A Time Stamped Virtual WORM System

A. Apvrille[1] & J. Hughes[2]

1: Storage Technology European Operation,
1, Rd Point Général Eisenhower,
31106 Toulouse, France
Axelle_Apvrille@storagetek.com
2: Storage Technology Corp.
7600 Boone Avenue North,
Minneapolis, MN 55428, USA
jim@network.com

Abstract

When backup operators have to handle giga bytes of information daily*, they usually find the task hard enough to leave security aside. This paper consequently intends to focus on proving documents' authenticity regarding both content and date. In the context of WORM technology, it proposes a media independent tamper evident system - using original cryptographic hash function techniques - and secure time stamping within a given accuracy. Finally, time stamping upgrade mechanisms are proposed to extend security guarantees over years.*

1. Introduction

The increasing digitalization and computerization of business processes have led to a burst in data volumes. For instance, BNP Paribas faces a 150 Tera-Bytes backup weekly, Volkswagen roughly 100 Tera-Bytes daily... Dealing with such large amounts of data is far from being an easy task, but keeping that data over years *as legal evidence* adds up a significant challenge to the problem. As a matter of fact, several countries have recently agreed to the suitability of electronic records in court trials in respect to a few constraints [Esg00] [Adap00, §1316-1]. Commonly, electronic records are given the same legal value as any other handwritten evidence, *provided they are considered authentic by the court*.

Typically, banks, insurances or Intellectual Property archive centers are concerned by the possibility of reproducing some former document in front of a court. For such organizations there is a clear need for a *trustworthy archival system* whose integrity and reliability (over years and volume) could not be argued with. More precisely, the requirements of such a system are *data security* (protecting against or detecting any alteration of data), *longevity* (dating documents precisely, and retrieving them after years of storage) and *performances* (being able to process thousands of records without significantly downgrading performances).

The traditional long-term electronic document storage media is optical disc [Afn01], commonly referred to as "WORM" (Write Once Read Many). But, actually, the WORM technology widens to *any* reliable system (a media, some piece of hardware equipment, software etc.) onto which data is written only once, and is trusted not to be modified: this is perfectly adapted to data integrity requirements. Unfortunately, regarding specific constraints of long term record archival, existing WORM technologies put a few security issues aside. This paper therefore intends to propose a *new* kind of WORM system - a *time stamped virtual WORM system* - to improve those points.

The paper is organized as follows. Existing WORM systems are described and evaluated in section 2. Then, we propose and explain the mechanisms of our time stamped virtual WORM system in section 3, and analyze the improvements which have been made in section 4. Finally, section 5 deals with how time stamped virtual WORM systems can offer strong security features throughout years and future work to be done.

2. WORM devices

In this section, we first give an overview of existing WORM technologies. Then, we define precisely the threat model which is considered. Finally, we evaluate existing technologies with that threat model.

2.1. Overview of existing WORMs

WORM is a technology designed for permanent data records. Data may only be written once onto media and then becomes permanent (neither rewritable, nor erasable). On the contrary, read operations remain unlimited. Williams [Wil97] has made the following classification:

P-WORMs (Physical - Write Once Read Many) are the best known. Recording creates a permanent physical change in the surface of the support in such a way that this area cannot be restored to its original state. Security is located at media level (see figure 1). For instance, a CD-R is a P-WORM.

E-WORMs (Coded - Write Once Read Many) use a factory pre-recorded write once code located on the media itself. The code is pre-recorded by the manufacturer and then later recognized by the firmware that switches to an overwrite prevention mode. Note data may actually be recorded on a rewritable media. Security is at driver level (embedded code) and media level (pre-recorded code) - see figure 1. For instance, StorageTek's VolSafe$^{(TM)}$ [Abs00] technology is an E-WORM implementation.

S-WORMs (Software Write Once Read Many) use software to protect against overwriting (see figure 1). Consequently, an S-WORM is media-independent.

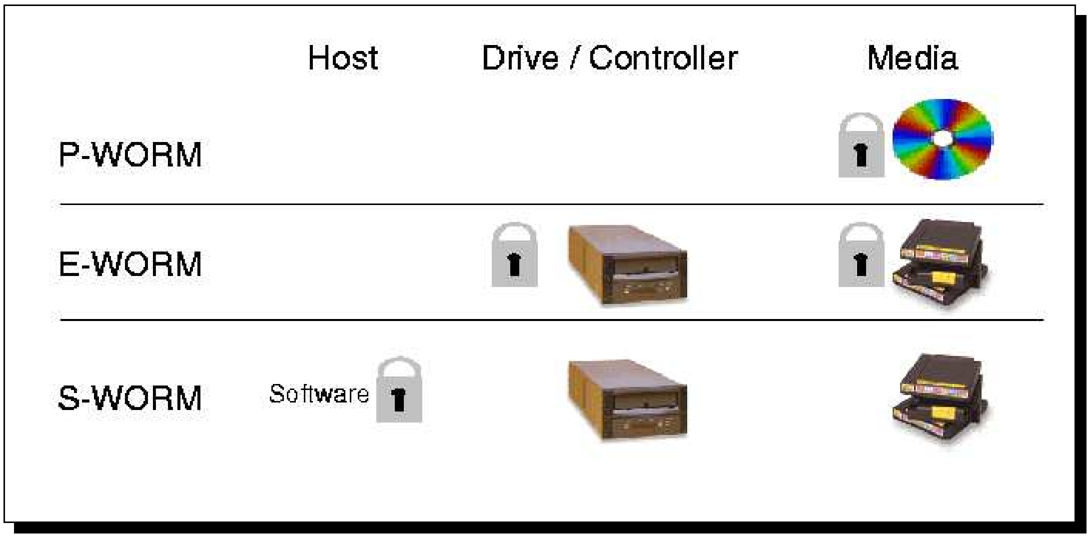

Figure 1: *Comparison between different types of WORMs. The lock on the figure indicates where security actually resides.*

2.2. Threat model

Basically, this paper is going to evaluate archival systems with a simple question: *given access to electronic records, is it possible to guarantee their authenticity over years ?*

More precisely, we suppose the attacker has physical access to stored data (for instance he can retrieve the tape cartridge or the disk which contains the data) and can use proper equipment for its manipulation. The purpose of this section is to make sure he cannot forge data *undetectably*.

A few possible vulnerabilities in this threat model have been detected:

- data integrity threat: attacker can possibly modify, truncate or erase data from the archival system. More precisely, there are two levels of security: protecting against tampering (making it "impossible" to modify) and detecting modifications (modifications are *possible* but will be *detected*).
- copy integrity threat: during a copy or a migration, attacker can possibly write something different onto the copy, or ignore data he does not wish to copy. For long-term archival systems, this threat is important as data is very likely to be migrated from one system to another one (because former system has become obsolete, or because it has expired...). To be sure document's authenticity will not be disputed, one needs to prove copied data is strictly identical to original.
- timing threat: attacker can successfully change dates of records. For many formal documents, date is an important information. For instance, insurance contracts, business agreements or buying orders should not be backdated. Attackers can use two different approaches: forging dates, or using clock accuracy problems. If an attacker can set up the date of an agreement so it has expired, he goes as far as altering the document's validity. Combined with data integrity threats, he can also forge a completely new buying contract and assign the date of his choice. Concerning time accuracy, requirements depends on security level to be obtained. At least, it is important to be able to provide a precision, and for instance state that clock is correct within a given accuracy.
- hardware support dependency threat: attacker takes advantage of the fact security information cannot be migrated to new support. This threat concerns the storage system's longevity. Suppose in a ten-year time backup operator would like to transfer all data onto a newer support that has better performances, or that he has chosen for other reasons. If the secure storage system depends on hardware support data is written to, then migrating data without losing security information might not be possible.
- bad authentication threat: attacker takes advantage of the fact document is unproperly authenticated to dispute its reliability. For instance, imagine a company has signed a partnership agreement with another company. If one of them wishes to prove the agreement existed, he'll have to reproduce the agreement signed by both of them. Actually, whether to sign or not to sign a document - and who should sign it - is a specific property of the document itself, and does not globally concern the storage system. Consequently, this paper will not address this issue, and will assume that this task is *already achieved before getting to the archival system*. The archival system will take signed or unsigned documents as input, whether this is required or not for each document.

2.3. Defeating existing WORM technologies

This paragraph intends to evaluate existing WORM technologies, according to the threat model established in §2.2. Results are summarized at table 1.

First of all, concerning data integrity, S-WORMs show an obvious security hole: basic S-WORMs provide no data integrity detection, and protection is poor as support itself is not protected. Any skilled user can perform an undetected record modification (or deletion) by simply by-passing the WORM software and using a less restrictive one. E-WORM's data integrity protection is not 100% sure, but is however more difficult to by-pass: attacker needs to ruin or reload the pre-recorded code. P-WORMs are relatively secure, as they are

Table 1: *Existing WORM features at a glance.*

	Data integrity protection	Data integrity detection	Copy integrity	Hardware independency	Secure time reference
P-WORM	yes (though not perfect)	no	bit-comparison (long)	no	no
E-WORM	yes, medium	no	“	no	no
Basic S-WORM	yes, but poor	no	“	yes	no
Required features	yes (or detection)	yes (or protection)	yes (optimized)	yes	yes

inherently non-rewritable. However, a closer analysis may reveal potential data integrity threats: for instance, a CD-R consists in a pattern of pits and lands that encode the information. Pits being permanent, it is impossible to re-write the disc with new information, however it is still possible to add new pits, and consequently slightly modify data on the CD-R.

Second, concerning time integrity, all existing WORMs unfortunately lack secure dating of documents. In best cases, referenced time merely gives a vague idea of document's creation date. For instance, on CD-Rs, when a file is written, CD File System also stores the creation date of the file on the media. However, there is absolutely *no guarantee of accuracy* for that date: it is provided “as is”.

Third, S-WORMs are inherently independent of any hardware support, which is a good point. On the contrary, P- and E-WORMs are not. If data stored on a CD-R is moved to a CD-RW (re-writable media), anybody can obviously overwrite that data. Same problem occurs on E-WORMs: if you transfer data onto another system which does not recognize the pre-recorded code, the system will not switch to a non-writable mode and it will be possible to modify, overwrite, truncate or erase data without any restriction.

Finally, checking copy integrity is possible (i.e. checking a copy is identical to the original), but with poor performances. For existing WORMs, there is basically no other solution than comparing each bit of data with the original.

2.4. Virtual WORM proposal

Actually, the whole problems boils down to the fact that existing WORM systems all pay attention to securing mechanisms that write information onto the media, but not to data itself. For instance, P-WORMs use physically non-rewritable non-erasable media. E-WORMs restrict and control writing operations. However, user data remains inherently unsecured.

Virtual WORM's basic idea is to focus on **data's security**, instead of its writing mechanisms[1]. Then, when data is taken in charge by the system's mechanisms and physically written, data is already secured.

Mainly, the solution we propose builds on S-WORMs. In §2.3, we have seen S-WORMs offer poor data integrity protection, and unfortunately, there's not much to do about that. So, with virtual WORMs, we suggest to improve data integrity *detection* techniques that are suitable for legal evidence documents. To do so, virtual WORMs make an extensive use of cryptographic mechanisms. Moreover, a signature-based time stamp protocol is added to virtual WORMs to provide a secure time reference functionality (see section 3).

[1] US-patent 2001-075-TOU “Virtual WORM method and system” pending.

Finally, we'll demonstrate the resulting time stamped virtual WORMs are independent of hardware support and optimize copy integrity (section 4).

3. A Time Stamped Virtual WORM System

In this section, we'll first explain how our time stamped virtual WORM system works, and then analyze its resistance to previously exposed threat model in §2.2.

3.1. General overview of the system

The system this paper proposes is based on a chain hashing technique (using one-way hash functions[MOV96, §9] over blocks of data) and on secure time stamping with digital signatures:

- step 1: chain hashing the input
  1. split data D in multiple *data blocks* D=$B_1 \dots B_n$,
  2. hash each block with H a one-way hash function: $H_1 = H(B_1)$ and $\forall i, 1 < i \leq n, H_i = H(H_{i-1}, B_i)$,
  3. and then store hashes along with blocks.
- step 2: secure time stamping
  1. time stamp the last block hash,
  2. digitally sign the time stamp,
  3. store time stamp and its signature.

3.2. Down into time stamping mechanism

Let us now have a closer look to the time stamping mechanism. From a generic point of view, [Roos99] defines a time stamp as a "token that binds information about *time* with the bit string". Actually, this definition does not make any assumption about security. However, in the context of this paper, time stamps are useless if time cannot be *certified within a given accuracy*. So, basically, the stamping protocol consists in signing a *time stamp token* containing current time τ and hash value $H(D)$ of document to time stamp: $\sigma_{PrvK}(H(D), \tau)$. The signing key pair (PrvK, PubK) belongs to an entity named *Time Stamp Authority* (shortened TSA), and is certified by a public key certificate. Time stamp's security is guaranteed by the signature. The verification process consists in comparing document's hash with the H(D) contained in time stamp token, verifying validity of TSA's certificate and finally verifying time stamp's signature.

On a performance point of view, time stamping being a "long" operation (RSA signatures are much longer than SHA-1 hashes for instance), this paper proposes an improvement for virtual WORM systems. Similarly to [BHS93, §2.2] where multiple blocks are time stamped together in a *round*, we suggest to time stamp together multiple blocks. To do so, we simply time stamp $H_{n,r}$ the last block hash. For a given round, with input $D = B_{1,r} \dots B_{n,r}$, time stamped virtual WORM's output is:

$$B_{1,r}H_{1,r}, \dots, B_{n,r}H_{n,r}, (H_{n,r}, \tau), \sigma_{PrvK}(H_{n,r}, \tau)$$
$$\forall i > 1, H_{i,r} = H(H_{i-1,r}, B_{i,r})$$

Block hashes are chained so $H_{n,r}$ depends from $H_{n-1,r}$ and $B_{n,r}$, and $H_{n-1,r}$ from $H_{n-2,r}$ and $B_{n-1,r}$... Recursively, $H_{n,r}$ is linked to $B_{1,r} \dots B_{n,r}$. So, time stamping $H_{n,r}$ is equivalent to time stamping $B_{1,r} \dots B_{n,r}$: all blocks are time stamped together through last block hash (note this is true because we are using chain hashing: if we were hashing each block independently, each block would need to be timestamped). This increases performances as time stamping will only be done once in a while, but of course, it is impossible to order chronologically blocks within a same round: all blocks of round r are considered to exist at τ the time stamp time of $H_{n,r}$. Consequently, implementations must find a balance between performances (time stamping less frequently) and time stamp accuracy (time stamping often).

3.3. Cryptographic hardware on time stamped virtual WORM

Time stamp protocol described in §3.2 relies on an unconditional trust of the TSA. If TSA's keys are compromised, or if TSA can be bribed, time stamp can no longer be trusted. Many researchers have worked on solutions to loosen this trust. [HS91, §5.1,§5.2] have proposed a linking scheme in which each time stamp response is linked to previous responses, and a distributed scheme for which the final time stamp is built upon responses of multiple TSAs. [BHS93, BdM91] have then proposed tree schemes that publish a time stamp response out of the root of a binary tree of time stamp requests. Finally, [BLLV98] have suggested a binary linking scheme based on rounds and hash functions.

Those solutions do loosen trust level of TSAs, but they unfortunately complicate the verification process of time stamp responses: either multiple time stamps need to be retrieved, or a round value, or previous time stamps etc (see [PRQMS98] for more precisions). For long term storage system, such design are usually impratical: for instance, in the linking scheme, verification process would have to retrieve all previous time stamps of the last 10 or 20 years...

In this paper, we'd like to find a compromise between convenience and trust of TSA. Consequently, we suggest to use a dedicated tamper-evident (and if possible tamper-proof) time stamper hardware card. This card would consist in:

- on-board cryptographic processors dedicated to time stamping, responding to a limited API such as `timestamp()`, `generateNewTSAKeys()` and `getPublicKey()`. Access to TSA's private key should be strictly forbidden.
- two internal clocks: first clock is set up at factory level and cannot be updated, second clock is used to time stamp documents and is synchronized from time to time through a secure link to an external reliable clock (an atomic clock for instance) under some very restrictive conditions[2]. For instance, second clock can only be re-synchronized if suggested update is very close to its current value, and if it hasn't drifted too far away from first clock. This idea is interesting because it does not require a permanent link to an external reference clock but only from time to time.

The whole time stamper card should behave as an opaque module, strictly forbidding (and detecting) any access to its internal components. For instance, FIPS 140-2 [NIST01] classifies different levels of security requirements for cryptographic modules, and a few selected trusted/governmental organizations accept to test and certify equipments according to FIPS 140-2 levels. Unconditional trust of TSA is then moved forward to unconditional trust of certifying organization. Depending on the security level which has been attributed to the card, a time stamp verifier can decide whether or not to trust the equipment.

3.4. Performances of our implementation

Practically, a Time Stamped Virtual WORM prototype has been implemented in Java. It is capable of processing a given input stream (multiple files, directories or a tape), adding security data (we also refer to this as the "WORMing" process) and writing the output to a directory or a tape. In our prototype, data is split into fixed size blocks (default is 256KB), then chain hashed using SHA-1. At the end of each file (or at each tape mark on a tape), the last block hash is time stamped using a Sha1WithRSA signature (key size can be configured) and the

[2]Pending US-patent number 2001-072-TOU "Trusted High Stability Time Source".

host's local clock. The time stamp's format complies to [TSP01][3]. All cryptographic operations are currently done by the software (standard algorithm implementations of Sun's JDK), the only hardware we attach to our prototype are two 9840 tape drives we read and write to.

Figure 2 shows the processing rates of our prototype for RSA-512, 1024 and 2048 signatures, compared to pure read/write rates on 9840 tapes (reading or writing files on tapes, without securing them). As we had expected it, the graph shows that time stamping less frequently gives better performances. A compromise needs to be found for each application, depending on security and performance requirements.

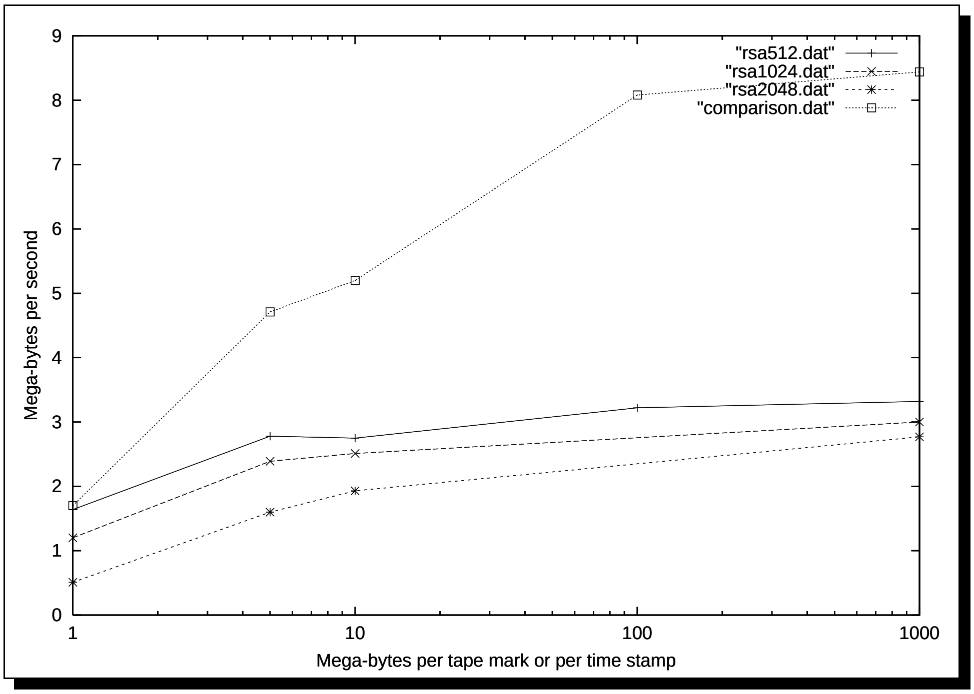

Figure 2: *Performance of a Java Time Stamped Virtual WORM prototype, on an Ultra Sparc 10 (400 MHz), connected to two 9840 tape drives (one for input, the other for output). The highest curve is provided here for comparison: it is the copying rate from one tape to another without using our prototype. The other results show the "WORMing" rate in mega-bytes per second for RSA-512, 1024 and 2048 bit keys, according to how frequent time stamps are done. Time stamping is performed each time a tape mark is encountered. Each test has been done over a total volume of 10000 MB. On the X-axis, a logarithmic scale has been used.*

4. Security analysis of time stamped virtual WORM systems

In the light of threat model in §2.2, this section intends to analyze results of proposed time stamped virtual WORM system regarding data, copy and time integrity, and hardware support dependency.

[3]Actually, the time stamp *request* strictly complies to [TSP01], but work is currently under progress to make the *response* really completely match [TSP01]. However, the differences should not impact our performance results.

4.1. Data integrity

Concerning data integrity, over long periods of archival, both accidental (for instance a media support failure ?) or intentional errors (a human attacker) may alter data. We believe in best cases the chain hashing mechanism will detect it, or at worst the time stamp's digital signature.

For instance, if block B_i is replaced or erased (accidentally, or intentionally) by $B_i\prime$, the low probability of collisions of hash function H guarantees that $H_i\prime = H(H_{i-1}, B_i\prime) \neq H_i = H(H_{i-1}, B_i)$. Modification is consequently detected in its block hash.

Now, let us suppose both data block and block hash are modified[4](see table 2), with $B_i\prime$ and $H_i\prime = H(H_{i-1}, B_i\prime)$. By the way, note such an attack is probably a human attack as it is highly unlikely an accidental error would produce the corresponding value for $H_i\prime$. Such an error is not spotted immediately:

1. Before index i, there's no problem: hashes match blocks.
2. At index i, we read $B_i\prime$, calculate $R = H(H_{i-1}, B_i\prime)$, and read $H_i\prime$. Error is not detected yet because $H_i\prime = R$.
3. Then, at index i+1, we read B_{i+1}, and calculate $R = H(H_i\prime, B_{i+1})$, which is different from $H_{i+1} = H(H_i, B_{i+1})$. That's where we detect the error.

Table 2: *A "smart" attack: block data is replaced by a new value, and block's hash is forged to correspond to new block*

Expected data		Real data	
B_1	$H_1 = H(B_1)$	B_1	$H_1 = H(B_1)$
B_2	$H_2 = H(H_1, B_2)$	$B_2\prime$	$H_2\prime = H(H_1, B_2\prime)$
B_3	$H_3 = H(H_2, B_3)$	B_3	$H_3 = H(H_2, B_3) \neq H(H_2\prime, B_3)$
B_4	$H_4 = H(H_3, B_4)$	B_4	$H_4 = H(H_3, B_4)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
B_n	$H_n = H(H_{n-1}, B_n)$	B_n	$H_n = H(H_{n-1}, B_n)$

So, the attack does not actually succeed. It only differs and diffuses the error, but the error *will be* detected. If the attacker wants to delay detection of B_i's modification as much as possible, he'll have to modify all H_j with $j \geq i$, but then error will be detected at time stamp's verification. As a matter of fact, either the last block hash has been modified to $H_n\prime$ but not the time stamp, and then time stamp does not correspond to $H_n\prime$, or both last block hash and time stamp token are modified to $H_n\prime$ and $(H_n\prime, \tau)$, but then time stamp token's signature fails because $\sigma_{PrvK}(H_n\prime, \tau) \neq \sigma_{PrvK}(H_n, \tau)$. The attacker cannot build a fake signature because he does not own TSA's private key.

In such cases, one may consequently jump to the conclusion that the intermediate chain hashes are useless, and that the whole system could work only with digitally signed time stamps. Technically speaking, when dealing with human attacks, this is true, but one should keep in mind that the system is meant to store thousands of tera bytes and that we are therefore very much concerned by performance... and accidental failures. In most cases, two records will be accidentally swapped, or a record will be set to zeros ($000 \ldots 000$) or ones ($111 \ldots 111$). Based on frequent errors, a recovery program could try the most obvious fixes and check whether data then corresponds to its hashes (see figure 3). How the recovery program is implemented is far beyond the scope of this paper and highly depends on the support data is written to. The important point here is that if the recovery program suggests a fix, it is possible to *prove* with a 100% guarantee if that fix is correct or not. For legal evidence, this point is very important: on standard systems, if data has been damaged and the operator

[4]Same analysis may be applied if block and its hash are erased.

finally manages to recover it, there is no proof recovered data is identical to the original. Concretely, this means recovered data can be disputed.

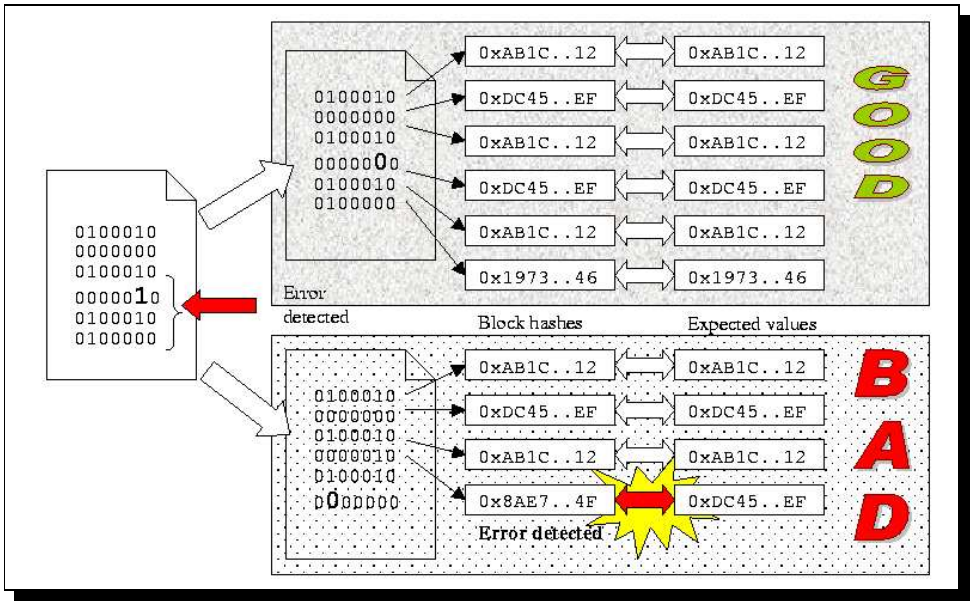

Figure 3: *Sure validation of recovery using chained hashed blocks. An error is detected in the retrieved file. The recovery program suggests two different solutions. The first solution is okay, verifying chained hashes proves the recovery is correct. Second solution is erroneous : an error is detected in chained hashes.*

So, the benefits of chain hashing are:

- performances: they make it possible to time stamp multiple blocks in a round (see §3.2),
- error location: except for human attacks, intermediate hashes will detect accidental failures without having to wait till the next time stamp (in 5, 10 or 20 MB for instance),
- proof of correct recovery: a fix can be proved to be correct or not.

4.2. Copy integrity with time stamped virtual WORMs

"Classical" WORM systems may achieve copy integrity by comparing bit to bit original data and its copy, but this is clearly not very efficient. This paper proposes to use block hashes that have been stored for data integrity.

Suppose we have a time stamp virtual WORM tape V_1 that stores secure documents :

$$B_1H_1, \ldots, B_nH_n, (H_n, \tau), \sigma_{PrvK}(H_n, \tau)$$

V_1 is copied to V_2. Verifying copy integrity on V_2 consists in:

1. checking hash blocks (H_i) on V_1 and V_2 are identical ;
2. checking time stamp signature on V_1 and V_2 are identical.

Using 256 KB blocks, SHA-1 and RSA-2048, each block hash has a size of 20 bytes, and time stamp token and its signature have an approximate size of 1 KB (256 bytes for the signature, 20 bytes for the hash, and the rest for TSA's certificate or other parameters – this is an approximation). Consequently, for a 1 Giga byte file, copy integrity algorithm will only have to check $\frac{20.2^{30}}{2^{18}} + 2^{10} \approx 80KB$. Only 0.008 % of the file has been bit to bit compared...

4.3. Time integrity

Concerning time integrity, an attacker cannot alter date τ contained in the time stamp token. If he tries to change τ to $\tau\prime$, verification of time stamp token's signature fails : $\sigma_{PrvK}(H_n, \tau) \neq \sigma_{PrvK}(H_n, \tau\prime)$.

4.4. Hardware support dependency

Time stamped virtual WORM are independent of hardware *supports* as they are based on S-WORMs (§2.1). For instance, if data on a first time stamped virtual WORM is output to a magnetic tape, and needs to be transferred onto a disk, there is no security problem. Actually, data written on the tape is already secured, and whatever support data is transfered to, it remains secured. So, time stamped virtual WORMs are truly independent of hardware support. One should however note that they are not entirely hardware independent: in §3.3 use of a time stamper card has been suggested. Fortunately, this should not be too limitative as the card should be built as a stand-alone module responding to strict API specifications. If in 10 years the card needs to be replaced (for instance because the on-board clock is out of service), we just need to find another hardware card compatible with the API.

5. Extending security over years and future work

Over long periods, storage systems will have to face multiple problems, such as :

- hardware drive failures,
- recovery of lost data (disk crash...),
- upgrade to newer media support (with more capacity, better resistance...),
- synchronization of time stamper's internal clock,
- renewal of TSA's key pairs,
- compromission of keys or algorithms...

We believe this paper has addressed problems which concern hardware related problems. For instance, hardware support independency will help use another support in case of a drive failure (§4.4), copy integrity will help migrate all data onto another media (§4.2) and intermediate hashes help recover data (§4.1).

This paper has also explained how time stamping clock could be kept accurate, using from time to time a secure link with an external clock (§3.3).

Concerning TSA-related problems, a few solutions have already arisen but are beyond the scope of this single paper. Basically, work is currently under progress for:

- "just-in-time" upgrade procedures: changing keys or algorithms just before they might become compromised,
- and Certificate Revocation List (CRL) system for the TSA to be able to renew its key pairs from time to time.

In §3.4, we have presented the performances of our software prototype. Issues now concern integration of this prototype to other high performance storage systems, such as RAIT[HMD01] (Redundant Array of Inexpensive Tapes), and optimization of performances on a dedicated hardware time stamper card prototype.

6. Conclusion

This paper has tried to contribute to security aspects of storage systems meant to keep document over years, with direct application to documents to be reproduced as legal evidence.

The concept of *virtual WORM* has been introduced as a new solution to existing WORMs' limitations. Instead of securing the media, the drive or the system which manages data, virtual WORM focus on directly securing data itself.

To do so, a data integrity detection technique based on cryptographic hash functions is used, and followed by a digitally signed time stamp. This makes it possible to detect any modification of data, and provides a high probability of valid recovery over accidental modifications. Using both intermediary hash functions and digital signatures offers a good compromise between performance (hash functions are fast) for large volumes of data and security (signature cannot be forged). Finally, we have demonstrated time stamped virtual WORMs improve hardware independency, and time and copy integrity features. Those properties are particularly useful for data meant to be kept over long periods.

7. Acknowledgements

We would like to thank Jacques Debiez for time he has accepted to spend with us sharing along his ideas concerning physical WORM systems and clock synchronization. We also wish to thank Vincent Girier for his helpful reviews and comments.

References

[Adap00] *Adaptation du droit de la preuve aux technologies de l'information et relative à la signature électronique*, Loi n. 2000-230, in *Journal Officiel du 14 mars 2000*, p. 3968, France, March 13, 2000, in French.

[Afn01] *Archivage électronique - Spécifications relatives à la conception et à l'exploitation de systèmes informatiques en vue d'assurer la conservation et l'intégrité des documents stockés dans ces systèmes*, French standard AFNOR, NF Z42-013, December 2001, in French.

[Abs00] L. Absher, *VolSafe(TM): A Discussion of Non-erasable, Non-rewritable Tape for the business environment*, White Paper, Louisville CO, July 2000.

[BdM91] J. Benaloh and M. de Mare, *Efficient Broadcast Time-Stamping*, Technical Report TR 91-1, August 1991.

[BHS93] D. Bayer, S. Haber and W. S. Stornetta, *Improving the Efficiency and Reliability of Digital Time-Stamping*, In R.M Capocelli, A. de Santis and U. Vaccaro, editors, *Sequences II: Methods in Communication, Security and Computer Science*, pp 329-334, Springer Verlag, New York, 1993.

[BLLV98] A. Buldas, P. Laud, H. Lipmaa and J. Villemson, *Time-Stamping with Binary Linking Schemes*, in *Advances on Cryptology* (CRYPTO'98), H. Krawczyk Ed., vol. 1462 of *Lecture Notes in Computer Science*, Springer, pp 486-501, 1998.

[Esg00] *Electronic Signatures in Global and National Commerce Act ("E-Sign")*, Public Law 106-229, U.S.A, June 2000.

[HMD01] J. Hughes, C. Milligan, J. Debiez, *High Performance RAIT*, Tenth NASA Goddard Conference on Mass Storage Systems and Technologies and Nineteenth IEEE Symposium on Mass Storage Systems, Maryland, U.S.A, April 2002.

[HS91] S. Haber and W. S. Stornetta, *How to Time-Stamp a Digital Document*, Journal of Cryptology, Vol.3, No.2, pp.99-111, 1991.

[MOV96] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, ISBN 0-8493-8523-7, October 1996.

[NIST01] National Institute of Standards and Technologies, NIST FIPS PUB 140-2, *Security Requirements for Cryptographic Modules*, U.S. Department of Commerce, August 17, 2001.

[PRQMS98] B. Preneel, B. Van Rompay, J-J Quisquater, H. Massias, J. Serret Avila, *TIMESEC: Design of a timestamping system*, Technical Report WP3, 1998.

[Roos99] M. Roos, *Integrating Time-Stamping and Notarization*, Master's thesis, 1999.

[TSP01] C. Adams, P. Cain, D. Pinkas, R. Zuccherato, *Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)*, Network Working Group, RFC 3161, August 2001.

[Wil97] R. Williams, *P-WORM, E-WORM, S-WORM Is a Sausage a Wienie ?*, Chicago IL, January 1997.