Available online at www.ijpe-online.com

International Journal of Performability Engineering

Vol. 13, No. 5, September 2017, pp. 711-720
DOI: 10.23940/ijpe.17.05.p14.711720

Approach of Tamper Detection for Sensitive Data based on Negotiable Hash Algorithm

Jing Lin*, Chuqiao Mi, Yuanquan Shi

School of Computer Science and Engineering, Huaihua University, Huaihua, 418000, Hunan, China

Abstract

Sensitive data is a very important to information safety. The real-world sensitive data is often illegally altered because database administrators (DBAs) have special identity and permissions in database system. However, the traditional secure measures, such as user authentication and access control, do not work well for them. For this case, it is necessary to identify effectively whether the sensitive data in database in enterprise trusted domain is illegally altered or not. Therefore, combining active detection at the security server with passive detection at the security client, a detection approach of the tampered sensitive data based on negotiable hash algorithm is proposed in this paper. Experiments show our algorithm can performs well for sensitive data tamper detection, and it is adapt to protect sensitive data in medical database.

Keywords: Tamper detection; Negotiable hash algorithm; Sensitive data; Medical data

(Submitted on March 24, 2017; Revised on June 29, 2017; Accepted on August 21, 2017)

1. Introduction

In many applications, such as public health service platform (e.g., medical data) and competitive intelligence system, a lot of sensitive data are stored in internal databases and frequently maintained by their owners in enterprise trusted domain. At the same time, outsourced database (ODB) model is also adopted by some owners. Although outsourced data is vulnerable to various internal and external threats which challenge the data integrity, more serious secure issues of sensitive data are arising because of the important data in databases being easily changed illegally and hard detected, which leads to be a challenging task of tamper detection for sensitive data.

About secure data, most studies on tamper detection are aiming at coarse-grained images and files using digital watermarking (fingerprinting) technique [1,3,12,17,23]. Researches on digital watermarking [4,7,8,10] for copyright protection, tamper proofing and authentication of relational databases have been extensively conducted in decades. In some works, the approach was proposed to detect the tampered sensitive data by using one-way hash algorithms [2,9,14], which tackled the tampered problem after important data was illegally altered in database and mitigated the vulnerability against collusion attacks [15,16,19] using only one single algorithm.

Due to the external untrusted service providers, preserving the security of the outsourced databases becomes another popular research topic today [5,6,11,13,18,20,21]. Some works consider the standard SQL queries involving SELECT clauses, which typically result in selection of a set of records matching a given predicate [11,13]. For record-level integrity, MAC-s is used in the ODB model. But in the more general Multi-Querier and Multi-Owner models, MAC-s are not useful since they would require all owners and legitimate queriers share the MAC key [11]. Existing immutable signatures, encryption and decryption in public key infrastructure are very computation and communication costly [5,13,21,22], which makes them impractical for real-life applications [21], and unsuitable to the multi-user setting [22].

* Corresponding author.
E-mail address:LinjingL99@126.com.

Although traditional secure measures, such as user authentication and access control, can sometimes protect data safety, it is still very difficult to prevent DBAs from deliberately modifying sensitive data. The detail reasons are as follows: In MAC-s model, in particular, the data owner needs to possess a secret key and apply an HMAC to all entries shipped to the database. In contrast, a digest per record is merely stored in database in our method. Whenever a record is retrieved, its integrity can be verified. Therefore, we focus on the tamper detection of the fine-grained sensitive data updated and accessed frequently by the internal users, especially DBAs, in in-house database. Their manipulation behavior should be checked and audited effectively for further safety of sensitive data.

The rest of this paper is organized as follows. In Section 2, we introduce some related work. Hash detection model, including the model design, approach principle and sensitive data structure, is described in Section 3. In Section 4, we present the negotiable security layer and its implementation. Experimental results are given in Section 5. The paper is then concluded with a brief summary.

2. Related Work

Different protective measures should be applied to different data. Watermarking is an information hiding technique to identify the sources of data. Fingerprinting is a technique that inserts digital marks into data to identify the recipients who have been provided with data. Both techniques can help protect data from piracy which has become a severe threat to database applications. Watermarking is often used to preserve the copyright of the digital asset, such as multimedia, software and database. Zhang et al. [23] presented a dual watermarking algorithm for medical images with copyright protection and content authentication. Waleed et al. [1] proposed an SVD audio watermarking which could embed a same watermark several times in audio signal, enhancing the detectability of the watermark in the presence of severe attacks. Stelvio et al. [3] designed a general watermarking model scheme combined with visual cryptography, which can provide some important solutions for tampering verification and the resolution of disputes on the ownership of a given image. On the other hand, for relational database, many studies were carried out in terms of copyright and data integrity protection using digital watermarking. For examples, Li, Gao, et al. [4,10] presented a fragile watermarking scheme for detecting malicious modifications. Hamadou et al presented a hybrid water-marking scheme for copyright protection and tamper proofing [8] and a fragile zero-watermarking technique for authentication [7]. Steve, Richard T et al. [12,17] used fingerprinting (digest) techniques for tamper identification of the audit log and file. But a significant matter of concern in fingerprinting is collusion attacks [2,14,19]. Lin et al. [9] applies MD5 hash function to detect the tamper of the record-level data in database.

Recently, database as a Service (DBaaS) has become the promising technology and ODB is becoming popular quickly. When the ODB service provider is potentially untrusted, sensitive data may have leaked. It is essential to provide adequate security measures to protect the data from both malicious outsider attacks and the Database Service Provider itself. An asymmetric fully homomorphic encryption algorithm using public key of the owner was designed to encrypt the data for its confidentiality, integrity and availability. Its disadvantage is high overhead of computation/communication and malpractice of the outsourced data is hard to detect [5]. Zhang et al. [22] proposed two efficient ID-based public auditing protocols for the outsourced data by combing Water's signature and public auditing for the data against the adversary's forgery attack in the standard security model. Immutable Authentication and Integrity Schemes for ODB was proposed to be resistant to single point vulnerability [21]. Due to high computation and communication overhead, outsourced scheme is impractical for real-life applications and not suitable to the multi-user scenario.

However, the scheme of using a single hash algorithm is deficient against collusion attacks. Therefore, we concentrate on the record-level data tamper detection in internal database using hash detection model with the negotiable hash algorithm and apply it to Medical Data.

3. Hash Detection Model

3.1. Model Description

To ensure the security of important data, such as user roles, password, permissions, privacy information and key business data etc., a hash detection model is presented (Figure 1) in this study. The model includes three layers as follows:

- The acquisition layer, accomplishing data collection and maintenance functions
- The negotiable security layer, composing of the hash negotiator, digester, security client and security server
- The data layer, saving data and responding to the data requests of users

The detection procedure using the model can be done by the following steps.

When the model runs the first time, the hash negotiator will negotiate a hash function only once, and the hash function is stored at step (1). The other components of the negotiable security layer can use it in the future. At step (2)-(4), the initial hash values are generated and assembled into R" by the digester, and then stored in database. The security client at step (5) and the security server at step (6) conduct strict tamper detection towards the sensitive data in the security layer, respectively.

Its principle can be described as follows:

By matching the hash values of a record, calculating twice using the hash algorithm, our approach can detect whether the data has been tampered or not. After the digester generates a legal initial hash value (i.e., Certification), the security client and/or the security server compare the hash value calculated from the current data with initial one (i.e., Verification). If differences were found out, the system will automatically print the information of alarm. In addition, the initial hash value can be legally updated by the digester. The minimal bandwidth overhead and the low computation overhead at the client make our model work well.

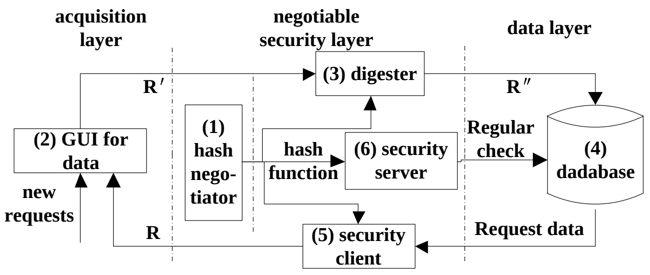

Figure 1. Hash detection model for sensitive data in internal database

3.2. Data Formalization

In our model, there are two identification tables: the protected data table and the system hash table. For brevity, we write the protected data table as T_R and the system hash table as T_H.

Let T_R ($a_1, a_2, \ldots, a_n$) be an attributes set. Where a_1 is primary key, in which the sequences of continual records are stored; a_2 means the logical deleted flag of invalid record; a_i ($3 \leq i \leq n$) denotes the other important attributes and n indicates the number of attributes of the protected table. *Supposed it is regarded as unlawful modification when any record is removed*. In order to check tampered data effectively, two new attributes (p and h) are added to T_R. Where p, a pointer of the negotiated hash algorithm, is used to save the ID of the algorithm; h denotes hash field, for storing the digest of sensitive data. Therefore, the schema of protected table T_R in this paper can be described as ($a_1, a_2, \ldots, a_n$, p, h).

Let T_H (b_1, b_2, b_3) also be formalized as an attribute set, where b_1 is a pointer of the negotiated hash algorithm; b_2 indicates a pointer of the protected data table; b_3 denotes a switcher controlling the negotiation mode in our approach. The negotiated hash algorithm, the name of the protected table and the negotiation mode are stored in hash table.

4. Negotiable Security Layer

4.1. Hash Algorithm Principle

One-way hash functions can process a message to obtain a condensed representation is called a digest. They enable the determination of a message's integrity: any change to the message will result in a different digest.

Taking SHA-1[1] for example, it is used to hash a message, here having a length of ℓ bits ($0 < \ell < 2^{64}$). Preprocessing will be carried out:

[1] SECURE HASH STANDARD, FIPS publication 180-2 (dated August 1, 2002). http://csrc.nist.gov/publications/ fips/fips180-2/fips180-2.pdf

1) Append the bit “1” to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation $1 + \ell + k \equiv 448 \bmod 512$.

2) Parse the padded message into N 512-bit message blocks, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$.

3) Set the initial hash value, $H^{(0)}$, consisting of the five 32-bit words in hex.

Then each message block ($M^{(1)}, \ldots, M^{(N)}$) is processed in order, the using steps can be described as follows:

```
For i = 1 to N:
{
    (1) Prepare the message schedule, {$W_t$}:
        $$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$
    (2) Initialize the five working variables, a, b, c, d, and e, with the $(i\text{-}1)^{st}$ hash value:
        $a = H_0^{(i-1)}; b = H_1^{(i-1)}; c = H_2^{(i-1)}; d = H_3^{(i-1)}; e = H_4^{(i-1)}$
    (3) For t = 0 to 79:
        {
          $T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t; e = d; d = c; c = ROTL^{30}(b); b = a; a = T$
        }
    (4) Compute the $i^{th}$ intermediate hash value $H^{(i)}$:
        $H_0^{(i)} = a + H_0^{(i-1)}; H_1^{(i)} = b + H_1^{(i-1)}; H_2^{(i)} = c + H_2^{(i-1)}; H_3^{(i)} = d + H_3^{(i-1)}; H_4^{(i)} = e + H_4^{(i-1)}$
}
```

After repeating steps one through four a total of N times, the resulting 160-bit digest of the message, M, is $H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}$.

Where *ROTL*^{n}*(.)* indicates circular left shift operation and $f_t(.)$, a sequence of logical functions, is defined as follows:

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (x \wedge z) & 0 \le t \le 19 \\ Parity(x,y,z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x,y,z) = x \oplus y \oplus z & 60 \le t \le 79 \end{cases}$$

where $\oplus$ is the exclusive-OR operation, $\wedge$ means bitwise AND operation. Other details can be seen in FIPS publication 180-2.

One-way hash algorithm is used is to produce message digests to detect tampered data. MD5, SHA series and RIPEMD are negotiable to be used in our model. This results in finding out the current algorithm more difficultly, which attenuates the vulnerability of using a single hash algorithm. Furthermore, hash algorithm is improved by shuffling its input, which is described as follows.

4.2. Improving Hash Algorithm

Due to the important data in internal database never being encrypted, for those who can update the data, it is easy to guess the hash algorithm in use according to the length of hash field value, which poses a great threat to the safety of these data. Therefore, a negotiation mechanism is introduced in our approach. Firstly, we negotiate a hash algorithm to calculate a digest, thus it increases the difficulty of discovering the hash algorithm for the modifier. Secondly, we refine the algorithm input by implanting hidden factor and shuffling the fields of records. The refinement can be suitable to a variety of hash algorithms, so that it is more difficult to know the input of algorithm. Hash value (R_h) is typically calculated using the formula (1) as follows:

$$R_h = hash(Di_{ins}(R_p), Ds) \qquad (1)$$

where $Di_{ins}(R_p)$ denotes an instance of digester with initiate parameter R_p, a pointer of the hash algorithm; *hash(.)* is the improved digest function; Ds indicates the algorithm input and is assigned a value using following rule (2):

$$Ds = shuffle(R_d, R_p, I_{anti}) \qquad (2)$$

where R_d (i.e., (a_1, a_2, ... , a_n)) is the protected data, and I_{anti} is hidden information, so it can be null. R_p, which is described in formula (1), can also be null. A default hash function will be used in our method, when R_p is null. And *shuffle(.)*, a function, can array a set of data according to the specific order.

Cost of cracking an improved algorithm can be calculated using the rule in the literature [9]. When the hash algorithm is unknown, the cracking process will cost more time. It illustrates that the improved algorithm is more advantageous in tamper detection.

4.3. Hash Negotiator

For hash negotiator, in our approach, there are three ways: global (DB-level), local (table-level) and frequent (record-level). In the global way, a hash algorithm negotiated only once at first run will be applied to protect all tables in our system. In the local mode, the same negotiated hash algorithm is shared by each of the protected tables. And in the frequent way, the negotiator is called once when a record is inserted into the table, and different hash function can be applied to different record, but it is rarely used in fact. In practice, depending on the specific situations, we can select a negotiation way and configure it from GUI of the system. The negotiated algorithm ID is stored in the system hash table and shared by the digester, security client and security server, the name of corresponding algorithm is judged in our program code.

4.4. Digester

After negotiating the hash algorithm R_p, a digester is instantiated with initiate parameter R_p. When the sensitive data record R' ($=R_d+R_p+R_h$) is input, firstly, the digester isolates the data R_d and R_p from R', and generates a new digest, H', using formula (1), and then it assembles the record R'' with R_d, R_p and H', finally, it outputs R'' ($= R_d+R_p+H'$). The core algorithm of digester is described as follows (Algorithm 1):

Algorithm 1 Digester Algorithm

```
Input: Record R′
Output: Record R"
1. $R_d \leftarrow$ getData(R′, a).  //to isolate the data $R_d$ from the input.
2. $R_p \leftarrow$ getData(R′,p).  //to isolate the hash function $R_p$ from the input.
3. Instantiate a digester Di with $R_p$.
4. $H' \leftarrow$ Hash(Di,shuffle($R_d$,$R_p$,getIanti())).  //implant "$I_{anti}$" and shuffle then generate a digest H′.
5. $R'' \leftarrow$ setData(R′, h, H′).  // to write H′ back to record.
```

4.5. Security Client

The security client is responsible for strict safety checks of sensitive data from the database. After a sensitive record, R, retrieved, at first, the client calls the digester to produce a new digest H′. Then it matches H′ with R_h, the initial one of the record. It would respond the request when successful match obtained. Besides printing the message of system alarm and writing the tampered record to the log file, when data tamper is detected, the request will be refused. No matter whether the sensitive data were modified or the digest was altered, it is impossible to escape from the security client. Therefore, it ensures that the application system can run safely at low cost. The client is designed to work in a passive way, which algorithm is as follows (Algorithm 2):

Algorithm 2 Security Client's Algorithm

```
Input: RecordSet Rs
Output: RecordSet Rs
1. for each R in Rs, do
2.   $R_d \leftarrow$ getData(R′,a).
```

```
3.  $R_p$←getData(R′,p).
4.  $R_h$←getData(R′,h).
5.  Instantiate a digester Di with $R_p$.
6.  H′←Hash(Di, shuffle($R_d$, $R_p$, getAntibody ())).
7.  **if** Compare(H′, $R_h$) **then**  // to compare the new digest with the initial one.
8.      Log(R).       // to write the tampered record R to the log file.
9.      Alarm(R).  // to alarm and deal with other functions.
10. **end if**
11. **end for**
```

4.6. Security Server

To take the initiative to find the tamper behavior, periodically, the security server starts a scan procedure for routine records test. The server and the client constitute a collaboration mechanism of active testing and passive testing together. This mechanism enhances the capability of the proposed approach to detect unauthorized tamper behavior in time. The security server's workflow is similar with that of the security client, which is described as below (Algorithm 3):

Algorithm 3 Pseudo-code of Check Procedure

```
**Input:**
**Output:**
1. Initialize a variable of waking up a check procedure: start←false.
2. Set time interval of waking up the check procedure.
3. Initiate timer(1), where 1: to start timer, 0: to stop timer, and start←true if the time interval is over.
4. **If** start is true **then**
5.     start the check procedure, namely check(String* pdt_ptr), to detect the protected data table, where pdt_ptr is a pointer of the protected
   data table.
6.     start←false.
7. **end if**
```

5. Experiments

In this section, we demonstrate the performance of our proposed model with respect to effectiveness by dealing with medical data. Our model has been implemented using JDK 1.7 and deployed in the Medical Information Integration Platform (MI2P). Performance experiments were run on 2 "IBM xSeries 365" servers with Xeon MP 2.7G CPU, 8G DDR and SCSI 600GB HD for developer. One acts as a database server that its DBMS is MySQL server 5.5. The other is web server on which Tomcat 6.5 was installed. Windows 2005 was installed on both servers.

5.1. Descriptions of Data Set

We use the real-world data from MI2P. In our model, a doctor's advice table, namely DoctorAdvice, was used. Our data set has 2,500 selected records in DoctorAdvice. Each record consists of 18 fields, such as serial_number (primary key), patient_name, department, content, dose, usage, frequency, doctor, nurse, checker, start_time, stop_time, conduct_time, hash_algorithm, hash_value, deleted_flag, and so on. They are shown in detail in Table 1.

We randomly selected 800/1000/900 records for illegal modification detection, added 350/500/400 new records for illegal insertion detection, and deleted 200/300/250 random records for illegal remove detection. Then, we randomly removed and last added 150/200/200 records for illegal tamper detection. The security client and the security server were executed to detect tampered data respectively.

5.2. Performance Experiments

The negotiated 160-bit SHA, 128-bit RIPEMD and 128-bit MD5 are used to conduct tamper detection, and the statistics is shown in Table 2, Table 3 and Table 4 respectively.

Figure 2 shows that primary key, deleted flag, digest, hash algorithm and the other fields were modified on purpose and the statistics about tamper detection is shown in this figure. Due to the particularity of the primary key (i.e., PK), what we should pay attention to is we don't have to modify. So the number of PK is 0. Only three hash algorithms were selected in our experiments, thus the number of updating algorithm is small, only 6.

Table 1. Description of DoctorAdvice fields

Field name	Type&length		Field name	Type&length
serial_number	varchar2(50)		frequency	varchar2(50)
patient_name	varchar2(40)		doctor	varchar2(100)
department	varchar2(40)		nurse	varchar2(100)
case_number	varchar2(100)		checker	varchar2(100)
advice_number	varchar2(40)		start_time	datetime
content	varchar2(2000)		stop_time	datetime
dose	varchar2(100)		conduct_time	datetime
usage	varchar2(40)		deleted_flag	varchar2(1)
hash_algorithm	varchar2(4)		hash_value	varchar2(512)

Figure 3 contrast statistics of detection performances using the three hash algorithms. Especially, the deletion operation cannot be checked out by the security client. Therefore, the bar of "delete" does not appear in the area of the "client" at the right of this figure.

S and C are the symbol of security server and security client respectively. FDLA denotes the abbreviation of "first delete last add" in Table 2, Table 3, Table 4 and Figure 3.

Table 2. SHA testing statistics

Testing way	Experiment 1 (160-bit SHA)							
	Add record/row		Update record/row		Delete record/row		First delete last add record/row	
	add	detect	update	detect	delete	detect	FDLA	detect
Security server	350	350	800	800	200	200	150	150
Security client	350	350	800	800	200	0	150	150
Success rate	100%		100%		S:100%; C:0%		100%	

Table 3. RIPEMD testing statistics

Testing way	Experiment 2 (128-bit RIPEMD)							
	Add record/row		Update record/row		Delete record/row		First delete last add record /row	
	add	detect	update	detect	delete	detect	FDLA	detect
Security server	500	500	1000	1000	300	300	200	200
Security client	500	500	1000	1000	300	0	200	200
Success rate	100%		100%		S:100%; C:0%		100%	

Table 4. MD5 testing statistics

Testing way	Experiment 3 (128-bit MD5)							
	Add record/row		Update record/row		Delete record/row		First delete last add record /row	
	add	detect	update	detect	delete	detect	FDLA	detect
Security server	400	400	900	900	250	250	200	200
Security client	400	400	900	900	250	0	200	200
Success rate	100%		100%		S:100%; C:0%		100%	

The experimental data in Table 2, Table 3 and Table 4 show:

(1) For any record added/inserted directly or updated optionally into the database by DBA, it can be detected by the security client and the security server with success rates of 100%.

(2) For all records deleted freely from the table (DoctorAdvice) by DBA, they all can be detected by the security server, but cannot be found out by the security client. It is known that the client only tests the results of user queries, whereas the deleted records are not included in the query results. Therefore, the success rate of deletion for the security client is 0, while it is 100% for the security server. The reason is that a continuous sequence is assigned to the primary key when the table is designed, and the application only marks logical deletion for invalid records rather than physically deletion. It would be hard

to detect illegal deletion when physical deletion is allowed. The contrast statistics, in the above tables, show that the tamper detection ability of the proposed approach greatly increases using the collaborative detection mechanism.

(3) For the records deleted casually from and then inserted directly in DoctorAdvice by DBA, they all can be detected by the client and the server. The success rates are 100%, too. A reasonable explanation for this is that the primary key is assigned automatically by DBMS and cannot be artificially specified, and it is a part of the input of hash algorithm. Therefore, the new digital fingerprint necessarily mismatches with the initial one, and it has the same effect on the client and on the server.

(4) The MD5 method in literature [9] lacked 'FDLA' activity, although it achieves the same performance as the proposed method in this paper in the detection of illegal addition, update and deletion. And in literature [9], the scheme of using a single hash algorithm is deficient against collusion attack. But in this paper, the hash algorithm is negotiable and unknown, the cracking process will cost more time and be more difficult. It is more advantageous in tamper detection.

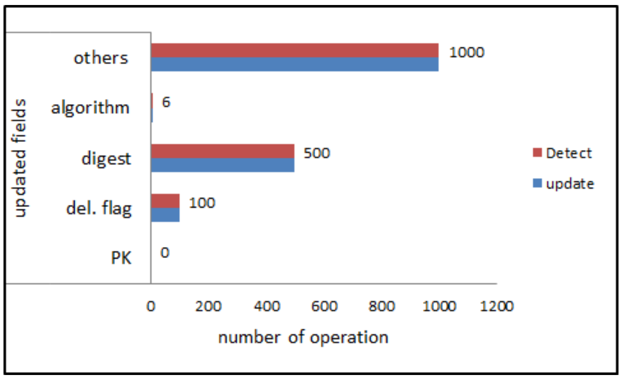

Figure 2. Updating & detecting statistics

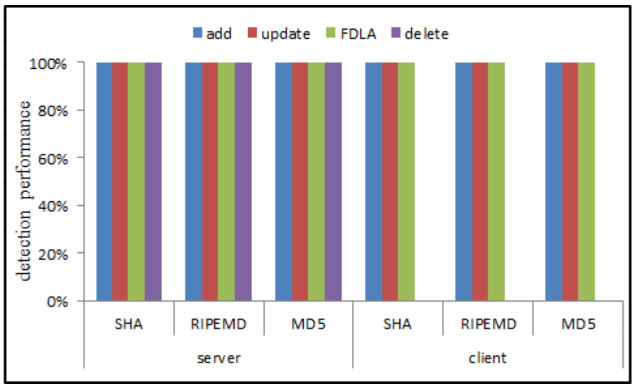

Figure 3. Contrast statistics of detection performance

The experimental data from the Figure 2 shows:

(5) Whether the digest (digital fingerprint), the hash algorithm or the sensitive data were tampered, all of them can be detected.

The contrast statistics about detection performance in Figure 3 indicates:

(6) The hash functions, namely MD5, SHA and RIPEMD, have the same excellent capability of data tamper detection with the method in this paper.

Finally, only a digest per record reduces communication cost. For calculating the record digest a time at the client, computation overhead is low.

6. **Conclusions**

In this paper, an approach of detecting tampered data in in-house database is proposed based on negotiable one-way hash algorithm, which implants hidden information into hash algorithm for improvement. It first generates an initial digest, and then stores it with data into database. The security server and the security client constitute the collaboration mechanism of active testing and passive testing, which enhances the capability of tamper detection in time and avoids the vulnerability of using a single hash algorithm. The experimental results show that this method can run effectively and detect unauthorized modification of the protected data in MI2P. Future study is needed to investigate time performance improvement of the proposed method and to explore applying our method to outsourced database scenario for detecting tamper behaviors.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No.41301084), Hunan Provincial Natural Science Foundation of China (No.2017JJ3252), Hunan Province Educational Science 13th Five-Year Planning Program (No.XJK016QXX003) and Key Laboratory Funds of Intelligent Control Technology for Wuling-Mountain Ecological Agriculture in Hunan Province (No.ZNKZ2014-8).

References

1. W. Al-Nuaimy, M. A. M. El-Bendary, A. Shafik, F. Shawki, A. E. Abou-El-azm, N. El-Fishawy, S. M. Elhalafawy, S. M. Diab, B. M. Sallam, F. E. A. El-Samie, and H. B. Kazemian, “An SVD Audio Watermarking Approach Using Chaotic Encrypted Images,” *Digital Signal Processing*, vol. 21, no. 6, pp. 764-779, 2011
2. C. Cida, “Recent Developments in Cryptographic Hash Functions: Security Implications and Future Directions,” *Information Security Technical Report*, vol. 11, no. 2, pp. 100-107, 2006
3. S. Cimato, C. N. Yang, and C. C. Wu, “Visual Cryptography Based Watermarking: Definition and Meaning,” *Lecture Notes in Computer Science*, vol. 7809, pp. 435-448, 2013
4. H. Guo, Y. Li, A. Liu, and S. Jajodia, “A Fragile Watermarking Scheme for Detecting Malicious Modifications of Database Relations,” *Information Sciences*, vol. 176, no. 10, pp. 1350-1378, 2006
5. S. Greeshma and R. Jayapriya, “Securing Database Server Using Homomorphic Encryption and Re-Encryption,” *Security in Computing and Communications. Springer International Publishing*, pp. 277-289, 2015
6. V. Gupta and I. J. Rajput, “Privacy Preserving in Data-Mining: A Survey on Security of Outsourced Transaction Databases,” *Compusoft International Journal of Advanced Computer Technology*, vol. 3, no. 12, pp. 1377-1385, 2014
7. A. Hamadou, X. Sun, L. Gao, and S. A. Shah, “A Fragile Zero-Watermarking Technique for Authentication of Relational Databases,” *International Journal of Digital Content Technology & Its Applications*, vol. 5, no. 5, pp. 189-200, 2011
8. A. Hamadou, X. Sun, S. A. Shah, and L. Gao, “A Hybrid Watermarking Scheme for Relational Databases Copyright Protection and Tamper Proofing,” *International Journal of Advancements in Computing Technology*, vol. 3, no. 8, pp. 18-28, 2011
9. J. Lin and Q. S. Huang, “Method of Data Tamper Detection by Using Improved MD5 Algorithm,” *Computer Engineering & Applications*, vol. 44, no. 33, pp. 148-150, 2008 (In Chinese)
10. Y. Li, H. Guo, and S. Jajodia, “Tamper Detection and Localization for Categorical Data Using Fragile Watermarks,” in *Digital Rights Management(DRM), Proceedings of the 2004 4th ACM Workshop on*, pp. 73-82, October,2004
11. E. Mykletun, M. Narasimha, and G. Tsudik, “Authentication and Integrity in Outsourced Databases,” *ACM Transactions on Storage*, vol. 2,no. 2, pp. 107-132, 2006
12. S. Mead, “Unique File Identification in the National Software Reference Library,” *Digital Investigation*, vol. 3, no. 3, pp. 138-150, 2006
13. A. R. Pathak and B. Padmavathi, “A Secure Threshold Secret Sharing Framework for Database Outsourcing,” in *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 IEEE International Conference on*, pp. 1642-1649, IEEE, May, 2014
14. R. L. Rivest, “The MD5 Message-Digest Algorithm,” RFC 1321, 1992
15. M. Stevens, “Single-Block Collision Attack on MD5,” *Cryptology Eprint Archive Report*, pp. 1-11, 2012
16. M. Stevens, “Fast Collision Attack on MD5,” *Cryptology Eprint Archive*, 2006
17. R. T. Snodgrass, S. S. Yao, and C. Collberg, “Tamper Detection in Audit Logs,” in *Very Large Databases(VLDB), Proceedings of the 2004 Thirtieth International Conference on*, vol. 30, pp. 504-515, August, 2004
18. M. V. Venkatesh and M. P. Parthasarathi, “Enhanced Audit Services for the Correctness of Outsourced Data in Cloud Storage,” *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 2, no. 2, pp. 564- 567, 2013
19. X. Wang and H. Yu, “How to Break MD5 and Other Hash Functions,” *Lecture Notes in Computer Science*, vol. 3494, pp. 561-561, 2005
20. M. Xie, H. Wang, J. Yin, and X. Meng, “Integrity Auditing of Outsourced Data,” in *Very Large Data Bases(VLDB), Proceedings of the 2007 33rd International Conference on*, pp. 782-793 , September, 2007
21. A. Yavuz, “Immutable Authentication and Integrity Schemes for Outsourced Databases,” *IEEE Transactions on Dependable & Secure Computing*, pp. 1-14, 2016
22. J. Zhang, P. Li, and J. Mao, “IPad: ID-based Public Auditing for the Outsourced Data in the Standard Model,” *Cluster Computing*, vol. 19, no. 1, pp. 127-138, 2016
23. Z. Zhang, L. Wu, H. Li, H. Lai, and C. Zhang, “Dual Watermarking Algorithm for Medical Image,” *Journal of Medical Imaging and Health Informatics*, vol. 7, no. 3, pp. 607-622, 2017

Jing Lin received MA degree in the Faculty of Information Engineering and Automation at the Kunming University of Science and Technology, in 2009. During 1995-1998 he worked as a software engineer with the Shenzhen Jinbada Comunication Company, where he was in charge of developing a detecting system for telephone device fault. During 1999-2005 he was a software manager in the Shenzhen Ncking Technology INC., LTD, analyzing and designing hospital information system. Since 2009, he has been an associate professor at Huaihua University. His current research interests include intelligent information system and information safety, data mining.

Chunqiao Mi received the Ph.D. degree in China Agricultural University in 2012. Now, he is an associate professor at Huaihua University. His research interests include agricultural informatization and big data.

Yuanquan Shi received his Ph.D. degrees in Sichuan University in 2011. Now, he is an associate professor at Huaihua University. His research interests include network security technology and application, intelligent information system.