New Technique to Insure Data Integrity for Archival Files Storage (DIFCS)

Mohannad Najjar
Computer and Information Technology
University of Tabuk
Tabuk, Saudi Arabia

***Abstract*—In this paper we are developing an algorithm to increase the security of using HMAC function (Key-Hashed Message Authentication) to insure data integrity for exchanging archival files. Hash function is a very strong tool used in information security. The algorithm we are developing is safe, quick and will allow the University of Tabuk (UT) authorities to be sure that data of archival document will not be changed or modified by unauthorized personnel through transferring in the network; it will also increase the efficiency of network in which archived files are exchanged. The basic issues of hash functions and data integrity will be presented as well.**

In this research: The developed algorithm is effective and easy to implement using HMAC algorithm to guarantee data integrity for archival scanned documents in the document management system.

Keywords—cryptography; hash functions; data integrity; authentication; HMAC; file archiving

I. Introduction

Information and data security in the different systems at UT are one of the most critical issues for the university authorities. Ensuring data in these systems are not modified in an unauthorized fashion is a fundamental goal. UT departments use different kinds of information systems: the academic system, the ERP system, document management system etc. All of these systems don't have any tool to guarantee the integrity of their data.

Data Integrity is one of the fundamental components of information security. Data integrity is a tool used to insure that data (documents, messages, emails, files, etc.) can't be changed, modified, deleted by unauthorized personnel, thereby insuring accuracy and consistency.

When a message is sent through the local network or Internet to a Receiver; data integrity tools are used to insure that the message was not altered and that it is identical to that sent from the Sender. There are many tools to insure data integrity, such as: parity bit, checksum, encryption and hash functions. Hash functions are one of the most used tools because of simplicity, speed and being free of charge.

Insuring Data Integrity is already an important tool used in data exchange in telecommunications and networking systems. For UT the use of DIFCS (Data Integrity File Checking System) algorithm will guarantee that data stored in all applications will be safe and reliable. This solution also will increase the safety of the university information systems, in a convenient and effective method. Additionally the DIFCS algorithm will increase the effectiveness of the whole files archive system.

We depend in our improved DIFCS algorithm on using HMAC function to insure data integrity, authentication and we will add additional improved techniques to increase the effectiveness of the algorithm in the local network.

List of important symbols used in the paper:

H	The hash function, MD5 or SHA-1
B	The number of bits in the block in the hash function
IV	The initial value for the hash function
M	The data input to HMAC
Y*i*	The *i*th block of m, 0≤*i*≤(l-1)
L	The number of blocks in m after padding
N	The length of hash code
K	The secret key, if K length is greater than *b* then K=*h*(K)
K+	The K padded with zeros on the left so the result has *b* bits
ipad	The inner pad; the byte 36 (in hexadecimal) repeated b/8 times
opad	The outer pad; the byte 5c (in hexadecimal) repeated b/8 times
h(*m*)	The value of the HMAC; the length of the data is n bits, where the maximum value for n depends on the hash function used, MD5 or SHA-1
Y	Set of all possible hash results

II. Hash Functions

Hash function is a function h: M→Y that has, as a minimum, two properties:

- it compresses a sequence $m \in M$ of bits of arbitrary length, including the empty sequence, into a sequence $h(m) \in Y$ of the constant (fixed) length,
- for any $m \in M$ it is easy to compute $h(m)$.

The hash function transformation of the message $m = m_1 \| m_2 \| \ldots \| m_t$ divided into fixed length blocks $m_1, m_2, \ldots, m_t$ can be described as follows (see Fig. 1):

$$H_0 = IV,$$

$$H_i = \varphi(m_i, H_{i-1}) \text{ for } i = 1,2,\ldots,t;$$

Where; IV is an initial value, H_i is a chaining variable, φ is a compression function (also called a round function) and ψ is an output transformation. As a result we obtain h(m) of fixed length. In cryptographic literature [2,5] the resulting sequence $h(m)$ has been given a wide variety of names: hash result, hash code, hash total, imprint, fingerprint, message digest, cryptographic checksum, authenticator, authentication tag, compression, compressed encoding, condensation, Message Integrity Code (MIC), etc. In the sequel $h(m)$ will be called hash result.

The structural model of the hash function is presented in Figure 1. [2]. It works well if the length of m_t is of the same length as each previous block $m_1, m_2, \ldots, m_{t-1}$. If it is not a case then extra bits must be appended to an input string before hashing to make m_t as long as $m_1, m_2, \ldots, m_{t-1}$.

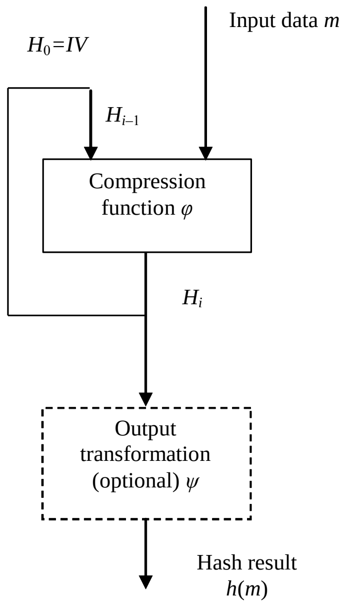

Input data m
$H_0=IV$
H_{i-1}
Compression function φ
H_i
Output transformation (optional) ψ
Hash result $h(m)$

Fig. 1. General model of the hash function *h*

III. Data Integrity

Any information system is deemed secure if it has at least three properties: Confidentiality, Data Integrity and Availability. So data integrity is one of the most important aspects of security according to data.

It insures the accuracy and consistency of data stored or transmitted from one point to another. There are many methods for insuring data integrity: physical and logical.

Physical tools like RAID (Redundant Array of Independent Disks). And logical like parity bit, CRC, Checksum, Encryption and Hash functions. In our paper we will improve a logical tool that will use hash function to insure data integrity of archived documents and files.

We will focus on insuring data integrity by using hash functions. And we will explain some algorithms that use hash functions (by using SHA-256 hash algorithm) to insure data integrity and (something more like) authentication and confidentiality.

Algorithm1:

Process file m_j by using a hash function SHA-256 h to calculate hash result $h(m_j)$. Save file m_j in the archive folder and save $y_j=h(m_j)$ in the secure folder of hash results. When you want to read m_j from its original folder then hash m_j by the same hash function h to calculate actual $x_j=h(m_j)$. If $y_j=x_j$ then the file was not changed, if not then the file was changed.

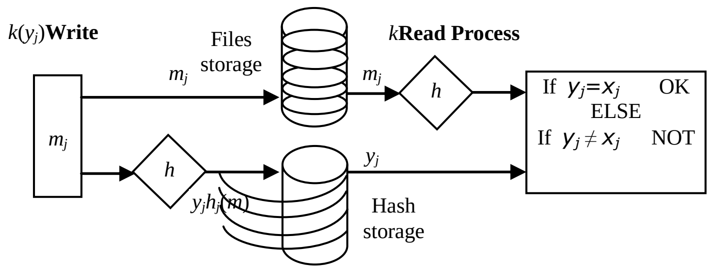


Fig. 2. Algorithm1

In this algorithm it is required to download the original file and hash result each time from the files storage and the hash storage, which are usually located on server decreasing the effectiveness of the whole reading process. Also there is no confidentiality for the files, or authentication for the source of the file where Man in the Middle attack can be a big threat.

Algorithm2:

Process file m_j by using a hash function h to calculate hash result $h(m_j)$ and encrypt it by using private key k_d. Save file m_j in the archive folder and save $k_d(y_j)= k_d(h(m_j))$ in the secure folder of hash results. When you want to read m_j from its original folder then hash m_j by the same hash function h to calculate actual $x_j=h(m_j)$. and decrypt $k_d(y_j)$ by using system public key k_e to recover y_j. If $y_j=x_j$ then the file was not changed if not then the file was changed.

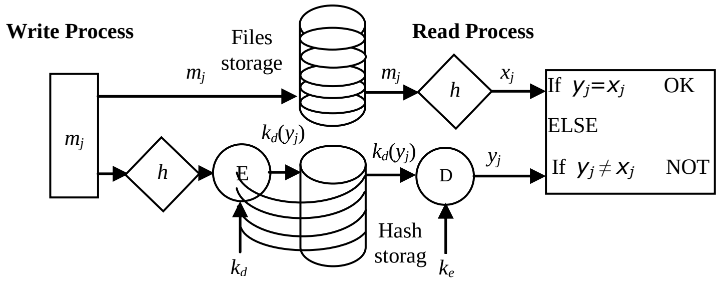


Fig. 3. Algorithm 2

Algorithm 3:

Process file m_j by using a hash function h to calculate hash result $h(m_j)$ and encrypt it by using secret symmetric key k. Save file m_j in the archive folder and save $k(y_j)= k(h(m_j))$ in the secure folder of hash results. When you want to read m_j from its original folder then hash m_j by the same hash function h to calculate actual $x_j=h(m_j)$. Decrypt $k(y_j)$ by using same symmetric key k to recover y_j. If $y_j=x_j$ then the file was not changed, if not then the file was changed.

In this algorithm it is required to download the original file and hash result each time from the files storage and the hash storage, which are usually located on server decreasing the effectiveness of the whole reading process. Also there is no confidentiality for the files. In the other hand, authentication of file source is insured.

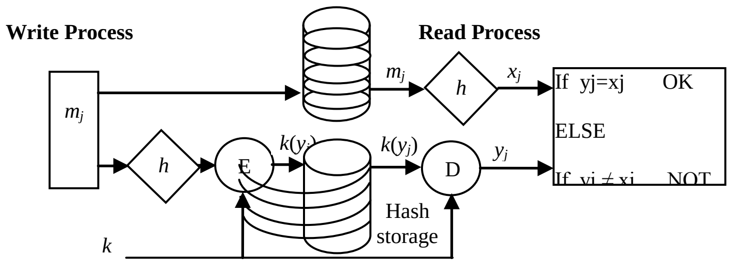


Fig. 4. Algorithm 3

Algorithm 4:

Pad secret p serial of bits to m_j and then process file $m_j||p$ by using a hash function h to calculate hash result $h(m_j||p)$. Save file m_j in the archive folder and save $y_j= h(m_j||p)$ in the secure folder of hash results. When you want to read m_j from its original folder then pad secret p serial of bits to m_j and hash $m_j||p$ by the same hash function h to calculate actual $x_j=h(m_j||p)$. If $y_j=x_j$ then the file was not changed, if not then the file was changed.

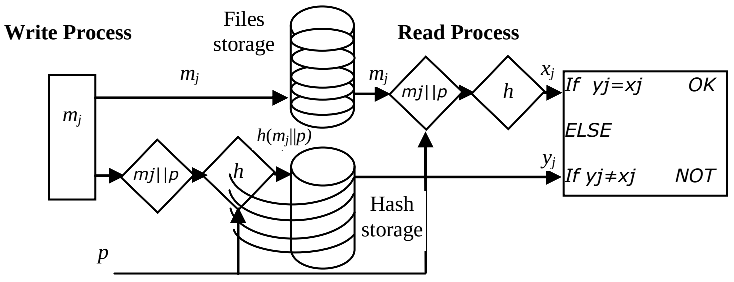


Fig. 5. Algorithm 4

In this algorithm it is required to download the original file and hash result each time from the files storage and the hash storage, which are usually located on server decreasing the effectiveness of the whole reading process. Also there is no confidentiality for the files but the authentication of file source is insured. Additional powerful cryptographic characteristic is fulfilled, where for $m_1= m_2$ then $h(m_1)\neq h(m_2)$.

If we want to make the saved files secret we can apply an additional operation where we encrypt m_j by using symmetric or asymmetric encryption algorithm.

In this paper we will use a special case of the fourth algorithm, where we will use HMAC (Key-Hashed Message Authentication code), which is used as an authentication cryptographic tool.

IV. HMAC

The main goals behind the HMAC construction [20] are:

- To use available hash functions without modifications; in particular, hash functions that perform well in software, and for which the code is freely and widely available.
- Preserve the original performance of the hash function without incurring a significant degradation.
- Use and handle keys in a simple way.
- Gain a well-understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the underlying hash function, and to allow easy replacement ability of the underlying hash function if it will be faster or more secure.

HMAC requires a cryptographic hash function, which we denote by h, and a secret key K. We assume h to be a cryptographic hash function where data is hashed by iterating a basic compression function on l blocks of data. We denote by b the bit-length of such blocks (where $l*b$ equal to the length of m in bits after padding), and by n the bit-length of hash outputs (n=128 bits for MD5, n=160 bits for SHA-1). The authentication key K can be of any length up to b, the block length of the hash function. Applications that use keys longer than b bits will first hash the key using h and then use the resultant n bit string as the actual key to HMAC. In any case the minimal recommended length for K is n bits (as the hash output length).

HMAC can be calculated as follows (Fig. 6):

$$\mathbf{HMAC}_K(m) = h(K^+ \text{ XOR } opad,\ h(K^+ \text{ XOR } ipad, m))$$

It can be done in 7 steps:

1) Append zeros to the left end of K to create a b-bit string K+.

2) XOR (bitwise exclusive-OR) K+ with ipad to produce the b-bit block Si.

3) Append m to Si.

4) Apply h to the stream generated in step 3.

5) XOR K+ with opad to produce the b-bit block S0.

6) Append the hash result calculated in Step 4 to S0.

7) Apply h to the stream calculated in step 6 and output the result.

Because of using such different fixed values of *ipad* and *opad* and doing two times hashing function we avoid the situation where the XORing operation between K^+ and *ipad* or K^+ and *opad* to have zero's value.

Keys

The key for HMAC [21] can be of any length (keys longer than *b* bits are first hashed using *h*). However, less than *n* bits is strongly discouraged as it would decrease the security strength of the function. Keys longer than *n* bits are acceptable but the extra length would not significantly increase the function's strength. A longer key may be advisable if the randomness of the key is considered weak.

Keys need to be chosen randomly (or using a cryptographically strong pseudo-random generator seeded with a random seed), and periodically refreshed. Current attacks do not indicate a specific recommended frequency for key changes as these attacks are practically infeasible. However, periodic key refreshment is fundamental security practice that helps against potential weaknesses of the function as well as the keys, and therefore limits the damage of an exposed key.

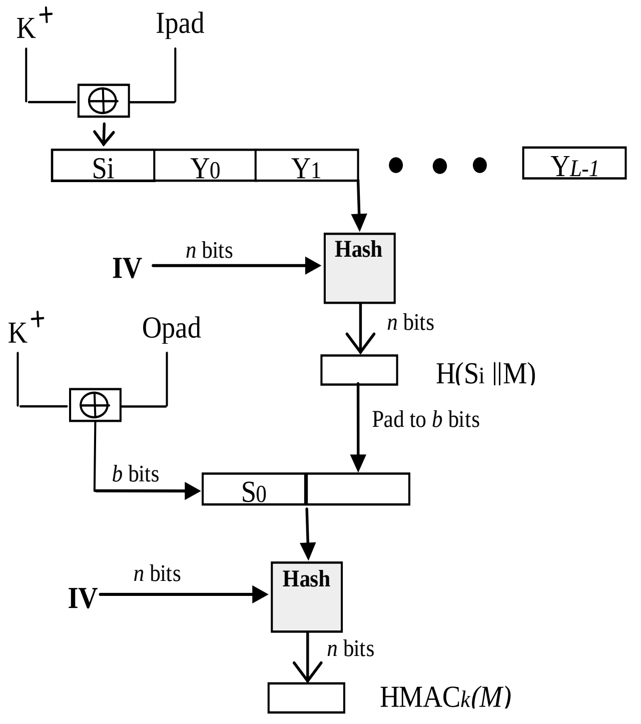
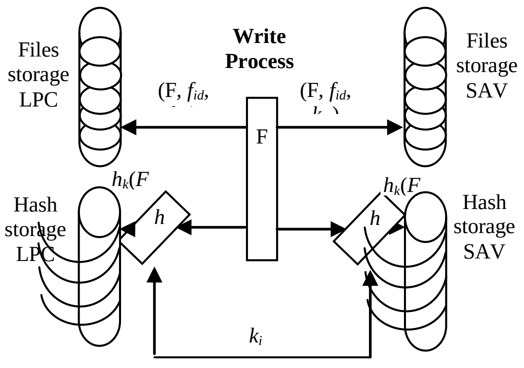


Fig. 6. HMAC function

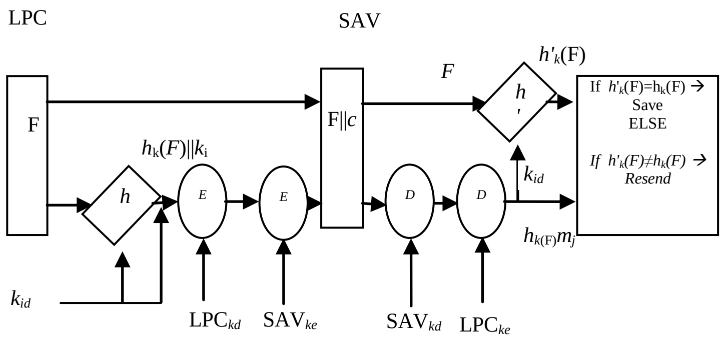
V. Data Integrity File Checking System (DIFCS)

We will focus in our research on insuring data integrity by using HMAC [19]. As HMAC is open to use any hash function with it. So in our paper we recommend to use at least SHA-256, which still secure against brute-force attack. In the future we recommend using even hash results with 1024 bits length. In any document management system, each department in the organization has to archive its uploaded files in a central archival warehouse. In the implementation of such solution we will face two important issues: the insuring of data integrity for archived files through transmission and the performance of the network where the transfer of these files is done from the server to the local computers.

Usually each department has an access to its own archived files only and not to the files of the whole archival warehouse. The improved algorithm we developed depends on this factor, that most of the retrieved files requested by the department's user are usually uploaded by the same department.



(a)
Fig. 7. Sending and saving process of file F to SAV



(b)

Fig. 7. Sending and saving process of file F to SAV

In this paper we are implementing an efficient algorithm to insure data integrity and authentication for the archived files and at the same time to insuring better performance for the network. HMAC algorithm will be used to insure data integrity and authentication and a temporary local storage on local PC of most used archival files, which will increase the efficiency of the network.

In the proposed solution uploaded files will be saved in two storage devices: in the local PC of the uploaded user (LPC) and in the Central Archive Server (SAV). Additionally in SAV and LPC we will apply HMAC with a secret key.

Uploading process:

When the user uploads F on his LPC, this file is saved in the temporary matrix storage on LPC and it is also sent and saved in SAV server. This saving process is explained in fig. 7, where each file F will have unique identifier f_{id} identifying F in a unique way on LPC and SAV. LPC will calculate hash result h_{kid}(F) for F by using HMAC algorithm and random secret unique key k_{id} then it will encrypt h_{kid}(F) and k_{id} by using User public key LPC$_{kd}$ to insure authentication and then result is encrypted by SAV public key SAV$_{ke}$ to insure confidentiality.

Encrypted result *c* and F together are sent through network to SAV.

On SAV encrypted *c* is decrypted by using SAV private key SAV$_{kd}$ and then again decrypted by using LPC public key LPC$_{ke}$ to recover h_{kid}(F) and k_{id}. By using k_{id} recovered from *c* SAV calculates hash result h'_{kid}(F) for F by using HMAC with key k_{id}. SAV compares recovered sent hash result h_{kid}(F) of F with the calculated one h'_{kid}(F), If they are equal then F and f_{id} and k_{id} are saved on SAV else SAV must sent a request to retransmit all again from LPC.

Downloading process:

We will have two situations, when file F with f_{id} and k_{id} exist on LPC, where only LPC will request for h_{kid}(F) from SAV, fig 8. (a). And second one when you have only f_{id}, Where we need file F with f_{id} and k_{id} and h_{kid}(F), fig. 8. (b).

When LPC requires a file F with f_{id} identifier from SAV, the following steps will be done:

1) Check if file F with f_{id} and k_{id} exist on LPC, If yes then go to 2 else go to 7,

2) Send a request to the SAV with f_{id} to retrieve hash result $h_{kid}(F)$,

3) SAV Search for $h_{kid}(F)$ according to f_{id},

4) SAV sends hash result $h_{kid}(F)$ and f_{id} to LPC m= $h_{kid}(F) \parallel f_{id}$),

5) LPC retrieve k_{id} and calculates hash result $h'_{kid}(F)$ for F,

6) If $h_{kid}(F)= h'_{kid}(F)$ then retrieve F from LPC and end, else go to 7,

7) LPC sends request to the SAV with f_{id} to retrieve

8) SAV Search for F according to f_{id} ,

9) SAV sends F and hash result $h_{kid}(F)$ and f_{id} and secret k_{id} encrypted by LPC public key LPC_{ke}, (m= F $\parallel$ $h_{kid}(F) \parallel f_{id} \parallel LPC_{ke}(k_{id})$),

10) LPC receives m= F $\parallel$ $h_{kid}(F) \parallel f_{id} \parallel LPC_{ke}(k_{id})$ and recovers k_{id} by using the private key of LPC_{kd},

11) User application on LPC calculates the hash result $h'_{kid}(F)$ for F,

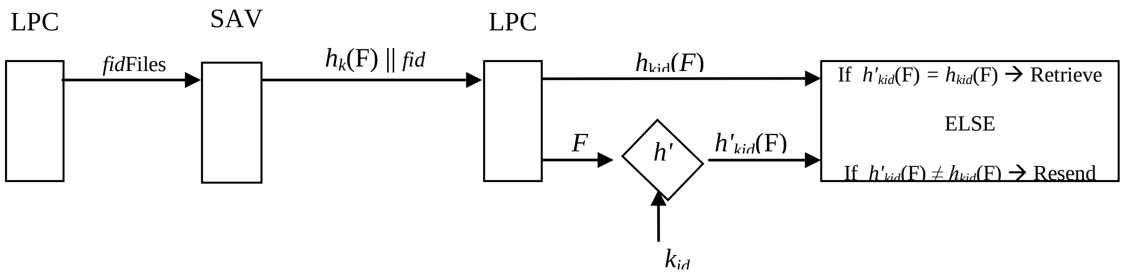
12) If $h_{kid}(F)= h'_{kid}(F)$ then retrieve F from LPC and save F and $h_{kid}(F)$ and fid and kid on LPC, else file is corrupted and resend again.

If additional security is required like confidentiality then symmetric key algorithm is used to insure confidentiality to F. Public key algorithm will be used to exchange the secret key between SAV and user working on the LPC.

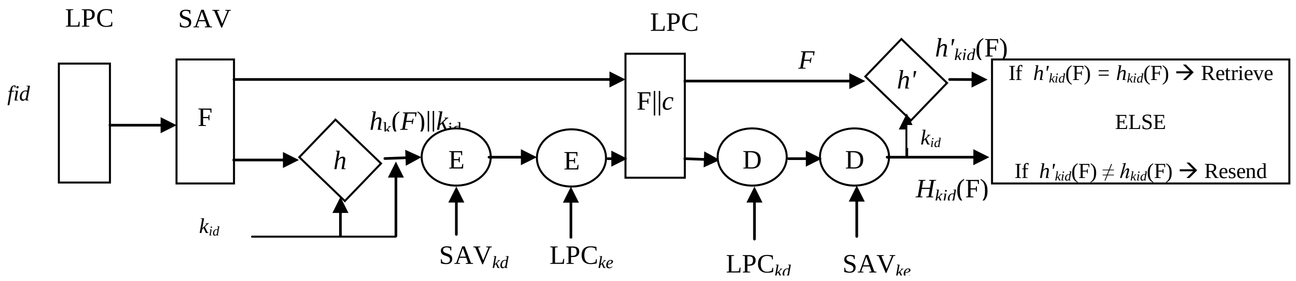
In our improved algorithm DIFCS we increased the cryptographic characteristics of the whole process of saving the file and its hash result on server and reading the files and their hash results from the same server. If we will compare the developed algorithm cryptographic characteristics with the other mentioned algorithms in this paper we can easily conclude the following:

a. In DIFCS algorithm the original file is saved on the local machine so it is not required to download each time the original file from the files storage located usually on server, which increases the effectiveness of the whole archive file retrival process.

b. Authentication of file source is insured.


LPC
SAV
LPC
fidFiles
h_k(F) || fid
h_kid(F)
F
h'
h'_kid(F)
k_id
If h'_kid(F) = h_kid(F) → Retrieve
ELSE
If h'_kid(F) ≠ h_kid(F) → Resend

(a)



(b)

Fig. 8. Downloading process from SAV

c. Additional powerful cryptographic characteristic is fulfilled, where for if we have two messages m1 and m2, where m1= m2 then h(m1)≠ h(m2).

d. Confidentiality for the files or hash results can be implemented according to the user requirements

VI. Conclusions

In this research we developed a new algorithm called DIFCS, which uses HMAC function to insure data integrity and authentication for archival file systems. DIFCS also uses a new technique for retrieving and checking if the archive files are authentic. The main function of DIFCS is to increase the efficiency of the files archival system and the local network. Such an algorithm insures data integrity for archived files and makes them immune against unauthorized manipulation and Man in the Middle attack. It also insures authentication between LPC and SAV.

In future work, we will develop the algorithm to make it a distributed algorithm: where archival files will be distributed and saved in different places according to a known mechanism. Such a development will increase the efficiency of the system.

References

[1] R.C. Merkle, A Certified Digital Signature. In proceedings of Advances in Cryptology, Lecture Notes in Computer Science (435), Springer-Verlag, California, USA, 1989, pp. 218-238.

[2] Menezes A. J., van Oorschot P.C., Vanstone S. A., Handbook of Applied Cryptography. CRC Press, Boca Raton, FL, 1997.

[3] Pieprzyk J., Sadeghiyan B., Design of Hash Algorithms. LNCS 756, Springer, Berlin, 1993.

[4] Wayner P., Digital Cash. AP Professional, Bostan, 1996.

[5] Preneel B, The state of the cryptographic hash functions. Damgård I. (ed.), Lectures on Data Security. Modern Cryptology in Theory and Practice. LNCS 1561, Springer, Berlin, 1999, 158−182.

[6] Qu C., Sebbery J., Pieprzyk J., On the symmetric properties of homogeneous bent functions. Pieprzyk J., Safavi-Naini R., Seberry J. (eds.), Information Security and Privacy. LNCS 1587, Springer, Berlin, 1999, 26−35.

[7] R. Tamassia, N. Triandopoulos, On the Cost of Authenticated Data Structures. In Proc. European Symposium on Algorithms, LNCS (2832), Budapest, Hungary, 2003.

[8] Y.H. Chen, E.J. Lu, Design of a secure fine-grained official document exchange model for e-government, Information & Security 15(1), 2004, pp. 55-71.

[9] J. Woerner, H. Woern, A security architecture integrated co-operative engineering platform for organised model exchange in a Digital Factory environment, Computers in Industry 56(4), 2005, pp. 347-360.

[10] G. Yee, Y. Xu, L. Korba, K. El-Khatib, Privacy and Security in ELearning, Future Directions in Distance Learning and Communication Technologies. Idea Group, Inc. 2006.NRC Publication Number: NRC 48120.

[11] IBM, 2008. Data integrity. Available at:

[12] http://publib.boulder.ibm.com/infocenter/tpfhelp/current/index.jsp?topic =/com.ibm.ztpf-ztpfdf.doc_put.cur/gtps5/s5dint.html (Accessed 12 December 2008).

[13] H. Maruyama, K. Tamura, N. Uramoto, Digest Values for DOM (DOM-HASH), RFC2803. Available at: http://www.landfield.com/rfcs/rfc2803.html (Accessed 13 November 2008).

[14] Bret Mulvey, Evaluation of SHA-1 for Hash Tables, in Hash Functions. Accessed April 10, 2009.

[15] Boritz, J. Efrim. "IS Practitioners' Views on Core Concepts of Information Integrity". International Journal of Accounting Information Systems. Elsevier. http://www.fdewb.unimaas.nl/marc/ecais_new/files/boritz.doc. Retrieved 12 August 2011

[16] Trust and Privacy in Digital Business: Third International Conference, TrustBus 2006, Krakow, Poland, September 4-8, Springer 2006, Proceedings (Lecture Notes in Computer Science / Security and Cryptology, ISBN-13: 978-3540377504.

[17] RFC1321, The MD5 Message Digest Algorithm, R.Rivest, April, 1992.

[18] FIPS-180-1, SHA-1 Secure Hash standard algorithm, April, 1995.

[19] Mihir Bellare Ran Canettiy Hugo Krawczykz, "Keying Hash Functions for Message Authentication", Crypto 96 Proceedings,Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

[20] H. Krawczyk, M.Bellare, R. Canetti HMAC: Keyed-Hashing for Message Authentication, RFC 2104, 1997.

[21] H. Krawczyk, M.Bellare, R. Canetti: Message Authentication using Hash Functions – The HMAC Construction, CryptoBytes, Vol. 2, No. 1 Spring 1996.

[22] NIST FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication Issued March 6, 2002.

[23] RFC4868, Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec, S. Kelly, S. Frankel, May,2007.