Research on computing verification algorithm on body sensing data in cloud computing environment

Yanfang Gao\textsuperscript{1,2}, Lan Yao\textsuperscript{*}, Jinyue Yu\textsuperscript{1}

\textsuperscript{1}School of Computer Science and Engineering, Northeastern University, Shenyang, Liaoning, 110169, China
\textsuperscript{2}Medical IT Department, Neusoft Corporation, Shenyang, Liaoning, 110179, China
*Email: yaolan@mail.neu.edu.cn

Abstract. Cloud computing enables users with limited resources to accomplish complex computation. Meanwhile, the massive human body sensing data generated by Wireless Body Area Network (WBAN) can be stored in cloud servers. However, cloud storage of human body sensing data is challenged by some issues. How can users ensure that cloud servers can return computing results and at the same time confirm that the results returned are correct? To address this problem, based on the argument system of commitment scheme, this paper designs a DMP verification algorithm. By compiling the hash of the header node into a constraint set and verifying the accuracy of the data of the intermediate node through a hash list, the algorithm reduces the compiling time of the intermediate node and improves the verification efficiency.

1. Research status of computational verification algorithms

In recent years, the theory of program execution verification has been further developed, and some theoretical researches with great practical significance have made substantial breakthroughs: the verifiable computing protocol based on IPs (Interactive Proof System \cite{1}\cite{2}), PCPs (Probabilistic Checkable Proof System \cite{3}\cite{4}) and cryptography theory \cite{5}\cite{6} can detect the correctness of the results of program execution returned by remote servers with high accuracy, and does not need to make any assumptions about remote servers. Cormode et al. designed CMT protocol using interactive proof system \cite{7}. Shai, Kushilevitz and Ostrovsky proposed that IKO verification system uses linear PCPs to construct verification system, which reduces the time complexity of user authentication \cite{8}. This simplifies the user's verification process and makes the whole verification simpler than before.

The interaction processes of the above verification computing are not encrypted. If you need to hide the verification process, you need to pre-encrypt the queries in the verification. Users use encryption protocols to achieve verification, so they cannot get the actual content of the query.

2. Design of verifiable computing algorithms DMP

2.1. Design of hash compilation algorithms

In order to represent hash function as constraint set, we need to represent hash computing as a compilable program and modularize hash algorithm. Firstly, we encapsulate the hash algorithm in GetBlock and PutBlock functions. Using the hash value of each data block in the DMB+ tree, we compile the hash algorithm from the original node and construct verifiable hash operations.
The verification algorithm determines whether the data of the data block is returned correctly by the name of the data block. If the correct hash value is returned, \( H_{\text{Chain}}(\text{block}) = \text{name} \) holds. Then the correct data can be obtained. If the wrong hash value is returned, it means that the block of data is wrong, and the user does not accept the computing results returned by using the block of data. GetBlock and PutBlock functions are shown in Algorithm 1 and 2.

### Algorithm 1. GetBlock.

**Input:** name \( n \), block;

**Output:** block;

1. \( H = \text{Hash(block)} \);
2. If \( (n = \text{Hash(block)} \)
   - Return Block;
3. Else
   - Return false;
   - Return Block;

### Algorithm 2. PutBlock.

**Input:** block \( b \);

**Output:** Hash \( n \);

1. \( n = \text{Hash}(b) \);
2. store\( (n, b) \);
3. return \( n \);

The following describes the process of converting a hash module into a constraint set: compiling \( b = \text{GetBlock}(n) \) into a constraint set \( C_{H^{-1}} \), where the input variable \( X \) is a name and the output variable is a block. When \( B \in H^{-1}(n) \) (i.e. \( H(b) = n \)), the constraint \( C_{H^{-1}}(X = n, Y = b) \) is satisfied. \( N = \text{PutBlock}(b) \) can be compiled into a constraint set \( C_H \) in the same way, but the input \( X \) of the PutBlock method is Block and the output is name. If and only if \( n = \text{Hash}(b) \), the formula \( C_H(X=b, Y=n) \) holds.

### 2.2. Description of verification algorithms using constraint sets

When DMB+ tree stores data into leaf nodes, it stores data continuously, arranges and connects leaf nodes according to the precursor and suffix of keywords. Each non-leaf node has a hash value as its name, and each leaf node has a hash chain value as its name. This storage method will reduce the hash computation of the verification process, usually in the following way. Load guarantees that the address value is consistent with the hash of the current information. The specific ways are as follows:

Each node is named by the hash (represented as \( h \)) of its content. The content of the internal node is the hash value (name) of the left and right child nodes of the node. Each leaf node corresponds to an address and contains the value currently stored on that memory address. Then, the hash value \( D \) of the stored content is the hash of the content of the root node. If \( A \) holds \( d \), if entity \( B \) claims that the value at address \( A \) is \( \text{Value} \), then \( B \) can prove its declaration by giving \( A \) a witness path: the hash of sibling nodes of \( A \) and the hash of their parent nodes. Then, \( A \) can check whether the hash relationship is valid and match \( D \). For \( B \) to succeed in a false statement, it must identify the hash algorithm used in the computation.

The following algorithm provides the required hash to solve the verification problem in DMB+ tree. The specific algorithm is shown in Algorithm 3 and Algorithm 4.

### Algorithm 3. Load.

**Input:** address \( a \), head node \( h \), tail node \( t \), hash \( d \);

**Output:** value;

1. Block \( \leftarrow a \); // Find data blocks by address
2. \( h \leftarrow d \); // Find the Head Node
3. \( \text{node} \leftarrow \text{GetBlock}(h) \);
4. value \( \leftarrow \text{head.value} \);
5. while \( (h \neq t) \):
   - if \( (h \neq t) \)
     - \( h \leftarrow h.\text{next} \);
     - value.add\( (\text{head.value}) \);
   - else
     - return false;
   - End if;
6. End while;

### Algorithm 4. Store.

**Input:** head node \( h \), tail node \( t \), hash \( d \);

**Output:** hash;

1. \( \text{path} \leftarrow \text{LoadPath(head,tail,d)} \);
2. \( l \leftarrow \lceil \log N \rceil \);
3. \( \text{node} \leftarrow \text{path}[i - 1] \);
4. \( \text{node} . \text{value} \leftarrow v \);
5. \( d' \leftarrow \text{PathBlock(node)} \);
6. For \( i = 1 \) to \( 1 \):
   - \( \text{node} \leftarrow \text{path}[i - 1] \);
   - \( d' \leftarrow \text{PathBlock(node)} \);
7. End for;
8. Return \( d \);
In Algorithm 3, an algorithm for loading data is given. In this algorithm, the accuracy of data is first judged by the hash value of data. If the data is accurate, the block of data will be loaded. The Algorithm 4 gives an algorithm for storing data. In this algorithm, the address of the data storage is found first, then the hash value of the block data is computed, and finally the data and hash value are stored and updated.

Because the storage address of the data is verified to be correct, only the leaf nodes of the stored data need to be verified in the DMB+ tree. In DMB+ tree, leaf nodes are designed as an immutable linked list, and data blocks are connected in series using hash values. Therefore, the integrity and accuracy of intermediate nodes can be guaranteed only by ensuring the accuracy of the head node and the end node in the computing verification part. To invoke loading or storage, the program must begin with the head node of the storage chain. In this method, we call the GetBlock function and the PutBlock function as described earlier. According to the leaf node list returned by the DMB+ tree, the verifier takes the head node hash as input, and according to the storage structure. The hash of data is directly provided by DMB+ tree, so it does not need to be recomputed, thus reducing the number of hash operations.

In the verification process, \( C_{\text{Load}} \) is used to represent the constraint of load coding. Only when the hash value \( h \) is consistent with the value \( v \) of address \( a \), can \( C_{\text{Load}}(X = a, h, Y = v) \) be satisfied and the equation can be established. This is the verification guarantee for the loaded function provided by the verifier. In the Store function, the node content in the list is first placed in the local variable, and then a new verifiable block is created. In order to satisfy the constraints of Store compilation, the verifier must compute the new path through the path, keep the address consistent with \( h \), and compute the new hash, which is consistent with the update of the old path storage. If the same, the head and tail node storage is correct.

Next, we will give a verification algorithm to verify the data of the head and end nodes of the storage structure. The specific algorithm is shown in Algorithm 5.

Algorithm 5. Verify head and tail node Store.

Input: Program \( f(x) \);
Output: True/False;
Run \( f(x) \); // Execute \( f(x) \)
Node \( h \leftarrow f(x) \); // Get the header node based on input \( f(x) \)
Node \( t \leftarrow f(x) \); // Get the tail node based on input \( f(x) \)
Address \( a_h = h.\text{address} \);
Address \( a_t = t.\text{address} \);
\( V_{\text{head}} = h.\text{HChain} \);
\( V_{\text{tail}} = t.\text{HChain} \);
\( C_{\text{HLoad}} = \text{Compile}(h); \ldots \)
\( C_{\text{TLoad}} = \text{Compile}(t); \)
\( C_{\text{HStore}} = \text{Compile}(h); \)
\( C_{\text{TStore}} = \text{Compile}(t); \)
If \( C_{\text{HStore}}(X = (a_h,\text{Hash}(h),h), Y = V_{\text{head}}) \&\& C_{\text{TStore}}(X = (a_t,\text{Hash}(t),t), Y = V_{\text{tail}}) \&\& C_{\text{HLoad}}(X = (a_h,h), Y = V_{\text{head}}) \&\& C_{\text{TLoad}}(X = (a_t,t), Y = V_{\text{tail}}) \)
Return true;
Else
Return false;

2.3. Description of verification algorithms using hash link lists

When the constraint set is used to verify the header node of the data, the data of the header node is guaranteed to be correct and complete. We will use hash linked list to verify the intermediate nodes, to ensure the accuracy of the data and the integrity of the linked list. Starting from the second node, the hash value of the current node needs to be recomputed according to the precursor of the current node. Comparing the newly computed hash value with the stored hash value, we can get whether the data is accurate or not according to the comparison results. For the data block \( b \) in a node, according to the definition of Hash list, we can simply give the hash chain value of \( b \). If the list is incomplete, then the
verification result of Hash list will not be correct, and the user will refuse to accept the computation result. The specific verification algorithm is shown in Algorithm 6.

Algorithm 6. Verify intermediate nodes.

- **Input:** Node head, Node tail;
- **Output:** True / False;

1. Node n = head.next;
2. While (n != tail)
   - h = HChain(Hash(n.date) || n.T.HChain);
   - If (h != n.HChain) Return False;
   - Break;
3. Else n = n.next;
4. End while;
5. Return true;

3. Validation of computational verification algorithms

In the previous section, several algorithms for internal verification of DMP computational verification algorithm are introduced. Next, we will introduce the overall verification idea of computational verification algorithm in detail.

(a) *Initialization stage:* (1) user input program f, and input variable x; (2) The server receives program f and variable x.

(b) *Computational verification stage:* (1) Compute according to the data, get the result of data calculation, and return it to the user; (2) Users compile program f into constraint set C using PCP compiler system in local system, and send verification query request to cloud server through query vector; (3) The cloud server receives the query vector and returns the verification results.

(c) *Data validation stage:* (1) Obtain data link according to program f and input variable x; (2) The user compiles the hash operation of the head and tail nodes into the constraint set, and sends the verification query request to the cloud server through the query vector; (3) Users verify the data correctness of intermediate nodes according to Algorithm 6; (4) The cloud server receives the query vectors and returns the verification results.

4. Algorithmic overhead analysis

The main cost of the algorithm is the computation of hash functions in GetBlock and PutBlock functions. Because of the representation of $C_H$ and $C_{H^{-1}}$ constraints, a large number of hash values need to be computed. This algorithm sacrifices the flexibility of data, stores human body sensing data sequentially, and improves the method of searching the next data block with hash method to the method of searching sequential structured data, thus reducing the computational complexity of hash operation in computation.

5. Implementation and evaluation of algorithms

5.1. *Experimental environment*

This experimental system is based on the open source experimental system provided by the research team of Texas University. In this paper, Pantry system in the system is used as a prototype system to implement the algorithm.

The experimental system is divided into three entities: cloud server, authentication user and data owner. Among them, cloud server is used and Tencent cloud server is used; the data set used in this paper is patient information, case information and body sensing data collected by a hospital. Because these data and patient's information are processed separately in database, we need to connect these data first, and connect the same patient's information. On the premise of guaranteeing the uniqueness of each data, we can reduce the dimension of the data. In the experimental environment, the data set is reduced to four dimensions, as shown in Table 1.
| name   | patient's medical record         | heartbeat | recording time         |
|--------|----------------------------------|-----------|------------------------|
| Li Si  | Hypertension, symptoms ***       | 72        | 2017-06-01-22:45:49    |
| Li Si  | Hypertension, symptoms ***       | 77        | 2018-01-09-13:34:12    |
| Zhang  | Heart disease, symptoms ***      | 73        | 2018-07-12-07:12:45    |
| Wang   | Diabetes, symptoms ***           | 93        | 2018-08-14-00:01:23    |

5.2. Verifiable computing scheme

The interaction among the three entities (data owner, user and cloud server) of the authentication scheme is divided into four stages: data upload, authorization, request query service or computing service and user authentication. Fig. 1 shows the flow chart of the scheme. As can be seen from Fig.1, the data owner has the authority to authorize the user. Data owners can encrypt their data with a unique key and then authorize it with the public key of the user (data visitor). At the same time, the scheme does not rely on trusted third party management, but allows users to authorize access to data visitors. Cloud servers can provide query and computational verification services for users. They will return the response data according to the user’s computational request, and the user will verify the verification object locally.

5.3. Performance analysis of computational verification algorithms

In the task of computing verification algorithm, this paper chooses the main computing methods of body sensing data: linear regression analysis and covariance computing as the main tasks to verify the performance of the algorithm. Two computational programs are submitted separately. In the scheme of argument system based on commitment scheme, we choose Pantry algorithm which is more practical in current research as a comparison algorithm and compare it with the DMP model proposed by us.

5.3.1 Performance Analysis of Data Storage Verification

Next, according to the number of GeteBlock and PutBlock methods encapsulated by DMP algorithm, we analyze the overhead of DMP algorithm in generating constraint sets, and compare it with the number of constraint sets generated by Pantry algorithm. We use $|C|$ to denote the number of constraints set. During the experiment, we divide the contrast experiment into two parts. We compare the number of constraints set generated by Load operation primitive and Store operation primitive in the verification system under the condition of different number of data blocks. Before the experiment, the hash value of
each data block has been computed, and the hash value has been stored in memory. The specific comparison is shown in Table 2 and 3.

Table 2. Number of Load primitive constraint sets.

| Number of data blocks for Load operation | Size of $|C|$ generated by Pantry | Size of $|C|$ generated by DMP |
|-----------------------------------------|----------------------------------|-------------------------------|
| $2^2$                                   | 51129                            | 11281                         |
| $2^5$                                   | 56473                            | 17717                         |
| $2^{10}$                                 | 67183                            | 28916                         |
| $2^{15}$                                 | 83749                            | 39720                         |
| $2^{20}$                                 | 93179                            | 47097                         |

Table 3. Number of Store primitive constraint sets.

| Number of data blocks for store operation | Size of $|C|$ generated by Pantry | Size of $|C|$ generated by DMP |
|-----------------------------------------|----------------------------------|-------------------------------|
| $2^2$                                   | 54012                            | 15187                         |
| $2^5$                                   | 70238                            | 20732                         |
| $2^{10}$                                 | 90128                            | 34371                         |
| $2^{15}$                                 | 152793                           | 42877                         |
| $2^{20}$                                 | 190548                           | 57367                         |

From Table 2 and Table 3, it can be seen that the size of constraint set generated by DMP algorithm is better than that generated by Pantry algorithm in both storage and loading operations. In the process of verification, the data accuracy of intermediate nodes of DMP algorithm relies on Hash list in DMB+ tree to verify, instead of using constraint set verification method in original algorithm. Table 4 compares memory overhead and network overhead when storing and verifying 500,000 pieces of data.

Table 4. Memory cost and network overhead comparison

|                                      | DMP     | Pantry  |
|--------------------------------------|---------|---------|
| Memory cost                          | 25MB    | 25MB    |
| Input network overhead               | 25MB    | 3KB     |
| Network overhead in argumentation    | < 50B   | 534B    |

As can be seen from Table 4, DMP and Pantry algorithms require additional memory to store hash values and constraints generated after compilation in verification. In the verification phase, Pantry algorithm reduces network overhead by compiling all hash operations locally into constraint sets. DMP algorithm sends hash values to users for verification, so it needs more network overhead.

5.3.2 Analysis of computational verification performance in client-side

We select linear regression analysis and covariance computing tasks, submit two types of computing programs to the system as their tasks, and compare the results of client-side execution of the two kinds of computing in the improved experimental system to verify and analyze the feasibility of our proposed DMP algorithm. In the experiment, we selected 500,000 data to compute. In the computing process, we select two fields in the record to determine whether there is a linear relationship as the computing task of linear regression analysis, and compute the covariance of the two fields as the covariance calculation task. In the experiment, the number of constraints generated by DMP algorithm and Patry algorithm is compared, and the verification time at the user side is also compared. The experimental comparison of the number of constraint sets is shown in Table 4. The comparison of authentication time between client and cloud services is shown in Table 5 and 6.

Table 5. Linear regression computing task time cost comparison

|                | Pantry | DMP    |
|----------------|--------|--------|
|                |        |        |
|                  | Pantry | DMP  |
|------------------|--------|------|
| Initial time (s) | 58     | 73   |
| Argumentation time (s) | 1432 | 732  |

Table 6. Covariance computing task time cost comparison

From Table 5 and Table 6, we can see that Pantry's verification time expenditure mainly concentrates on the verification stage, and the verification time accounts for 70%~85% of the total verification time expenditure. Because both algorithms use the same compilation method in the computational compilation stage, there is no obvious difference between DMP algorithm and Pantry algorithm in the initialization stage. In the Argumentation stage, the time of DMP algorithm is reduced by 45%~55% compared with Pantry algorithm.

6. Conclusion

Although the research on verifiable computing has achieved more results, it is still in the theoretical stage, and there is no algorithm that can be fully used and applied in practice. Based on the reality, this paper designs a verifiable computing algorithm according to the characteristics of body sensing data, and implements the verification algorithm model in the experimental system. As an application case of the proposed algorithm model, the feasibility of the computing verification of the model is proved by the results of the actual task, and the advantages of this method are proved by comparing the existing algorithms.

Acknowledgments

This work was supported by National Natural Science Foundation of China (61602106).

References

[1] Goldwasser S, Kalai Y T and Rothblum G N 2015 Delegating Computation: Interactive Proofs for Muggles J. Acm. 62 1-64
[2] Peikert C and Vaikuntanathan V 2008 Noninteractive statistical zero-knowledge proofs for lattice problems Pro. of conference on cryptology: advances in cryptology. Berlin (Springer-Verlag), 536-53
[3] Jaikumar R and Madhu S 2007 On Dinur’s Proof of the PCP Theorem Bulletin of the American Mathematical Society, 44 19-61
[4] Yuval I, Mohammad M and Amit S 2012 On Efficient Zero-Knowledge PCPs Theory of Cryptography. Berlin (Springer-Verlag) 151-68
[5] Gennaro R, Gentry C, and Parno B 2010 Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers Conference on Advances in Cryptology Berlin (Springer-Verlag) 465-82
[6] Gentry C and Wichs D 2011 Separating succinct non-interactive arguments from all falsifiable assumptions Proc. of the annual acm symposium on theory of computing (ACM) 99-108
[7] Cormode G, Mitzenmacher M and Thaler J 2012 Practical verified computation with streaming interactive proofs Innovations in Theoretical Computer Science Conference (ACM) 90-112
[8] Ishai Y, Kushilevitz E and Ostrovsky R. 2007 Efficient Arguments without Short PCPs IEEE Conference on computational Complexity (IEEE) 278-91