Data Assurance in a Conventional File System

Rudan, Sasa; Kovacevic, Aleksandra; Milligan, Charles; and Milutinovic, Veljko

Abstract— ***In the introductory part, this paper defines the general environment for this research and defines the terms of interest. Then we define the research problem: How to find a mechanism that guarantees that a file stored in a conventional file system, on disk, has not been modified; this problem is important because it enables security of the imperceptible changing of data, which is the main problem with digital data acquisition. Existing solutions to the problem are briefly surveyed in the third part, and their drawbacks are underlined; each surveyed piece of research is analysed according to the same template. Then, the essence of the proposed solution is presented: Our proposal is a smart card based Digital Sealed File System (DSFS). The conclusion is from the point of view of performance/complexity ratio.***

1. Introduction

Cryptographic techniques play an important role in data protection, since accessing data can be limited to those who hold the proper key.

There are several basic security services which cryptography may provide: confidentiality, data integrity, authentication, authorization, and non-repudiation. In many applications, the combination of cryptographic services is desired [2].

Common security systems imply a kind of Write Once Read Many (WORM) storage systems. This is not what we need, since almost all WORM systems relay on the specific format of data storage/archiving [4]. As soon as we transfer the data of interest to another medium/system, security of the data is broken.

2. Problem Statement

The goal of our research is not to protect document content, but only to protect unnoticed modification of the document. The idea is to allow any person to check if stored document or data is consistent and not modified. On the other hand, no one is allowed to be an authorized person and to do imperceptible change of the document content. The medium dependency has to be avoided and also any specific operating or file system organization.

At first, one can think that our system is much easier to implement then a classical security system. What we need to achieve is to prevent any unnoticed modification of the document, and not to protect content of the document against disclosure. In the classical encryption system there is a document owner; thereby this person has absolute rights over the document and its security. On the contrary, in our case there is no person who we can trust. What is more interesting, we have to be at most careful with the owner of the document (i.e. in the case of financial books, government wants to protect itself from any malicious modification by the accountant; Government wants to put a kind of digital seal over the document that is inspected).

3. Existing Solutions and Their Criticism

The data security systems are based on the classical cryptography principles [2]; two of them are symmetric and asymmetric.

Symmetric-key cryptography principles are presented in the "*Applied Cryptography,*" by B. Schneier [1]. It is a mechanism where the same key is used for both encrypting and decrypting. It is intuitive principle because of its similarity with, for example, using the same key for locking and unlocking a door.

Principle of asymmetric cryptography is presented in "*New Directions in Cryptography*," by D. Whitfield and M. Hellman [2]. Encryption-decryption process involves key pair - one for encrypting, and the other for decrypting. Given a key pair, data encrypted with the public-key can only be decrypted with its private-key and conversely, data encrypted with the private-key can only be decrypted with its public-key. The public key is distributed widely and freely. The private key is never distributed and must be a secret.

The solution for our problem using these techniques requires following participants: the client, the server, and the third party. Client is a user who wants to check the document authenticity. Server is an interface to the document and the document itself. Third party is an auxiliary service that provides verification of server (i.e. "Document is not modified").

The scenario for this solution can be described as follows: client is accessing the server with a certain request (typically, to check if the document is modified). Upon receiving the request, server is answering (typically, with claim

Manuscript received October, 2004. This work was supported in part by the StorageTech.

Sasa Rudan, Aleksandra Kovacevic, and Veljko Milutinovic are with Faculty of Electrical Engineering, University of Belgrade.
Charles Milligan is with StorageTech.

"Document is (not) modified"). Client refers to the third party, in order to verify correctness of the answer. It is necessary that both, client and server have a trust in the third party (Fig. 1):

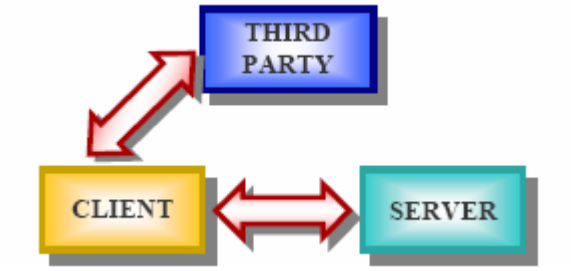

Figure 1: Possible solution scheme

However, requirements for our system imply that it is not allowed to have an intermediary authoritative body (third party).

4. Proposed Solution and why It Is Expected to Be Better

In our solution, it is used the public key encryption technology and the digital signing. Cryptography algorithms are public available and confidentiality of cryptographic methods relies on the confidentiality of the private key. Therefore, the most important point of every public-private key system is to secure the location where the private key is stored. The solution for that problem is using the specialized hardware components that have enough memory to store all relevant cryptographic data and enough processor power to perform basic cryptographic operations independently of operating system and software applications. It has been used the characteristic of the smart card: the resistance of reading their contents; upon attempt of opening smart card, it will destroy itself [11].

4.1 System Environment

The server in our system contains trusted documents, created by some users which are used by the other users. There are three actions over the documents that have to be provided: access to the documents, verifications of the documents, and document signing.

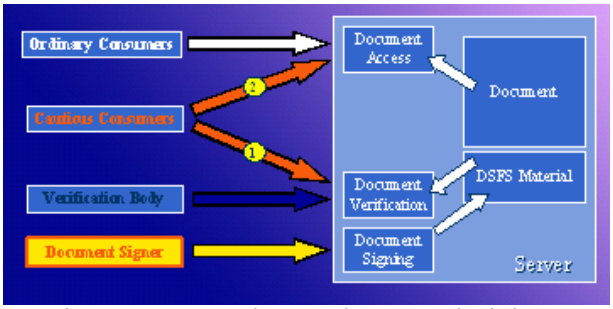

Figure 2: System environment for proposed solution

It can be identified four types of the document users or use cases (Fig. 2):

Ordinary consumers - they use document, not bothering with document integrity, so they only do "document access" action.

Cautious consumers - before using the document they want to verify document integrity, so they do both actions: verification and access.

Verification bodies – they are charged for document integrity verification and not interested in the document content.

Document signer - signer is a person (equipment) to whom is granted trusted period for signing the document.

4.2 System Components

The following components exist in the proposed solution (fig. 3):

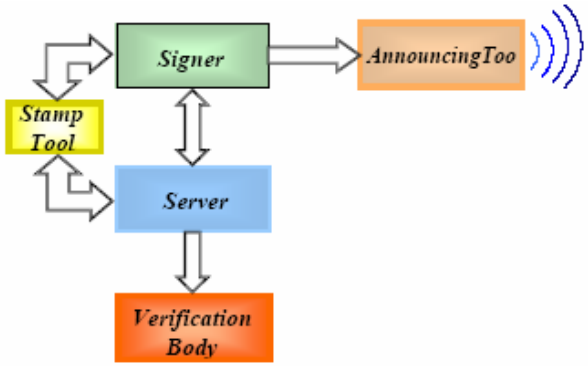

Figure 3: Block diagram of DSFS architecture

The central "actor" is a classical file server, or more generally, document server. It stores and publishes documents, but also contains DSFS material (document stamps, public keys, etc). DSFS material on server is secured by the classical file - system security policy. It means that it is liable to classical file-system attack. This is the reason why server is used just as secondary DSFS-material container. Also, there is a signer of the document, which is an authorized person that could be later interested in verification of the document authenticity. The signer uses stamp tool (or signing tool) for document signing (sealing). In our case it is the smart card based tool. After process of document signing, DSFS material has to be safely published. This is performed by announcing tool. Verification body verifies document authenticity. This is usually document signing initiator. Finally, there is user, the most active member of the system. He or she uses (none) signed documents.

4.2.1 Scenario of System Usage

User uses the system, almost in a same way as in a non-signed system. If the document is signed, only reading of the sealed document or data is allowed to the user. Writing activity may be allowed by the system, if the document is not sealed, otherwise user will influence the consistency of the system (i.e. the seal will be broken).

4.2.2 Server Specification

We can imagine server as a standalone or network file server. It provides the following functions:

Reading of the document – activities for signed document do not differ from the unsigned ones. If the sign of the document is broken, user should be warned about it, and the system should

provide an opportunity to give up from reading activity.

Writing of the documents – if the document is signed, it has to be forbidden to user to proceed with writing operation.

Associating stamp to the document – upon creating the sign for a given document, signer will forward it (using special protocol) to the server.

Reading sign associated to the document – reading the sign and forwarding to the verification body.

4.3 Document Signing

Signing algorithm has the following phases:

Phase 1 (system –> signing tool/stamp tool):

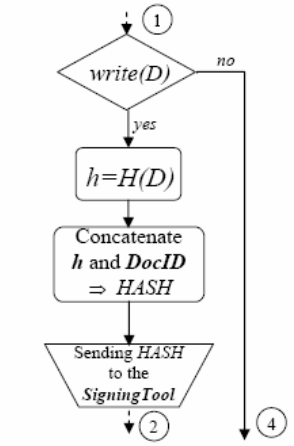

Figure 4: Phase 1 signing algorithm

First, server has to check the type of the incoming request: document writing request, associating stamp to the document, or document read request. In the case of document read request whole signing process is skipped. In the case of seal creation or writing request server initiates signing process; hash *h* of the document *D* has to be created [7]. Next step is concatenation of the document identifier (*DocID*), the unique document identification (it is necessary to have an evidence of all signed documents) with calculated hash *h,* creating document *HASH* and sends it to the stamp tool (Fig. 4). The following components exist in the data, which signing tool expects from the system: *DocID*, *HASH* – hash value of the document that is requested.

Phase 2 (signing tool):

After receiving *HASH*, stamp tool concatenates *HASH* with the time stamp, creating *THASH* object [6]. After that, stamp tool checks if it has already received this *THASH* before. If the answer is yes, error is reported, because *THASH* uniquely identifies specific document. Additional *THASH* insertion would imply the possibility of the document resigning, which is not allowed. If the answer is no, stamp tool can finally generate private/public key pair KP_U, KP_R [13]. The next step is *THASH* encryption with generated private key, producing digital signature *DS* [5]. When DS is generated there is no need for private key existence, so it can be destroyed (Fig. 5).

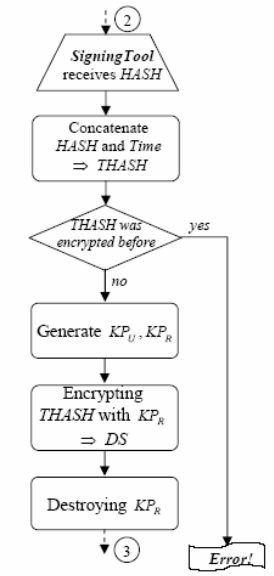

Figure 5: Phase 2 signing algorithm

The following components exist in the signed data stored on server: *Digital Sealed Document* – consists of DocID, document itself, Time Stamp and Encrypted HASH, *Publishing Material* – consists of DocID (identical with the previous one), Time Stamp (also identical with the previous one) and generated public key corresponding to the destroyed private key.

Phase 3 (signing tool –> system):

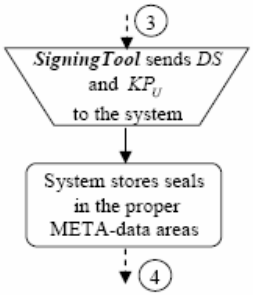

Figure 6: Phase 3 signing algorithm

Signing tool forwards seals to the system together with the public key (Fig. 6). When server receives DSFS material from stamp tool, it stores material inside the DSFS META-data area. DSFS META-data area should not be specifically implemented (an ordinary XML document is quite acceptable).

These are the following components of the data, which signing tool gives as output: the first one is *Digital Seal* – consisting of encrypted *DocID*, *Time Stamp* and *HASH*. DocID is identical with the DocID mentioned above. Time Stamp is generated by signing tool, using internal real time clock to provide information of signing time. HASH is encrypted with already destroyed private key. Another component is *Publishing Material* which consists of DocID (identical with the previous one), Time Stamp (also identical with the previous one) and generated public key,

corresponding to the destroyed private key. Public key is the main reason for the existence of the publishing material. If there is possibility that someone can replace pubic key, than he or she can modify document, calculate new hash, choose new key pair, encrypt time and hash with new private key, store them in DSFS META-data area, and replace old public key with the new one.

In general, all three phases can be represented as followed:

$$X: h = H(D)$$
$$HASH = Concatenate[h, DocID]$$
$$X \rightarrow Y: HASH$$
$$THASH = Concatenate[HASH, Time]$$
$$Y: DS = E_{KP_R}[THASH]$$
$$Y \rightarrow X: DS, KP_U$$

X - System,
Y - SigningTool
DS - Digital Seal

4.4 Publishing Scenario

The next step is publishing DSFS material to the real publishing storage. A type of publishing storage depends on what publishing policy is being chosen. Here are presented three techniques of public key publishing.

4.4.1 Publishing on a Public Site

In this scenario, anybody who is interested in document integrity (verification body), could access to one of those public sites, retrieve DSFS material, and check document integrity. However, this policy is liable to attack, since we use classical server technologies for publishing. It is required to make all public sites resistant to afterwards public key publishing of the existing document. This approach is acceptable for some level of security. It is not difficult to create an adequate service with append-only and random-read features. Also, some kind of secure communication between publishing tool and public sites is required.

We can improve publishing security using distributed and non-correlated sites. In this case Verification Body should access to as many as possible public sites and retrieve from each of them DSFS material for the specific document. If all of them are not equal it means that there is possibility that someone tried to obstruct document integrity.

4.4.2 Publishing with the Smart Card

In this scenario, it has to be used two different smart cards: primary (signing tool smart card) and secondary (publishing tool smart card). We use secondary smart card for storing public keys and distributing them to each user of DSFS. On a document verification request, associated corresponding DSFS material should be returned. The most secure solution is when signing tool smart card publishes directly to the publishing tool smart cards. It has to be defined some secure broadcast protocol which will transfer DSFS material created on signing tool to publishing tool [11]. Number of publishing tool smart cards must be equal or greater comparing with the number of parties interested in verification of a document.

Problem with this solution is requirement that all secondary smart cards have to be updated. Also, there is possibility that someone can replace a publishing smart card with identical (from the interface point of view) smart card, with different DSFS material.

4.4.3 Publishing with the Signing Tool

The most secure process would be storing DSFS material inside the signing tool. This means that signing tool is also the publishing tool. This is the most secure solution, since there is no need for open communication between signing tool and publishing tool. Also, in this case we can easily check existence of the same HASH in the process of document signing. However, in this case the problem that can appear is limited storage space.

4.5 Verification Scenario

Depending on the technique applied in public key publishing, there are various verification methods.

Figure 7 shows simplified process of making signed hash.

First HASH value of the current document has to be calculated as HASH'. Than, DSFS material has to be used to extract original HASH using adequate public key which is also stored in DSFS material. If the original HASH is the same as the calculated HASH', that means the document is not modified after it was signed. Otherwise it is modified, and error should be reported.

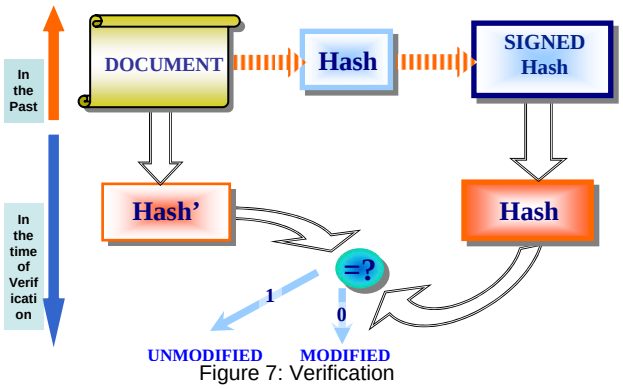

Figure 7: Verification

The first verification model requires from system to calculate the HASH of the current state of the document and to compare it with the HASH stored in META-data area.

The second one is based on forwarding necessary publishing material (retrieved from publishing tool) to the system and system calculates new values. This happens when verification body suspects in reliability of DSFS META-data area, and not in the system reliability.

The most secure case is when verification body requests download whole document from the server in order to personally check the

document integrity using DSFS material obtained from publishing tool.

4.6 Potential Problems

In this part it would be underlined some of the potential problems of this system.

4.6.1 Communication Snooping

Smart card – server communication is basically a serial communication which means that there is no real danger of snooping activities. There is only possibility of hooking in somewhere in operating system.

Snooping smart card – publishing tool communication is similar with the previous one. In the case of second policy we have serial communication, so there is no danger of snooping activities. In the last publishing scenario signing tool and publishing tool are the same. It means that there is no possibility of snooping.

4.6.2 Distribution

One of the possible problems is replacing publishing smart card with a new one. Protection against this is time stamp, which is generated inside the smart card. We can use unique smart card ID to detect smart card replacement.

4.6.3 Intra Snooping

First possibility of *copying content* of current smart card is technically impossible. Another possibility is *scanning content* of smart card. We can scan electro-magnetic field to find out what is the values of stored bits. To scan more precisely, we can use liquid oxygen to slow down changes in smart card. Also there are *snoop activities* inside the smart card, using statistic methodology. On the other hand, technology progress almost eliminated any possibility of those kind of scanning and snooping, so we can presume that we are completely safe using smart cards.

5. CONCLUSION

One of the advantages of our system is that it does not require any special equipment. It can be implemented on conventional servers/desktops that are extended only with classical smart card reader.

In conclusion, the proposed approach provides needed security of the document, if it is assumed that the security of the system components (i.e. smart card) is guaranteed. The proposed solution is of interest to those who have to prevent imperceptible changing of data.

REFERENCES

[1] Schneier, B., “Applied Cryptography,” 2nd edition, *Wiley*, 1996.

[2] Diffie, W., Hellman, M.,”New Directions in Cryptography,” *IEEE Transactions on Information Theory*, 1976.

[3] Apvrille, A., Hughes, J., Girier, V., “Streamed or Detached Triple Integrity for a Time Stamped Secure Storage System,”

[4] Wang, Y., Zheng, Y., “Fast and Secure Magnetic WORM Storage Systems,” www.verisign.com/resources/wp/enterprise/management/management.pdf

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

[6] Apvrille, A., Hughes, J., “A Time Stamped Virtual WORM System,”

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

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

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

[10] HP Automated Storage Division: “Safeguarding Data with WORM: Technologies, Processes, Legalities and Standards,” *Technical report*, Hewlett Packard, 1999.

[11] StorageTek: “Encryption Key Management System using multiple Smart Cards,”

[12] The ISO 7816 Smart Card Standard, http://www.cardweb.com.tw/card/iso/ISO7816.htm

[13] Blaze, M., “Key Management in an Encrypting File System,” AT&T Bell Laboratories, www.crypto.com/papers/cfskey.pdf

[14] VerSign, “Enterprise Key Management,” *VerSign White Paper,* www.verisign.com/resources/wp/enterprise/management/management.pdf