Toward Standardization of *Self-Encrypting Storage*

Security in Storage Workshop
Baltimore, September 25, 2008

Laszlo Hars
Laszlo.Hars@Seagate.com

Abstract For ensuring confidentiality and integrity of stored data in mass storage devices the least expensive secure solution is local data encryption, data authentication together with access control and physical tamper detection. This paper considers many aspects, like functionality, security, trust, testability for the most common such secure storage devices: self encrypting disk drives, which are already in production. Millions of secure disk drives are being deployed, so it is important that security professionals learn about their possible features, designs and security tradeoffs. Standardization of the used algorithms, interfaces and features would be beneficial for both customers and for manufacturers, because cost reduction, faster time to market, better trust can be realized if vendors architect their self encrypting drives in a standard way.

Table of Contents

1. Introduction

There are various means available for ensuring confidentiality and integrity of the data stored in mass storage devices, including physical protection or keeping data in a secure remote location, but the least expensive and highly secure storage systems employ local data encryption, data authentication together with access control and physical tamper detection. In this paper we concentrate on the features, advantages of self encrypting dives, and the need for their standardization. Other related aspects, like side channel attack resistance of the electronics (see e.g. [9], [11], [12] or [13]), the design of a cryptographic core, or the random number generator (see e.g. [14]), key management (see e.g. [7]), security related services, data leakage from imperfect magnetic media and other HW components, key generation and management, secure firmware update, interface specifications, etc. are only superficially discussed.

Samples of self encrypting disk drives have been available since early 2006, while mass production ramped up in 2007. A lot of such devices are being deployed first by laptop computer manufacturers, followed by desktop PC's and data center applications. Portable media players (embedded disks), TV broadcast video recorders represent another large market segment. Because millions of secure disk drives are being deployed, it is important that security professionals know about their possible features, advantages, limitations, designs and security.

Need for secure storage

There have been many recent cases of information getting into unauthorized hands from lost or stolen laptops or insiders accessing unattended enterprise computers or storage devices. The Privacy Rights Clearinghouse maintains a long list of such reported cases [18].

Physical protection and using remote locations are two means of keeping stored data confidential. The least expensive secure-storage systems use local data encryption with optional data authentication, together with access control and physical tamper detection. Such devices, self encrypting disk drives, are now being mass-produced. Manufacturers are deploying them by large numbers in laptops, desktop PCs, data-center applications, portable media players, and TV broadcast video recorders.

The IEEE P1619 Security in Storage Working Group [19] has developed standard architectures for external encryption modules and tape drives. However, there’s no standard yet for self encrypting sealed storage devices, where developers can adapt the data layout to security needs and provide access control to the encrypted data. An attacker can only see the ciphertext after disassembling the drive and examining the internal structures with multimillion-dollar equipment. Because of the attacks’ destructive nature, if the storage device is returned after an attack, the owner will notice the tampering and won’t trust the stored information. This effectively renders all kinds of data-modification attacks harmless.

Legislation

With loss of data beyond the critical level, federal and state governments are enacting legislation to protect those who are inadvertently put at risk. As of early 2008 over 35 states have enacted laws that require breach notification. However, informing customers of lost data is not required - if that data was encrypted. There are six significant federal U.S. bills under consideration that impact privacy:

- S 239 Notification of Risk to Personal Data Act of 2007
- HR 516 Federal Agency Data Privacy Protection Act
- HR 836 Cyber-Security Enhancement and Consumer Data Protection Act of 2007 (criminal law)
- HR 1685 Data Security Act of 2007
- S 495 Personal Data Privacy and Security Act of 2007
- S 1178 Identity Theft Prevention Act (Inouye)

These laws focus on also the protection requirements for storage devices. Implementing the right security metrics for data storage can avoid penalties and no actions required when data is lost. For example, HR 516 provides an exception for data that is not in usable format that includes data maintained or communicated in an encrypted form.

Existing federal laws already include encryption clauses. The Health Insurance Portability and Accountability Act of 1996 (HIPAA) that has radically changed data handling in the healthcare industry, provides technical safeguard for encryption to electronically protect personal health information

What can we do?

Physically protecting storage devices (guard, tamperproof enclosure, self destruction…) is expensive and not always feasible, like for portable devices. Cryptography provides less costly, simpler, and always available solutions. Encryption ensures privacy, secrecy; user authentication provides access control; data authentication can optionally protect from data changes, corruption.

Where to encrypt?

Data and keys have to be separated if the encrypted media are moved around, accessed from different encrypting devices (e.g. CD-ROM, Tape). However, in sealed storage the stored data and the encryption engine are not separated, allowing simpler key management with less cost, without the necessary secure infrastructure to manage: generate, store, transfer, load encryption keys. Keys could be generated and used inside the self encrypting storage devices. This alleviates many security risks, reduces costs.

There is only one key to manage outside the storage device: the user authentication key (password). This makes the trust boundary include only the sealed storage device, not the key management infrastructure.

Below we will elaborate also on other questions, like How to encrypt?

2. Scope of a Standard

There are storage standards for external encryption modules (IEEE P1619) and for tape drives and similar devices (IEEE P1619.1) [19]. They don’t address the needs of self encrypting storage,

although some aspects are the same. They address: Block (sector) oriented storage devices, with random access (addressable blocks).

A self encryption standard should cover Low level functionality, that is

- Encryption
- Authentication (user/data)
- Access control
- Key generation, store

High level functionality usually is not specific to self encrypting storage. We can often use TCG; Key management standards... Not considered higher level of functionality includes

- Interfaces, command format/payload
- Services (SP's, stash storage…)
- Authorization/rights management…
- Timer, logging, administration of other services…

3. Concerned Parties

Concerned parties could be categorized in three groups:

- Sealed storage manufacturers, including disk drives and solid state storage vendors
- Computer manufacturers, because they are the ones who order storage devices, design around and build them in to computers; they need equivalent second source, which is hard to achieve without a commonly accepted standard. They are also the ones who Interact with end users, and know their needs.
- End users, because their data is to be protected. They include
  - Data center operators, Healthcare providers, Banks, Insurance companies
  - Privacy groups
  - Government agencies…

A self encrypting storage standard needs wider audience, therefore it might not consider special needs, like what the military, intelligence agencies have.

4. Need for standards

A self encrypting storage standard provides significant advantages for the vendors of storage devices and their users:

1. Allows avoiding custom-design security architecture, and provides secure architecture, which already met public scrutiny
2. Simpler security analysis is enough for conforming devices, because the standardized features need not be covered
3. Reduces development costs, time to market
4. Users trust nonprofit organizations more than profit oriented manufacturers
5. Allow manufacturing drives of equivalent security and functionality by different vendors, providing second source of drives for OEMs
6. Following a standard provides marketing advantages: users perceive public standards as of good security

5. Threat Model, Attack Scenarios

An attack can try to access private user data, encryption or authentication keys, internal data structures of the device, its firmware, etc., or it could attempt to modify these, in the hope that these tampering would help a subsequent data theft. We can distinguish the attacks based on the frequency of the attackers' access to the drive:

- One time access: Stolen drive, lost laptop
- Intermittent access: Boarder guard making snapshot, hotel employee, another conference participant…
- Regular illegitimate access: Night guard, colleague at work…

We can also look at, what damage a specific attack can cause:

✓ Lost laptop / stolen (off-line) drive. The attacker can extract information from the drive (spin stand, redirect head signal to a second controller). In some cases 1-2 previous block contents can also accessed (analyzing magnetic saturation level, edge overlap patterns)
  - The attacker can perform a key search at known plaintext, like the MBR, Partition table, OS files, etc.
  - The attacker could try sending host interface messages, like authentication attempts, password trials, unauthorized data read/write, control commands
- Interrupted, tampered protocols (active, on-line attacks), to try to learn keys, reusable info (authenticate other partitions..)
- Users can also attack secrets in their own drive (on-line), when the data owner is different, like in the case of DRM protected files, online games, electronic cash…
  - Side channel attacks, fault injection…
- Spying hardware can be employed (on-line), including key loggers, cable snoopers, logic analyzers… We can distinguish cases between laptops (which are usually in sight when working) and drives in data centers (usually operating out of site of the personnel). The spying device can be inserted
  - Inside the host
  - On any point of the cable between the host, the storage controller, and the storage device (drive). It can attempt stealing data, attacking secure authentication protocols, etc.
  - Inside the storage device on wires, busses between components
- Malicious software could be installed in Host OS
  - Stealing small amount of data
  - Spying keys (key loggers, clipboard/control snoopers)
- Side channel leakage can be observed in Host
  - Resource usage monitoring (SW, HW)
- Side channel leakage can be observed in unauthenticated Drive
  - Radiation, heat, power fluctuation…
- External influences can be administered to the drive, like attempting fault injection in the internal computation
  - Changing magnetic field, EM radiation, supply voltage / internal resistance modulation, extreme temperature…

✓ After disassembling the storage device, an attacker can access the raw data on the storage media

- At most one time, because of the necessary destruction
- In some cases, 1-2 previous (latent) contents can be recovered at some data locations
- Ciphertext: encrypted user data. This enables the attacker
  - To perform traffic analysis (examining the patterns of data changes)
  - Actively change the data
    - bit flip: (almost) randomize plaintext ⇒ 1 bit info
      - many (>2^{64}) different plaintext versions at an LBA
    - arbitrary overwrite (data destruction, key search…)
    - copy other blocks over
    - copy back previous content (of the same or other block)
- Not only user data is subject to attack, but hidden system data

6. Goals

Self encrypting storage devices attempt to protect the confidentiality of the user data, encryption and authentication keys, and their own internal data and firmware.

- Data confidentiality to be provided by encryption, not by physical protection
- Access control has to be used to prevent
  - Ciphertext inspection
  - Tampering with ciphertext
- Some key/password management is needed
  - to provide different authorities to different users
  - to function without secret keys/passwords entered while the OS runs (authentication from the BIOS, from pre-boot environments)
- Allow fast secure erase of the stored information (sanitation before reuse)
- Breaking one disk drive (discovering its keys) should not help in breaking others
  - Could use Globally Unique Root Key in electronics, based on manufacturing process variations / one time programmable memory
- No backdoors, manufacturer keys should be employed
  - True random user keys generated in the drive at the users' request, as often as needed (with automatic internal re-encryption)
  - Secure key export could be provided, but only *when authorized*
    - functionality, correctness, entropy… to be verified
    - for data recovery when electronics fail
  - Key import could be provided, but only *when authorized* (accepting the associated security risks)
    - Recovery of plaintext from (failed) drive must be hard (except with escrowed keys)
  - Industry standard crypto algorithms have to be used: AES, RSA, ECC, DH…
  - The security architecture must be published, and made verifiable
- Transparent data layout alterations can be employed, e.g. reserving hidden space out of bounds, between sectors, in system area; or combining, splitting sectors, etc.
- Can use many forms of metadata (not considered in the IEEE P1619 standards). These include LBA, Age of data, Drive ID, Track geometry, Error Correction Codes...

- Security has to be provided, even if the (meta) data layout on media is known to an attacker. Not enough information should be stored in the storage device, which allows decrypting the user data (except brute-force key search, breaking cipher), when discovered
- Commercial grade security is provided, not military grade, not for top secrets
  - Physical attacks must fail ***before*** authentication
    - including side channels, etched away chip covers: microscope, probes…
  - Physical attacks might succeed ***after*** authentication
    - stealing powered up drive: data lost (could be prevented with secure networking)
    - side-channel attacks on drive in use (could be hindered by secure HW design, to be covered by a secure HW standard)
    - focused ion beams, micro probes (could be prevented by expensive tamper proof enclosures)
  - Unauthenticated ciphertext access should only be possible after disassembling the drive. Users must be able to detect this (by tamper detectors, noticing the inaccessibility of missing drives…)

7. Alternatives of Self-Encrypting Storage

There are many implementation possibilities for secure storage. One can use physical protection, remote locations, or cryptographic means. Encryption can performed in different places in the data path between the application SW and the storage medium.

- Encryption in the HOST (with dumb drive)
  - Provides some (insufficient) protection of the data in transit (security constraints are different for long term and transient storage)
  - LBA (data block address) is in the clear
  - Ciphertext freely available (see below)
  - Open physical environment
    - Key loggers
    - Logic/bus analyzer
    - Frozen RAM
    - DMA: FireWire (IEEE 1394), Peripheral buses
    - PCI(e/x...) bus monitor card
    - Side channels
  - Open SW environment
    - Malware
    - Net vulnerabilities
    - Debuggers, unprotected memory
    - DMA (FireWire, Disk commands, Video cards)
    - SW side channel leakage (resource use, latency...)
    - Virtual-memory/swap-to-disk, hibernation file
    - User errors
- Encryption in the application (SW)
  - Knows its protection needs, but it runs in unknown environment
    - Capabilities, Vulnerabilities of the OS, the HW
    - Memory cached, swapped to disk, to hibernation file
    - Deleted sectors erased or marked free
    - Indexing services

        - Recent files lists…
    - The host vulnerabilities all still apply
- HW Encryption *outside* of storage
        - Ciphertext available
        - LBA is in the clear (see below)
        - Need: Long lived keys for storage ⇔ session keys for transit
            - Storage encryption for transit: Security risk (fixed keys)
            - Network security for storage
                » Need key management for ephemeral keys: Cost, Complexity
                » Security risk: Data and key are separated (needs tracking)
    - HW Encryption in the Host
        - Crypto coprocessor
        - Crypto microprocessor extensions (Intel AES, VIA…)
        - TPM (secure key storage…)
    - External encryption module (P1619.0)
    - Encrypting disk controller

- **Problems** when the *LBA is in the clear* and the *Ciphertext is available*:
    - We had a high speed, mass encryption device (export / import control)
    - Traffic analysis (observation of data access patterns)
        - Internet cache, File system, Virtual memory, Hibernation file…
        - Response to seat reservation, database update/query…
    - Hiding data (from virus scanners) by temporarily moving it
    - CRC integrity manipulation with enough changes in one block
    - DRM data manipulation (e.g. number of times a video was viewed)
    - Data destruction at bit flip (selective small block, often remains undetected)
    - Data modification (malleability) with little collateral damage
        - Instruction-, address patching: 2^{-8}, 2^{-16} chance to jump to a desired place
        - Function corruption: (if no crash) critical security functions can be disabled
            - Key erase, Input validation, Protocol step/abort…
        - Changing behavior of malicious programs which test data blocks
        - Activation versions of documents, with embedded data for their selection

8. Advantages/Possible Features of Self-Encrypting Storage

Securing stored data is best at the closest point to the data, that is, inside the storage device. External crypto modules and software encryption are more expensive, when used with un-encrypting drives they leak information and offer more attack possibilities. Below we list a few of the numerous advantages of implementing cryptographic and access control functions inside the storage devices.

1. Low cost. Basic features can be incorporated in commercial pricing models
2. High security
    a) True random encryption keys: no weak user key can degrade security
    b) Keys are not stored in drive: no physical attack gets data in lost drives
    c) Keys do not leave the drive
        - No need for secure external key storage, key tracking, auditing
        - No accidental key mishandling (e.g. encrypting the key with itself)

d) Closed system: no malware infection
e) Crypto HW (single chip): no debugger, bus analyzer attacks
f) No SW side channel attacks *by malicious processes*

3. Protection from malicious host software
    a) Encryption keys are generated in drive, never leave the drive in the clear
    b) User authentication
        - Done with protected code in drive
        - Passwords are not stored (their iterated MAC could be stored, dependent on unique, secret HW root keys)
        - Authentication host code can be run from host BIOS (clean ROM) or attested pre-boot code
4. Fast and secure disk sanitation by erasing keys
    – With proof of erasure; even on (partially) failed drives!
5. Authentication key encrypts data-keys in drive
    – Instant password change (no user data re-encryption), for regular password change, mitigate breached password problems, at employee leaving…
6. Automatic protection of
    a) virtual memory (swap file)
    b) hibernation file
    c) boot record, partition table
    d) file cache, indices, recent files list…
7. User experience
    a) easy to deploy: always there, always active
    b) easy to setup: no SW or extra HW to install
    c) cannot mis-configure: all data encrypted (index, swap, hibernate...)
    d) automatically satisfy privacy laws, no need for data classification
    e) easy to use: transparent, no changes in activities, no learning curve
8. High speed encryption (at interface speed)
    – 0 host processor load, delay
9. Low power consumption: dedicated, optimized HW
10. Transparent: no need for HW or SW changes
11. Initial set up in the factory: operational, secure,
    a) Internally generated, true random secret keys
    b) Default passwords, ID's provided on paper
12. Different keys for different partitions
    a) Users, OS'es, applications are separated (sandboxes)
    b) Un-mounted partitions safe from
        i. Malware
        ii. User errors
        iii. Lunch-time attackers…
13. Pre-boot environments supported
    – A special MBR for cold boot, Swap MBR after authentication, then Warm boot
14. Support for Multi-level Authentication
    a) BIOS to open LBA band (~partition)
    b) (Pre-boot) OS: complex authentication
        i. Network access
        ii. Passwords, Biometrics
        iii. Tokens…
15. Support for third party security services

a) Virtual smart cards, dCards
b) Hidden, secure (stash) storage for keeping secrets even on authenticated (open) drives
    i. housekeeping info, integrity check for sensitive data, SW, drive ID
    ii. user passwords, accounts, sensitive personal info
    iii. data for digital rights management, electronic cash, game state…

16. Access control for stored data: ciphertext inaccessible
    – No data modifications, traffic analysis, snapshots
17. Strong authentication: number of failed logins are restricted (slow password dictionary attacks)
18. Authentication-key management, key hierarchy
    E.g.: enterprise key, master key, backup key…
19. Host communication can be (additionally) encrypted:
    protecting information flow in the "cable" (network: IPSec, TLS…)
20. On-line re-encryption: time shifted secure communication
    a) data is buffered
    b) encryption keys are negotiated at data access
21. DRM systems are helped, without high processor load or "dirty" tricks (rootkits)
    a) Clean design, Better system stability
    b) Secure computation in the drive (scripting)
    c) Secure, hidden storage
22. Security services
    a) secure, fast random number generator
    b) public key encryption, signatures for network applications
    c) key agreement protocols
    d) symmetric key encryption of bulk data (when allowed)
    e) secure authentication for third parties
    f) certificates with secure storage…
23. Storage can be tied to specific devices (mating):
    TV recorder HW, motherboard, controller, music-video players
24. Forensic logging, usage history, error logs are provided
25. Support for automatically closing a partition
    a) after a certain time
    b) after certain idle time
26. Support for disk arrays
    a) Unchanged SCSI Protection Info (routing ID)
    b) Valid error checking info (checksum, CRC)
    c) Internal XOR engine for RAID (on plaintext or ciphertext)
27. Support for background data services (on plaintext)
    By the Host or by the drive on opened partitions
    a) De-duplication
    b) Compression
    c) Indexing
    d) RAID rebuild
    e) Virus scanning, media fingerprints…
28. Open interface, easy support for all OS'es

8.1 Details Example: Hardware Root Key

Usually, in self encrypting storage devices there is some sensitive information, which has to be permanently saved. This includes password signatures (used for user authentication), encryption keys for host communication, etc. They are often stored on the main storage medium, and have to be encrypted with a secret key, which, of course, cannot be stored on the same medium.

This root key has to be of very high entropy, such that an exhaustive key search has little chance to discover it. One possibility was to externally generate it and store in the electronics (like in anti-fuse PROM). This method has technological difficulties (the need for a large charge-pump circuit) and security weaknesses:

- The root key generation and burn-in has to be performed in a supervised, secure environment
- There is a theoretical possibility for a back door, key escrow, or intentional weak key generation, so users have to fully trust the manufacturer. Operator errors, hardware malfunction or sabotage can cause serious consequences for a large number of drives.
- The ROM block is relatively easy to identify on the chip with a microscope, and the stored data (causing physical changes) can sometimes be discovered by visual observation.

There is a more secure and less expensive alternative: The Root Key can be derived from random manufacturing variations of the circuit layout, deviations of the substrate structure. See e.g. [8], [15] or [16]. If both inputs of a symmetric comparator are connected to the same input voltage, the logic level of the output will depend on random manufacturing and material variations of the circuit layout, which are not visible even with electron microscopes.

Existing memory cells can be used (or imitated by cross connected inverters). They tend to always settle to the same output level after power-on. It depends on random manufacturing and material variations, not easily seen by microscopic inspection.

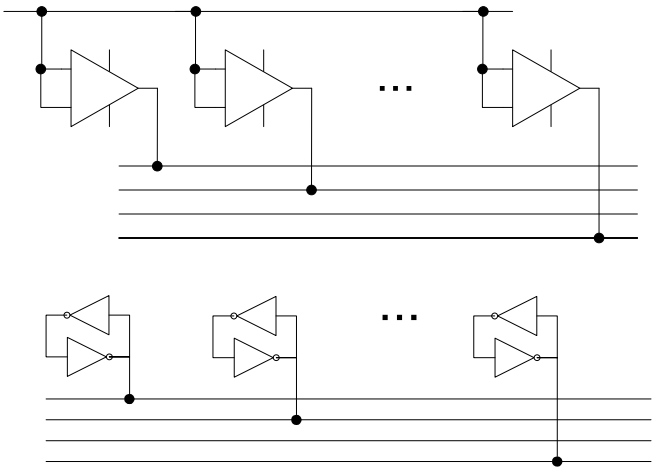

Figure 1. Raw key from circuit layout variations

The output of these blocks is effectively uniform random across different circuits. We will refer to them as "raw" key. Unfortunately, with time, temperature and supply voltage variation some bits may flip: a few of the comparators could have very little asymmetry, causing the

corresponding bits to become instable. Our experiments showed that at initial tests 0…17 of 256 bits were uncertain, and at most a couple of bits change later, during years of use in the field. One could compute and store on the storage medium an Error Correction Code of the resulting root key. It is complicated, and the error correction code leaks information about the root key. The following method is simpler and does not leak; therefore the resulting root key is of higher security.

At manufacturing the bits of the raw key are tested at all extreme conditions (4-corner tests with temperature and supply voltage). The *indices* of the ones, which do not change are recorded. The first 128…256 of them will be used as the root key. We don't know, however, if all those bits will stay constant as the circuit ages. If they don't, our root key could become corrupt. At standard manufacturing technologies there are no more than a *couple* of flipped bits (which became instable in the field). In this case we can still reliably recover the root key without error correction codes (and information leak):

- At factory setup read all the comparators hundreds of times (starting with cold, finishing with warm chip, each case with and without disk activities, low and high supply voltage).
- Build a table T of the indices of the stable bits, store it on the disk (replicated, digitally signed).
- If there are more than 20 unstable bits, declare the ASIC unsuitable for secure storage.
- Construct the root key from the first 224 stable bits.
- Calculate a cryptographic hash S (SHA-2) of the root key and store it alongside with the table T on the storage medium (replicated, digitally signed).

Each time the ASIC starts up, it reads the table T and hash S from the medium and assembles the root key from the comparator bits specified in T. If the hash of the newly constructed root key matches S, we can be reasonably sure, that it is correct. If they don't match, (one or two comparators changed their minds) we do the following:

- Flip all possible ways one or two (or even 3) of the used comparator bits (listed in T) until the hash of the generated root key match the stored one (S).
- Replace the indices corresponding to the newly found bad comparators with unused ones from the end of table T. If the corresponding bit value is different from the previously used bit in that position, mark the index (e.g. with a negative sign).
- Digitally sign the new table T and write it to the storage medium.

This way we can tolerate root key bits turning unreliable in the lifetime of the drive, as long as there is enough reserve and each time no more than 3 bits are affected. If more than a few bits flip at a time, the key search becomes slow, but still possible. Assuming 200 Mb/s hashing speed of the SHA-2 engine (with 512 bit blocks), 400,000 bit flips can be tested a second. The number of possible 1, 2 and 3 bit flips are 128, 16384 and 2,097,152 respectively. All of the possible 3-bit flips can be tried in 5 seconds (worst case), 2 bit flips are corrected in the average in 20 ms. However, these actions are rarely or never needed.

9. Possibilities with External Support

- Complex authentication
  - Multi-factor (with network access, biometric data, tokens...)
  - Pre-boot environments (duplicate OS functions)
- Communication (Interface/Network) security

  - Negotiate communication keys (key exchange)
  - Different exchange keys for multiple hosts talking to same drive
  - Encrypt command, LBA with session keys, nonces…
- Different authentication/encryption for different files (need OS support, because drives don't automatically have file system information, - except OSD)
- Re-authentication after spin down/up (BIOS, OS), while the computer is still powered up
- …

10. Trust: Open-source / Errors / Certification

There is a long debate if closed source security SW or HW could be trusted. Ideally, all the design details should be open, so everybody could verify the correctness, the lack of back doors, etc. In the practice this does not work.

- Open source SW *could* be verified, but it is *not*. Recent news:
  - 33 years old Unix bug (yacc) [22]
  - 25 years old BSD flaw (*dir() library) [27]
  - Debian Open SSL bug (wrong signatures) [28]
  - TLS bug [29]
  - Defeating Encrypted and Deniable File Systems [26]
  - 2nd worst: Joomla! open source content management system [25]
  - Fortify's Open Source Security Study: "most widely-used open source SW exposes users to significant unnecessary risks" [21]
    - 22,826 cross-site scripting and 15,612 SQL injection issues in versions of 11 SW packages
    - in two-thirds of the cases no contact or no response
- HW design is very hard to be verified (lag technology evolution)
- For protecting manufacturer's IP some details are kept secret
- Alternative: independent, trusted certifications

- HW (or SW) Bugs: hard to find (~Intel FDIV)
  - Data dependent faults (wrong algorithm, carry delay, load violation, clock/timing conflicts, meta-stability…)
- There is no way for a user to verify if a chip is manufactured according to a published design
- Intentionally planted faults are practically impossible to find
  - Enough if 1 out of $(2^{64})^2$ or more inputs rigged
- Single wrong output ⇒ broken encryption [30]

  ⇒ User must trust manufacturer, independent verifier

- Political trust issues:
  - Large companies / newcomers
  - Democratic / oppressive countries
  - Government agencies / nonprofit orgs / businesses
  - User groups / privacy advocates ⇐ political agendas

Some users want to be able to verify that a self encrypting storage device really encrypts their data, in the claimed encryption mode, with the right key. Allowing this opens up security weaknesses (the ciphertext becomes available – although controlled, the encryption key leaves the drive – making its tracking difficult and a key erasure unverifiable), and tells very little about the actual security (backdoors, predictable keys, rare faults will not be found). Consequently, it is better when verifications are done in controlled environments, by trusted entities, only.

11. Interfaces

The interface between the host computer and self encrypting storage devices is already standardized in TCG. The key management interfaces between the host and key servers are also (being) standardized in may standards bodies, like IEEE-P1619.3 [19].

- Only 2/4 extra commands are added to disk interface standards
- Payload carries arbitrarily many security commands
- Defined in TCG specs
- Could use key management standards

12. Encryption

The data is secured in disk drives by ***access control*** and ***encryption***. Several encryption modes are applicable for the secure disk drives. They include CBC, various counter mode with changing IV, location tweaked ECB mode, wide block encryption modes (see [1], [5] or [6]). The underlying cipher is uniformly AES in currently available secure storage.

- ECB leaks equality of blocks at (1-time) ciphertext access
- Large change granularity is desired: Small plaintext change ⇒ large ciphertext change
- If ciphertext is freely accessible: authenticate by data redundancy
  - Long block encryption: EME2, XCB, BitLocker…
- CBC with encrypted LBA as IV (to prevent watermarking attacks)
  - Link blocks in 1-direction
  - Ciphertext is one time accessible: No malleability weaknesses
  - For speed: Interleaved sub-chains are processed in parallel
- Counter mode, initialized with LBA
  - When previous block contents can be recovered: leaks changed bits
- Counter mode, started with: block-version counter || LBA

? MAC-Counter mode (≈ LH 02/2006 in P1619):
  - Asymmetric
    - plaintext change = total scramble
    - ciphertext change = limited effects (irrelevant at access controlled storage)

  $M = MAC(LBA, P_2, P_3…)$, and
  $C_1 = AES(P_1) \oplus M$
  C_1: initial count for Counter mode encryption of other blocks

13. Data integrity, authentication

Data authentication (like MAC for data communication) can be provided in secure storage devices for reasonable costs, and it can be offered as an optional security component.

When an adversary gains write access to the storage medium, he can cause damage, by writing back older contents of the same storage blocks, or by copying over data from other locations. Relying on the mechanical difficulty of dis/assembling disk drives and on physical tamper detectors, like (mechanical or electronic) seals on the storage device enclosure is not always satisfactory for *high security* applications. Secure disk drives can offer cryptographic means to detect unauthorized changes of the stored data with little degradation of performance, and give a warning to the user when his data has been tampered with. These data integrity checks rely on secret keys, which an unauthenticated attacker does not know and cannot access. These keys are associated with LBA ranges (disk partitions), and they can be different for each.

There are several alternatives:

1) A MAC on each LBA, stored separately. It wastes typically 16/512 ≈ 3% storage at 512 Byte disk blocks (0.4% at 4KB blocks), but it is fast to generate and verify. If the MAC uses different keys from the encryption, this method has the advantage that somebody could be granted the right to verify data, without enabling him to decrypt it. It also provides an additional error check, but does not protect against replay attacks (copying back older content to the same location together with its MAC), therefore it is not sufficiently secure.

2) Updateable (or incremental) MAC on many storage blocks. See details in [17]. At writing a single block of data, the MAC can be updated without reading the other blocks, which contributed to the MAC. At verification all conjoined blocks have to be read, but only once in a power cycle.

3) Merkle tree of MAC values of several blocks. See details in [2] and [3]. Similarly to the updateable MAC method, less memory is used than at LBA authentication, but more processing time at write and after boot up but *no read penalty* later.

Updateable MAC

Data authentication can be provided by "updateable", keyed hash, like GMAC in [17], or by the P1619 XTS encryption, when the ciphertext blocks are added together. They have the property, that at a change in the middle of the data, the signature can be updated, without re-processing the entire data set, but most data changes, including moving data blocks can be detected.

However, for verification of the integrity of the data, every block has to be read, which is a slow process, even though it has to be performed only once per session, after a successful user authentication (and before log-off, terminating the session). It is faster to maintain many MAC values, corresponding to smaller sets of data blocks. For example, for every track of a disk drive an updateable MAC could be maintained.

(a) When a data block is accessed, which has not yet been authenticated, the corresponding set of blocks gets authenticated by the firmware, and a warning is given the user at a failure.

(b) When the storage device is idle, it goes over all the stored MAC values as a background process and verifies them. The ones, which pass, are marked in a volatile table (erased at log-off), so subsequent accesses to them are instantaneous.

The drawback of one MAC per track is that an attacker can still replay the whole track, together with its MAC. This usually causes large changes in the file system and so software crashes, inconsistent data, so this replay attacks are often detected, but not always. For higher security more data have to be authenticated together.

Balanced Tree of MAC's

Another alternative is a static tree structure imposed on the data blocks. A node contains the MAC of the MAC's of its children nodes, at the lowest level, the MAC of the data in the corresponding storage blocks. At a read or write operation, only the MAC's on the path from the changed block to the root have to be verified or updated, respectively. This is similar to Merkle Trees (hash trees: [2]) for files, except our tree is static and its structure is optimized for fast access.

In the bottom all LBA's of a disk track are hashed together, because during one disk revolution all the track data can be read or written, regardless of the starting point. With fewer LBA's the disk access has to wait until the data arrives at the read/write head, and so the efficiency of the MAC computation is reduced. The MAC values can be stored in non-volatile memory or in a hidden partition of the disk (cached to RAM during processing).

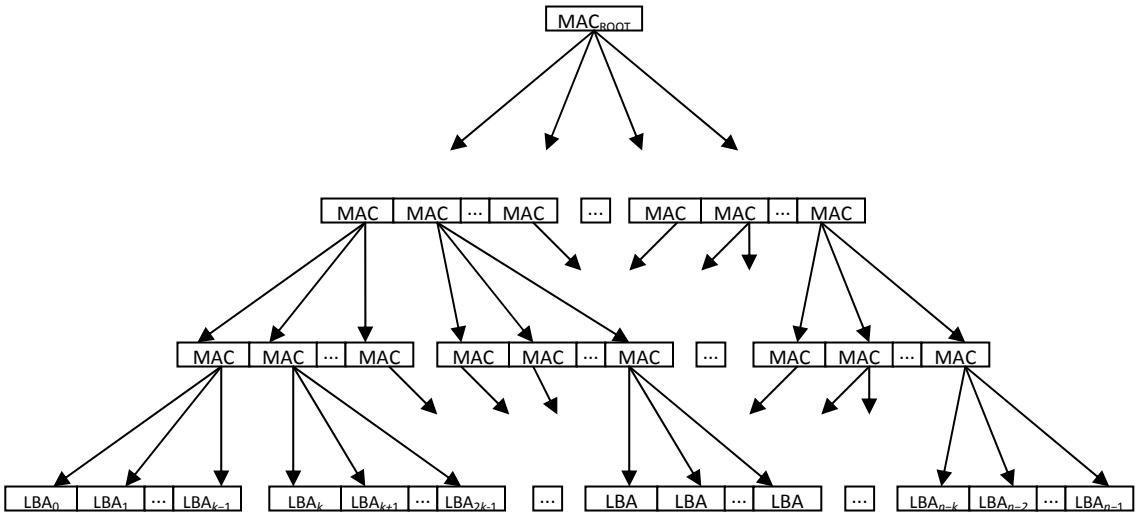

Figure 2. Balanced MAC tree

There is large flexibility in the tree structure. The first level of nodes could have 16 children, higher level nodes could have more or less. Assuming 2^{32} LBA's (17 TB with 4 KB blocks) and 16 children for every node, the path from an LBA to the root is of length 8.

At a block-write operation: all other data blocks of the track have to be read and processed, and on the path to the root node, 7 other MAC operations have to be performed (on cached data). It does need some extra work, but the overhead in processing time is manageable, because the data blocks processed together are next to each other, or they are cached in memory, and so they are accessed fast.

The storage overhead is also small in case of disk drives. 128-bit MAC values need little more than 16 byte storage for every 16 LBA's, that is one byte per LBA, or less than 0.2% loss of disk capacity with 512B blocks, 8 times less with 4KB blocks. Because of this moderate storage overhead the MAC values could even be stored in FLASH, or other non-volatile memory.

Because the MAC at the root of the tree depends on every LBA, the only replay attack possible is to restore the whole disk partition to an earlier state. If the root MAC is kept also in nonvolatile memory in the crypto ASIC, even this full restore attack fails, providing full protection against data alteration attacks.

14. Key Management

Data encryption keys can be formed by secure combinations (cryptographic hash, encryption) of different entities. They include

- a user key (a random, secret, normally, not even the user knows)
- the platter signature (unchangeable, unique physical fingerprint)
- the root key (securely stored in the electronics, globally unique)

The user key is randomly generated, but it is not stored in the drive, only its mixture with the user's password or ID. (The user password is not stored, either, only its signature for authentication, access control. For slowing down dictionary attacks, when this signature is leaked, iterated MAC could be used for the signature, depending on a secret HW key.) This way the encryption key is of high entropy even if the user password is weak. It is important: after erasing the user authentication information (disk erase, sanitation), a key search of the attacker has to deal with full key entropy. According to the estimates from [7], the first character of a typical user password contains 4 bits of randomness, and each subsequent character adds 2 more bits of randomness. This suggests that an average 8-character password only possess about 18 bits of randomness, which an attacker could brute-force search in roughly 2^{18} operations. This amount of processing is within the reach of any serious attacker.

There can be several keys, with different rights associated. Arbitrary hierarchy of these keys can be established, with higher level keys providing more rights. Since there are several passwords, the encryption key cannot be derived from any of them.

The drive can restrict the number of login attempts with invalid passwords. This way the negative effects of weak user passwords are mitigated, because a dictionary search aborts after a few unsuccessful attempts, locking up the drive. The key is true random, stored in a secure key-store, (different versions mixed with the corresponding passwords) so even a disassembled storage device is not susceptible to key guessing or to breaking in the key store. Only the master password has to be of high entropy, which unlocks a locked drive, where too many password guesses were tried.

There can be key export or import commands supported, where the key is wrapped in a secure cryptographic envelope. This requires keeping tracks of each key copy, storing them securely for high costs and complications, but helps at data recovery should the device electronics fail. Key escrow might be mandated by the local authorities in certain markets.

15. User Authentication

Users are authenticated by proving the knowledge of a secret or the possession of some unique object with secret information (token, fingerprint…). Authentication can be as simple as providing a valid password (could be used in cell phones), or more complex, such as challenge-response protocols with random or sequential nonces (against message replay-, or man-in-the-middle type attacks), multifactor-, biometric authentication, secure tokens, or accessing the up-to-date user credentials over a network. See more in [5]. Mutual authentication is easy to support, so user don't reveal their secrets to fake hard drives or the drive does not tell secrets to rogue hosts.

The number of failed authentication attempts can be restricted: for example, after entering 5 wrong passwords, the drive locks up, requiring higher level of authorities to unlock. This hinders an external trial-and-error search for weak user passwords. The most basic authentication structure uses a User- and Master password. The later is used, among others, to reset locked-up drives (after too many failed user authentication attempts).

16. Access Control

Without proper authentication the disk drive does not accept read/write commands, so an attacker cannot access the encrypted data, and gains no information about the locality of changes since an earlier snapshot.

Access control to the encrypted data improves security, and may be *required* by export/import control authorities in the country where the disk drive is manufactured and/or where it is used. (These regulations are constantly changing, and they are different for mass-market, inexpensive storage devices and for expensive systems.) If the encrypted data was freely accessible, a secure disk could be used as a standalone *mass encryption* device, which is controlled in some markets.

Absolute access control cannot be provided for reasonable costs. When the magnetic platters are removed from a disk drive, and placed on a spin stand, the encrypted data can be accessed. The storage media can also be connected to a rogue internal controller. However, this is only a one time access (although in some case a couple of the previous content of a storage block could be reconstructed, with very high costs and taking a lot of time and computation). By employing tamper detectors and because of the delicate mechanical construction, disk drives cannot be assembled back again for reasonable costs, without the legitimate owner detecting the tampering. In this case the owner will not trust this drive anymore, so an attacker cannot make a second snapshot on new data, encrypted with the same keys, nor can he cause damage by illicit changes of the ciphertext.

An example, when ciphertext modification attacks can cause catastrophic consequences is when the attacker knows or guesses where a certain program, document or the OS system files are stored (the OS files are usually stored in the beginning of the boot partition). With a bit flip some part of the executable code can be randomized. This could be in system functions, where parts are known to contain jump addresses. There is a non-negligible chance to alter it to a specific value (followed by random-looking bits), resulting in a predictable change of the behavior of the function.

17. FW Updates, Diagnostics, Testability

Storage devices, as other complex HW have to provide diagnostic modes of operation, with direct read-write access to the stored data. (This includes the SCSI "LongRead" command, which returns the raw data of a sector, together with error correction codes.) It is needed for initial testing in the factory and investigating defects for quality improvements. Diagnostic mode direct access defeats any access control, so it is mandatory to only allow it before the first key is activated (passwords are set), or when a special authentication is performed, with, for example, some dongle or other HW device. This should require the drive to be shipped to an authorized service center, and performed in a trusted environment. Still, the user must not trust his data after diagnostic mode access (without full data authentication), and have to receive some warnings.

Diagnostic *ports*, like the JTAG port allow diagnosing HW or SW problems on microprocessor controlled systems. Standard, off-the-shelf equipment can be used to directly control the connected microprocessor, reading/writing memory, peripheral ports or performing CPU commands, trace program execution, stop at breakpoints, etc. Such diagnostic ports are invaluable tools for searching for the cause of unexpected behavior of a system, therefore, we should keep them in the system and not permanently disable or remove them before shipping the drives to the customers.

On the other hand, diagnostic ports pose a security threat: an adversary can connect a debugger to it and access secrets in a secure system. Therefore, we have to control their use.

Standard solutions

They include fuses, which can be blown to permanently disable diagnostic ports after their last use in the factory; HW or SW authentication mechanisms (passwords, tokens, biometrics etc.); Special SW (FW) versions to be authenticated and loaded to open up the ports; Switches or jumpers to activate special resident FW versions, which open up the ports. The main drawbacks of these methods are:

- Activation of the diagnostic ports require the authentication to be fully functional, so problems affecting that functions cannot be diagnosed
- Permanently changed security systems (fuses) cannot be diagnosed again after the initial diagnosis, for investigating problems at later stages
- Opening up the ports have high security risks: all the user secrets stored in the system become available, which could not have been deleted because of a system failure
- Inconvenience: Confidential data need to be deleted before diagnosis and restored afterwards

Desired features:

- No secrets (user data, cryptographic keys) should be accessible via the diagnostic port
- No damage should be done to the users' data (no erasure of keys, other secrets)
- Low cost
- Low complexity
- Reliable operation
- Open design, easy verification for trust, liability

- Firmware code can be diagnosed from the very first instruction (boot-up diagnosis)
- Port de/activation should not involve any FW action (to enable debugging FW bugs)
- Real FW code used in the field is diagnosed, not special versions
- Security hardware functions should be accessible
- Non-security functions should not be affected
- Tamper evidence: some simple extra physical detection mechanism (glue, paint, tape, plastic connector, oxidizer, snap) has to indicate to the user that diagnosis has been activated

A Possible Diagnostic Port Protection Design

We assume that the system has a *root key*, as described above. It encrypts (one or more) *user key*(s), stored only in encrypted form in the nonvolatile memory of the system (disk, Flash, ROM, etc.). These user keys encrypt all the confidential information stored in the system. We also assume an unchangeable part of the FW (*ROM code*), which performs integrity and authenticity check on the rest of the FW (downloadable) and the nonvolatile storage.

The key component is a *switch*, which enables the diagnostic port. When it is activated while the system is already started, it will *reset* the system, erasing the content of the memory, key registers, etc. If the system is started with the switch activated, the diagnostic port gets opened with the *root key blocked*. All access to it results in all zeros, or another predetermined value. In this way no user secrets can be decrypted, but they are accessible.

This way the boot-up process, the firmware authentication and every part of the system can be diagnosed. Even though a new, unauthorized FW can be downloaded, the next time the system boots up in non-diagnostic mode, the ROM code will detect the rogue FW and *halts*.

The switch can be implemented as a *sensory circuit*, which detects if a connector is attached to the diagnostic port and activates the diagnostic port with all security features the system has. This way no extra action is needed to start the diagnosis, only a suitable cable has to be attached.

18. Summary

Many aspects of self-encrypting storage are worth standardizing because the storage industry and computer manufacturers, data center operators need standards and benefit from them and, because self encrypting storage has many advantages, like it is secure, simple to use, inexpensive, transparent, integrated, fast and of low power…

19. References

[1] S. Halevi and P. Rogaway. *A Tweakable Enciphering Mode*. http://eprint.iacr.org/2003/148

[2] R. Merkle, *Secrecy, authentication, and public key systems*, Ph.D. dissertation, Dept. of Electrical Engineering, Stanford Univ., 1979.

[3] M. Szydlo, *Merkle Tree Traversal in Log Space and Time*, Eurocrypt '04, available at http://www.szydlo.com/szydlo-loglog.pdf

[4] D. E. Knuth. *The Art of Computer Programming. Volume 2. Seminumerical Algorithms*. Addison-Wesley, 1981.

[5] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[6] C.H. Mayer and S. M. Matyas, *Cryptography: a New Dimension in Computer Security*, John Wiley & Sons, 1982.
[7] NIST Key Management Guidelines SP800-57. http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf and http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part2.pdf
[8] Pim Tuyls, Geert-Jan Schrijen, Boris Skoric, Jan van Geloven, Nynke Verhaegh and Rob Wolters (Philips Research Laboratories) *Read-Proof Hardware from Protective Coatings*, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2006), Yokohama, Japan
[9] Francois-Xavier Standaert, Eric Peeters, Cedric Archambeau and Jean-Jacques Quisquater, *Towards Security Limits of Side-Channel Attacks*, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2006), Yokohama, Japan
[10] Ran Canetti, Shai Halevi, and Michael Steiner. *Mitigating Dictionary Attacks on Password-Protected Local Storage*. Advances in Cryptology - CRYPTO '06. LNCS vol. 4117, pages 160-179. Springer-Verlag, 2006.
[11] M. Joye and P. Paillier and B. Schoenmakers, *On Second-Order Differential Power Analysis*, Workshop on Cryptographic Hardware and Embedded Systems CHES 2005, Yokohama, Japan
[12] Eric Peeters and Francois-Xavier Standaert and Nicolas Donckers and Jean-Jacques Quisquater: *Higher-Order Side-Channel Attacks: Concrete Results*, Workshop on Cryptographic Hardware and Embedded Systems CHES 2005, Yokohama, Japan
[13] Werner Schindler, Kerstin Lemke and Christof Paar: *A Stochastic Model for Differential Side Channel Cryptanalysis*, Workshop on Cryptographic Hardware and Embedded Systems CHES 2005, Yokohama, Japan
[14] M. Epstein, L. Hars, R. Krasinski, M. Rosner, H. Zheng: *Design and Implementation of a True Random Number Generator Based on Digital Circuit Artifacts*, Workshop on Cryptographic Hardware and Embedded Systems CHES 2003, Cologne, Germany (2003)
[15] Lofstrom, K.; Daasch, W.R.; Taylor, D. *IC identification circuit using device mismatch*, Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International, Volume , Issue, 2000 Page(s): 372 – 373
[16] Hirase, J.; Furukawa, T.: *Chip Identification using the Characteristic Dispersion of Transistor*, Test Symposium, 2005. Proceedings. 14th Asian Volume , Issue , 18-21 Dec. 2005 Page(s): 188 – 193
[17] David McGrew, *Efficient Authentication of large, dynamic data sets using Galois/Counter Mode (GCM)*, 3rd International IEEE Security in Storage Workshop, December 13, 2005. http://ieeeia.org/sisw/2005/PreProceedings/10.pdf
[18] http://privacyrights.org/ar/ChronDataBreaches.htm
[19] http://siswg.net/
[20] Maxwell Cooter, Open source could learn from Microsoft, http://www.techworld.com/security/news/index.cfm?newsid=102204
[21] Fortify's Open Source Security Study, http://www.fortify.com/l/oss/oss_report.html
[22] Matthew Broersma, Developer fixes 33-year-old Unix bug, http://www.techworld.com/security/news/index.cfm?newsid=102127
[23] Sumner Lemon, Intel chips open to attack, http://www.techworld.com/security/news/index.cfm?newsid=102143
[24] Robert McMillan, Encrypted hard drives may not be safe, http://www.techworld.com/security/news/index.cfm?newsid=102171
[25] IBM Internet Security Systems: X-Force 2008 Mid-Year Trend Statistics report. http://www-935.ibm.com/services/us/iss/xforce/midyearreport/xforce-midyear-report-2008.pdf
[26] A. Czeskis, D. J. St. Hilaire, K. Koscher, S. D. Gribble, T. Kohno, and B. Schneier: Defeating Encrypted and Deniable File Systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications, http://www.schneier.com/paper-truecrypt-dfs.html
[27] http://www.osnews.com/story/19731/The-25-Year-Old-UNIX-Bug
[28] *Hovav Shacham*: insecure.iacr.org, Crypto'08 Rump session
[29] *Hal Finney*: Looking over virtual shoulders, Crypto'08 Rump session
[30] *E.Biham, Y.Carmeli, A.Shamir*: Bug Attacks, Crypto'08