Chapter 16
Techniques to Certify Integrity and Proof of Existence for a Periodical Re-encryption-Based Long-Term Archival Systems

A. H. Shanthakumara and N. R. Sunitha

Abstract The periodical re-encryption-based archival systems have many specific characteristics such as actively re-encrypting the stored data objects periodically with or without conscious to the owner. For such a system, traditional techniques cannot be applied to check the integrity and proof of existence of a data object. For example, most traditional systems check integrity and proof of existence by comparing hash value of a data object stored in the archival system to the corresponding hash value with the owner. It may not be realistic solution due to continuous change in the bit patterns of the archivals due to periodical re-encryption. Therefore, we present a solution that is not only suitable to a specific periodical re-encryption-based archival system but also to any existing storage systems from long-term point of view.

Keywords Certify Integrity · Proof of Existence · Archival Systems

16.1 Introduction

The impact of digital information environments has been remarkably universal, with the generation of vast quantities of information. Sometimes these data are stored in a special infrastructure called digital preservation or archival system. Along with newly produced data, many countries have digitalized and stored the documents which are on papers to save physical space. The land registers [4, 18] in Europe and the digitization of records of high courts in India [20] are to name a few. The main goal of such a system is to secure the long-term persistence of information in digital form.

There are many reasons to keep on changing the bit patterns of the stored data objects periodically in the archival systems [1], with or without conscious to the

A. H. Shanthakumara (✉) · N. R. Sunitha
Siddaganga Institute of Technology, Tumakuru, Karnataka, India

N. S. V. Rao et al. (eds.), *Proceedings of International Symposium on Sensor Networks, Systems and Security*, https://doi.org/10.1007/978-3-319-75683-7_16

owner of the data object. Today, we have several protection solutions based on cryptography. However, such solutions are not guaranteed to be secure in future as computer power and cryptanalysis evolve [24]. For example, we can use quantum computer techniques to attack applications currently using RSA signatures. Also, there is a possibility of strengthening the cryptographic security with advanced computation power by increasing the size of the key or block size. In order to protect the data for long term, there can be a change in key or key size or block size or cryptographic algorithm itself or format transformations resulting in change of bit pattern of data.

This work is based on our previous work, which is the periodical re-encryption-based archival system [23]. The archived data object may transform into several versions in a short period of time. In order to preserve the assurance of integrity all the way back to that of the original data object is a real challenge. Traditionally, integrity is checked by comparing the stored hash value with a newly computed hash from a data object being archived. In periodical re-encryption-based system, the bit patterns of the stored data object will keep on changing periodically. Hence it is not a feasible solution to compare newly computed hash value to the stored hash value with the owner in order to ensure integrity. We have many systems [8, 10, 12, 16, 22, 25] which address this problem through reregistration process of a data object to the archival system when it is re-encrypted with new credentials. It also involves the verification of the integrity and authenticity before the data object is transformed into a new format or version. This process is not practical, because the owner or organization who attached the data objects may not be online always or may no longer be available over a long period in order to authenticate during reregistration process. The data stored in archival systems which provides periodical re-encryption is useful, if their integrity (data objects are unaltered) is protected over a long term and also a proof of existence (a time reference when the data object is witnessed) is provided.

16.2 Related Work

In this section, we describe some common integrity checking and proof of existence techniques traditionally used in a digital archival systems.

16.2.1 Cryptographic Hash Function

Cryptographic hash function, as defined by Wenbo Mao [14], is a deterministic function which maps a bit string of an arbitrary length to a bit string of fixed length called a hash value or digest or simply hash and satisfies important properties: (a) hash value $h(x)$ for an input x should be computationally indistinguishable from a uniform binary string in the interval $(0, 2^{|h|})$ (Mixing transformation); (b) it should be computationally infeasible to find two inputs x and y with $x \neq y$ such that $h(x) = h(y)$ (collision resistance); and (c) given a hashed value h, it should be computationally infeasible to find an input string x such that $h = h(x)$ (pre-image resistance). The security of traditional archival systems relies on the hardness of defeating one of the hash function properties. Therefore, today's secured hash function becomes insecure in future as cryptanalysis evolves, and thus, no single hash function can be secure from long-term point of view[15].

16.2.2 Digital Signature

A digital signature scheme consists of three algorithms [15, 16, 25]. An efficient key generation algorithm generates private and public key such that the message m is encrypted using private key that can be decrypted by using public key or vice versa. An efficient signing algorithm generates a signature on a given message m and by using private key. An efficient verification algorithm verifies and decides the signature is valid or not. RSA [21] and variants of ElGamal [6, 13] are popular digital signature schemes.

16.2.3 Merkle Tree

A Merkel tree [15, 17] is one of the most widely used hash linking schemes. The leaves of the Merkle binary tree are the hash values of the data objects being processed. The value stored at each internal node is the concatenated hash values of the children. The value computed at the root of the tree is called Merkle root, which represents the compressed value of all the data objects to be processed.

For example [8, 22], let us consider four data objects D_1, D_2, D_3, and D_4 with a corresponding hash values $h1$, $h2$, $h3$, and $h4$ which are to be processed at a time T_0. The corresponding Merkle tree is shown in Fig. 16.1.

The values of the internal nodes are obtained by $h12 = h(h1||h2)$, $h34 = h(h3||h4)$, and Merkle root $MR = h(h12||h34)$. To check the proof of data object D_2 whose hash value is $h2$, the required path is $D2 \rightarrow h2 \rightarrow h12 \rightarrow MR$. We can compute MR mathematically $h(h(h1||h2)||h34)$ with the information $h1$, $h2$, and $h34$. This information is called authentication path to the data object D_2. The

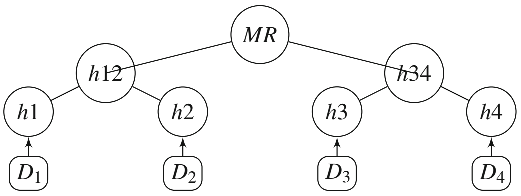


Fig. 16.1 Merkle tree with time stamp

compressed value MR is used as a proof of existence of a data object. Change to any of the data objects will result in a different MR value.

16.2.4 Time Stamp

Whenever an archivist, who manages the data objects, has a copy to be time-stamped, he or she transmits the hash value of a data object to a trusted time stamping authority (TSA). An authority records the date and time the hash value was received and retains a copy for safekeeping. Any challenger can check the integrity by comparing the archivist hash value with the TSA record [2, 7, 8, 11, 15, 22].

16.2.5 Evidence Record Syntax (ERS)

The Evidence Record Syntax (ERS), proposed by Gondrom et al. in RFC 4998 [8, 15], holds an evidence for each data object. The evidence record contains a sequence of certificates issued by archivist, and each certificate contains time stamp issued by a trusted time stamping authority (TSA) based on the archivist request on a Merkle root. Each time stamp covers a group of data objects by using Merkle tree.

16.2.6 One-Way Accumulators

One-way accumulator [3] is a function which takes two arguments from comparably sized domains and produces a result of similar size. In other words, it is a quasi-commutative hash function with two input arguments and produces fixed-size digest. The combination of quasi-commutative and one-way properties is used to develop one-way accumulator. The desired property of one-way accumulator is obtained by considering function $Acc : X \times Y \rightarrow X$ and asserting that for all $x \in X$ and for all $y1,y2 \in Y$, $Acc(Acc(x, y1), y2) = Acc(Acc(x, y2), y1)$.

16.2.7 Patricia Trees

It is a space-optimized data structure in which each node with a single child is merged with its parent [9]. It supports comparably high-speed search and insertion of a new node to the data structure. Unlike regular trees, the key at each node is compared considering a group of bits, where the quantity of bits in that group at that node is an r-ary tree. It is binary when r is 2. The example as shown in Fig. 16.2 represents a binary tree containing the strings 000, 011, 100, and 101. A new string is easily inserted into a data structure as each node represents a unique string and

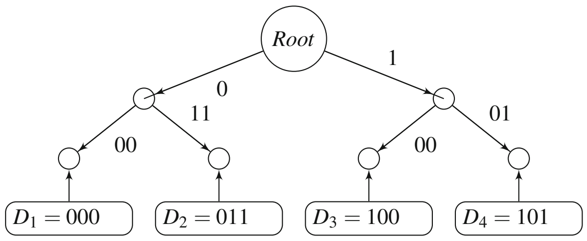


Fig. 16.2 Patricia tree containing the strings 000, 011, 100, and 101

its position is uniquely determined by its value. The root node represents an empty string, and leaf node represents the actual string in a data structure. The searching is as easy as tree traversal starting from a root and following left path if the bit of a string is 0 and, otherwise, the right path. The process is repeated until string is found or all bits of a string are exhausted.

16.3 Proposed Scheme

We propose a scheme by considering three actors in an archival system: data owner, who generates data object that needs to be archived for long-term usage; an archivist, who manages the archived data object in a secured manner; and a trusted middle layer between the owner and an archivist, which is defined as below.

16.3.1 Lightweight Integrity Management Layer (IML)

It computes necessary operation to provide proof of integrity and existence of data object. It uses an XOR operation to compute proof of integrity. In order to provide a proof of existence of a data object, it computes Merkle root on a special data structure called Patricia tree. The special feature of this data structure is to support time stamping scheme [19]. The value for proof of existence of a data object is computed based on the Patricia tree over a group of hash value of a data object.

16.3.2 Scheme Description

The basic computations required are defined in Fig. 16.3. The data owner computes $h \leftarrow Hash(D_i)$ before archiving the data object D_i. The notations B_i and A_i indicate the hash values of data object D_i before and after the encryption/re-encryption, respectively. The record $E_i \leftarrow (F_i, S_i)$ provides proof of integrity, and

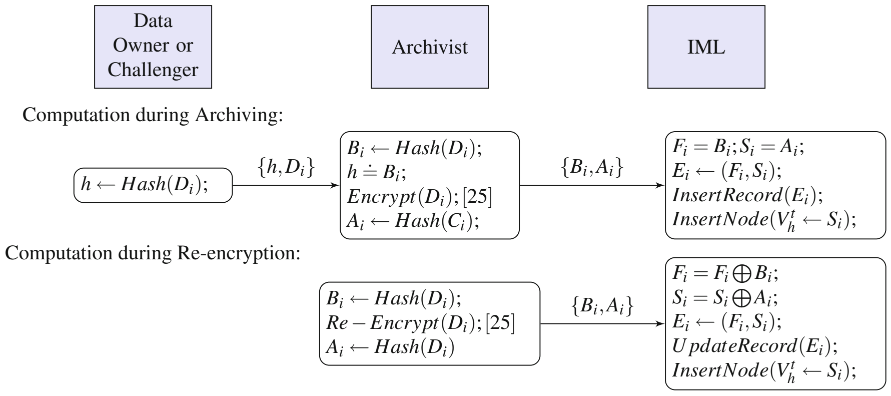

Data Owner or Challenger
Archivist
IML
Computation during Archiving:
$h \leftarrow Hash(D_i);$
$\{h, D_i\}$
$B_i \leftarrow Hash(D_i);$ $h \doteq B_i;$ $Encrypt(D_i);[25]$ $A_i \leftarrow Hash(C_i);$
$\{B_i, A_i\}$
$F_i = B_i; S_i = A_i;$ $E_i \leftarrow (F_i, S_i);$ $InsertRecord(E_i);$ $InsertNode(V_h^t \leftarrow S_i);$
Computation during Re-encryption:
$B_i \leftarrow Hash(D_i);$ $Re-Encrypt(D_i);[25]$ $A_i \leftarrow Hash(D_i)$
$\{B_i, A_i\}$
$F_i = F_i \bigoplus B_i;$ $S_i = S_i \bigoplus A_i;$ $E_i \leftarrow (F_i, S_i);$ $UpdateRecord(E_i);$ $InsertNode(V_h^t \leftarrow S_i);$

Fig. 16.3 The basic computations

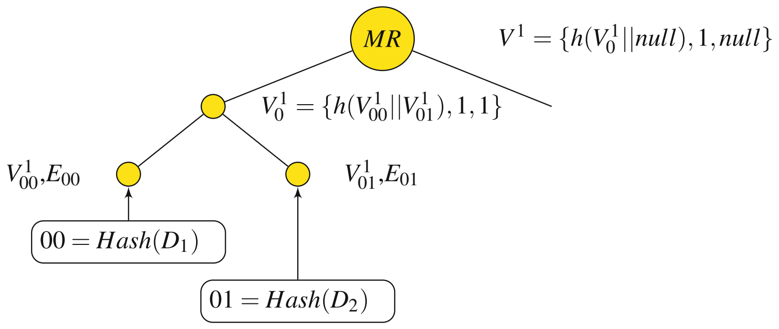
the node V_h^t computed and inserted to Patricia tree at the time interval t will be used to confirm the existence of data object D_i.

Any challenger or the owner, who wishes to confirm the integrity of the data object D_i, can collect the record E_i from IML and current hash value A_i from the archivist. He/she computes $F_i \bigoplus A_i \doteq S_i \bigoplus h$ to confirm the integrity of the data object. If it does not match, there is a loss of integrity somewhere in the re-encryption stage. In order to find the exact interval where the integrity was lost, IML must store the record E_i for all re-encryption stages.

We organize our data structure based on Patricia tree. It is implemented as binary tree indexed by original hash value, h, computed at the time of archiving to enable efficient search. The internal nodes of the tree store small metadata that mainly contains a set of computed hash values by using hash values of its children at different stages. This set of hash values at each node is ordered by time intervals t. It also stores two additional values to represent left and right child records used at the time of computing hash value. Specifically, we represent it with the notations $V_n^t = \{h(V_l^t || V_r^t), l, r\}$. The record V is stored at the node n during the time interval t. The symbols l and r indicate the left and right child records used to calculate hash value of the node. The leaf node of the tree holds one additional record E to store the values for proof of integrity.

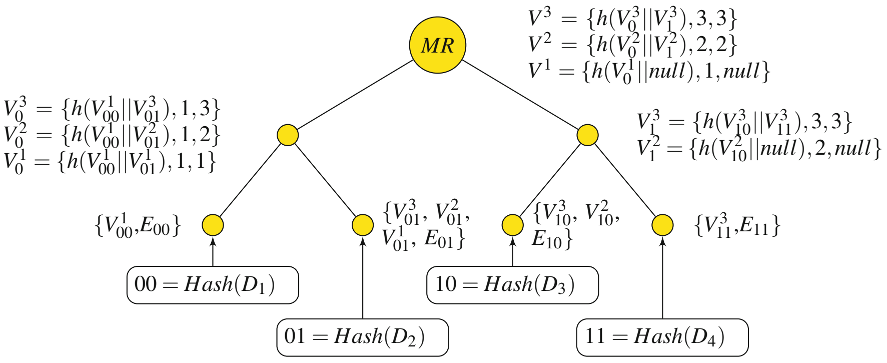
Let us consider the two data objects D_1 and D_2 that are to be archived at a time T_1. Let us assume that 00 and 01 are hash values of data objects D_1 and D_2, respectively. The IML computes the values for proof of integrity and updates the records to the corresponding internal nodes of the tree. At each leaf node, top record V holds the current hash value of the data object and, bottom record E holds the values for proof of integrity. Figure 16.4 shows a data structure at time T_1. The value of the record at root node $V^1 = \{h(V_0^1 || null), 1, null\}$ will be used as Merkle root. Authentication path is computed with the records V_{00}^1, V_{01}^1, and V^1.

Similarly, in a time interval T_2, let us assume that the data object D_2 is re-encrypted and data object D_3 with hash value 10 is inserted. In a time interval T_3,



MR
$V^1 = \{h(V_0^1||null), 1, null\}$
$V_0^1 = \{h(V_{00}^1||V_{01}^1), 1, 1\}$
V_{00}^1, E_{00}
V_{01}^1, E_{01}
$00 = Hash(D_1)$
$01 = Hash(D_2)$

Fig. 16.4 Data objects D_1 and D_2 are archived at time interval T_1



MR
$V^3 = \{h(V_0^3||V_1^3), 3, 3\}$
$V^2 = \{h(V_0^2||V_1^2), 2, 2\}$
$V^1 = \{h(V_0^1||null), 1, null\}$
$V_0^3 = \{h(V_{00}^1||V_{01}^3), 1, 3\}$
$V_0^2 = \{h(V_{00}^1||V_{01}^2), 1, 2\}$
$V_0^1 = \{h(V_{00}^1||V_{01}^1), 1, 1\}$
$V_1^3 = \{h(V_{10}^3||V_{11}^3), 3, 3\}$
$V_1^2 = \{h(V_{10}^2||null), 2, null\}$
$\{V_{00}^1, E_{00}\}$
$\{V_{01}^3, V_{01}^2, V_{01}^1, E_{01}\}$
$\{V_{10}^3, V_{10}^2, E_{10}\}$
$\{V_{11}^3, E_{11}\}$
$00 = Hash(D_1)$
$01 = Hash(D_2)$
$10 = Hash(D_3)$
$11 = Hash(D_4)$

Fig. 16.5 Data structure at the time interval T_3

the data objects D_2 and D_3 are re-encrypted, and the data object D_4 with a hash value 11 is inserted. Figure 16.5 shows the corresponding data structure.

The records at the root of the tree can be used as Merkle root and, index of the record can be considered as time stamp at which the Merkle root is constructed. The owner or any challenger can construct Merkle root with the help of authentication path and specific time interval as record index and confirms the existence of their data object. For example, in order to confirm the existence of D_2, the records V_{01}^3, V_{00}^1, V_1^3, and V^3 are required. Similarly, to confirm D_1, the records V_{00}^1, V_{01}^1, and V^1 are required. The IML periodically removes the records which are out of scope in order to reduce the space overhead. For example, with reference to Fig. 16.5, at the time interval T_3, all records with index 2 (time interval 2) except V_{10}^2 are irrelevant and can be removed from the tree. The record V_{10}^2 is required to prove that the data object D_3 with hash value 10 is archived at a time interval T_2.

16.4 Implementation

We implemented IML layer using Java code on a desktop machine and performed some experiments using Enron's employees email data set [5]. These experiments run on an algorithm called a re-encryption [23]. We implemented Patricia tree-based data structure to hold the records in IML layer. The following functions describe the overview of IML layer:

1. The **SearchNode** enables to search the data object given by hash value. It is basically traversing from root node to corresponding leaf node in the tree.
2. The **InsertNode** enables to insert a new node to the tree during new archival.
3. The **InsertRecord** enables to insert a new record to a node. It could be used when there is a change in the metadata of the node.
4. The **UpdateRecord** enables to update the values for proof of integrity after re-encryption of a data object occurred.
5. The **DeleteRecord** enables to delete a record of a node. It is used when IML wants to remove irrelevant records from the tree.

The IML maintains a special list to make a note on the data objects, which are undergone in the process of re-encryption. Periodically, IML calculates Merkle root on hash values of all those data objects and accordingly inserts new records to the corresponding nodes in the data structure.

16.5 Results and Discussion

In the long-term archival system, it may not be possible for the owner or client to be online always in order to receive and store the certificates issued by the archivist. Therefore, client server-based systems may not be strongly advisable solutions for long-term and re-encryption-based archival systems.

The major problem of the ERS schemes [8] is the size of linking information. The size of the metadata is linearly increasing with the number of data objects of an archival system. In addition, it may not be a practical solution due to frequent re-encryption events of the data object.

The major problem of one-way accumulator-based scheme is the high complexity involved in computing a one-way accumulation on every re-encryption of a data object. If the archival node takes one microsecond per data object to generate a certificate with a new accumulator value, it could process only around 8% data objects compared to ERS schemes.

The major complexity is eliminated by using simple XOR operation in our scheme. The simulation of IML shows that it could process 2–3% more data objects than ERS scheme in the time period of 30 ms. We selected 90,000 emails randomly of Enron's employees with an average size of 1.9 KB. We treat each email with all its attachment as a single data object. The simulation started with ten initial data objects

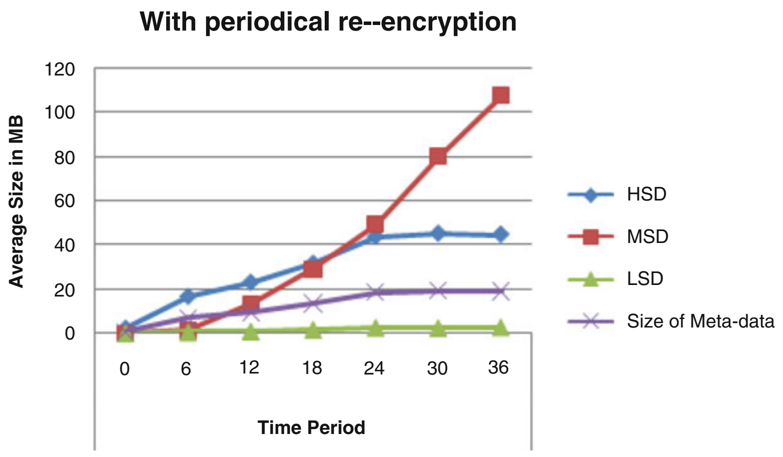


Fig. 16.6 Storage requirements for data objects and metadata when considering the periodical re-encryption

and uniformly added as defined in [23]. On every millisecond (a time interval), IML calls to calculate the proof of existence among the hashes of the data objects which are re-encrypted during that time interval.

Over 5 runs of simulation of periodical encryption algorithm [23] could process up to 81,400 emails in 30 time intervals, out of which nearly 23,600 data objects are highly sensitive data objects (HSD). It also shows that there is a linear increment in medium sensitive data objects (MSD) over a period of time. Figure 16.6 shows the storage requirements for data objects and metadata. It shows that the required size of the metadata for our scheme is stabilized to around 20 MB as stabilization is observed with respect to HSD objects. Figure 16.7 shows the storage requirement of metadata that is linearly increasing with respect to number of data objects in the absence of periodical re-encryption. It shows that our scheme could also be used with traditional system.

The IML scheme is lightweight and easy to implement on any periodical re-encryption-based archival systems. Storage space required for storing the information with respect to integrity checking and proof of existence is comparatively reduced.

16.6 Conclusion and Future Work

The obsolescence of hash functions or cryptographic security algorithms affects the solutions of archival systems from long-term point of view. The traditional, reregistration process of data object to an archival system whenever the version or format changed may not be a realistic solution. Ensuring the integrity and proof of

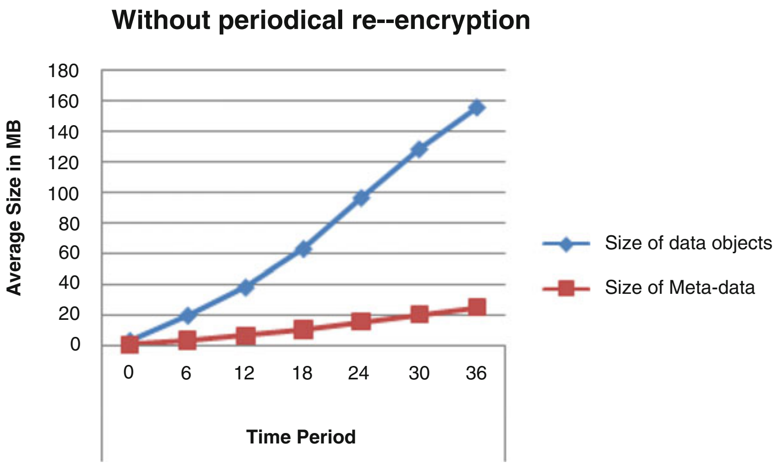


Fig. 16.7 Storage requirements for data objects and metadata when without considering the periodical re-encryption

existence of a data object whenever the owner needs confirmation is a real challenge of any archival systems.

In this paper we have shown the challenge and need of a specific technique to certify long-term integrity and proof of existence for a periodical re-encryption-based archival system. Our scheme uses most suitable time stamping technique and focus more on realistic implementation in the archival systems. The experiments demonstrate that metadata required to provide proof of integrity and proof of existence are compact in size, which is reaching 20 MB for 81,000 emails. The simulation results show that the storage requirement for metadata is stabilized over a period of time instead of linearity. This scheme is very lightweight and easy to implement. The owner need not to store all the certificates issued by the archivist, and data object integrity and proof of existence are easily verifiable by any challenger at any point of time.

The unfocused issue in our scheme is that if the data object is altered, it is unable to generate a caution alarm message to authorized person or archivist.

References

1. Baker, M., Shah, M., Rosenthal, D.S.H., Roussopoulos, M., Maniatis, P., Giuli, T., Bungale, P.: A fresh look at the reliability of long-term digital storage. In: Proceedings of EuroSys 2006, pp. 221–234, April 2006
2. Bayer, D., Haber, S., Stornetta, W.: Improving the efficiency and reliability of digital time-stamping. In: Sequences II: Methods in Communication, Security, and Computer Science, pp. 329–334. Springer, Berlin (1993)

3. Benaloh, J., de Mare, M.: One way accumulators, a decentralized alternative to digital signatures. In: Advances in Cryptology - EUROCRYPT '93, LNCS 765, pp. 274–285. Springer, Berlin (1994)
4. Centre of Registers and Information Systems: e-Land Register. http://www.egov-estonia.eu/e-land-register. Accessed 20 Jan 2015
5. Cohen, W.: Enron email dataset. http://www.cs.cmu.edu/~enron. Accessed 20 Jan 2015
6. Gamal, T.E.: On computing logarithms over finite fields. In: Williams, H.C. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 218, pp. 396–402. Springer, Berlin (1985)
7. Giuli, P., Maniatis, P., Giuli, T., Baker, M.: Enabling the long-term archival of signed documents through time stamping. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies. FAST '02, USENIX Association, pp. 31–46 (2002)
8. Gondrom, T., Brandner, R., Pordesch, U.: Evidence Record Syntax (ERS) Securing Electronic Business Processes, in RFC 4998, pp 367–375 (2007)
9. Goodrich, M., Tamassia, R., Hasic, J.: An efficient dynamic and distributed cryptographic accumulator. In: Proceedings of Information Security Conference (ISC). Lecture Notes in Computer Science, vol. 2243 , pp. 372–388. Springer, Brelin (2002)
10. Haber, S.: Content integrity service for long-term digital archives. In: Archiving 2006, pp. 159–164. IS&T, Ottawa (2006)
11. Haber, S., Stornetta,W.S.: How to time-stamp a digital document. J. Cryptol. **3**(2), 99–111 (1991)
12. Haber, S., Kamat, P., Kamineni, K.: A content integrity service for digital repositories. In: 3rd international conference on open Repositories, Southamton, UK (2008).
13. Johnson, D., Menezes, A., Vanstone, S.A.: The elliptic curve digital signature algorithm (ecdsa). Int. J. Inf. Sec. **1**(1), 36–63 (2001)
14. Mao, W.: Modern Cryptography Theory and Practice. Hewlett Packard Professional Books, 1st edn. Prentice Hall, Upper Saddle River (2006)
15. Martin, A., Gagliotti, V., Daniel, C., Alexander, W., Johannes, B.: Authenticity, integrity and proof-of-existence for long-term archiving: a survey. IACR Cryptology ePrint Archive, p. 499 (2012)
16. Martin, A.G. Vigil, Moecke, C.T., Custdio, R.F., Volkamer, M.: The Notary Based PKI A Lightweight PKI for Long-Term Signatures on Documents. Public Key Infrastructures, Services and Applications. Lecture Notes in Computer Science, vol. 7868, pp. 85–97. Springer, Berlin (2013)
17. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO '89. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer, Berlin (1989)
18. Ministere de la Justice: Livre Foncier. https://www.livrefoncier.fr. Accessed 20 Jan 2015
19. Oprea, A., Bowers, K.D.: Authentic time-stamps for archival storage. IACR Cryptology ePrint Archive 2009, p. 306 (2009)
20. RFP for Scanning and Digitization of Records of High Court of Himachal Pradesh. http://hphighcourt.nic.in/pdf/TenderDigitiz21112014.pdf. Accessed 20 Jan 2015
21. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
22. Sangchul, S., Joseph, J.: Techniques to audit and certify the long-term integrity of digital archives. Int. J. Digit. Libr. **10**(2–3), 123–131 (2009)
23. Shanthakumara, A.H., Sunitha, N.R.: A novel archival system with dynamically balanced security, reliability and availability. In: 2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), Bangalore, pp. 27–33. IEEE (2016). https://doi.org/10.1109/CSITSS.2016.7779435
24. Storer, M.W., Greenan, K.M., Miller, E.L.: Long term threats to secure archives. In: Proceedings of the 2006 ACM Workshop on Storage Security and Survivability (2006)
25. Troncoso, C., De Cock, D., Preneel, B.: Improving secure long-term archival of digitally signed documents. In: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability. StorageSS '08, pp. 27–36. ACM, New York (2008)