The ISC Int'l Journal of Information Security

ISeCure

January 2019, Volume 11, Number 1 (pp. 3–14)

http://www.isecure-journal.org

A Trust-Based Probabilistic Method for Efficient Correctness Verification in Database Outsourcing

Simin Ghasemi [1], Mohammad Ali Hadavi [2,*], and Mina Niknafs [3]

[1] *Department of Computer Engineering, Payame Noor University (PNU), Iran*
[2] *Malek Ashtar University of Technology, Tehran, Iran*
[3] *Department of Computer Engineering, Vali-e-Asr University of Rafsanjan, Rafsanjan, Iran*

ARTICLE INFO.

Article history:
Received: 26 July 2018
Revised: 8 December 2018
Accepted: 22 December 2018
Published Online: 30 January 2019

Keywords:
Database Outsourcing, Security, Correctness Verification, Trust, Merkle Hash Tree.

ABSTRACT

Correctness verification of query results is a significant challenge in database outsourcing. Most of the proposed approaches impose high overhead, which makes them impractical in real scenarios. Probabilistic approaches are proposed in order to reduce the computation overhead pertaining to the verification process. In this paper, we use the notion of *trust* as the basis of our probabilistic approach to efficiently verify the correctness of query results. The trust is computed based on observing the history of interactions between clients and the service provider. Our approach exploits Merkle Hash Tree as an authentication data structure. The amount of trust value towards the service provider leads to investigating just an appropriate portion of the tree. Implementation results of our approach show that considering the trust, derived from the history of interactions, provides a trade-off between performance and security, and reduces the verification overhead imposed on clients in database outsourcing scenarios.

1 Introduction

Nowadays keeping and maintaining data is exceedingly expensive, so many organizations prefer to outsource their data to a third party service provider. Furthermore, lack of appropriate hardware and software infrastructures, not having professional experts, and tendency to concentrate on the main organization's missions are among other factors that encourage organizations to outsource their data. In data outsourcing scenarios, or the Database-As-a-Service (DAS) model (Figure 1), data and its management are outsourced to a third party service provider, which may be untrustworthy. That is, data outsourcing, in spite of its clear advantages, brings security challenges including data confidentiality and correctness of query results as the most critical ones. Organizations have to delegate control of their data to a service provider whose honesty cannot be guaranteed in practice. Therefore, they have to counteract data integrity violations by the verification of query results. Consequently, there should be a way to inspect to what extent the returned result satisfies a query condition. This problem refers to the correctness verification of query results, which includes completeness, freshness, and integrity of the results.

As shown in Figure 1, in a generic data outsourcing scenario, there is a Data Owner (DO), a Service Provider (SP), and Clients(C). To audit the correctness of a query result, SP provides a proof corresponding to the result. This proof is also known as Verification Object (VO). While C receives a result, checks its

* Corresponding author.
Email addresses: s.ghasemi@pnu.ac.ir, hadavi@mut.ac.ir, m.niknafs@vru.ac.ir

ISSN: 2008-2045 © 2019 ISC. All rights reserved.

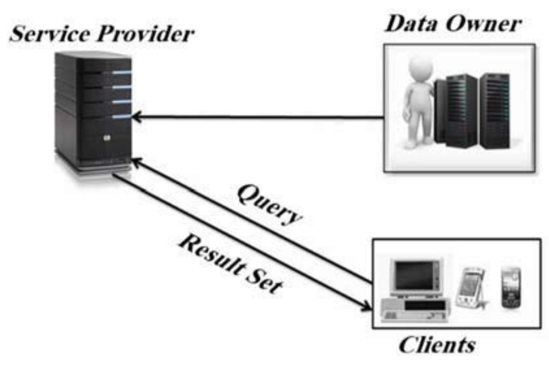


Figure 1. The Generic DAS Model

correctness using the associated *VO* returned by *SP*.

Some approaches have been proposed for correctness verification of query results in the literature [1–6]. Most of them use digital signatures and data authentication structures as their basic primitives. They often impose high computation and communication overheads making them unacceptable for real scenarios. On the other hand, some approaches, namely probabilistic approaches, while reducing the imposed overheads, do not guarantee the correctness of results. The probability of correctness is measured through the probability of success of a dishonest *SP*. Such approaches offer a trade-off between security and performance. In this paper, we propose a probabilistic approach for correctness verification, which uses the history of *C*-*SP* interactions to adjust the imposed overhead. The history of *C*-*SP* interactions is the basis of computing the notion of *trust* in our approach, which is constructed upon the Goodrich *et al.*'s approach [1]. The authors in [1] suggest signing some specific levels - in addition to the root - in their authentication data structure. We use only a portion of these additional signatures based on the computed trust value to trade-off between security and efficiency. For each transaction, the current trust value towards *SP* adjusts computation overhead during correctness verification. The trust value itself is updated resulting from the verification process. If the correctness is verified, the trust value increases. As much as the trust value increases, the verification process takes less effort.

We leveraged our approach in Content Distribute Networks, where numerous *SP*s deliver services to customers distributed all over the world in order to load balancing. As an example, the content of this service can be the geographical location of restaurants. For this service, customers frequently interact with service providers. In this scenario, the information (query results) is not very sensitive and some possible mistakes in results are admissible. In such a scenario, probabilistic versification is intuitive and considering the notion of trust can improve the method. The trust model for this scenario can be an agreement-based model, which is suitable for not too critical applications. Trust establishment is based on Service Level Agreement (SLA) signed by *SP* for delivery of correct answers to the customers. Indeed, the SLA is prepared based on customers' requirements [7].

The result of our implementation shows that using the history of interactions, in addition to attaining an acceptable correctness probability, makes the verification process efficient enough, leading to a more practical solution.

The remainder of this paper is organized as follows. Related works are reviewed in Section 2. Our approach is described in Section 3. The results of implementation are discussed and analyzed in Section 4. Finally, Section 5 concludes the paper.

2 Related Work

Trust concept has been mentioned in many security areas and database subjects especially in access control policies [8–10]. There are also some works, which investigate the role of trust in outsourcing IT services [11–13]. We use this concept for correctness verification of outsourced databases. In accordance with the aim of this paper, in this section we only concentrate on the works related to the correctness verification in the DAS model. We divide the proposals into three categories including Merkle Hash Tree (MHT) based, digital signature based, and probabilistic approaches.

2.1 MHT

MHT, introduced for the first time by R. C. Merkle [14], has been used in several proposals [6, 15, 16] as an authentication data structure. Generally, an MHT is constructed on a database relation whose tuples are sorted with respect to a searchable attribute. Hash values of the tuples construct leaves of the tree. The value of a parent node is computed by a hash function over the concatenation of its immediate children. The root is calculated by a recursive concatenation process. Finally, *DO* signs the root by its private key. In addition to the database relation, *DO* outsources the constructed MHT for each searchable attribute. When *C* submits a range query, *SP*, in addition to the result set, sends the required nodes in the MHT so that the MHT root can be reconstructed by *C*. Having the MHT root and its signature, *C* can verify the correctness of the result. Figure 2 depicts an example of an MHT. According to the figure, h_{12} is calculated by the concatenation of h_1 and h_2. With r_3 as the query result, *SP* besides r_3 sends h_4 and h_{12} as *VO* to *C*.

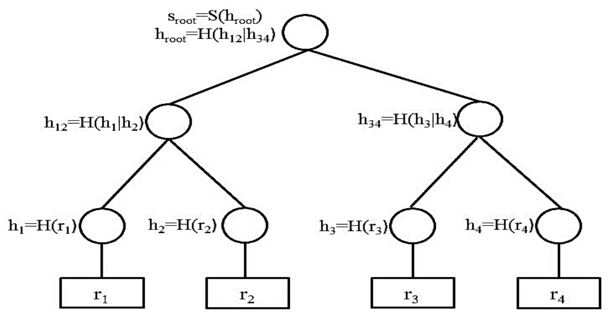


Figure 2. A Sample Merkle Hash Tree

Integrating B^+-tree with the MHT, Li *et al.* [17] proposed MB-tree (Merkle B-Tree). In B^+-tree, each node contains multiple values, kept together on disk. This property leads to low cost of read/write from/to disk. The time complexity of this method is $O(log(h))$ where h is the height of the constructed tree. Embedded MB-Tree is the next method proposed by Li *et al.* [17] in which a tree is embedded into each node of the main tree. The embedded trees are constructed based on the outsourced data in each node. This approach reduces the size of *VO*.

Goodrich *et al.* [1] proposed an efficient method for correctness verification, which aims to authenticate query results based on a specific attribute. In this method, MHT is recursively divided into $O(log^*n)$ specific levels whose nodes are signed by *DO*. While, building *VO* for a query result, *SP* is expected to send only those nodes by which *C* can verify the signature of a node in the specific level, instead of the signature of the root. In this way, *C* tries to construct the smallest enveloping subtree, which covers all the result tuples.

Wang *et al.* [18] proposed a data integrity verification scheme with designated verifiers for dynamic outsourced databases. They presented a new authentication data structure by combining skip lists with polynomial functions and accumulators. They constructed data verification scheme for range queries in a dynamic scenario. The communication cost in this scheme is independent of the size of the authenticated dataset.

The main advantage of the tree-based methods is the small number of signatures. However, their initial construction cost, as well as the cost of data update is considerable compared to signature based approaches. Moreover, the size of *VO* for a query result depends on the database size instead of the result size.

2.2 Digital Signature

In digital signature based approaches data is signed by *DO*'s private key. *C* verifies the correctness of digital signatures by using *DO*'s public key. In this method, *DO* usually signs all tuples by its private key and outsources them to *SP* beside the relation. When *C* submits a query, *SP* sends result tuples together with their signatures to *C*. On account of high storage and computational overhead of digital signatures, aggregated signature was proposed [4]. In this way, *SP* aggregates all signatures and sends an aggregated signature to *C*. The main problem in [5] is that it cannot verify the completeness of query results. To solve the problem, Narasimha and Tsudik [19] proposed a method based on signature chaining. Their suggestion is to consider the immediate predecessor tuple of a tuple into the tuple's signature. In [20] a method was proposed for completeness verification of diverse kinds of queries including selection, join, and projection.

In digital signature based approaches, each tuple can be updated solitary. Therefore, the data update process is independent of the database size which is a notable advantage. On the other hand, the main weakness is their considerable initialization cost, especially for fine grained verification.

2.3 Probabilistic Approaches

In probabilistic approaches such as [2, 3, 21–23], the correctness of query results cannot be guaranteed while verification process is performed more efficiently. Sion [2] proposed a probabilistic approach using a challenge-response method. In this method, *C* sends queries in batch mode along with a challenge to *SP*. An honest *SP* finds the challenge response and sends it to *C*. In contrast, a lazy *SP* works properly just until finding the challenge response, so correctness of query result is probabilistically verified. In [21] a method for the proof of data ownership at server side was proposed. In this method, there is not any query, but *DO* occasionally sends a cryptographic challenge to *SP* to investigate the property of data ownership.

In database outsourcing scenarios it is probable that some parts of data on *SP* be deleted (intentionally or unintentionally), while *SP* conceals the problem. Xie *et al.* [3] proposed a probabilistic method for detecting unruly behavior of *SP*. In their approach, the correctness of query results is verified by introducing some fake tuples among the real ones. If all the expected fake tuples satisfying query conditions are returned by the server, *C* probabilistically verifies the completeness of query results. The probabilistic method by Xie *et al.* has been improved in [22] in which the past behavior of *SP* determines the number of outsourced fake tuples. As much as SP works properly, the number of fake tuples decreases which in turn, increases the verification efficiency.

3 The Proposed Approach

Our approach is based on the proposed method by Goodrich *et al.* [1]. In their method, a Merkle Hash Tree is built over tuples sorted with respect to a searchable attribute. Then, as shown in Figure 3, in addition to the root, the nodes located in some specific levels of the tree (log^*n where n is the number of tuples) are signed in a recursive mode. The main reason of signing such nodes is to reduce the height of the verification tree. Thus, C verifies the root of the smallest tree which covers all the tuples in the query result. In fact, the root of the smallest enveloping tree is verified. Clearly, a smaller tree has fewer nodes, so computation overhead reduces at C side. Although this situation is desirable, sometimes outspread results lead to a big authentication subtree close to the main tree. As a result, C has to reconstruct the whole authentication tree for verifying the query result. This is an important objection to Goodrich *et al.*'s method, which results in imposing high computation overhead on C. Our proposal tries to solve it by minimizing the verification subtree considering the previous *C-SP* interactions.

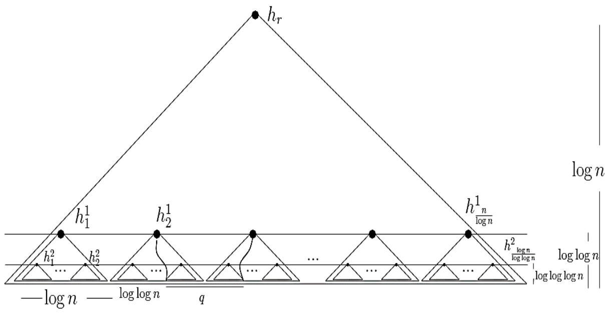


Figure 3. Goodrich *et al.*'s authentication structure [1]

3.1 Our History Based Trust Model

In most of the database outsourcing applications, there are consecutive interactions between SP and C. We use the history of these interactions, each of which is followed by a verification process, to reduce computation overhead at C side. That is, the history of *C-SP* interactions is the basis of our proposed trust model.

We remark that there is neither a unique reference model nor a unique definition of trust. However, the intuition says that trust can be interpreted as a belief towards future behaviors based on deduction from the past behavior. The following definitions of trust confirm our interpretation of trust:

- Trust is a subjective expectation an agent has about another's future behavior based on the history of their encounters [24].
- Trust is the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context [25].

According to the above definitions, a trust value T indicating C's belief toward proper functionality of SP is defined based on its past behavior. Higher trust values lead to lighter verification process in C, and consequently less computation overhead. According to T, a subtree of the enveloping tree is chosen for the verification. We name our approach "Trust Based Data Authentication Structure (TBDAS)". TBDAS is a history-based and light-weight method for C through which the correctness of query results in the DAS model is probabilistically verified.

We assume that at the beginning of interactions, SP is untrustworthy and the trust value is 0, so C has to verify the whole enveloping subtree similar to the Goodrich *et al.*'s method. As much as the number of faithful interactions grows, C's belief towards SP's behavior increases, so a smaller subtree of the enveloping tree is reconstructed for the verification. It is worth to mention that C randomly chooses a smaller subtree from the enveloping tree so that SP cannot infer anything about the chosen subtree to deliberately manipulate the result without being detected at C side.

In Section 4.3, we analyze our probabilistic approach to show that although the correctness is not guaranteed, the escape probability of a malicious SP is insignificant. Moreover, verification process at C side is gradually diminished providing SP's honest behavior.

3.2 Database Outsourcing

Figure 4 depicts the scheme of TBDAS in two phases including *Database Outsourcing* and *Query Execution*. In the first phase, *DO* sorts relation tuples with respect to a searchable attribute, builds an MHT, and signs its root plus the nodes in specific levels of the tree similar to Goodrich *et al.*'s method [1]. With h as the height of the tree, log^*h determines the specific levels. In the simplest case, only the first level of the tree, i.e. the root, is signed. To clarify, let h be the height of the tree. Then, $x = log(h)$ is the first level whose nodes must be signed by DO. Other specific levels are recursively specified by $x = log(x)$, until reaching $x = 1$ indicating the root as the last specific level. For example, for a tree with $h = 16$, levels 4, 2 and 1 are the specific levels. After signing the nodes of specific levels, *DO* outsources database relations as well as their corresponding signed MHTs to SP. For data update, *DO* have to do all of the above procedure. Figure 5 describes the pseudo-code of database outsourcing phase.

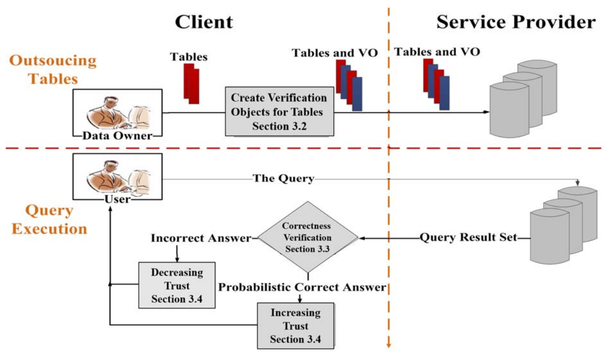

Client
Service Provider
Outsoucing Tables
Tables
Tables and VO
Tables and VO
Create Verification Objects for Tables Section 3.2
Data Owner
User
The Query
Query Execution
Incorrect Answer
Correctness Verification Section 3.3
Query Result Set
Decreasing Trust Section 3.4
Probabilistic Correct Answer
Increasing Trust Section 3.4

Figure 4. TBDAS Scheme

```
Inputs:
 T: table to be outsourced
 attr: searchable attribute in T
 HF: hash function
Variables:
 R_i:the i^th tuple of T
 T_srtd: sorted T
 HR_i: hashed value of tuple i in T
 h: height of Tree
 x: levels of Tree
Procedure:
 T_srtd= Sort (T, attr)           /* sorts T with respect to attr              */
 For each tuple i
 HR_i=HF(R_i)                     /* creates hash value for each tuple of T    */
 Tree = MHTCons (T_srtd, HF)      /* constructs a simple MHT over the sorted table
                                     T_srtd using the hash function HF        */
 h = height (Tree)                /* specifies the height of Tree              */
 x = log (h)
 While (x ≥ 1){
     Tree=SignLevel(Tree, x)      /* signs the nodes of level x in the Tree    */
     x = log (x)}
 Outsource (T, Tree)              /* outsources the signed Tree beside T to SP */
```

Figure 5. Pseudo-code of Database Outsourcing phase in TBDAS

3.3 Correctness Verification

In the second phase of TBDAS (Figure 4, Query Execution), C sends a query to SP. For the sake of simplicity, we suppose that the query is a simple range query on a relation. SP executes the query on the DO's last updated data and finds a set of tuples as query result. In addition to the query result, SP sends some extra intermediate nodes of the signed MHT to C. It includes the nodes by which C can reconstruct the smallest signed subtree. Also, SP sends all signatures of the enveloping subtree to C.

When C receives a query result plus its VO, considering the current trust value towards SP, verifies the result . To this purpose, C chooses a set of tuples to construct the subtree. A higher trust value results in a lower number of tuples to verify. For instance, if the query result contains 1000 tuples and the current trust value is 0.4, C chooses 600 tuples for verification. C constructs an enveloping subtree by using both the selected tuples and received hash values. If the root of the constructed subtree belongs to a specific level, its signature is available and C can verify it. Otherwise, C has to continue constructing the subtree until it reaches a specific level having a signed node. In the worst-case, the constructed subtree is the same as the enveloping subtree which covers all the result tuples. In such a case, if the root of the enveloping subtree is verified, the query result is certainly correct. That is, the correctness is guaranteed. If the constructed subtree becomes smaller than the spanning subtree and C verifies its signature, the result is probabilistically correct. Equation (1) calculates the $CorrectnessProbability$ (CP) of the result.

$$CP = \frac{(TupNum - TupNum') \times T_{current} + TupNum'}{TupNum} \quad (1)$$

Equation (1) indicates that the verified tuples $(TupNum')$ are certainly correct, but non-checked tuples $(TupNum - TupNum')$ are probably correct relying on the current trust value toward SP $(T_{current})$.

After the verification, if the answer is probabilistically correct, the trust value increases. Otherwise, the trust value decreases to show dishonesty of SP. In Section 3.4 we show that how the result of correctness verification impacts on the amount of trust. Intuitively, a verified result should slightly increase the trust value and vice versa a detected incorrect result should sharply decrease it.

It is worth to mention that lower computation overhead on C leads to a bit more communication overhead between SP and C. Considering C as a low power client, more efficient computation at C is worth having a bit more communication overhead. In Section 4.2, we show that the extra communication overhead in our approach is negligible compared to that of Goodrich *et al.*'s method.

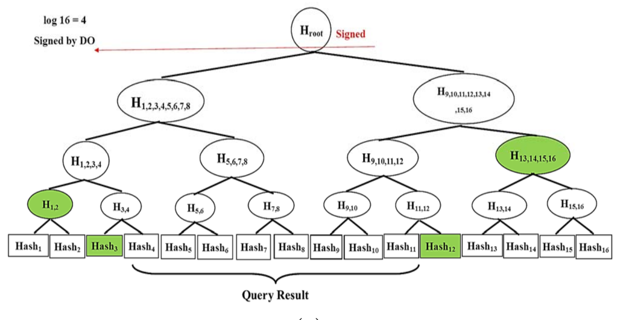
Let us clarify the difference between TBDAS and Goodrich *et al.*'s method regarding SP's functionality. In Goodrich *et al.*'s method, SP sends only the root signature of the enveloping subtree beside the tuples of the query result and necessary hash values. In TBDAS, in addition to these data items, signatures of the lower levels of subtree are also sent to C. These extra signatures enable C to verify a smaller subtree, which is proportional to the amount of trust toward SP. The following example illustrates the difference.

Example - Suppose that there is a relation Market with three attributes $ProductID$, $Price$, and $Count$ (Figure 6). This relation has 16 tuples and there is a unique hash value for each tuple. The relation has been sorted upon the $ProductPrice$.

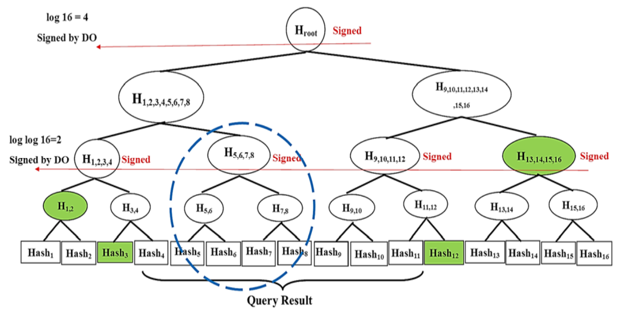
Figure 7a shows a naïve MHT over the Market relation and Figure 7b shows an MHT based on TBDAS and Goodrich *et al.*'s method. In a naïve MHT, the root is signed by DO but in the other two methods, in

ProductID	ProductPrice	Count	Hash Value
1	100	1	Hash1
2	115	2	Hash2
3	130	1	Hash3
4	141	1	Hash4
5	156	2	Hash5
6	172	2	Hash6
7	189	1	Hash7
8	200	2	Hash8
9	210	2	Hash9
10	212	2	Hash10
11	219	2	Hash11
12	223	1	Hash12
13	226	2	Hash13
14	231	1	Hash14
15	265	1	Hash15
16	270	1	Hash16

Figure 6. Market relation



(a)



(b)

Figure 7. Constructed Nodes for Market relation (Figure 6) (a) A naïve MHT, (b) An MHT based on Goodrich *et al.*'s Method

addition to the root, the nodes in specific levels l where $l = \{2, 4\}$ are also signed. Assume that after data outsourcing, a user poses a sample query as below:

SELECT * FROM Market WHERE 140 $\leq$ *Price* $\leq$ 220

According to Figure 6, the answer includes tuples 4 to 11. The highlighted nodes in Figure 7a and 7b are the VO corresponding to the result. In a naïve MHT, the user constructs the whole tree to get the hash of the root and verifies the answer by the root signature. Considering Goodrich *et al.*'s method, there is not a low level signed subtree which covers all tuples of the query result. Therefore the user reconstructs the whole tree as well as in the naïve MHT. In TBDAS the current trust value towards SP, $T_{current}$, determines the number of tuples for verification. For example, with $T_{current} = 0.5$ the client builds a subtree for half of the result tuples. In this case, if tuples 5 to 8 are selected by C, they form a subtree whose root (H5678 in Figure 7b) must be verified with its signature (the dotted blue space in Figure 7b). As we see in this example, TBDAS usually leads to construction of a smaller subtree, which decreases the client computation overhead.

3.4 Updating Trust

Trust value (T) is a value in [0..1] indicating client's trust toward SP. Initially, it is set to zero assuming SP is fully untrustworthy. The initial value depends on C's belief to SP. For example, if SP is a reputed organization, or a trusted third party introduces SP, the initial trust value can be higher than zero. However, we assume that there is no prior trust to SP and initialize it to zero. On the other hand, the trust value is limited to an upper bound, Trust Threshold (TT), to force a light verification process even for an honest SP. TT is initialized by C and each user may have her own TT. As much as TT is closer to 1, reduction of the computation overhead is more tangible. In this paper, we set TT to 0.95.

In TBDAS, the trust value is calculated and updated after correctness verification for each query. On the other hand, the number of verified tuples in a query result depends on the trust value. Receiving a query result from SP, if $T = 0$, C has to construct the smallest subtree which covers all tuples in the result and verify its root signature. If it is verified, the query result is definitely correct. For non-zero trust values, it is enough for C to verify the correctness of a part of tuples whose number is determined according to Equation (2). In Equation (2), $T_{current}$ is the current trust value, $TupNum$ denotes the number of tuples in the result, and $TupNum'$ denotes the number of selected tuples for verification.

$$TupNum' = TupNum \times (1 - T_{current}) \tag{2}$$

Equation (2) indicates that as much as distrust to SP is higher, the number of tuples selected for verification is closer to the total number of returned tuples. To verify the chosen set of tuples, C constructs merely the subtree, which covers all the chosen tuples. Clearly, this tree is smaller than the tree, which contains all the result tuples, so lower computation overhead is imposed on C.

In order to decrease the risk of sudden changes in the behavior of SP, C gradually increases the trust value according to Equation (3). On the other hand, if correctness is not verified for a query result, the trust value sharply decreases as formulated in Equation (4). Hereon, C penalizes SP according to a suggested mechanism in the service level agreement signed between them. In summary, the trust value increases additively having correct query results and decreases sharply receiving an incorrect answer.

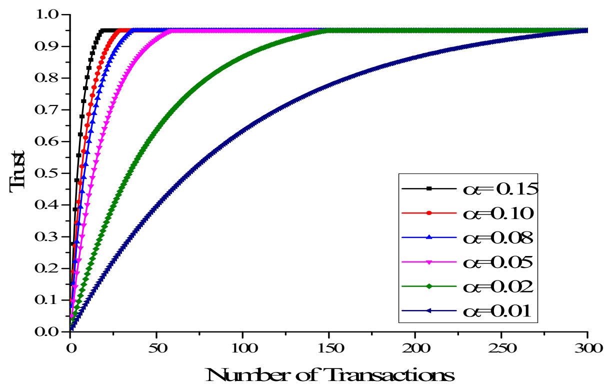
$$T_{new} = \min(T_{current} + \alpha \times \frac{TupNum'}{TupNum}, TT) \quad (3)$$

$$T_{new} = T_{current} - \min(T_{current}, \beta \times \frac{TupNum'}{TupNum}) \quad (4)$$

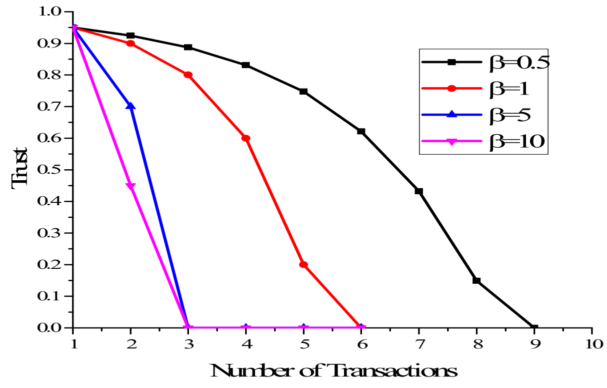
In both Equation (3) and Equation (4), $T_{current}$ is the current trust value to SP, TT is the trust threshold, $TupNum$ is the number of tuples in the query result, and $TupNum'$ refers to the number of selected tuples for verification where $TupNum' \leq TupNum$. In Equation (3), α is a coefficient to adjust the changing rate of trust. Similarly, in Equation (4), β is a coefficient larger than zero to specify the rate of decreasing trust.

Figure 8a depicts an example of trust increment. In this example, we assume that a fixed query is executing during many transactions with $TupNum = 1000$. The trust value, initialized to zero, increases each time for a correct result. We evaluated this example by multiple values for α. The figure shows that for $\alpha = 0.15$, after about 25 transactions, the trust value reaches to its upper bound. For $\alpha = 0.01$, it takes almost 300 transactions to reach the trust threshold. The figure also confirms that trust increment occurs with a gentle slope. However, the slope depends on the value of α chosen by C. Our experimentation indicate that $0 < \alpha < 0.2$ is an appropriate range for α. Obviously, its exact value depends on the application setting and the frequency of interactions between C and SP.

Figure 8b shows the other side of changing trust towards SP, i.e., trust decrement. The figure shows that at first, the trust value is 0.95, as the trust threshold. We evaluate the example by multiple values for β. As depicted in the figure, for $\beta = 0.5$, after seven transactions of incorrect result, detected by C, the trust value reaches to zero. For $\beta = 10$, just after one erroneous transaction, trust value reaches to zero. This example shows that the risk of trust to SP is low. This is because with only one unverified result, the value of trust sharply reduces, which in turn enforces a more rigorous verification of subsequent results. Considering the application in use, the client chooses a proper β as well as a proper α. Figure 9 represents pseudo-code of Query Execution phase including query submission, updating the trust value, and result verification at C side. We would like to remark that our trust model



(a)



(b)

Figure 8. Examples of Trust Variation for (a) Trust Increment, (b) Trust Decrement

and its calculations is not the only way to consider SP's past behavior. One can suggest other formulas, which compute and reflect the trust towards SP in the verification process. However, our analysis and evaluations (Section 4) show the efficacy of our proposed model.

4 Evaluation and Analysis

In this section we discuss on the efficiency improvement as well as on the security of our approach. We empirically show that our approach relying on the notion of trust reduces the client side verification overhead compared to that of Goodrich *et al.*'s method. In the second subsection, the effect of the approach on the communication overhead has been evaluated. We also theoretically assess the security of our approach in terms of the escape probability of a malicious server. Escape probability is the probability with which a dishonest server manipulates a query result and escapes from the client detection.

```
Inputs:
  T: Initial Trust value to SP = 0
  TT: Trust Threshold = 0.95
  Alfa: A coefficient in (0, 0.2)          /* a coefficient for trust increment */
  Beta: A coefficient > 0                  /* a coefficient for trust decrement */
  N: Number of queries

Variables:
  i: Counter for queries
  Qi: the ith query
  RSi: Result Set of Qi
  TupNum: Number of tuples in RSi
  TupNum': Number of tuples to be verified according to T
  STi: Smallest enveloping signed subtree for Qi
  R: Verification Result (true or false)
  SSi: Signature Set for construction of enveloping subtree

Procedure:
  i = 1;
  while (i≤N)
  {
   Snd_Query(Qi,SP)                  /* client sends query Qi to SP */
   (RSi,SSi) = Exec_Query(Qi)        /* SP executes Qi and sends the result RSi and signatures
                                        SSi as VO to C */
   TupNum' = TupNum × (1-T)
   STi = Blt_MHT(RSi,TupNum')        /* C builds the smallest enveloping subtree covering
                                        TupNum' tuples */
   R = Vrf_Sign(STi)                 /* C verifies root signature */
   if (R is True)
    {
     T = min(T + Alfa × TupNum'/TupNum, TT)           /* trust increment */
     CP = ((TupNum - TupNum') × T -TupNum')/TupNum    /* Correctness Probability
                                                          of RSi */
     }
   else
    {
     T = T - min(T, Beta × TupNum'/TupNnum)           /* trust decrement */
     Return False                                     /* rejects RSi */
     }
     i++
  }
```

Figure 9. Pseudo-code of Query Execution phase in TBDAS

4.1 Performance Evaluation

The goal of this section is to evaluate our claim in reducing computation overhead in TBDAS. To this purpose, we implemented TBDAS based on Goodrich *et al.*'s method in order to show the effect of adopting trust on their method. In the experiments, we show that how gradual increase of the trust value decreases the overhead of verification processes. In our implementation, we used Java language on a PC with a Pentium IV Core i5 processor and 4GB of RAM to act as SP and MySQL DBMS as the database server. We executed our system on a shopping relation with five attributes including ID (unique), $Name$, $Price$, $Timestamp$, and $Header$. $Timestamp$ is set by DO to show the expiration date of tuples. $Header$ is the tuple digest computed through a hash function over the tuple. C sends a query with the following template to SP:

SELECT * FROM TableName WHERE A $\leq$ *Price* $\leq$ B

In the implementation, we set $\alpha = 0.15$ and $T = 0$ as the initial trust value. We suppose that in this experiment, SP acts honestly and the trust continuously increases. The results of implementation are shown in Figure 10a and 10b for 1000 and 1000000 tuples, respectively. The results have been calculated as the mean of 10 repetitions for a query execution. Each line in the figures belongs to a query with the specified result size. The starting point of each line, indicated by a star sign, shows the functionality of Goodrich *et al.*'s method in the same state. Assuming the honesty of SP, the trust value increases to eventually reach the threshold. In contrary, a detected sabotage leads to a sharp decrease of trust value as explained in Section 3.4. Vertical axis shows the number of constructed nodes to reach the root of covering subtree for each query execution. According to Figure 9, the most significant factor in query result verification time at C side is the time complexity for constructing the smallest enveloping subtree, which directly depends on the number of constructed nodes.

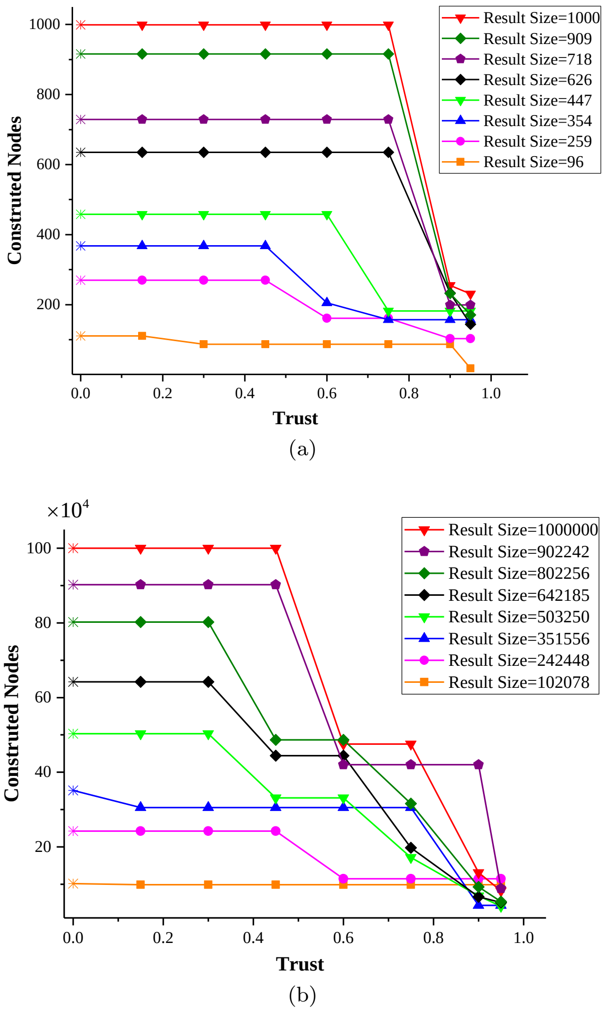


Figure 10. TBDAS Analysis in a table with (a) 1000 tuples, (b) 1000000 tuple

Figure 10a and 10b show the effect of trust value on decreasing the number of constructed nodes for verification per different result sizes. Comparing the execution of similar queries over relations with different sizes, the figures show that as much as the table is larger, considering trust is more effective. They also depict that as much as T increases, the number of constructed nodes by C decreases especially when T is closer to TT. Moreover, it can be inferred from the figures that the effect of considering trust is more tangible for larger databases. The efficiency effect appears even with lower trust values when the database size becomes larger. On the other hand, for a small database (in our experiment, a table having 1000 tuples) although the effect of considering trust appears with higher trust values, it causes a sharp decrease in the number of constructed nodes. In overall, Figure 10a and 10b acknowledge that adopting trust significantly decreases verification overhead in application settings having high volume databases and queries with large result sets.

4.2 Communication Overhead

In TBDAS, SP sends some extra signatures as a part of VO to enable C to verify the root of a smaller subtree. The extra signatures while reducing client side verification process, lead to a bit more communication overhead. In this section we experimentally measure the amount of communication overhead including the size of VO sent from SP to C in both TBDAS and Goodrich *et al.*'s method. We use the same setting mentioned in the previous section for computation overhead evaluation. We pose queries with various result size. The result of this experiment for each result size is the average of several executions with different ranges in the attribute domain but a fixed result size.

Figure 11 shows the amount of communication overhead in KB per different result sizes on a table having one million tuples. As Figure 11 depicts, as much as the size of result increases, the amount of communication overhead decreases. This is because the number of necessary signatures and extra nodes for verification goes less when the result size increases. Moreover, the communication overhead for TBDAS and Goodrich *et al.*'s approaches are so close together and their difference is negligible with respect to the total overhead.

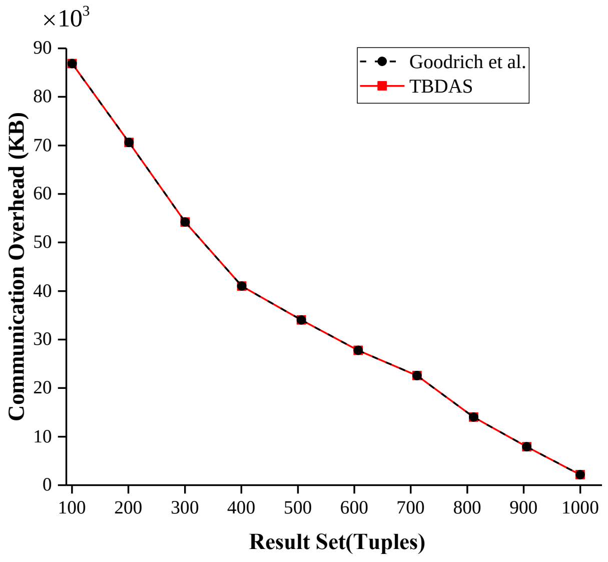


Figure 11. TBDAS communication overhead in a table with one million tuples

4.3 Security Analysis

Although TBDAS offers lower verification overhead, it probabilistically verifies the correctness of query results. That is, the correctness of results are not guaranteed. This is because TBDAS relies upon the past behavior of SP and prefer not to verify all the nodes but a subset of them whose size is proportional to the amount of trust to SP. Therefore, it is probable in theory that a highly trusted SP abuses the method and manipulates some tuples, which are not chosen at client side to be in the verification subtree, so the result is mistakenly verified. Now, we calculate the probability of an unauthorized result manipulation by SP and being undetected at client side. We call such probability as $EscapeProbability$ (EP).

In TBDAS, we suppose that SP cannot learn about the chosen verification subtree at client side. Therefore, it manipulates some tuples and hopes to escape the client detection. EP is calculated by (5) where t is the number of manipulated tuples by SP, $TupNum$ is the result size (in tuples), and m is the number of tuples, which are not among the chosen tuples at client side. It depends on the current trust towards SP, $T_{current}$, and is calculated by $m = T_{current} \times TupNum$.

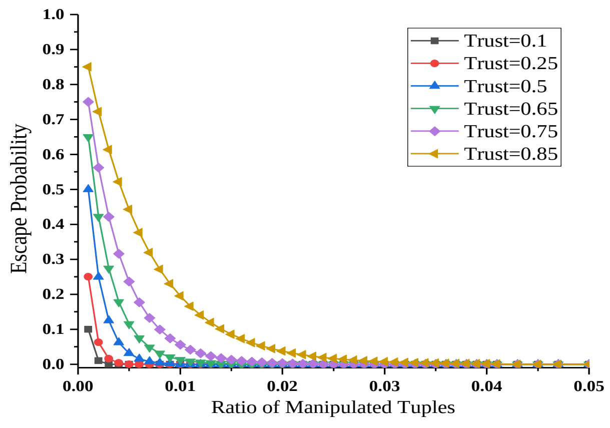
$$EP = \begin{cases} \frac{\binom{m}{t}}{\binom{TupNum}{t}} & m \geq t \\ 0 & m < t \end{cases} \tag{5}$$

Equation (5) says that if SP manipulates more than m tuples the client definitely detects SP's misbehavior. Otherwise, there is a small but non-zero probability for SP to escape. Simplifying (5), we have (6), which is another form to represent EP.

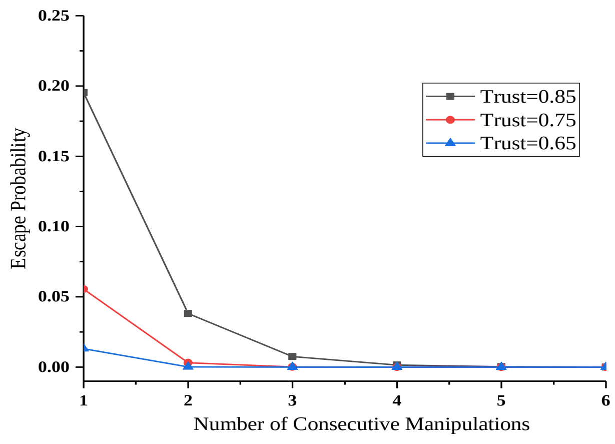
$$\begin{aligned} EP = &\frac{m-t+1}{TupNum-t+1} \times \ldots \times \\ &\frac{m-2}{TupNum-2} \times \frac{m-1}{TupNum-1} \times \\ &\frac{m}{TupNum} \end{aligned} \tag{6}$$

Equation (6) says that when $t \ll TupNum$, $EP \approx \left(\frac{m}{TupNum}\right)^t = (T_{current})^t$. That is, with very small t, the escape probability is independent from the result size, and is equal to $(T_{current})^t$, which also comes from our intuition as more trust to SP prepares potential to escape, though it diminishes exponentially for frequent escapes if $t > 1$.

From (5) we can simply compute EP when SP manipulates a ratio of result tuples. Figure 12a is the diagram depicting EP for different ratios of manipulations. Each curve in the figure shows EP for a trust value. The figure shows that EP exponentially decreases and being close to zero with increasing the manipulation rate. It also clearly shows that even with a very little change in the result, e.g. 2%, EP is a negligible value close to zero. We observe in Figure 12a that when $T_{current} > 0.8$, it is likely for SP to escape from client detection when it manipulates a very small percentage of the result. However, Figure 12b shows that SP cannot consecutively takes advantage of the client trust. It is assumed in this figure that SP manipulates 1% of result tuples for client queries. The figure indicates that the probability of more than two consecutive escapes is almost zero even for a highly trusted SP. Both of the experiments in Figure 12a and 12b are conducted on 1000 tuples as result size.



(a)



(b)

Figure 12. Escape Probability for (a) ratio of manipulated tuples, (b) consecutive manipulation

5 Conclusion

In this paper, we proposed TBDAS as a trust-based method, which efficiently verifies the correctness of query results returned by the service provider in the database as a service model. The notion of trust in our MHT-based method, helps to reduce the size of verification subtree. The amount of trust is obtained from the history of client-server interactions. Trading off between security and performance, TBDAS enables the client to tune its trust towards the server. Although the correctness of results is not guaranteed, we showed that the server cannot manipulate a query result and escape client detection unless with an insignificant probability. Reducing computation overhead imposed on the client makes our approach appropriate for outsourcing scenarios having numerous client-server interactions with computationally weak clients. Reduction of the verification process makes TBDAS appropriate for crowd-sourcing setting alongside outsourcing applications.

As future work, we plan to extend our work to more complex queries such as join queries over more than one table.

References

[1] Michael T. Goodrich, Roberto Tamassia, and Nikos Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'08, pages 407–424, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-79262-7, 978-3-540-79262-8.

[2] Radu Sion. Query execution assurance for outsourced databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 601–612. VLDB Endowment, 2005. ISBN 1-59593-154-6.

[3] Min Xie, Haixun Wang, Jian Yin, and Xiaofeng Meng. Integrity auditing of outsourced data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 782–793. VLDB Endowment, 2007. ISBN 978-1-59593-649-3.

[4] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2):107–138, May 2006. ISSN 1553-3077.

[5] R. Tamassia and N. Triandopoulos. Efficient content authentication over distributed hash tables. Technical report, CS Department, Brown University, 2005.

[6] Charles Martel, Glen Nuckolls, Premkumar Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, January 2004. ISSN 0178-4617.

[7] Ayesha Kanwal, Rahat Masood, Muhammad Awais Shibli, and Rafia Mumtaz. Taxonomy for trust models in cloud computing. *The Computer Journal*, 58(4):601–626, 2015.

[8] P. N. Mahalle, P. A. Thakre, N. R. Prasad, and

R. Prasad. A fuzzy approach to trust based access control in internet of things. In *Wireless VITAE 2013*, pages 1–5, June 2013.

[9] Jorge Bernal Bernabe, Jose Luis Hernandez Ramos, and Antonio F. Skarmeta Gomez. Taciot: multidimensional trust-aware access control system for the internet of things. *Soft Computing*, 20(5):1763–1779, May 2016. ISSN "1433-7479.

[10] Andrew G. West, Adam J. Aviv, Jian Chang, Vinayak S. Prabhu, Matt Blaze, Sampath Kannan, Insup Lee, Jonathan M. Smith, and Oleg Sokolsky. Quantm: a quantitative trust management system. In *Proceedings of the Second European Workshop on System Security, EUROSEC*, pages 28–35, 2009.

[11] B. Dong, R. Liu, and H. W. Wang. Trust-but-verify: Verifying result correctness of outsourced frequent itemset mining in data-mining-as-a-service paradigm. *IEEE Transactions on Services Computing*, 9(1):18–32, Jan 2016. ISSN 1939-1374.

[12] Ron Babin, Kim Bates, and Sajeev Sohal. The role of trust in outsourcing: More important than the contract? *Journal of Strategic Contracting and Negotiation*, 3(1):38–46, 2017.

[13] Jae-Nam Lee and Byounggu Choi. Effects of initial and ongoing trust in it outsourcing: A bilateral perspective. *Information & Management*, 48 (2):96 – 105, 2011.

[14] Ralph C. Merkle. A certified digital signature. In *"Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238. Springer New York, 1990.

[15] Roberto Tamassia. Authenticated data structures. In *Algorithms - ESA 2003*, pages 2–5. Springer Berlin Heidelberg, 2003.

[16] Premkumar Devanbu, Michael Gertz, Charles Martel, and Stuart G. Stubblebine. *Authentic Third-Party Data Publication*, pages 101–112. Springer US, 2001. ISBN 978-0-306-47008-0.

[17] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 121–132, New York, NY, USA, 2006. ACM. ISBN 1-59593-434-0.

[18] X. Wang, Y. Lin, and G. Yao. Data integrity verification scheme with designated verifiers for dynamic outsourced databases. *Security and Communication Networks*, 7(12):2293–2301, Jan 2014.

[19] Maithili Narasimha and Gene Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications*, DASFAA'06, pages 420–436, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-33337-1, 978-3-540-33337-1.

[20] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 407–418, New York, NY, USA, 2005. ACM. ISBN 1-59593-060-4.

[21] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-703-2.

[22] Simin Ghasemi, Morteza Noferesti, Mohammad Ali Hadavi, Sadegh Dorri Nogoorani, and Rasool Jalili. Correctness verification in database outsourcing: A trust-based fake tuples approach. In *Information Systems Security*, pages 343–351, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[23] Morteza Noferesti, Simin Ghasemi, Mohammad Ali Hadavi, and Rasool Jalili. A trust-based approach for correctness verification of query results in data outsourcing. *JOURNAL OF COMPUTING AND SECURITY*, 1(1):3–14, January 2014. ISSN 0178-4617.

[24] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 2431–2439, Jan 2002.

[25] T. Grandison and M. Sloman. A survey of trust in internet applications. *IEEE Communications Surveys Tutorials*, 3(4):2–16, Fourth 2000. ISSN 1553-877X.

Simin Ghasemi was born in Zanjan, Iran in 1987. She received her B.S. in 2010 from Institute for Advanced Studies in Basic Sciences (IASBS), and M.S. from Sharif University of Technology on data security in september 2010. She is already a faculty member of Payam Noor University of Zanjan. Her research interests include data and database security, trust based models and database outsourcing.

Mohammad Ali Hadavi received his Ph.D. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2015. He received his M.S. and B.S. degrees in software engineering from Amirkabir University of Technology, Tehran, Iran, in 2004, and from Ferdowsi University of Mashhad, Iran, in 2002, respectively. Focusing on information security, he has published more than 30 papers in national and international journals and conference proceedings. His research interests include software security, database security, and security aspects of data outsourcing.

Mina Niknafs received her B.S. degree in information technology engineering from Isfahan University of Technology in 2009 and her M.S. degree in information technology from Sharif University of Technology in 2012. Her research interests include security and trust management, attack tolerance of reputation systems, data and network security.