Streamed or Detached Triple Integrity for a Time Stamped Secure Storage System

Axelle Apvrille[1] James Hughes[2] Vincent Girier[1]

[1]Storage Technology European Operations
Toulouse Research & Development Center
1 Rd Point Général Eisenhower
31106 Toulouse, France
{Axelle_Apvrille, Vincent_Girier}@storagetek.com

[2]Storage Technology Corp.
7600 Boone Avenue North
Minneapolis, MN 55428, USA
jim@network.com

Abstract

Organizations and companies with integrity concerns for their archivals are currently left with very few and unconvenient solutions. To cope with those needs, a Time Stamped Virtual WORM system has been proposed previously, but only its concepts and theory have been examined yet.

Hence, this paper focuses on defining practical block formats to help implement this system in reality. But there are several pitfalls on the path of implementation, and this paper has to be extremely cautious not to introduce any limit - or security flaw - into virtual WORMs. With such requirements, two different block formats are successfully defined: a streamed *format where security data is inserted within user's documents, and a* detached *format where security information is written in a different location.*

Finally, the detached format is studied in the sample case of a tamper-evident FTP server.

Keywords

INTEGRITY, TIME STAMP, STORAGE, DIGITAL SIGNATURE, WORM, XML.

1 Introduction

THE burst in data volumes has led to a growing concern for security of archives. Storing data is already an interesting feature, but providing *triple integrity* guarantees (data, time and copy integrity) is even better.

No on-the-shelf solution being available, we have previously proposed in [AH02] a *Time Stamped Virtual WORM* (Write Once Read Many) system. Using cryptographic hash functions and digitally signed time stamps, it defines security information to secure user data. This system is media-independent and it meets triple integrity requirements. However, only a theoretical study has been done previously, though there is a strong need for a real implementation which (1) does not introduce any security flaw, (2) is adaptable to any kind of media and (3) may evolve easily throughout years.

To do so, this paper proposes two generic block formats: a *streamed* format where security is written within the user data stream, and a *detached* format where security data is kept apart from user's data. XML Schemas for this format are proposed, and help define extensible, easy to adapt solutions.

The paper is organized as follows. Section 2 introduces previous work concerning secure virtual WORM storage, and explains how this system achieves "Triple Integrity". Section 3 explains the need for a data format and proposes a streamed data format for the system. This solution is

improved in section 4 where security data does not need any longer to be inserted in data to secure. Finally, section 5 discusses about a sample application of our proposition, over a secure tamper-evident FTP server.

2 Previous work on tamper-evident storage systems

2.1 Using WORM systems for security

Multiple studies have already focused on secure storage. Globally, all of them have settled down for WORM systems using media where one can only write *once* but possibly read multiple times. For instance, [Kah00] has studied use of WORM optical disks for archival of legal evidence documents.

Unfortunately, "physical" WORMs have shown their limits in [ASD99], because whatever technology is used, a skilled user - with appropriate equipment - can always manage to alter documents. Moreover, we have stated in [AH02] that secure time stamps were not taken into account in such systems, even though date could be an important information for most legally archived documents.

So, different sorts of WORM technologies have arisen and are classified in [Wil97]. On one side, E-WORMs *embed* protection code, but they have not been very successful because they do not significantly improve data security compared to physical WORM supports (see [AH02]). On the other side, S-WORMs offer software protection for data, but they have initially been abandonned because it seemed too easy to by-pass software protection (see [Wil97]).

The use of cryptography has given back interest in S-WORMs, and it has led us to propose in [AH02] a new kind of WORM technology: *virtual WORMs*. Behaving like physical WORM supports, they focus on securing data itself, independently of hardware support: data can be secured on a support which does not provide physical security, like for example a magnetic tape, or a hard disk. Furthermore, secure time stamping functionalities are offered. This system has consequently been named *Time Stamped Virtual WORM*.

On a legal point of view, such systems are acceptable as (1) several countries have agreed on the suitability of electronic records in court trials (for instance see [ESI00, LAW01]), and (2) media or technology to be used for storing important documents are very rarely mentioned, leaving laws open to any possible evolution. Till 1997, a rare exception to this was the Securities and Exchange Commission (SEC) in the U.S., but finally, they amended their rule in [SECS97] to expand electronic storage solutions for brokers and dealers. Similar work is also currently under progress in AFNOR and ISO recommendations such as [AFN01].

So, *technically*, Time Stamped Virtual WORM systems offer a real secure storage alternative to traditional optical disks, and *legally*, they are accepted by most regulations.

2.2 Basics of Time Stamped Virtual WORM systems

The general concept of *Time Stamped Virtual WORM* systems has been proposed in [AH02]. Basically, data is secured during a "WORMing" process at figure 1.

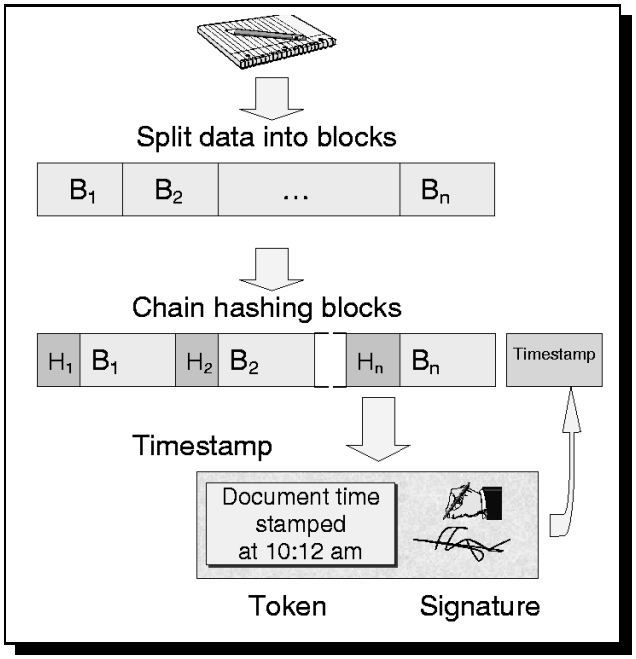

Figure 1: *Description of Time stamped virtual WORM mechanism. Both one-way hash functions and digital signatures are used to secure documents.*

User data is first split into blocks. Then, blocks are chain hashed [AH02, §3.1]: this consists in hashing each block with the previous block hash. Finally, the last memorized block hash is time stamped and digitally signed using for instance [ACPZ01]. Naturally, the "WORMing" process is reversible: documents may be "UnWORMed" by simply taking off all block hashes and time stamps.

Document's validity may be checked upon request from a *validator*, at any time after it has been secured. Depending on situations, the validator may be the user, or a trusted third party. A good way to do that, for instance, is to develop an Open Source validation program, so that anybody can check the sources and improve them. As the validation program is not owned by any specific party, it can be more easily trusted not to be corrupt.

2.3 Triple Integrity for Time Stamped Virtual WORMs

In this paper, *triple integrity* makes reference to the combination of *data* integrity, *time* integrity and *copy* integrity. Time Stamped Virtual WORM systems have been designed to meet those requirements:

- **data integrity** requirements are met by both block hashes and the time stamp's digital signature. Technically speaking, the block hashes are not strictly necessary for data integrity, but (1) they make it possible to time stamp less frequently (hence improving performances) and (2) in simple cases, they help spot accidental write failures. Security details may be found in [AH02, §3.2,§4.1].

- **time integrity** is taken into account by digitally signed time stamps. In [AH02, §3.3], we have stated those time stamps are impossible to forge *provided the Time Stamp Authority (TSA) is trusted*. Various methods have been proposed to loosen this trust [HS91, BdM91, BHS93]. On our side, we have suggested use of a dedicated physically secure hardware card meeting FIPS 140-2 [NIS01] level 3 or 4 requirements.

- **copy integrity** is the ability to prove a copy is strictly identical to the original, and is meant to prevent people from making fake copies of documents. Basically, Time Stamped Virtual WORMs require bit-to-bit checking of less than 0.01% of user data. Details of copy integrity's importance, and how it is solved may be found in [AH02, §2.2,§4.2].

2.4 Limits to Time Stamped Virtual WORMs

Time Stamped Virtual WORM systems are an interesting alternative to physical WORMs because they are media independent and because they offer strong cryptographic level security such as triple integrity.

Yet, previous work has only presented the theoretical concepts of such systems, and direct implementation of this work is impossible. If we merely write a first block hash H_1, then user data B_1 etc till H_n, B_n and the final time stamp, there is absolutely no way to know where user data blocks start and end, when we're dealing with a time stamp or a block hash. This leads us to defining block formats for Time Stamped Virtual WORMs. In section 3, we'll propose a block where security information is written within the stream of user data, and in section 4, a block format which separates security data from user flow.

3 Streamed data format

Defining a data format is a step towards implementation of Time Stamped Virtual WORM. However, there are several pitfalls our data format should be extremely cautious about:

1. **it should not introduce any security flaw**. If triple integrity is no longer met when the data format is used, something is definitely wrong.

2. **it should be extensible**. Time Stamped Virtual WORMs were intended for long-term storage. There's a very high probability new needs, new technologies will arise in a ten-year's time, so data format should be ready to evolve.

3. **it should introduce as few limits as possible to the theoretical model**. For instance, the storage system is media-independent, this feature should not be jeopardized by the data format.

3.1 Block description

§2.4 has shown that data format's main goal was (1) to set boundaries between user and security data and (2) to say whether security data contains a block hash or a time stamp. As this information is not included in user or security data, it should be added somewhere and should be easily accessible on any media. File systems offer random access, but tapes are more limited with only sequential access. Consequently, we chose to add all missing information in a *WORM header*, and not a footer. Then, we chose to append a security data block, and a user data block: this forms a *WORM unit* (see figure 2). Actually, the order between security and user data is not important, but a choice had to be made.

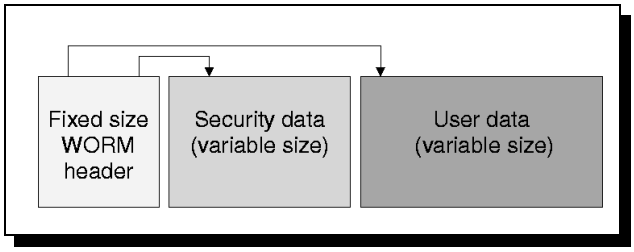

Figure 2: *Components of a WORM unit: header, security data and user data.*

Placing a header at the beginning of each WORM unit is not enough to make it easy to read on any media. As a matter of fact, tapes can only read a record if provided buffer is big enough to contain the *whole* record (impossible to split a record). Unfortunately, no operation is able to return the size of a next record. So, if header's size is not fixed, one should always allocate a large enough buffer to be sure to read it: this is not a good option, because this maximum buffer size depends on the tape drive. As this did not introduce any prejudice to other supports, we chose to define the WORM header as a *fixed size* block. A block layout example of a WORMed tape is given at figure 3.

On tapes, each file is represented by multiple records, and ended by a *tape mark*. For a WORMed tape, each file is a sequence of WORM units - with the last unit containing a time stamp.

This solution uses one WORM header for every security / user data couple. Instead, a single global WORM header could have been written for all blocks $H_{i,1}, B_{i,1} \ldots H_{i,n}, B_{i,n}$ till time stamp T_i. This reduces the number of WORM headers from $n+1$ to 1 (on tapes, this is interesting because writing a new record is time consuming). The header should have memorized the number of security blocks ($n+1$), and the number of user data blocks (n). Nonetheless, this solution has not been chosen because:

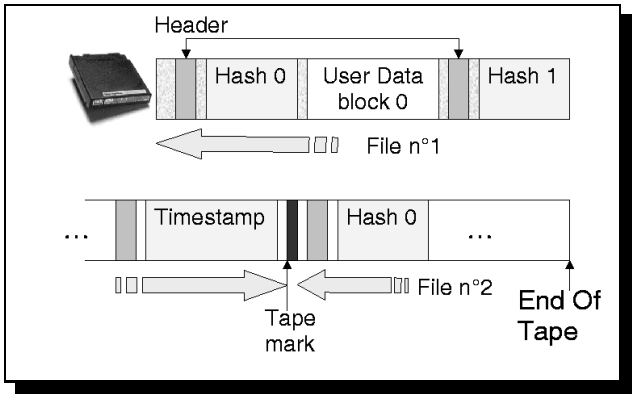

Figure 3: *Block layout overview of a WORMed tape.*

- it implies the fact that *the number of security and user blocks* is known *at the beginning of the WORMing process*. This is not true. Virtual WORMs are meant to operate on input *streams*, so they might not have the knowledge of when a document might end.

- it implies the fact that *all user blocks have the same size* and *all security blocks too* (or the header should contain a map of all blocks, with their respective size, but this is quite complicated). Although user blocks may be of fixed size, virtual WORMs have actually never required this, so we do not wish to introduce an additional constraint. As for security blocks, their size cannot be fixed because for instance, if Time Stamp Authority's keys are changed, a new public key certificate (with possibly a different size) needs to be inserted into the time stamp.

3.1.1 WORM headers

§3.1 has explained the use of one fixed size WORM header in every WORM unit, and that it should be able to set boundaries between user and security data (possibly of variable size) and indicate the content of security data.

As elements' order in a WORM unit is fixed (security and then user data), header just needs to memorize sizes. Security data size and user data size have been allocated 4 bytes each (see figure 4). This limits their size to 4GB, but

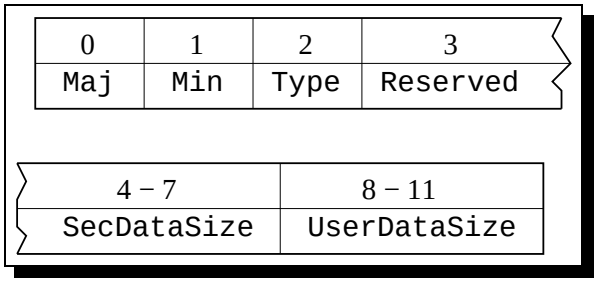

Figure 4: *WORM header format. The upper line indicates the number of bytes allocated for each field.*

this seemed reasonable to us. A type indicator is also included in the header (`Type`), offering 256 possible types. Only 2 of them are used at the moment: block hash and time stamp.

Finally, to keep this format extensible, a version number has been included, represented by a major (`Maj`) and a minor (`Min`) number.

3.1.2 Security data

Both types of security data (block hash or time stamp) are built upon the same model: first, an identifier on 2 bytes and then, context dependent information. Total size is indicated in the `SecDataSize` field of the header.

For block hashes, the identifier represents the mechanism which has been used to obtain the block hash, and it is simply followed by the hash data (ex: SHA-1 output a 20-byte digest): refer to table 1. The identifier is different from a hash algorithm identifier, because, for instance, it may represent *chain hashing* with SHA-1 hash functions. At times where ASN.1 [ITU97a] is being challenged by XML, we haven't thought it wise to use ASN.1 object identifiers to represent the block hash mechanism, as this would have meant depending from that standard.

Table 1: *Block hash format.*

`MechID`	`DigestValue`

For time stamps, the identifier represents the *format* of the time stamp (see table 2). As a matter of fact, there are multiple ways of representing time stamps: DER encoded [ITU97b] time stamp response from [ACPZ01], an XML time stamp (see [AG02]), a proprietary format etc. The identifier is followed by the time stamp itself. Its size may vary but it can be deduced from the WORM header's `SecDataSize` field.

Table 2: *Time stamp format.*

`TimeStampType`	`TimeStampValue`

3.1.3 User data block

Finally, the user data part is the simplest: it is just plain, raw, user data. User data should not be modified. For instance, if user data ends by a few trailing zeros, those zeros should not be truncated or this will be considered as data tampering.

One should also note that user data might be empty. For example, the last information which is written about a WORMed document is its time stamp. This consists in a WORM header, and security data being the time stamp, but there is *no user data*. The header should mention this by setting its `UserDataSize` field to 0.

3.2 Triple Integrity with streamed data format

This data format does not impact the content of security data. So, security data still secures user data regarding

data, time and copy integrity: the triple integrity features of Time Stamped Virtual WORMs is preserved.

The only possible attack one may attempt is to modify *the WORM block format* itself: a WORM header, or the mechanism and time stamp identifiers. For instance, an attacker can corrupt sizes of security or user data. As header is not sealed, this modification is not detected. However, it is important to note that:

1. the attacker cannot modify *user data* undetectably. At most, he can merely ruin the system. Unfortunately, this has always been true as Virtual WORMs are not tamper resistant.

2. an analysis program reading carefully data flow might reveal simple attacks, and even be able to recover original data. For instance, if a WORMed document references use of chained hashes with SHA-1 for all blocks except block B_{28}, an analysis program could send an alarm and try to recover the document by setting back the block hash mechanism identifier to "chained hashing with SHA-1".

To conclude, section 3 has proposed a data format to help implement Time Stamped Virtual WORM systems. This data format meets defined criterias as triple integrity is still achieved, only few minor limits are introduced (blocks cannot exceed 4 GB), and evolution is possible (version number is included, and fields are allocated more bytes than necessary).

This data format is said to be *streamed* as security data is written in the flow of user data. As this is not convenient in all situations, in next section, we'll try to improve this format by detaching security data.

4 Making a detached triple integrity certificate

4.1 The need for detached security data

Block format defined in section 3 is efficient for processing input streams of sequential data. However, with such a format, secured document now contains additional information: headers and security data blocks have been inserted.

This might not be very convenient for two reasons. First, extremely secure environments might require user document is left *strictly* unmodified: nothing should be inserted in it (and, actually, it is quite paradoxal that working on document integrity and non-modification, the storage system is in reality allowed to modify them). Second, with such a block format, user's and validator's needs are incompatible: user cannot work with a *secured* document because headers and security blocks pollute it, and validator cannot validate anything without security information. Consequently, they cannot communicate easily with each other because they do not need the same information.

So, actually, there is a need for a format where securing a document does not "corrupt" it, and where user and validator both get the information they need, but no more.

4.2 The detached block format

To solve this problem, this paper proposes to store separately headers and security data blocks from user blocks. Hence, user data remains strictly unmodified, and it is easy to send only user data to user, and everything (headers, security and user blocks) to the validator.

An example of detached block format is represented at figure 5. Tape 1 only contains user data, whereas tape 2 contains detached security data. Tapes may be stored in separate locations.

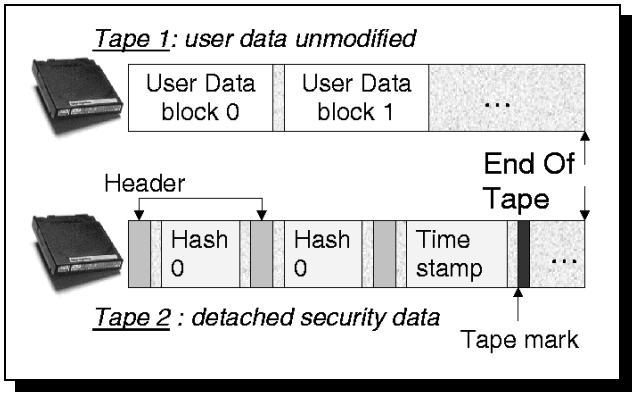

Figure 5: *Example of detached block format on tapes.*

The sequence of headers and security data blocks actually are a sequence of special WORM units: their head-

ers mention user data, but those blocks are never present. This is not possible in the streamed data format, so we need to indicate this is a detached mode. For instance, we suggest to add a `detachedMode` flag in the `Reserved` area of the WORM header (see figure 4 in §3.1.1).

So, a document now undergoes the following process:

- at storage time, the document goes through the WORMing process. Output dispatches user data in a given location and, at some other place, WORM headers and security data.
- at retrieval time, there is no processing to be done: the system can just send user data blocks as is.
- at validation time, the validator requests WORM headers and security data from the storage system. Then, he retrieves the unsecured document to be validated. Note the user can send him the document: it is now easy for them to communicate. Finally, he checks document's authenticity: he's got all elements to complete his task.

There is also a concern about size of information sent to the validator. If a 1 GByte file is WORMed, the validator surely won't be willing to receive an additional 1GB of security information ! Fortunately, size of WORM headers and security data is very small compared to user data: for a 1 GB file, under reasonable conditions, table 3 evaluates security information to only roughly 140 KBytes.

Table 3: *Approximate size of a WORM certificate for a 1GB file, using 4096 x 256KB blocks, chained SHA-1 hashing, and 2048 bit RSA signatures.*

$4096\times$	(	12 bytes		*header*
	+	22 bytes	)	*security data*
+		12 bytes		*header*
+		2048 bytes		*estimation of time stamp's size*
+		256 bytes		*time stamp's signature*
$\approx$		140 KBytes		

4.3 From detached block format to detached WORM certificates

When stored, WORM headers and security data are just plain binary information. We'd like to improve this and offer to the validator a better suited representation of the information he needs. Similarly to digital signatures where validators check certificates, this led us to the idea of building a *WORM certificate*.

A *WORM certificate* is merely the *representation* of WORM headers and security data (also called *"pre-certificate" information*). Multiple representations exist, so WORM certificates may be represented with different ways. For instance, figure 6 has represented two possible WORM certificates of the same "pre-certificate" information: a PEM-like WORM certificate, and an XML-like WORM certificate.

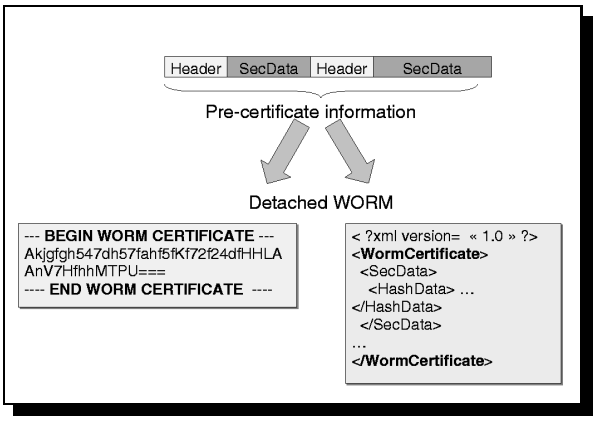

Figure 6: *Converting pre-certificate information into a WORM certificate.*

4.4 Example of detached WORM certificate representation using XML

A WORM certificate needs to be easy to process for the validator, extensible, and if possible customizable and human readable. XML meets all those criterias as multiple XML parser tools exist and help process XML documents, XML has been designed to be extensible (its name is *eXtensible Markup Language*), and finally its

```
<element name="WormCertificate" type="worm:WormCertificateType">      *1*
   <complexType name="worm:WormCertificateType">                       *1*
      <sequence>                                                       *1*
         <element name="SecData" type="worm:SecDataType"               *1*
          maxOccurs="unbounded"/>                              *1*
      </sequence>
   </complexType>
</element>
<complexType name="worm:SecDataType">                                  *2*
   <choice>                                                            *2*
      <element name="HashData" type="worm:HashDataType" />             *2*
      <element name="TimeStampData" type="worm:TimeStampDataType" />   *2*
   </choice>                                                           *2*
</complexType>
<complexType name="worm:HashDataType">
   <sequence>
      <element name="MechanismIdentifier" type="anyURI" />             *3*
      <element name="HashValue" type="ds:DigestValueType" />           *4*
   </sequence>
</complexType>
<complexType name="worm:TimeStampDataType" >
   <sequence>
      <element name="TimeStampIdentifier" type="anyURI" />             *3*
      <element name="TimeStampValue" type="tsp:TimeStampRespType" />   *4*
   </sequence>
</complexType>
```

Figure 7: *XML Schema for a WORM certificate.*

layout may be customized by using XML stylesheets. Consequently, using an XML representation of detached WORM certificate seemed quite adequate.

In this example, our representation of WORM certificates uses XML time stamps [AG02, §4.2] (the `tsp` namespace), and XML Signatures [ERS02] (the `ds` namespace). Used namespaces are listed at figure 8.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ourWormNameSpace"
  xmlns:tsp="http://www.isse.org/papers/2002
             /ApvrilleGirier"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:worm="http://ourWormNameSpace''
elementFormDefault="qualified" />
```

Figure 8: *Definition of XML namespaces used for an XML representation of WORM certificates.*

Then, we use XML Schema [W3C01] to define the content of an XML WORM certificate. Part 1 of figure 7 defines the certificate as a sequence of headers and security data couples. Part 2 states that security data is actually a choice between either a block hash or a time stamp. In *3* we can notice that block hashing mechanisms and time stamp identifiers are represented as URIs - which is common in XML. Finally, in *4*, we rely on the definition of digest values in [ERS02] (`DigestValueType`) and time stamp responses (`TimeStampRespType`) in [AG02].

4.5 Triple Integrity for the detached block format

With the detached mode, security data is the same, it is only stored in a separate location from user data. Consequently, using the detached mode does not change anything to security guarantees offered by Time Stamped Virtual WORM systems (refer to §3.2).

One would have maybe expected detached WORM certificates to be globally signed. However, this is unnecessary because there is already a signature: the signature of time stamps. Block hashes being chained, time stamp's signature secures all previous blocks. Adding a global signature will not improve integrity guarantees regarding *user's* data, it will merely prove *the WORM block format itself* has not been corrupted. Thus, we have not thought

an additional signature useful enough.

5 Building a tamper-evident FTP server

In this section, we propose to develop a sample application to WORM block formats: a tamper-evident file server.

5.1 Primary goals of a tamper-evident FTP server

A user has the following needs: he wants an easy way to archive multiple documents, and additionally (1) he requires his data cannot be tampered with undetectably, and (2) he wants to know for sure when he stored his document (so that, for instance, he can compare different versions). In a word, he needs a tamper-evident file server with secure time stamping capabilities.

As FTP [PR85] is a very famous protocol dedicated to file transfers, with multiple implementations on any platform (even standard browsers understand FTP), it has been selected for transferring user documents. For security guarantees, as user needs fit in triple integrity, a Time Stamped Virtual WORM service could handle them. So, basically, we need an FTP server to address transparently a Time Stamped Virtual WORM service. The resulting application - a *WORM FTP server* - should handle three different situations (refer to figure 9):

1. when user puts a file on the server, it is transferred via FTP, and then automatically secured by the Time Stamped Virtual WORM service,
2. user should be able to retrieve his file from the server. His file should be unmodified,
3. validator should be able to check integrity of a retrieved file.

5.2 Designing a WORM FTP Server with detached WORM certificates

When a user retrieves his file, he expects to receive his file and *only* his file. Headers and security data should be omitted. On the other hand, when the validator wishes to check a file, he needs all information. This is a typical scenario for using the detached format (section 4).

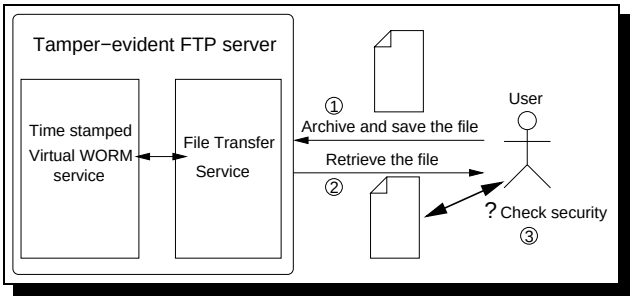

Figure 9: *Use cases for a WORM FTP server.*

So, we propose that when a user puts a file on the server, it is transferred via FTP, and then automatically secured, in a detached mode, by the Time Stamped Virtual WORM service (see figure 9). As detached mode is used, retrieval is very simple: when a user gets a file from the server, it can be sent over without any previous transformation. Note that in this case, the time-stamping authority is supposed to be part of the FTP server. Other schemes could be designed with the FTP server sending requests for time-stamps to an external authority.

For validation, there are two eligible solutions. First, the validator could be asked to send a *status* command to the server. WORM headers and security data blocks for a given file would then be converted into a detached WORM certificate and sent over to the validator. This solution is quite neat, but some FTP clients cannot easily send status commands. For instance, with a browser, you can merely authentify yourself and retrieve files.

Consequently, a second solution has arisen: when a file is put on the server, it is automatically processed in detached mode, so we could simply write this detached pre-certificate information as a file on the server (for instance, we could append a `.SEC` extension to its name). To retrieve a WORM certificate, the validator would then merely need to request the appropriate `.SEC` file. The server would receive the request, and transform the pre-certificate information into a WORM certificate and send it back to the validator. This solution is not perfect, because it duplicates the number of files on the FTP server,

however it offers an easy solution to retrieve validation data. Finally, this is the solution we have selected.

5.3 Implementing a WORM FTP Server

More precisely, if no *status* command is to be used, the `STOR` and `RETR` FTP commands should be modified (see figure 10).

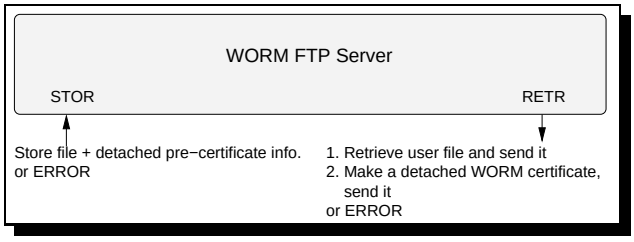

Figure 10: *Overview of FTP server commands to modify for use with a Time Stamped Virtual WORM service.*

The `STOR` command is used, with a pathname argument, to store a file on an FTP server. It accepts transferred data on a data connection. If the given pathname does not refer to any existing file yet, then transfer the file, and then WORM it. Two output files should be created: the user file unmodified, and a pre-certificate file with a `.SEC` extension appended to its name. If the file already exists, it should be WORMed again, and both resulting files should be overwritten with new information. If for some reason the WORMing process fails, the command should fail with a *permanent negative completion reply* error (see [PR85, §4.2]).

The `RETR` command is used, with a pathname argument, to retrieve a file stored on the FTP server. On the server, the file's content should remain unaffected. If the pathname refers to a pre-certificate file, the file should first be converted into a WORM certificate - using a given representation, for instance XML (see §4.4) - and then sent back over a data connection. If the pathname refers to user file, then the file can simply be sent back without any processing. If the file does not exist, or the conversion to a WORM certificate fails, the command should fail with an appropriate error.

5.4 Possible threats to the FTP WORM server

Let's look back at our FTP WORM server. If an attacker has access to the server, he cannot successfully modify a file undetectably: the pre-certificate information is there for detection of tampering. If he modifies pre-certificate information, then the information does not match the file any longer. If he modifies both, the time-stamp in the pre-certificate information will not fit. The WORM layer guarantees triple integrity for files stored on the server.

If the attacker hacks communications between the server and the client, then he can possibly modify the file to be stored on the server. But then, the wrong file gets WORMed: when the validator retrieves the file, he merely validates that the *attacker's* file has not been modified. Finally, if the attacker tampers the WORM certificate, then, either the block hashes or the time stamps will detect the error.

Such an FTP-server is not *tamper-resistant*. An attacker can probably corrupt data if he has access to the server (though, as mentioned in §3.2, slight modifications to the original data could be recovered with an appropriate program). However it is *tamper-evident*: the attacker cannot corrupt anything undetectably.

6 Conclusion

In the context of long-term archival, there is a strong need for a secure storage system offering triple integrity guarantees. As no real solution existed yet, previous work introduced the concepts of a new system named Time Stamped Virtual WORM. However, only the theoretical aspects of this system were investigated.

In this paper, we consequently focused on proposing possible implementations for this system. To do so, we defined two different block formats. The former is particularly efficient in situations where input is processed as a stream of data. The format of a WORM header and a security data block have been detailed. The latter improves the streamed format in situations where writing security information within user data is not acceptable. It consists in detaching previously defined headers and security blocks in a separate location. Such a format is open to conversions into well suited representations such as a

stand-alone XML WORM certificate.

Both formats have been designed to work over any kind of support, and are open to future extensions. We have also proved that they did not introduce any security flaw in Time Stamped Virtual WORM systems. Finally, we have illustrated the use of our detached format in a practical case, over a tamper-evident FTP server.

Future work should still be done in several areas. For instance, we should work over recovery of altered WORM headers, over implementation of a hardware time stamper card, and over the idea of a future WORM protocol to ease communications between different entities.

Acknowledgements

The authors would like to extend their acknowledgements to Gérald O'Nions, with whom discussion always turns out to be very useful. WORMing an FTP server was his initial idea and we believe it really shows use of detached triple integrity.

References

[ACPZ01] 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.

[AFN01] AFNOR. 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. NF Z42-013, December 2001. in French.

[AG02] A. Apvrille and V.Girier. XML Security Time Stamping Protocol. In *proceedings of the Information Security Solutions Europe Conference (ISSE 2002)*, Paris, France, October 2002

[AH02] A. Apvrille and J.Hughes. A Time Stamped Virtual WORM System. In *proceedings of the SEcurité des Communications sur Internet workshop (SECI 2002)*, Tunis, Tunisia, September 2002.

[ASD99] HP Automated Storage Division. Safeguarding Data with WORM: Technologies, Processes, Legalities and Standards. Technical report, Hewlett-Packard, 1999

[BdM91] J. Benaloh and M. de Mare. Efficient Broadcast Time-Stamping. Technical Report 91-1, Clarkson University, Department of Mathematics and Computer Science, 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*, pages 329-334. Springer Verlag, New York, 1993.

[ERS02] D. Eastlake, J. Reagle and D. Solo. *(Extensible Markup Language) XML Signature Syntax and Processing.* Network Working Group, RFC 3275, March 2002

[ESI00] Electronic Signatures in Global and National Commerce act (E-SIGN), Public Law 106-229, June 2000

[HS91] S. Haber, W. S. Stornetta, *How to time stamp a digital document*, Journal of Cryptology, Vol.3, No. 2, pp. 99-111. 1991

[ITU97a] ITU-T Recommendation X.680, *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation*, OSI networking and system aspects - Abstract Syntax Notation One (ASN.1), Series X: Data networks and open system communications, December 1997.

[ITU97b] ITU-T Recommendation X.690, *Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, OSI networking and system aspects - Abstract Syntax Notation One (ASN.1), Series X: Data networks and open system communications, December 1997.

[Kah00] R. Kahn. Evidentiary benefits and business implcations of write-once-read many ('WORM') optical

disk storage for record management. Technical report, Cohasset Associates, Inc. for Hewlett-Packard, December 2000.

[LAW01] Application de l'article 1316-4 du code civil et relatif à la signature électronique. Décret n. 2001-072, March 2001, in French.

[NIS01] NIST. *Security Requirements for Cryptographic Modules.* U.S. Department of Commerce, FIPS PUB 140-2, August 2001.

[PR85] J. Postel and J. Reynolds. *File Transfer Protocol (FTP).* Network Working Group, RFC 959, October 1985.

[SECS97] Securities and Exchange Commission (SEC). Reporting Requirements for Brokers or Dealers under the Securities Exchange Act of 1934, February 1997.

[W3C01] W3C. XML Schema, February 2001.

[Wil97] R. Williams. P-WORM, E-WORM, S-WORM, Is a Sausage a Wienie ? Technical Report, Cohasset Associates Inc., January 1997.