactly into a number of disk sectors. Authenticated encryption breaks this relationship, expanding the cipher-text data by a small amount, leaving lots of small, unaligned data in each file. The data expansion mandates the file system reorganizes its physical design—a complex task to be avoided if possible.

Figure 2: Performing authenticated encryption for secure deletion.



However, the extra space for the storage of stubs in secure deletion matches perfectly the storage demands of data expansion for authenticated encryption. In fact, the AON transform we use for secure deletion is authenticating. This synergy allows us to use the same extra data (the stub) for both authentication and deletion.

### Implementation

The technologies we have presented are implemented and released in the ext3cow file system, a freely available, open-source system designed for version management in the regulatory environment. In addition to its security features, ext3cow provides file versioning and file system snapshot, standard features in compliant storage. It also implements an intuitive, *time-shifting* interface that provides real-time access to past versions of data.


Based on our implementation, we have found that both the versioning and security features of ext3cow degrade performance minimally. Ext3cow was developed as a versioning extension to ext3: the standard file sys-

tem in most Linux distributions. The wide adoption of ext3 makes it a good platform for comparison. The versioning and security features of ext3cow degrade read and write throughput by less than 5% when compared with ext3.<sup>6,7</sup> Ext3cow has a lightweight snapshot/versioning model that interferes minimally with disk performance. Also, time spent performing disk I/O dominates the time spent on cryptographic operations. Thus, performance should be no obstacle to the adoption of security constructs that enhance regulatory compliance.

### Conclusion

Defining the technology underlying a compliant system is a confusing and evolving process. Ultimately, whether an organization is compliant may be decided in the courts and depends on vagaries such as best practices by peer organizations, business process within the organization, and the organization's intent.<sup>4</sup> The role of specific technologies are to support an organization's efforts to meet legislative guidelines.



Security constructs that provide irrefutable proof of compliance represent a major advance. They demonstrate not only that data management practices meet the specific requirements of legislation, but assist in the subjective evaluation of compliance as well. Organizations that employ them will be making best efforts to be compliant, which has a myriad of benefits including lower risk of liability and increased investor confidence. 

---

#### References

1. Bellare, M., Guérin, R., and Rogaway, P. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology Lecture Notes in Computer Science*, 963, Springer-Verlag, 15–28, 1995.
2. Bellare, M., and Namprempre, C. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology. Lecture Notes in Computer Science*, 1976, Springer-Verlag.
3. Gutmann, P. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the USENIX Security Symposium (July 1996)*, 77–90.
4. Kahn, R. A., and Blair, B. T. *Information Nation Warrior: Information Management Compliance Boot Camp*. ATIM International, May 2005.
5. Lamport, L. Password authentication with insecure communication. *Comm. ACM* 24, 11 (Nov. 1981), 770–772.
6. Peterson, Z. N. J., Burns, R., Ateniese, G., and Bono, S. Design and implementation of verifiable audit trails for a versioning file system. In *Proceedings of the USENIX Conference on File And Storage Technologies (Feb. 2007)*, 93–106.
7. Peterson, Z. N. J., Burns, R., Herring, J., Stubblefield, A., and Rubin, A. Secure deletion for a versioning file system. In *Proceedings of the USENIX Conference on File And Storage Technologies (Dec. 2005)*, 143–154.
8. Rivest, R. L. All-or-nothing encryption and the package transform. In *Proceedings of the Fast Software Encryption Conference (1997)*, 1267, 210–218. *Lecture Notes in Computer Science*.

---

**Randal Burns** (randal@cs.jhu.edu) is an Associate Professor in the Department of Computer Science at Johns Hopkins University, MD.

**Zachary Peterson** (zachary@cs.jhu.edu) is a Senior Analyst for Independent Security Evaluators and an Assistant Research Scientist at the Johns Hopkins University, in Baltimore, MD.

© 2010 ACM 0001-0782/10/0100 \$10.00