Fighting Mallory the Insider: Strong Write Once Read Many Storage Assurances

Radu Sion, Yao Chen
Stony Brook Network Security and Cryptography Lab
{sion, yaochen}@cs.stonybrook.edu

Abstract

Today's increasingly digital societies and markets mandate consistent procedures for the access, processing and storage of information. A recurrent theme is the need for regulatory-compliant storage as an essential underpinning enforcing long-term data retention and life-cycle policies.

Here we introduce a Write-Once Read-Many (WORM) storage system providing strong assurances of data retention and compliant migration, by leveraging trusted secure hardware in close data proximity.

This is important because existing compliance storage products and research prototypes are fundamentally vulnerable to faulty or malicious behavior, as they rely on simple enforcement primitives that are ill-suited for their threat model.

This is hard because tamper-proof processing elements are significantly constrained in both computation ability and memory capacity – as heat dissipation concerns under tamper-resistant requirements limit their maximum allowable spatial gate-density.

We achieve efficiency by (i) ensuring the secure hardware is accessed sparsely, minimizing the associated overhead for expected transaction loads, and (ii) using adaptive overhead-amortized constructs to enforce WORM semantics at the throughput rate of the storage servers ordinary processors during burst periods. With a single secure co-processor, on commodity x86 hardware, the architecture can support unlimited read throughputs and over 2500 write transactions per second.

I. Introduction

Over 10,000 regulations govern the management of information in the US alone [89], in financial, life sciences, health-care industries and the government. These regulations impose a wide range of regulatory policies, ranging from information life-cycle management (e.g., mandatory data retention and deletion) to audit trails and storage confidentiality. Examples include the Gramm-Leach-Bliley Act [65], the Health Insurance Portability and Accountability Act [94] (HIPAA), the Federal Information Security Management Act [95], the Sarbanes-Oxley Act

[96], the Securities and Exchange Commission rule 17a-4 [93], the DOD Records Management Program under directive 5015.2 [90], the Food and Drug Administration 21 CFR Part 11 [92], and the Family Educational Rights and Privacy Act [91].

In Europe, the Directive 95/46/EC of the EU on the protection of personal data, provides privacy and security guarantees for personal information [70]. In addition, many EU members have their own data protection laws; e.g., the UK Data Protection Act of 1998 [69] requires mandatory disposal of personal information after a given retention period, logging of alterations, and strict confidentiality. In Japan, the Financial Instruments and Exchange Law of 2006 [16] regulates financial settings and similar laws can be found in Canada [67], Australia [79] etc.

A recurrent theme to be found in these regulations is the need for regulatory-compliant storage as an underpinning to deliver Write Once Read Many (WORM) assurances, essential for enforcing long-term data retention and life-cycle policies. Main requirements are:

Guaranteed Retention. One main goal of compliance storage is to support WORM semantics: once written, data cannot be undetectably altered or deleted before their regulation-mandated life span end, even with insiders' physical access.

Secure Deletion. Once a record has reached the end of its lifespan, it can (and in some cases must) be deleted. Deleted records should not be recoverable even with unrestricted access to the underlying storage medium; moreover, deletion should leave no hints of their existence at the storage server.

Compliant Migration. Retention periods are measured in years, e.g., intelligence information, educational and health records have retention periods of over 20 years. Accordingly, *compliant data migration* mechanisms are required to transfer information from obsolete to new storages while preserving the associated security assurances.

An example of the above requirements can be found in HIPAA [94] (under SEC. 1173 (d), "Security Standards for Health Information"), which mandates "safeguards [...] to ensure the integrity [...] of the information" and "to protect against any reasonably anticipated [...] threats or hazards to the [...] integrity of the information" (e.g., once stored).

We note that **data confidentiality**, **authentication** and **access control**, are other essential security dimensions in such frameworks, but do not constitute the main focus here.

A common thread running through many of these regulations is the perception of powerful insiders as the primary adversary. These adversaries have superuser powers coupled with full access to the hardware. This corresponds to the perception that much recent corporate malfeasance has been at the behest of CEOs and CFOs, who also have the power to order the destruction or alteration of incriminating records. Since the visible alteration or destruction of records is tantamount to an admission of guilt in the context of litigation, a successful adversary must perform their misdeeds *undetectably*.

Major storage vendors have responded by offering compliance storage and WORM products, including IBM [44], HP [36], EMC [20], Hitachi Data Systems [35], Zantaz [98], StorageTek [84], Sun Microsystem [86] Network Appliance [66] and Quantum Inc. [73].

Unfortunately, these products and research prototypes do not satisfy the requirements outlined above. There are fundamental vulnerabilities to faulty or malicious behavior, because of the reliance on simple enforcement primitives such as software and/or simple hardware device- hosted on/off switches – ill-suited to the target (insider) threat model. In practice, these first-generation mechanisms allow an insider using off-the-shelf resources to replicate illicitly modified versions of data onto seemingly-identical storage units without detection.

As an example, consider a recent IBM US patent for a disk-based WORM system whose drives selectively and permanently disable their write mode by using programmable read only memory PROMs:

> "One method of use employs selectively blowing a PROM fuse in the ... hard disk drive to prevent further writing to a corresponding disk surface in the hard disk drive. A second method of use employs selectively blowing a PROM fuse in processor-accessible memory, to prevent further writing to a section of logical block addresses (LBAs) corresponding to a respective set of data sectors ..." [45].

Such methods do not provide strong WORM guarantees. Using off-the-shelf resources, an insider can penetrate storage medium enclosures to access the underlying data (and any flash-based checksum storage). She can then surreptitiously replace a device by copying an illicitly modified version of the stored data onto a identical replacement unit. Maintaining integrity-authenticating checksums at device or software level does not prevent this attack, due to the lack of tamper-resistant storage for keying material. By accessing integrity checksum keys, the adversary can construct a new matching checksum for the modified data on the replacement device, thus remaining undetected. Even if we add tamper-resistant storage for keying material [10], a superuser is likely to have access to keys while they are in active use: achieving reasonable data throughputs will require integrity keys to be available in main memory for the main (untrusted) run-time data processing components.

More generally, the design of compliance storage is extremely challenging due to the conflict between security, cost-effectiveness, and efficiency. Yet another complicating factor is the often decades-long retention periods; it is unrealistic to expect data to reside on the same device for so long, thus mandating efficient secure migration mechanisms. To defend against insiders, we need processing components that are both tamper-resistant and *active*, such as general-purpose trustworthy hardware. By offering the ability to run logic within a secured enclosure, such devices allow fundamentally new paradigms of trust. Trust chains spanning untrusted and possibly hostile environments can now be built. The trusted hardware will run certified logic; close proximity to data coupled with tamper-resistance guarantees allow an optimal balancing and partial decoupling of the efficiency/security trade-off. Assurances can now be both efficient and secure.

However, the practical limitations of trusted devices pose a set of significant challenges in achieving sound regulatory-compliance. Specifically, heat dissipation concerns under tamper-resistant requirements limit the maximum allowable spatial gate-density. As a result, general-purpose secure coprocessors (SCPUs) are significantly constrained in both computation ability and memory capacity, often being up to one order of magnitude slower than host CPUs.

Nevertheless, it is tempting to think that *tamper-proofing* disks [1] and/or entire server enclosures would solve the problem. We note that, due to the same reasons why SCPUs are hard to design and certify (the inability to properly dissipate heat through a tamper-proof enclosure, and the security assurance and certification process) this is a very difficult task to attain for any reasonable price and has not been achieved. This is one of the main reasons why the IBM 4764/4758 secure CPUs have taken half a decade and several tens of millions of dollars to develop and certify to FIPS 140-2 security level 4 [3, 6, 18, 83].

The constraints imposed by the tamperproof design of SCPUs mandate careful consideration in achieving efficient protocols. Straight-forward implementations of the full processing logic *inside* SCPUs are bound to fail in practice simply due to lack of performance. The server's main CPUs will remain starkly under-utilized and the entire cost-proposition of having fast untrusted main CPUs and expensive slower secured CPUs will be defeated. Here we propose to leverage secure trusted hardware to achieve strong and practical regulatory compliance for storage systems in realistic adversarial settings. We identify essential vulnerabilities in existing regulatory compliant systems and the requirement for strong mechanisms. We first introduce a design for a compliance storage system that offers guaranteed record retention and compliant migration. We show our architecture to be practical. With a single secure co-processor, on commodity x86 hardware, it can support over 2500 transactions per second.

II. Model

A. Economics and Threat Model.

In the threat model for compliance storage, a legitimate user Alice (e.g., the data owner, see Figure 1) creates and stores a record R onto WORM storage. Later, R's existence is regretted and Alice (acting in effect as a malicious "Mallory") will do everything she can to prevent Bob (e.g., federal investigators) from accessing R or inferring its existence. The main purpose of WORM is to defend against such an Mallory. Moreover, as she may have superuser powers, and direct physical access to the hardware, we cannot rely on conventional file/storage system access control mechanisms [30, 59] or data outsourcing techniques [17, 39, 54, 64] to ensure that records can only be modified in compliance.

[1] We note that *tamper-proof* disks are by definition and associated adversarial model (physical access attacks) very different from *encryption* disks [46, 76], a mature, successfully deployed technology, handling a different adversarial model (software attacks).

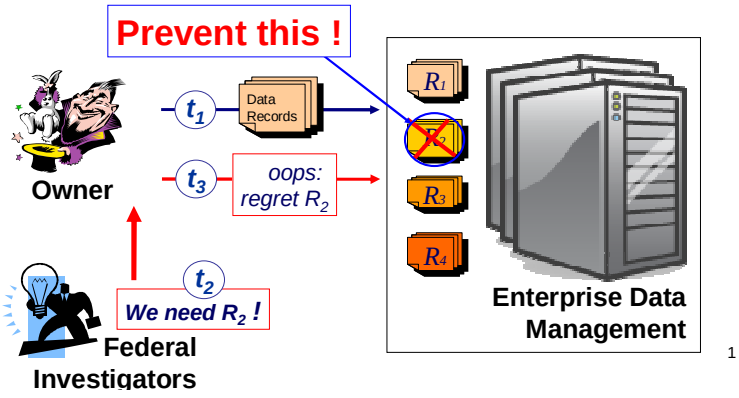

Fig. 1. WORM prevents history “re-writing”.

To prevent physical attacks, strong tamper-resistant and reactive hardware is required. Moreover, in financial industries insiders with multi-billion dollars worth of incentives to alter data records, are common. Given that the deployment domain of the WORM solution is not pre-determined, the tamper-proof assurances of the hardware device should be designed to match the most powerful of adversaries. The National Institute of Standard and Technology has established standards for the security of cryptographic modules and specifically for physical properties and tamper-resistance thereof [8]. The FIPS 140-2 Level 4 certification is at present the highest-attainable hardware security in non-classified domains. Accordingly, in this paper we deploy FIPS 140-2 Level 4 certified hardware.

Moreover, due to the data intensive nature of the application, such hardware should allow the execution of WORM logic inside its trusted enclosure. This is one of the reasons why passive hardware such as specified by the Trusted Platform Module [10] specifications of the Trusted Computing Group [9] can not be deployed here. The TPM specifies a micro-controller that stores keys, passwords and digital certificates. It should be designed to be connected to the main circuitry of computing devices (such as PC motherboards) and ensures that the stored data is secure from external software attacks or physical theft. However, the TPM

> “can only act as a ‘slave’ to higher level services and applications by storing and reporting pre-runtime configuration information. [...] At no time can the TCG building blocks ‘control’ the system or report the status of applications that are running” [10].

This passive nature limits the TPM’s utility in storage subsystems that require active processing, such as WORM, trustworthy indexing, deletion and migration. [5]. Moreover, the TPM cost, incentive and security models are simply too weak for the strongly secure requirements of major deployment settings of WORM, as has been shown repeatedly in recent, simple, yet effective attacks [1, 49] mounted with almost no or very limited cheap off-the-shelf resources.

Properties	Retention	Migration	Secure Deletion
Software Attacks	(a) premature deletion	(c) in-transit alteration	(e) insider "remembers" data past deletion
Physical Attacks	(b) media destruction	(d) replace physical disks	(f) The disk does not perform deletion

TABLE I

WORM ATTACKS. THIS PAPER DETECTS (A) THROUGH (D), YET DOES NOT HANDLE (E) OR (F). DATA OWNERS HAVE ALWAYS THE NATURAL CHOICE OF "REMEMBERING" RECORDS PAST THEIR REGULATION-MANDATED RETENTION PERIODS, OR DESTROYING HARDWARE TO DELETE RECORDS BEFORE EXPIRATION. YET, THE MAIN GOAL OF THIS WORK IS TO PROTECT DATA FROM BEING ALTERED OR DELETED BEFORE ITS REGULATION-MANDATED LIFE SPAN END **undetectably**, EVEN WITH PHYSICAL ACCESS.

Regarding the requirements of *secure deletion* (at the end of mandated retention periods) that can often be found in regulation, we note that Alice has always the natural choice of "remembering" records past their regulation-mandated retention period, e.g., in non-regulated outside storage. Thus, in the WORM adversarial model we can focus mainly on preventing Alice from "rewriting" history, rather than "remembering" it. Additionally, we prevent the rushed removal of records before their retention periods.

B. Deployment

Fig. 2. x3755

Given the applied nature of this work, we believe it is important to briefly discuss the deployment environment. A typical compliant storage cluster contains enterprise disk array subsystems (usually hosted within multiple physical racks) and a set of multi-CPU data management servers, interconnected by fast network switches. IBM's System Storage DS4200 Express Model 7V disk storage system [40] (Figure 3) and IBM's System x3755 [41] (Figure 2) are representative of the first two components.

Fig. 3. DS4200

To enforce strong WORM semantics, we are augmenting the servers with a set of trusted FIPS 140-2 Level 4 certified hardware components as main points of processing trust and tamper-proof assurances. Our architecture employs trusted general-purpose hardware such as the IBM 4758 PCI [3] and the newer IBM 4764 PCI-X [6] cryptographic coprocessors. [7]. The IBM 4764 (Figure 4) is PowerPC-based and runs embedded Linux.

Fig. 4. 4764

The 4758 is based on a Intel 486 architecture and is preloaded with a compact runtime environment that allows the loading of arbitrary external certified code. The CPUs can be custom programmed. Moreover, they (4758 models 2 and 23 and 4764 model 1) are compatible with the IBM Common Cryptographic Architecture (CCA) API [4]. The CCA implements common cryptographic services such as random number generation, key management, digital signatures, and encryption (DES/3DES,RSA). If physically attacked, the devices destroy internal state (in a process powered by internal long-term batteries) and shut down in accordance with the FIPS 140-2 certification.

Critical portions of the WORM logic run inside the trusted enclosure and benefit from its assurances against physical compromise. However, as explained above, these CPUs have limited computation ability and memory

(main heat producer) capacity, making them orders of magnitude slower than ordinary CPUs (see table IV). Therefore such hardware can be used only as a severely constrained aide.

Note on timestamps: At various points in this paper timestamps generated by the SCPU are deployed to assert the freshness of integrity constructs. In this context it is important to note that the considered SCPUs maintain internal, accurate clocks protected by their tamper-proof enclosure. This precludes the requirement for additional handling of time synchronization attacks by the insider adversary. Specifically, as long as the client clocks are relatively accurate (these clocks are not under the control of the server), time synchronization is not an issue. On the other hand, handling the possibility of failures and skews in the SCPU clock is out of scope here.

C. Cryptographic Tools

We use the term *encryption* to denote any semantically secure (IND-CPA) encryption mechanism [31], unless specified otherwise. The approaches discussed here do not depend on any specific instance thereof. A *one-way cryptographic hash* $\texttt{hash}()$ is a fixed-sized output function such that (i) it is computationally infeasible, for a given value $\texttt{V}'$, to find a $\texttt{V}$ such that $\texttt{hash}(\texttt{V}) = \texttt{V}'$, (ii) it is computationally infeasible, to find two values V, V', such that $\texttt{hash}(\texttt{V}) = \texttt{hash}(\texttt{V}')$, (iii) changing even one bit of the hash input causes random changes to all the output bits with 50% probability. (i.e., roughly half of them change even if one bit of the input is flipped). Candidates are the MD5 (fast) or the SHA 1/2 classes (secure) [26, 56, 78]. We consider ideal, collision-free hashes and strongly un-forgeable signatures.

Merkle (hash) trees [57] enable the authentication of item sets by using only a small amount of information. In the hash tree corresponding to data items $S = \{x_1, \ldots, x_n\}$, each node is a cryptographic hash of the concatenation (or other combination) of its children. The tree is constructed bottom-up, starting with cryptographic hashes of the leaves. The verifying party stores the root of the tree or otherwise authenticates it. To later verify that an item x belongs to S, all the siblings of the nodes in the path from x to the root are sufficient in reconstructing the root value and comparing it with the authenticated root value. The strength of this authentication mechanism lies in the above-mentioned properties of the cryptographic hashes. Merkle trees offer a computation-storage trade-off: the small size of the information that is kept at the authenticator's site is balanced by the additional computation (hashing $\log n$ items) and communication overheads. As suggested in the data outsourcing literature (where the adversary is an outsider), Merkle trees are a useful tool in guaranteeing data integrity. However, in a compliance storage environment, where new records are constantly being added to the store, Merkle tree updates ($O(\log n)$ costs) can be a performance bottleneck. Our solution will overcome this by deploying a simple yet efficient range authentication technique relying on the certifying entire "windows" of allocated records (with $O(1)$ update costs).

III. Related Work

In this section we discuss existing WORM mechanisms as well as a set of related research areas such as encryption and versioning file systems.

Tape-based WORM. The Quantum DLTSage predictive, preventative and diagnostic tools for tape storage environments [73] are representative of tape-based WORM. Their WORM assurances are provided under the assumption that only Quantum tape-readers are deployed.

> "DLTSage WORM provides features to assure compliance, placing an electronic key on each cartridge to ensure WORM integrity. This unique identifier cannot be altered, providing a tamper-proof archive cartridge that meets stringent compliance requirements to ensure integrity protection and full accessibility with reliable duplication." [73].

Given the nature of magnetic tape, an attacker can easily dismantle the plastic tape enclosure and access underlying data on a different customized reader. Relying on the physical integrity of a "plastic yellow label" to safeguard compliance data is unacceptable.

Optical-disk WORM. Optical disk WORM guarantees rely on irreversible physical primitive write effects to ensure the inability to alter existing content. However, this very inability makes optical disks unsuited for scenarios with variable retention periods. Moreover, slow performance, small data capacity, and the inability to fine-tune secure deletion granularity limit their use in compliance scenarios. Further, optical disks are vulnerable to simple data replication attacks. Nevertheless, because it is faster than tape and cheaper than hard disk, optical WORM is often deployed as a secondary, high-latency storage medium in the framework of a hard disk-based solution. Care needs to be taken in establishing points of trust and data integrity when information leaves the secured hard disk store for the optical media. As we will discuss below, such integrity assurances can be maintained with the help of the secure hardware hosted inside the main store.

Hard disk-based WORM. Magnetic disk recording currently offers better overall cost and performance than optical or tape recording. Thus almost all recently-introduced WORM storage devices are built atop conventional rewritable magnetic disks, with write-once semantics enforced through software ("soft-WORM"). The EMC Centera Compliance Edition [20] is representative of such soft-WORM offerings. It is a content addressed storage (CAS) product that also offers regulatory compliance capabilities. Each data record

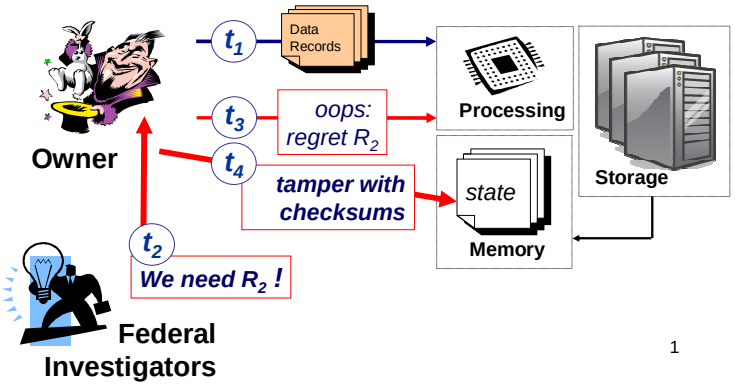

Fig. 5. "Soft-WORM" approaches without the support of tamper-proof hardware are vulnerable to adversaries with physical access to the data store.

> "has two components: the content and its associated content descriptor file (CDF) that is directly linked to the stored object (business record, e-mail, etc.). A digital fingerprint derived from the content itself is the content's locator (content address). The CDF contains metadata record attributes (e.g., creation date, time, format) and the object's content address. The CDF is used for access to and management of the record. Within this CDF, the application will assign a retention period for each individual business record. Centera will permit deletion of a pointer to a record upon expiration of the retention period. Once the last pointer to a record has been so deleted, the object will be eliminated" [20]

Given its software-only nature, this approach is vulnerable to simple software or physical direct disk-access attacks as described above. Data integrity can be easily compromised. The same problems arise with the soft-WORM compliance products from Hitachi [35], IBM's LockVault [42], the IBM System Storage Archive Manager [43, 44], Network Appliance's Snaplock [66], and Sun StorageTek's Compliance Archiving Software [85, 86].

Worth noting is the work by Huang et.al. [38], which introduces **"content immutable storage"**, a mostly soft-WORM storage system that must satisfy the following properties: "1) offer overwrite protection that is secure even against inside attacks; 2) efficiently support index mechanisms; 3) allow records to be properly disposed of after they have expired; and 4) be low cost yet reliable" [38].

Unfortunately, while the above desiderata are well formulated and appropriate, the proposed mechanisms stop short of delivering in a realistic adversarial setting. Specifically, insiders with super-user powers and physical access can simply open the magnetic drive enclosures and alter the underlying media while remaining un-detected. Attempts to prevent this by storing checksum data at locations logically un-addressable from user-land are bound to fail for an insider with full super-user privileges and physical access to the device.

Note on tamper-proofing commodity hardware: Proposing to provide tamper-proof disks to solve this is a worthy initiative, yet very difficult to attain in practice exactly due to the same reasons why SCPUs are hard to design and certify, namely the inability to properly dissipate heat through a tamper-proof enclosure. This is one

of the main reasons why the IBM 4764/4758 secure CPUs have taken half a decade and several tens of millions of dollars to develop and certify to FIPS 140-2 security level 4 [3, 6, 18, 83] [2].

Moreover, even if such devices would be designed, as in any normal system operation, the associated magnetic media MTBFs will lead to several failed disks per day. If each disk is indeed designed for tamper-proofness, their cost will be comparable to the cost of SCPUs – where a major part of the cost is the certification process and the tamper-proof enclosure – leading to maintenance costs of hundreds of thousands of dollars per day. This scheme thus violates requirement (4) as well as a desiderata of a having a "small trusted computing base" (after all, all the disks are now to be trusted).

We note that *tamper-proof* disks are by definition and associated adversarial model (physical access attacks) very different from existing *encryption* disks [46, 76], a mature, successfully deployed technology, handling a different adversarial model (software attacks).

Ultimately, without a reliable, cost-effective tamper-proof point of trust, this chicken-egg circularity can simply not be solved in the considered adversarial model.

In addition to the above drawbacks, it also seems that the paper does not respect the usual semantics associated with mandatory data retention by introducing and allowing an append operation. Imagine an email sent by an Enron executive that specifies "Sell immediately the stock". After this has been committed into a WORM store, it becomes clear that such a "sell" might be illegal. If the WORM store allows appends, all the executive has to do is append something like " after December 2020" yielding a message that loses its original semantics. In other words, most data retention regulations refer usually to the mandated retention not only of the data itself but mostly of the semantics thereof. Allowing un-checked append operations violates this requirement. Properly implemented appends should provide authenticated access logs, however, it is unclear what benefits that would bring at this layer. Instead, it seems a versioning file system could be implemented on top of a no-append WORM store to achieve the same functionality.

Versioning File Systems. Versioning file systems [62, 71, 77] trace and store file changes, making user actions revocable. In [77], the file system determines which old versions of a file to retain and for how long, using a combination of user specified policies and information extracted from file-edit histories. With digital audit trails for versioning file systems [71], the user publishes a small amount of data to a third party, thus committing to a version history. The published data can be later used in auditing to verify the contents of the file system.

While presenting some conceptual similarities to our work, such approaches suffer from (i) significant privacy concerns – few companies would trust third parties with their internal transaction data, (ii) a set of often impractical

[2]It is similarly tempting to suggest the ability to tamper-proof entire server enclosures. Due to the same reasons (above), this would be prohibitive in terms of performance, cost and time-to-market.

assumptions with grave scalability problems – the existence of a globally trusted third party that can support trillions of transactions per second – from the millions of companies registered in the U.S. alone, and (iii) performance drawbacks – network-limited bandwidth and high latency. Moreover, it is likely that the comparably small costs of SCPUs[3] would simply not justify a network-centric solution with all its drawbacks.

Provenance-Aware Storage. A Provenance-Aware Storage System (PASS) [63] automatically collects and maintains "provenance" of documents, which is defined as the complete execution history that produced them. In effect this enables the identification of the specific workflow that produced a certain record. Provenance information also allows the tracking of any data changes between program invocations [33].

Integrity-Assured Storage. Tripwire [50, 51] verifies stored file integrity at scheduled intervals of time. File systems such as I^{3}FS [48] and GFS [29], and Checksummed NCryptfs [82] perform online real-time integrity verification. Venti [74] is an archival storage system that performs integrity assurance on read-only data. SUNDR [53] is a network file system designed to store data securely on untrusted servers and allow clients to detect unauthorized accesses as long as they see each other's file modifications.

Encrypted Storage. Many researchers have proposed file-system-level support for encryption [13, 14, 32, 34, 47, 55, 58, 97]. If native disk support for encryption is not available, these software approaches can fill the role, though at slower speeds. Some of these systems also support data integrity, yet not under the WORM adversarial model.

IV. Architecture

A. Overview

We achieve strongly compliant storage in adversarial settings by deploying tamper-resistant, general-purpose trustworthy hardware, running certified logic at the server site. As heat-dissipation concerns greatly limit the performance of such tamper-resistant secure processors (SCPUs), it is essential to design a protocol stack with minimal impact on cost and efficiency. Specifically, we ensure the access to secure hardware is sparse, to minimize the SCPU overhead for expected transaction loads. We deploy special deferred-signature schemes to enforce WORM semantics at the target throughput rate of the storage servers main processors.

Design Vision. Overall, we believe in the general philosophy outlined by Hsu et al. [38] with the following principles: (i) increasing the cost and conspicuity of any attack against the system, (ii) focusing on end-to-end trust, rather than single components, (iii) using a small trusted computing base, isolating trust-critical modules and making them simple, verifiable and correct, (iv) using a simple, well-defined interface between trusted and untrusted components, and (v) trusting, but verifying every component and operation.

[3]IBM 4764 PCI-X SCPUs are currently priced below $8000.

In line with this philosophy, we believe it is important for the record-level WORM layer to be simple and efficient. Thus, the focus of this paper is on record-level logic only. Specifically, we *do not discuss name spaces, indexing or content addressing* [11, 19, 25, 37, 60, 61, 75, 87, 90, 94, 99] here, as these do not constitute the main thrust, and can be layered conveniently on top.

We would like to note that the mechanisms proposed here can be *layered at arbitrary points in a storage stack*. In most implementations we expect a placement either inside a file system (records being files, VRDs acting effectively as file descriptors), or inside a block-level storage device interface (e.g., for specialized, embedded scenarios with no namespaces or indexing constraints).

Small Trusted Computing Base. The main intuition behind our design is based on the use of the SCPU as a trusted witness to any regulated data updates (i.e., writes and deletions). As such, the SCPU is involved in *updates* but not in *reads*, thus minimizing the overhead for a query load dominated by read queries. The SCPU witnessing is designed to allow the main CPU to solely handle reads while providing full WORM assurances to clients (who only need to trust the SCPU). Specifically, upon reading a regulated data block, clients are offered SCPU-certified assurances that (i) the block was not tampered with, if the read is successful, or — if the read fails, either (ii) the block was deleted according to its retention policy, or (iii) it never existed in this store.

No Hash-Tree Authentication. To escape the $O(logn)$ per update cost of the straight-forward choice of deploying Merkle trees in data authentication, we introduce a novel mechanism with identical assurances but constant cost per update. To achieve this, we label data blocks with *monotonically increasing* consecutive serial numbers and then introduce a concept of sliding "windows" which can now be authenticated with constant costs by only signing their boundaries, due to their (consecutive) monotonicity (vs. going up the Merkle tree in $O(logn)$). In doing so we lose some Merkle-tree expressiveness not required here, namely the ability to handle arbitrary (non-numeric) labels.

Peak Performance. During high system-load periods, to further increase throughput we temporarily defer *expensive* witnessing operations (e.g., 1024-bit signatures) with less expensive *short-term secure* variants (e.g., on 512-bit). The short-term security is adaptive and ensures that the system can strengthen these weaker constructs later, during decreased load periods – but within their security lifetime. Thus, the protocols adaptively amortize costs over time and gracefully handle high-load update bursts.

B. Building Blocks

In the following we detail the above intuitions and main solution building blocks. We define:

1) **Data record.** A data item governed by storage-specific regulation. Data records are application specific and can be files, inodes, database tuples. Records are identified by descriptors (RDs).

Field	Description
SN	A system-wide unique 64-80 bit serial number.
attr	WORM-related attributes, including creation time, retention period, applicable regulation policy, shredding algorithm, litigation hold, f_flag, MAC, DAC attributes
RDL	The **R**ecord **D**escriptor **L**ist – a list of physical data record descriptors corresponding to the current VR $\{RD_1,\ RD_2,\ ...\}$.
metasig	SCPU signature on (**SN**, **attr**): $S_s\left(SN,\ attr\right)$.
datasig	SCPU signature on **SN** and a chained hash (or other incremental secure hashing [12, 15]) of the data records : $S_s\left(SN,\ Hash\left(data\right)\right)$.

TABLE II
OUTLINE OF A VRD.

2) **Virtual record (VR).** A VR basically groups a collection of records that fall under the same regulation specific requirements (e.g., identical retention period) and need to be handled together. VRs are allowed to overlap, and records can be part of multiple different VRs (being referenced through different descriptors). This enables a greater flexibility and increased expressiveness for retention policies, while allowing repeatedly stored objects (such as popular email attachments) to potentially be stored only once.
3) **Virtual record descriptor (VRD).** A unique, securely issued identifier for a VR. Its structure is outlined in Table II. A VRD is uniquely identified by a securely issued system-wide serial number (`SN`), and contains various retention-policy related attributes (`attr`), a list of physical data record descriptors (`RDL`) for the associated VR data records, and two trusted signatures (`metasig` and `datasig`) issued by the SCPU, authenticating the `attr` and `RDL` fields.
4) **Virtual record descriptor table (VRDT).** A table of VRDs indexed by their corresponding SNs maintained by the main (un-trusted) CPU on disk.

1) The VRDT structure: The untrusted main CPU maintains (on disk) a table of VRDs (VRDT) indexed by their corresponding serial numbers. These serial numbers are issued by the SCPU at each update. The SCPU securely maintains two private signature keys, s and d respectively, that can be verified by WORM data clients. Their corresponding public key certificates – signed by a regulatory or certificate authority – are made available to clients by the main CPU.

The SCPU deploys s for the `metasig` and `datasig` signatures in the VRD and d to provide deletion "proofs" that the main CPU can present to clients later requesting the deleted records. Specifically, when the retention period for a record v expires, in the absence of litigation holds, its corresponding entry in the VRDT is replaced by $S_d(v.SN)$ ($v.SN$ is the unique serial number for v). A VR v can be in one of two mutually-exclusive states:

1) *active*: data records and attribute integrity is enforced by the `metasig` $= S_s(SN, attr)$ and `datasig` $= S_s(SN, Hash(data))$ signatures, or
2) *expired*: with the associated "deletion proof" signature $S_d(v.SN)$ present in the VRDT.

Thus, the VRDT entries contain either the VRD for active VRs, or the signed serial number for records whose

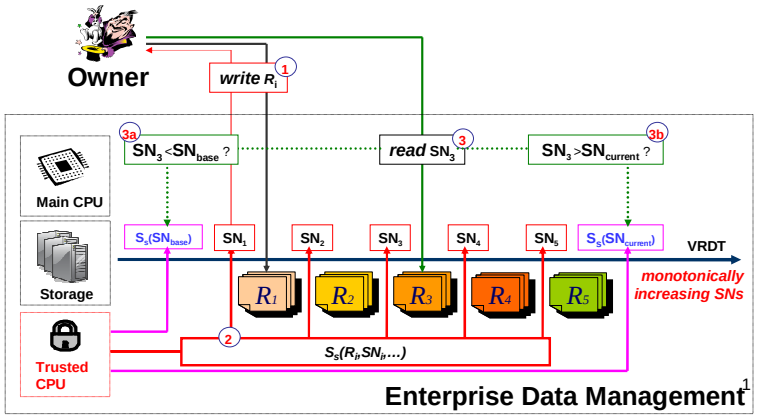

Fig. 6. Step 1 and Step 2 show that on every write, a new serial number will be generated. The SCPU cooperates with the main CPU to manage serial numbers. The VRDT entries contain monotonically increasing consecutive serial numbers within a specific window: $[SN_{base}, SN_{current}]$. In Step 3, for each read, the main CPU will check if the SN is outside the window. Any SNs outside this range have undoubtedly expired and have been deleted (or are not allocated yet) to limit the VRDT's storage footprint. The unique SN-to-record associations are certified by a trusted signature S_s.

retention periods have expired (see Figure 6).

Window Management. Serial number issuing and VRDT management are designed to minimize the VRDT-related storage. The main idea is to use a sliding window mechanism through which previously expired records' deletion proofs can be safely expelled and replaced with a securely signed lower window bound. To handle the fact that while some of retention expirations are likely to occur in the order of insertion, this is unlikely to hold for all records, an additional data structure controlling record expiration will be introduced later.

Specifically, we denote the lowest serial number among all the still *active* VRs (whose retention period has not passed and/or have a litigation hold) as SN_{base}. Let $SN_{current}$ be the highest currently assigned SN. Then the window defined by these two values contain all the active VRs (and possibly a few already expired ones). Any deletion proofs outside of this window are not of WORM-interest any more, and can be securely discarded. Now the main CPU can convince clients that any of the records outside of the current windows have been rightfully deleted (or have not been allocated yet) by simply providing $S_s(SN_{base})$ and $S_s(SN_{current})$ as proofs.

In order to prevent the main CPU using old $S_s(SN_{current})$ values to maliciously ignore recently added records, one of two mechanisms need to be applied: (i) upon each access, the client contacts the SCPU directly to retrieve the current $S_s(SN_{current})$, or (ii) $S_s(SN_{current})$ will also contain a timestamp and the client will not accept values older than a few minutes – and the SCPU will update the signature timestamps on disk every few minutes (even in the absence of data updates). In general cases, we believe (ii) should be chosen for the following reasons: in a busy data store, the staleness of the timestamp on $S_s(SN_{current})$ is not an issue, due to the continuously occurring updates; on the other hand, in an idle system, the small overhead of a signature every few minutes does

Function	Description
write(data,ret,pol,shr) returns: new **VRD**	Writes data record, associated with given retention, policy and shredding algorithm.
assoc(rd[],ret,pol,shr) returns: new **VRD**	Associates set existing RDs under given retention, policy and shredding algorithm.
read(sn)	Reads from an existing VR.
delete(data,serial)	Internal access point used by the SCPU to delete a VR. Not available to clients.
lit_hold(sn,C) returns: **VRD**	Notifies of a litigation hold to be set on a VR. This can only be invoked by authorized regulatory parties with trusted credential: $C = S_{reg}(sn\|current\_time)$
lit_release(sn,C)	Release a previously held litigation lock. Can only be invoked by the regulatory party owning it.

TABLE III
OUTLINE OF THE WORM INTERFACE.

not impact the overall throughput.

Naturally, to reduce storage requirements, a similar technique can be applied further for different expiration behaviors. Specifically, if records do not expire in the order of their insertion – likely if the same store is used with data governed by different regulations – we can define the following convention: the main CPU will be allowed to replace any contiguous VRDT segment of 3 or more expired VRs with SCPU signatures on the *upper* and *lower* bounds of this deletion "window" defined by the expired SNs segment. This in effect enables multiple active "windows", linked by these signed lower/upper bound pairs for the deleted "windows". Since the signatures result in additional SCPU overhead, we can deploy these storage reduction techniques during idle cycles.

It is important to note that the upper and lower deletion window bounds will need to be correlated, e.g., by associating the same unique random window ID to both (e.g., inside the signature envelope). This correlation prevents the main CPU to combine two unrelated window bounds and thus in effect construct arbitrary windows. Also, in order to avoid replay attacks of old $S_s(SN_{base})$ signatures they will include expiration times. Moreover, such replays would not achieve much, as the clients have always the option to re-verify the correct record retention upon read.

2) Retention Policy Conflict Resolution: Because data records are allowed to participate in multiple VRs, it is important to decide what happens if a record falls under the incidence of two different, potentially contradicting policies. In the WORM layer, where the main concern lies with securing retention policy behavior, the conflicts to be handled are likely the result of different associated expiration times. For such conflicts, several solutions are available: (i) do not allow the same data record to participate in multiple VRs (use copies thereof instead), or (ii) resolve the policy conflict according to predefined conventions.

In (ii), the pre-defined convention should relate to the interpretation of the specific conflicting regulations. One alternative could be to simply always delete the record at its earliest mandated expiration time. Note that special

care should be taken if the record is a shared block in the file system. Another alternative could be to force the record's retention until its last occurring expiration. The later can be enforced by associating each data record securely with a reference count of how many VRD are "pointing" to it, and erasing from media only when the reference count is zero. The implementation of such an association however, is non-trivial. A data structure similar in function to the VRDT will need to be maintained for the reference counters of each record.

3) WORM Operations: **Write.** In a **write**, the following operations are executed. The main CPU writes the actual data to the disk, and messages the SCPU with the resulting RDs and the corresponding attributes (such as regulation policy, retention period and shredding method parameters). *Data records and their RD descriptors are implementation specific and can be inodes, file descriptors, or database tuples*.

The SCPU increments the current serial number counter to allocate a SN for this new VR and then generates its `metasig` and `datasig` signatures. To create `datasig` the SCPU is required to read the data associated with the stored record. In section IV-C.1 we discuss how to reduce this overhead at burst-periods under a slightly weaker security model (the main CPU will be trusted to provide `datasig`'s hash; the hash will be verified during idle times). The evaluation in section V considers both models.

Next, the main CPU creates a VRD, associates it with the specified attributes, as well as `datasig` and `metasig`, both provided by the SCPU. The VRD is then written by the main CPU to the VRDT maintained unsecured storage.

Read. To perform a read, a record handle (i.e., the SN) needs to be provided to the WORM layer. It is important to note that, as discussed in section IV-A, the associated data indexing and SN management mechanisms are intentionally not discussed here.

A client's **read** operation only requires main CPU cycles. This is important, as query loads are expected to be often mostly read-only. If a read of a VR v is disallowed on grounds of expired retention, the main CPU will then either provide $S_d(v.SN)$ (proof of deletion), or prove that the serial number of v is less than SN_{base} (thus rightfully deleted) by providing $S_s(SN_{base})$. Similarly, in the multiple "windows" solution (see section IV-B.1), the main CPU will need to provide a SCPU-signed lower and upper bounds for the window of expired SNs that contains v, as proof of v's deletion.

In a successful read the client receives a VRD and the data. It then has the option of verifying the SCPU `datasig` and `metasig` signatures[4]. If the signatures do not match, the client is assured that the data (or the corresponding VRD) has been prepensely modified or deleted. This is so because the (consecutive) monotonicity of the serial numbers allow efficient discovery of discrepancies.

Record Expiration. Record **expiration** and their subsequent deletion is controlled by a specialized Retention

[4]The client must have access to the appropriate SCPU public key certificates (the server can provide them), and have access to a (roughly) synchronized time server.

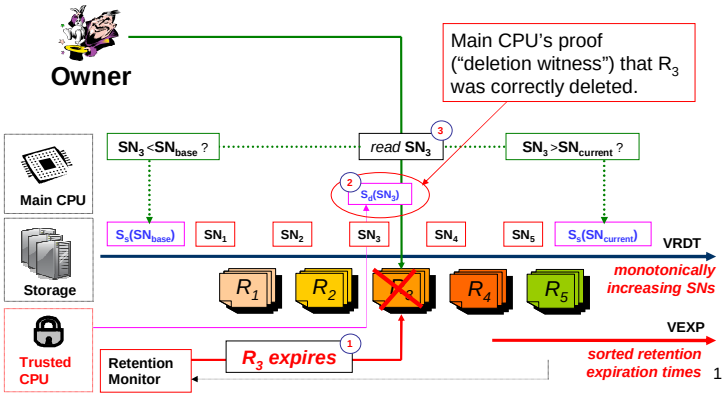

Fig. 7. The SCPU witnesses retention expiration events. When a record expires (Step 1), the SCPU provides an unforgeable proof of deletion (the signature $S_d(SN)$) to the main CPU (Step 2) to present for future read queries if necessary (Step 3). The SCPU maintains a sorted list (VEXP) of next-to-expire SNs and runs a retention monitor (RM) to ensure timely deletion of records.

Monitor (RM) daemon running inside the trusted SCPU enclosure [5]. To amortize linear scans of the VRDT while ensuring timely deletion of records, the SCPU maintains a sorted (on expiration times) list of serial numbers (VEXP), subject to secure storage space. The VEXP is updated during light load periods (e.g., night-time). As common retention rates are of the order of years, we expect this to not add any additional overhead in practice (we discuss alternatives to this assumption in Section V). The VEXP is deployed by the SCPU to enable efficient and timely deletion of records. To this end, the RM is designed to wake up according to the next expiring entry in VEXP and invokes the **delete** operation on this entry. It then sets a wake-up alarm for the next expiration time and performs a sleep operation to minimize the SCPU processing load. If a new record with an earlier expiration time is written in the meantime, the SCPU resets the alarm timer to this new expiration time and updates the VEXP accordingly.

To **delete** a record v (Figure 7), the SCPU first invokes the associated storage media-related data shredding algorithms for v (not discussed). It then provides the main CPU with $S_d(v.SN)$, the proof of v's rightful deletion of, which will replace v's entry in the VRDT. The main CPU can then show this signature as proof of rightful deletion to clients.

Note on secure deletion: As discussed in Section II, achieving secure deletion is not the main aim of this paper. In *any solution*, the data owner Alice has always the natural choice of "remembering" records past their regulation-mandated retention period. Thus nothing can be proven to outside parties to the opposite effect. Nevertheless, for the benefit of an innocent insider, it is important to provide the ability to ensure secure deletion is triggered (against any other future threats, internal or external)

Litigation. Often, records involved in ongoing litigation proceedings will reside in active WORM repositories. A court can then mandate a litigation hold to be placed on such active records, which in effect will prevent their

[5]Note this is another reason why TPMs cannot be deployed here.

deletion even if mandated retention periods have expired – such records cannot be deleted until litigation release. This is achieved through the **lit_hold** and **lit_release** entry-points. Both operations will alter the `attr` field to set a litigation held flag together with an associated timeout of the hold. This process will be performed by the SCPU, who will subsequently also update `metasig`.

Litigation holds can be set only by authorized parties identified with appropriate credentials. In their simplest form, these credentials can be instantiated as a verifiable regulation-authority signature on the record's SN, the current time stamp $C = S_{reg}(SN, current\_time)$ (and an optional litigation identifier). This signature can be stored as part of the `attr` field, e.g., to allow the removal of the hold by the same authority only (or other similar semantics). This will be achieved by invoking **lit_release**.

Note on failures and operation atomicity: In all of the above operations, failures in timely updates to the disk-hosted data structures (e.g., the VRDT) can impact the WORM semantics and leave the store in an inconsistent state. For example, apparently, failures in the deletion process could cause records to be physically deleted before their corresponding deletion proofs have been generated. To handle such failures, the recovery process will need to be carefully designed, e.g., to explore the entries in the VRDT and reconcile them with the records in the VEXP, ensuring deletion proofs will be generated (upon recovery) for all expired records. *Media failures* are also possible, and do not constitute the subject of the current work. It is unclear whether in general one could distinguish between a failure-induced read error and a malicious data altering attack. Appropriate resources should be deployed to prevent this (e.g., redundant storage mechanisms, multiple WORM-compliant off-site backups etc).

Note on forward security: By its very nature, any signature-based solution defending against insider adversaries by isolated TCBs in hostile environments would have trouble regenerating keys to ensure forward security – since signature verification must be available to both trusted and untrusted outsiders. It would be interesting to see if a private key evolution mechanism can be designed that allows for the evolution of also the public key under the certification authority's signature!

Yet, with the available RSA constructs currently forward security is not guaranteed, unless special assumptions can be made regarding a periodic refreshment of signature keys by trusted parties interacting with the SCPU at key points in time.

a) Migration: In long-lived data scenarios, it is important to enable the **migration of data** to newer hardware and infrastructures while preserving regulation specific assurances. Mechanisms need to be devised to allow the secure transfer of secure WORM-related state maintained by the SCPU (together with the underlying data) to a new data store, under the control of its untrusted operator. We propose a method that minimally involves regulatory authorities yet preserves full security assurances in the migration process. The main challenges are related to the creation of a secure trust chain spanning untrusted principals and networks. Specifically, the original SCPU

($SCPU_1$) should be provided assurances that the migration target environment ($SCPU_2$) is secure and endorsed by the relevant regulatory authority (RA).

To achieve this, the migration process is initiated by (i) the system operator retrieving a migration certificate (MC) from the RA. The MC is in effect a signature on a message containing the timestamped identities of $SCPU_1$ and $SCPU_2$. Upon migration, (ii) the MC is presented to $SCPU_1$ (and possibly $SCPU_2$), who authenticates the signature of the RA. If this succeeds, $SCPU_1$ is ready to (iii) mutually authenticate and perform a key exchange with $SCPU_2$, using their internally stored key pairs and certificates. $SCPU_2$ will need backwards-compatible authentication capabilities, as the default authentication mechanisms of $SCPU_2$ may be unknown to $SCPU_1$. This backwards compatibility is relatively easy to achieve as long as the participating certificate authorities (i.e., SCPU manufacturer or delegates thereof) still exist and have not been compromised yet. A cross-certification chain can be set up between the old and the new certification authority root certificates. Once (iii) succeeds, $SCPU_1$ will be ready and willing to transfer WORM and indexing state on a secure channel provided by an agreed-upon symmetric key (e.g., using a Diffie-Hellman variant). After the secure state migration is performed, main data records can be transferred by the main CPUs directly.

The migration process is controlled by an externally run user-land *Compliant Migration Manager* (CMM). The CMM is configured to interact with the RA and the certificate authorities, create the communication channels between the data migration source and target systems, and perform and monitor the raw data transfer between data stores once the inter-SCPU interaction is completed.

C. Optimizations

1) Limiting SCPU hashing overhead: In the process of creating a VR's `datasig` signature, the SCPU is required to read and hash the data records associated with the VR. As hinted at in section IV-B.3, to support higher burst-periods throughputs, we propose to reduce this overhead while only minimally impacting the adversarial model assumptions (section II-A). Specifically, in the present model, the user Alice is trusted to accurately provide the data to be stored – *only later does Alice regret the storage of certain records*. Accordingly, we will extend this assumption by trusting the main CPU (during high-load periods when no SCPU cycles are available) to accurately compute (on behalf of the SCPU) the data hash required in `datasig`. We do not trust blindly however. The SCPU verifies this trust assumption by re-computing and checking these hash values during lower-load times (e.g., when the update burst is over) or after a certain pre-defined timeout.

This extension does not weaken the WORM defenses significantly, because providing an incorrect hash will be detected immediately upon verification, and the window of time between record commitment and hash verification can be kept insignificant in comparison to typical year-long retention rates. In Section V we illustrate the

performance gains of deploying this scheme.

2) Deferring strong constructs: A second throughput-optimizing method we deploy is to temporarily defer *expensive* witnessing operations (e.g., 1024-bit signatures) with less expensive temporary *short-term secure* variants (e.g., on 512-bit). This is particularly important during update burst periods. The short-lived signatures will then be strengthened (e.g., by resigning with strong permanent keys) during decreased load periods – *but within their security lifetime*. In effect this optimization amortizes SCPU loads over time and thus gracefully handles high-load update bursts.

We use 512-bit RSA signatures as a reference security lower-bound baseline. 512-bit composites could be factored with several hundred computers in about 2 months around year 2000 [81]. However, despite numerous efforts [2, 21–24, 27, 52, 80] no significant progress has been made [28] beyond the Number Field Sieve (NFS) [72] techniques.

To be on the safe side, we assume that today, 512-bit composites can resist no more than a few *tens of minutes* (e.g., 60-180 mins) to factoring attempts by Alice, who may want to do so in order to alter the `metasig` and `datasig` fields. In the WORM adversarial model however, this can only rarely be of concern, as Alice is unlikely to regret record storage and succeed in breaking the signatures in such a short time.

The intuition then is to deploy fast shorter-lived signatures during burst periods to support high transaction rates. To achieve an adaptive behavior, optimally balancing the performance-security trade-off, we need to determine the maximum signature strength we can afford (e.g., bit-length of key) for a given throughput update rate. The main idea here is to understand how much faster a signature of x bits is, given as known baseline the time taken by an n bit signature. This is not difficult, however space constraints prevent further elaboration.

HMACs. We note that an even faster alternative is to replace short-lived signatures with simple and fast keyed message authentication codes (e.g., HMACs). This would practically remove any authentication bottlenecks during burst periods, thus allowing practically unlimited throughputs at levels only restricted by the SCPU – main memory bus speeds (e.g., $100 - 1000MB/s$). The only drawback of this method is the inability of clients to verify any of the HMACed committed records until they are effectively signed by the SCPU. We believe that in a production environment such HMACs will be the prevalent design choice.

3) Efficient Record Expiration Support Structures.: As discussed in Section IV-B, to ensure timely deletion of expired records, a sorted list of SNs for records in order of their expiration times is maintained in a special linear data structure (VEXP) inside the SCPU. Naturally, due to memory limitations, the VEXP may not hold the SNs for the whole database.

So far we considered a solution in which the VEXP is sufficiently large to keep up with the data specific regulation-mandated expiration rates. In Section IV-B.3 we proposed that the VEXP is updated with fresh entries

from the VRDT in times of light load (a scan of the VRDT is required to do so). We call this (1) the "VEXP" solution. While we believe this is a reasonable assumption – especially given the year-long retention periods that are usually mandated – we would like to also explore what happens in situations when the expiration rate is high enough to deplete the VEXP data structure before "light load" times come around.

Specifically, when depleted, the SCPU will have to suspend other jobs and scan the VRDT to replenish the VEXP. But linear scanning of the VRDT may be expensive due to the fact that records do not appear in the order of their expiration times. Thus, additional solutions are required to enforce more efficient deletion mechanisms.

In addition to updating the VEXP during light load periods, we propose two alternative solutions. One idea consists of maintaining an authenticated B-Tree index (in un-trusted storage) – instead of a SCPU-internal, limited size VEXP structure – sorting the entries in the VRDT in their increasing expiration times. The retention monitor (RM) running inside the SCPU will simply check the B-Tree to determine the records that are to be expired next. The B-Tree will be updated in the write operation at the same time as the VRDT. It will be authenticated by simply maintaining a hash-tree on top of it, enforcing its authenticity and structural integrity assurances as in the verifiable B-trees of [68]. Thus, when the VEXP empties, the SCPU can replenish it with a sorted list of SNs by just reading in the corresponding B-Tree leaves. We call this (2) the "pre-sorted" expiration handling solution.

Further, instead of updating the B-Tree for every record insertion, an update buffer can be deployed to reduce the update overhead during bursts. The intuition behind using a buffer is to amortize the cost for each update by buffering the insertions and committing them to the B-Tree in batches. Specifically, the buffer is used to cache the incoming write updates (to avoid the direct B-Tree update cost in real time). Then, periodically, the elements in the buffer are inserted in the B-Tree by bulk-loading. This is likely to yield significant benefits because a majority of incoming records are likely to not be expiring anytime soon, thus buffering wait-times are not a problem. Ultimately, the advantage of using a buffer is obtaining high instantaneous throughput in insertion burst periods while keeping the amortized performance roughly the same as the pre-sorted solution. To authenticate the buffer, a simple signed cryptographic hash chain checksum [78] is deployed that enables the SCPU to verify the buffer's integrity upon read. This is important to prevent the server from surreptitiously removing entries from the buffer before the SCPU had a chance to empty it into the B-tree by bulk-loading. We call this (3) the "pre-sorted with buffering" solution.

V. Evaluation

The introduced architecture naturally satisfies important WORM assurances: data integrity and non-repudiation.

Theorem 1: Data records committed to WORM storage can not be altered or removed undetected. *(data integrity)*

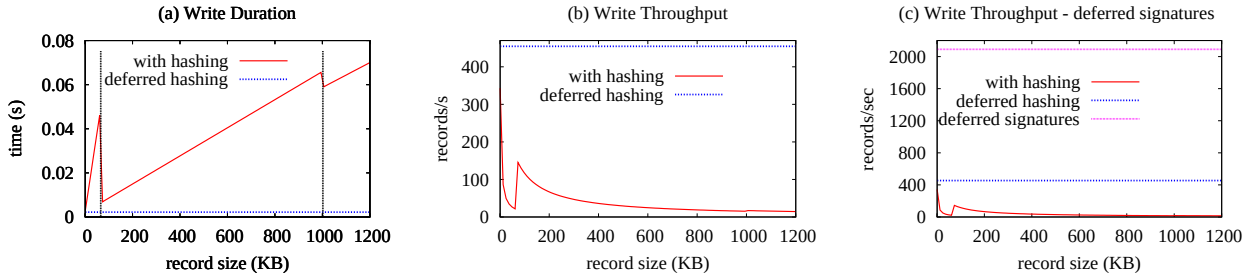

Fig. 8. (a) Write time variation with record size. The partially-linear time variation is due to hashing and input transfer speed. The optimization method with a deferred data hashing step (section IV-C.1) results in a 2ms constant update time regardless of record size. (b) Throughput variation with record size. Up to 350 updates/sec can be supported for smaller records. By deferring the data hashing we obtain a constant throughput of about 400-500 updates/second. (c) Throughput variation with record size (using the deferred strong constructs optimization). With deferred signatures, the improvement is significant, reaching 2000-2500 records/s.

Proof: Any adversarial attempt to delete or modify the data will be detected, since all data modifications are witnessed by the SCPU and signed for securely. The proof then reduces directly to the un-forgeability of the deployed signatures and the non-invertible, collision-free nature of the hashes. ■

Theorem 2: Insiders with super-user powers are unable to "hide" active data records from querying clients by claiming they have expired or were not stored in the first place. *(non-repudiation)*

Proof: Any claim of deletion need to be accompanied by a proof thereof. This proof is a strong, unforgeable signature that can only be generated by the SCPU at record expiration. Claiming previously committed records have not been actually stored is prevented by the (consecutive) monotonicity of the SNs. ■

Performance Upper-Bounds. We consider a single-CPU/SCPU system setup consisting of a unsecured main CPU (P4 @ 3.4 Ghz) and the IBM 4764-001 PCI-X Cryptographic Coprocessor [6]. In table IV we introduce several of the key performance elements of both the SCPU and the P4. We chose not to discuss here main CPU and storage I/O costs which are instance specific and do not pertain to the WORM layer. We aim mainly to determine the maximum supported transaction rates, in the presence of update witnessing by the SCPU. Specifically, we are concerned with the overheads introduced by SCPU data hashing, and the `metasig` and `datasig` signatures. For `datasig` we have:

$$T_{datasig}(x) = T_{h_d}(x) + T_{s_d} + T_{in_d}(x) + T_{out_d} \quad (1)$$

where x is the size of the data records, $T_{h_d}(x)$ represents the hashing time, T_{s_d} is the SCPU signature time, and $T_{in_d}(x)$ represents the transfer time for the inbound data in the hashing process. The overheads associated with `metasig` consist mainly in a SCPU signature on the SN and `attr` fields (<1KB in size) – we will approximate the $T_{metasig}(x)$ value with the SCPU signature time. We denote $T(x) = T_{datasig}(x) + T_{metasig}(x)$.

In Figure 8 (a) a plot of T(x) is presented for the considered hardware. Due to the hardware nature of the SHA-1 hashing engine we encountered a partially-linear variation of writing time, starting at approximately 3ms for small

Function	Context	IBM 4764	P4 @ 3.4Ghz
RSA sig.	512 bits	4200/s (est.)	1315/s
	1024 bits	848/s	261/s
	2048 bits	316-470/s	43/s
RSA verif.	512 bits	6200/s (est.)	16000/s
	1024 bits	1157-1242/s	5324/s
	2048 bits	976-1087/s	1613/s
SHA-1	1KB blk.	1.42 MB/s	80 MB/s
	64 KB blk.	18.6 MB/s	120+ MB/s
	1 MB blk.	21-24 MB/s	
DMA xfer	end-to-end	75-90 MB/s	1+ GB/s
CPU freq		266MHz	3400Mhz
RAM		16-32MB	2-4GB

TABLE IV

HARDWARE PERFORMANCE OVERVIEW. SCPUS (E.G., IBM 4764-001 PCI-X) ARE ORDERS OF MAGNITUDE SLOWER FOR GENERAL PURPOSE COMPUTATION THAN MAIN CPUS (PENTIUM 4, 3.4GHZ, OPENSSL 0.9.7F). ON THE OTHER HAND, THE CRYPTO ACCELERATION IN THE SCPU SHOWS IN DIRECT SPEEDUP OF CRYPTO OPERATIONS. ALSO OPTIMIZED KEY SETUPS MIGHT RESULT IN SLIGHTLY DIFFERENT NUMBERS FOR THE MAIN CPU.

records of a few KB (300 records/second). The two thresholds at 64 KB and 1 MB-records mark improvements obtained by hashing larger blocks of data. Specifically, the hashed block size is increased from 1024 bits to 64 KB and from 64 KB to 1 MB respectively (see table IV). The figure also depicts the writing time for the optimization method where SCPU hashing costs are deferred (see section IV-C.1). In this case, each write takes no more than 2ms/record (500 records/second). Figure 8 (b) shows throughput as a function of record size.

In Figure 8 (c) it can be seen that the deferred strong constructs optimization (section IV-C.2) yields significant throughput increases. With 512-bit signatures, burst update rates of over 2000 - 2500 records/second can be sustained for 60-180 minutes (the life-time of the short-lived constructs).

As the SCPU is not involved in **reads**, the only WORM-related overhead there is constituted by the optional records signature verification. We note that for normal operation this should not be an issue, as there is no reason why Alice should not trust the data store to provide accurate data, or with integrity ensured through cheaper constructs like simple MACs. However, WORM assurances at read time will likely be mandated in auditing scenarios when regulatory parties (e.g., federal investigators) are performing in-house audits. In that case the investigator's clients' hardware, likely a commercial x86-level CPU will handle the verification of WORM-related VRDT signatures. Given the figures outlined in table IV, a throughput of over 2500-2600 verified reads per second can be sustained.

In summary, the WORM layer (in a single-SCPU setting) can support per-second update rates of 450-500 in sustained mode, 2000-2500 in bursts of no longer than 60-180 minutes and 2500 reads (sustained). By construction

these results naturally scale if multiple SCPUs are available.

For these throughputs it is likely that even for single-CPU (but especially for multi-CPU) systems, I/O seek and transfer overheads are likely to constitute the main operational bottle-necks (and not the WORM layer). Typical high-speed enterprise disks feature 3-4ms+ latencies for individual block disk access [88]. These times are twice the projected average SCPU overheads and can become dominant, especially when considering fragmentation and multi-block record accesses.

Expiration Cost Evaluation. In the following we explore the overheads introduced by the three record expiration handling mechanisms discussed in Section IV-C.3. Specifically, here we are mainly concerned with I/O costs as these are likely the main bottle-necks in accessing the externally-stored B-Tree. This is in contrast to the previous section in which we explored the upper bounds of the supported transaction rates. We start by elaborating on the costs of the three record expiration handling solutions.

(1) In the **VEXP solution**, the cost for an insertion is just the cost of the **write** operation $T(x)$ we analyzed above

$$T_{VEXP-ins}(x) = T(x) = T_{datasig}(x) + T_{metasig}(x)$$

When the VEXP is depleted, the SCPU has to linearly scan the VRDT. Thus the amortized cost for a deletion is

$$T_{VEXP-del}(x) = \frac{T_{scan}}{y}$$

$$T_{scan} = \frac{VRDTSize}{disk_{bw}} + \frac{VRDTSize * disk_{frag} * disk_{seek}}{disk_{blocksize}}$$

where T_{scan} is the time-cost of scanning the VRDT, $disk_{frag}$ is the disk fragmentation rate, $disk_{bw}$, $disk_{seek}$ represent the disk bandwidth and seek time respectively and y is the size of VEXP. For simplicity and illustration purposes we assume that the record size is the same as the disk block size ($disk_{blocksize}$). This would correspond to a deployment inside a block-level device stack.

(2) In the **pre-sorted solution**, every new record has to be inserted into the B-Tree as well (in addition to the VRDT). The cost for an insertion becomes

$$T_{presorted-ins} = T_{datasig} + T_{metasig} + T_{seek} + T_{trans} + T_{update}$$

where we denote by $TreeHeight$ the height of the B-Tree.

$$T_{seek} = disk_{seek} * TreeHeight$$

is the disk seek time for traveling from the B-Tree root to the leaf level to insert the new entry. The transfer time for reading in the corresponding data blocks is

$$T_{trans} = \frac{disk_{blocksize} * TreeHeight}{disk_{bw}}$$

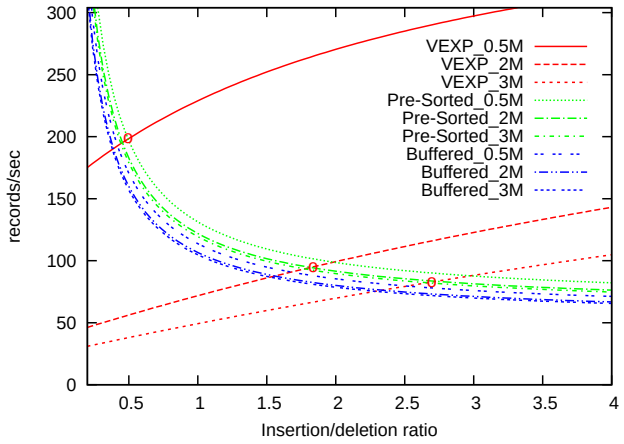

Fig. 9. Throughput variation (deferred hashing used) with database size and insertion/deletion ratio. Database sizes are 0.5M, 2M, 3M records respectively. The hardware parameters are taken from Table IV with 4KB block size, 0.1% disk fragmentation, 2 milliseconds disk seek time.

and

$$T_{update} = \frac{disk_{blocksize} * TreeHeight}{HashSpeed}$$

is the cost for updating the "verifiable" [68] portion of the B-Tree (which involves one hash computation per visited node), where $HashSpeed$ denotes the throughput of the deployed cryptographic hash function.

The cost of a deletion consists of the cost of reading in the sorted SNs from the B-Tree leaves (and then inserting them into the VEXP structure):

$$T_{pre-sorted\_del} = \frac{T_{trans\_list} + T_{update}}{y}$$

where T_{trans_list} is the time for populating the VEXP with the read SNs.

(3) For the pre-sorted with buffering solution, compared with the simple pre-sorted solution, there is an additional cost for maintaining the buffer. As a reminder, the buffer is used to cache the incoming write updates (to avoid the direct B-Tree update cost each time). Periodically, the elements in the buffer are inserted in the B-Tree by bulk-loading. The main cost component here lies in simply maintaining the chained hash checksum [78] that enables the SCPU to verify the buffer's integrity upon read.

$$T_{buffer-ins} = T_{presorted-ins} + T_{hash}$$

where T_{hash} is a constant time to recompute the new chained checksum for the newly inserted entry. The cost for a deletion is the same as the simple pre-sorted solution.

In Figure 9 we depict the impact of the above expiration handling solutions in the maximum supported throughput. The x-axis represents the ratio of record insert to (regulation-mandated) deletion rates. This effectively models the "growth" rate of the system. If the insertion rate is higher than the corresponding expiration rates, then the effective size of the data base is going to increase. The insertion/deletion rate ratio determines how fast this happens.

If the ratio is sub-unitary, the system effectively "empties". In this case, it can be seen that up to around a ration of 0.5, the pre-sorted methods do better than the VEXP solution. Between 0.5 and approx. 1.7, the VEXP mechanisms perform better but only for smaller database sizes (e.g., 0.5MB). On the other hand, for 2MB databases for example, the VEXP curve lies below the pre-sorted curves. Starting from a ratio of 1.7 onwards, the VEXP solutions start to out-perform the pre-sorted variants for database sizes of under 2MB. At a ratio of around 2.7 this holds also for database sizes over 3MB.

Naturally these data points are quite instance and parameter specific, yet the intent is to illustrate the overall behavior. As database size grows, the curves of the pre-sorted solutions mostly overlap, indicating that they are not influenced much. On the other hand, the performance of the VEXP solution are dropping largely. The reason for this is the increase in size of the VRDT, thus yielding more expensive scans thereof. Moreover it can be seen that as the insertion/deletion ratio increases, the throughput of the two pre-sorted solutions drops and the throughput of the VEXP solution increases. This is because pre-sorted solutions are paying more than the VEXP solution in insertions. In other words, the larger the ratio is, the more efficient it becomes to use the VEXP solution.

We note that the curves for the VEXP solutions and the pre-sorted variants intersect at certain points (depending on the corresponding database sizes). This suggests the deployment of an *adaptive solution* for different insertion/deletion ratios, choosing optimal expiration handling mechanisms. When the ratio is below certain thresholds (which can be regarded as the expiration burst periods) the pre-sorted solutions outperform the VEXP solution. As the ratio increases, VEXP features higher throughputs than the pre-sorted solutions. To prevent oscillations in the adaptive switching right at the threshold, hysteresis mechanisms can be deployed. Moreover, abrupt changes to the average insertion/deletion ratio are unlikely.

Note on buffering. Also, it seems that buffering does not help much in the pre-sorted buffered variant. Yet, although the average throughput of the simple pre-sorted solution is always higher than the pre-sorted with buffering, the later one can achieve higher instant throughput to serve insertion bursts. This might be essential in certain applications where burst availability is paramount.

Note on throughputs. It is important to note that the throughputs discussed here include I/O costs, unlike the throughputs discussed in the overall performance section where we chose to evaluate what potential transaction rates the WORM-specific cryptographic operations can keep up with – yet did not consider the system-specific I/O components. This is the reason why these values are lower than the theoretical upper-bound values in the general performance section.

VI. Conclusions

Recent compliance regulations are intended to foster and restore humans' trust in digital information records and, more broadly, in our businesses, hospitals, and educational enterprises. As increasing amounts of information are created and live digitally, compliance storage will be a vital tool in restoring this trust and ferreting out corruption and data abuse at all levels of society.

In this paper we first identified essential vulnerabilities in existing regulatory compliance systems. We outlined the requirement for strong mechanisms. We introduced a regulatory-compliant architecture offering strong Write

Once Read Many assurances by leveraging tamper-proof hardware. We have shown the architecture to handle throughput upper-bounds of over 2500 write transactions/second and unlimited reads.

In future research, it is important to explore traditional file system primitives layered on top of block-level WORM assurances.

REFERENCES

[1] Attacks on TPMs. Online at http://www.cs.dartmouth.edu/~pkilab/sparks/.

[2] TWIRL: The Weizmann Institute Relation Locator. Online at http://www.wisdom.weizmann.ac.il/~tromer/twirl/.

[3] IBM 4758 PCI Cryptographic Coprocessor. Online at http://www-03.ibm.com/security/cryptocards/pcicc/overview.shtml, 2006.

[4] IBM Common Cryptographic Architecture (CCA) API. Online at http://www-03.ibm.com/security/cryptocards//pcixcc/overcca.shtml, 2006.

[5] Trusted Computing Platforms Storage: Compliance, Security, and Policy. Online at https://www.trustedcomputinggroup.org/news/presentations/SNIA_Security_%Summit_2006.pdf, January 2006.

[6] IBM 4764 PCI-X Cryptographic Coprocessor. Online at http://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml, 2007.

[7] IBM Cryptographic Hardware. Online at http://www-03.ibm.com/security/products/, 2007.

[8] NIST Federal Information Processing Standards. Online at http://csrc.nist.gov/publications/fips/, 2007.

[9] Trusted Computing Group. Online at https://www.trustedcomputinggroup.org/, 2007.

[10] Trusted Platform Module (TPM) Specifications. Online at https://www.trustedcomputinggroup.org/specs/TPM, 2007.

[11] Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An Asymptotically Optimal Multiversion B-tree. *The VLDB Journal*, 5:264–275, 1996.

[12] Mihir Bellare and Daniele Micciancio. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In Walter Fumy, editor, *Advances in Cryptology — Proceedings of EuroCrypt*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192. Springer-Verlag, 11–15 May 1997.

[13] M. Blaze. A Cryptographic File System for Unix. In *Proceedings of the first ACM Conference on Computer and Communications Security*, pages 9–16, Fairfax, VA, 1993. ACM.

[14] G. Cattaneo, L. Catuogno, A. Del Sorbo, and P. Persiano. The Design and Implementation of a Transparent Cryptographic Filesystem for UNIX. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 245–252, Boston, MA, June 2001.

[15] Dwaine E. Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. Incremental multiset hash functions and their application to memory integrity checking. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 188–207. Springer, 2003.

[16] Protiviti Consulting. Frequently Asked Questions About J-SOX. Online at http://www.protiviti.jp/downloads/JSOXOverviewfinal_E.pdf, 2006.

[17] Premkumar T. Devanbu, Michael Gertz, Chip Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In *IFIP Workshop on Database Security*, pages 101–112, 2000.

[18] Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert van Doorn, Sean W. Smith, and Steve Weingart. Building the ibm 4758 secure coprocessor. *Computer*, 34(10):57–66, 2001.

[19] Malcolm C. Easton. Key-Sequence Data Sets on Indelible Storage. *IBM Journal of Research and Development*, May 1986.

[20] EMC. Centera Compliance Edition Plus. Online at http://www.emc.com/centera/ and http://www.mosaictech.com/pdf_docs/emc/centera.pdf, 2007.

[21] A.K. Lenstra et al. Analysis of Bernstein's Factorization Circuit. *Advances in Cryptology*, pages 1–26, 2002.

[22] J. Franke et al. SHARK: A Realizable Special Hardware Sieving Device for Factoring 1024-Bit Integers. In *Cryptographic Hardware and Embedded Systems*, 2005.

[23] W. Geiselmann et al. Scalable Hardware for Sparse Systems of Linear Equations, with Applications to Integer Factorization. In *Cryptographic Hardware and Embedded Systems CHES*, 2005.

[24] W. Geiselmann et al. A Simpler Sieving Device: Combining ECM and TWIRL. In *Intl. Conf. on Information Security and Cryptology*, 2006.

[25] Christos Faloutsos. *ACM Computing Surveys*, volume 17, chapter Access methods for text, pages 49–74. 1985.

[26] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley & Sons, 2003.

[27] W. Geiselmann and R. Steinwandt. Hardware for Solving Sparse Systems of Linear Equations over GF(2). In *Cryptographic Hardware and Embedded Systems CHES*, 2003.

[28] W. Geiselmann and R. Steinwandt. Special Purpose Hardware in Cryptanalysis: The Case of 1024-bit RSA. *IEEE Security and Privacy*, pages 63–66, January 2007.

[29] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29–43, Bolton Landing, NY, October 2003. ACM SIGOPS.

[30] E. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage, 2003.

[31] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[32] P. C. Gutmann. Secure filesystem (SFS) for DOS/Windows. `www.cs.auckland.ac.nz/~pgut001/sfs/index.html`, 1994.

[33] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing Secure Provenance. In *StorageSS*, 2007. Stony Brook Network Security and Applied Cryptography Lab TR 03-2007.

[34] J. S. Heidemann and G. J. Popek. File system development with stackable layers. *ACM Transactions on Computer Systems*, 12(1):58–89, February 1994.

[35] Hitachi Data Systems. The Message Archive for Compliance Solution, Data Retention Software Utility. Online at `http://www.hds.com/solutions/data_life_cycle_archiving/achievingregcomp%liance.html`, 2007.

[36] HP. WORM Data Protection Solutions. Online at `http://h18006.www1.hp.com/products/storageworks/wormdps/index.html`, 2007.

[37] W. Hsu and S. Ong. Fossilization: A Process for Establishing Truly Trustworthy Records. *IBM Research Report*, (10331), 2004.

[38] Lan Huang, Windsor W. Hsu, and Fengzhou Zheng. CIS: Content Immutable Storage for Trustworthy Record Keeping. In *Proceedings of the Conference on Mass Storage Systems and Technologies (MSST)*, 2006.

[39] HweeHwa Pang and Arpit Jain and Krithi Ramamritham and Kian-Lee Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *Proceedings of ACM SIGMOD*, 2005.

[40] IBM. IBM DS4200 Express Disk Storage System. Online at `http://www-03.ibm.com/servers/storage/disk/ds4000/ds4200/`, 2006.

[41] IBM. IBM System x 3755. Online at `http://www-03.ibm.com/systems/x/rack/x3755/`, 2006.

[42] IBM Corp. IBM System Storage N series with LockVault compliance software. Disk-based regulatory compliance solutions for unstructured data. Online at `http://www-03.ibm.com/systems/storage/network/software/lockvault/`, 2007.

[43] IBM Corp. IBM Total Storage Family: Tivoli System Storage Archive Manager. Online at `http://www-306.ibm.com/software/tivoli/products/storage-mgr-data-reten/%`, 2007.

[44] IBM Corp. IBM TotalStorage Enterprise. Online at `http://www-03.ibm.com/servers/storage/`, 2007.

[45] IBM Corporation and Daniel James Winarski and Kamal Emile Dimitri. United States Patent 6879454: Write-Once Read-Many Hard Disk Drive, 2005.

[46] Seagate Inc. Momentus 5400 Full Disk Encryption 2. Online at `http://www.seagate.com/docs/pdf/marketing/po_momentus_5400_fde.pdf`, 2006.

[47] Jetico, Inc. BestCrypt software home page. `www.jetico.com`, 2002.

[48] A. Kashyap, S. Patil, G. Sivathanu, and E. Zadok. I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System. In *Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004)*, pages 69–79, Atlanta, GA, November 2004. USENIX Association.

[49] Bernhard Kauer. OSLO: Improving the Security of Trusted Computing. In *USENIX Security Symposium*, 2007.

[50] G. Kim and E. Spafford. Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection. In *Proceedings of the Usenix System Administration, Networking and Security (SANS III)*, 1994.

[51] G. Kim and E. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *Proceedings of the 2nd ACM Conference on Computer Commuications and Society (CCS)*, November 1994.

[52] A.K. Lenstra and A. Shamir. Analysis and Optimization of the TWINKLE Factoring Device. In *EuroCrypt*, 2000.

[53] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure Untrusted Data Repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI 2004)*, pages 121–136, San Francisco, CA, December 2004. ACM SIGOPS.

[54] Maithili Narasimha and Gene Tsudik. Authentication of Outsourced Databases using Signature Aggregation and Chaining. In *Proceedings of DASFAA*, 2006.

[55] A. D. McDonald and M. G. Kuhn. StegFS: A Steganographic File System for Linux. In *Information Hiding*, pages 462–477, 1999.

[56] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[57] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Research in Security and Privacy*, 1980.

[58] Microsoft Research. Encrypting File System for Windows 2000. Technical report, Microsoft Corporation, July 1999. `www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp`.

[59] Ethan L. Miller, William E. Freeman, Darrell D. E. Long, and Benjamin C. Reed. Strong security for network-attached storage. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14, January 2002.

[60] Soumyadeb Mitra, Windsor W. Hsu, and Marianne Winslett. Trustworthy Keyword Search for Regulatory-Compliant Records Retention. In *Proceedings of VLDB*, 2006.

[61] Soumyadeb Mitra and Marianne Winslett. Secure Deletion from Inverted Indexes on Compliance Storage. In *Proceedings of the StorageSS Workshop*, 2006.

[62] K.-K. Muniswamy-Reddy and E. Zadok C.P. Wright, A. Himmer. A versatile and user-oriented versioning file system. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 115–128, 2004.

[63] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo I. Seltzer. Provenance-Aware Storage Systems. In *Proceedings of the USENIX Annual Technical Conference*, pages 43–56, 2006.

[64] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *ISOC Symposium on Network and Distributed Systems Security NDSS*, 2004.

[65] National Association of Insurance Commissioners. Graham-Leach-Bliley Act, 1999. `www.naic.org/GLBA`.

[66] Network Appliance Inc. SnapLock Compliance and SnapLock Enterprise Software. Online at `http://www.netapp.com/products/software/snaplock.html`, 2007.

[67] Ministry of Finance. Bill 198 of 2002. An Act to implement budget measures and other initiatives of the Government. Legislative Assembly of Ontario, 2002.

[68] HweeHwa Pang and Kian-Lee Tan. Authenticating query results in edge computing. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 560, Washington, DC, USA, 2004. IEEE Computer Society.

[69] British Parliament. Data protection act of 1998. Online at `http://www.staffs.ac.uk/legal/privacy/dp10rules/`, 1998.

[70] European Parliament. European directives. Online at `http://ec.europa.eu/justice\_home/fsj/privacy/law/index\_en.htm`, 2006.

[71] Zachary N.J. Peterson, Randal Burns, Giuseppe Ateniese, and Stephen Bono. Design and Implementation of Verifiable Audit Trails for a Versioning File System. In *Proceedings of the Conference on File and Storage Technolgoies (FAST), USENIX* , pages 93–106, 2007.

[72] C. Pomerance. A Tale of Two Sieves. *Notices of the ACM*, pages 1473–1485, December 1996.

[73] Quantum Inc. DLTSage Write Once Read Many Solution. Online at `http://www.quantum.com/Products/TapeDrives/DLT/SDLT600/DLTIce/Index.asp%x` and `http://www.quantum.com/pdf/DS00232.pdf`, 2007.

[74] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST 2002)*, pages 89–101, Monterey, CA, January 2002. USENIX Association.

[75] Peter Rathmann. Dynamic Data Structures on Optical Disks. In *1st International Conference on Data Engineering*, 1984.

[76] Robert McMillan and IDG News Service. Seagate Readies Secure Drive: Automatically encrypted Momentus is aimed at laptops containing sensitive data. Online at http://www.pcworld.com/article/id,127701-c,harddrives/article.html, 2006.

[77] Douglas J. Santry, Michael J. Feeley, Norman C. Hutchinson, and Alistair C. Veitch. Elephant: The file system that never forgets. In *Workshop on Hot Topics in Operating Systems*, pages 2–7, 1999.

[78] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley & Sons, 1996.

[79] Australian Securities and Exchange Commission. Clerp 9 corporate reporting and disclosure laws. Online at http://www.asic.gov.au, 2004.

[80] A. Shamir. Factoring Large Numbers with the TWINKLE Device. In *Cryptographic Hardware and Embedded Systems*, 1999.

[81] Robert D. Silverman. A cost-based security analysis of symmetric and asymmetric key lengths. Online at http://www.rsasecurity.com/rsalabs/bulletins/index.html. Note: Bulletin 13, 2000.

[82] G. Sivathanu, C. P. Wright, and E. Zadok. Enhancing File System Integrity Through Checksums. Technical Report FSL-04-04, Computer Science Department, Stony Brook University, May 2004. www.fsl.cs.sunysb.edu/docs/nc-checksum-tr/nc-checksum.pdf.

[83] S.W. Smith and S.H. Weingart. Building a High-Performance, Programmable Secure Coprocessor. *Computer Networks (Special Issue on Computer Network Security)*, 31:831–860, April 1999.

[84] StorageTek Inc. VolSafe secure tape-based write once read many (WORM) storage solution. Online at http://www.storagetek.com/, 2007.

[85] Sun Microsystems. Sun StorageTek Compliance Archiving Software. Online at http://www.sun.com/storagetek/management_software/data_protection/compl%iance_archiving/, 2007.

[86] Sun Microsystems. Sun StorageTek Compliance Archiving system and the Vignette Enterprise Content Management Suite (White Paper). Online at http://www.sun.com/storagetek/white-papers/Healthcare_Sun_NAS_Vignette_%EHR_080806_Final.pdf, 2007.

[87] T. Krijnen and L. G. L. T. Meertens. Making B-Trees Work for B.IW 219/83. The Mathematical Centre, Amsterdam, The Netherlands, 1983.

[88] Seagate Technology. Seagate Cheetah 15K.5. Online at http://www.seagate.com/www/en-us/products/servers/cheetah/cheetah_15k.5%/, 2007.

[89] The Enterprise Storage Group. Compliance: The effect on information management and the storage industry. Online at http://www.enterprisestoragegroup.com/, 2003.

[90] The U.S. Department of Defense. Directive 5015.2: DOD Records Management Program. Online at http://www.dtic.mil/whs/directives/corres/pdf/50152std_061902/p50152s.p%df, 2002.

[91] The U.S. Department of Education. 20 U.S.C. 1232g; 34 CFR Part 99:Family Educational Rights and Privacy Act (FERPA). Online at http://www.ed.gov/policy/gen/guid/fpco/ferpa, 1974.

[92] The U.S. Department of Health and Human Services Food and Drug Administration. 21 CFR Part 11: Electronic Records and Signature Regulations. Online at http://www.fda.gov/ora/compliance_ref/part11/FRs/background/pt11finr.pd%f, 1997.

[93] The U.S. Securities and Exchange Commission. Rule 17a-3&4, 17 CFR Part 240: Electronic Storage of Broker-Dealer Records. Online at http://edocket.access.gpo.gov/cfr_2002/aprqtr/17cfr240.17a-4.htm, 2003.

[94] U.S. Dept. of Health & Human Services. The Health Insurance Portability and Accountability Act (HIPAA), 1996. www.cms.gov/hipaa.

[95] U.S. Public Law 107-347. The E-Government Act, 2002.

[96] U.S. Public Law No. 107-204, 116 Stat. 745. The Public Company Accounting Reform and Investor Protection Act, 2002.

[97] C. P. Wright, M. Martino, and E. Zadok. NCryptfs: A Secure and Convenient Cryptographic File System. In *Proceedings of the Annual USENIX Technical Conference*, pages 197–210, San Antonio, TX, June 2003. USENIX Association.

[98] Zantaz Inc. The ZANTAZ Digital Safe Product Family. Online at http://www.zantaz.com/, 2007.

[99] Qingbo Zhu and Windsor W. Hsu. Fossilized index: the linchpin of trustworthy non-alterable electronic records. In *SIGMOD '05:*

Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pages 395–406, New York, NY, USA, 2005. ACM Press.