Information Sciences 258 (2014) 371–386

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Security and privacy for storage and computation in cloud computing

Lifei Wei [a], Haojin Zhu [a], Zhenfu Cao [a,*], Xiaolei Dong [a], Weiwei Jia [a], Yunlu Chen [a], Athanasios V. Vasilakos [b]

[a] *Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China*
[b] *Department of Computer and Telecommunications Engineering, University of Western Macedonia, Kozani, Greece*

ARTICLE INFO

Article history:
Available online 27 April 2013

Keywords:
Secure computation auditing
Secure storage
Privacy-cheating discouragement
Designated verifier signature
Batch verification
Cloud computing

ABSTRACT

Cloud computing emerges as a new computing paradigm that aims to provide reliable, customized and quality of service guaranteed computation environments for cloud users. Applications and databases are moved to the large centralized data centers, called *cloud*. Due to resource virtualization, global replication and migration, the physical absence of data and machine in the cloud, the stored data in the cloud and the computation results may not be well managed and fully trusted by the cloud users. Most of the previous work on the cloud security focuses on the storage security rather than taking the computation security into consideration together. In this paper, we propose a privacy cheating discouragement and secure computation auditing protocol, or *SecCloud*, which is a first protocol bridging secure storage and secure computation auditing in cloud and achieving privacy cheating discouragement by designated verifier signature, batch verification and probabilistic sampling techniques. The detailed analysis is given to obtain an optimal sampling size to minimize the cost. Another major contribution of this paper is that we build a practical secure-aware cloud computing experimental environment, or *SecHDFS*, as a test bed to implement SecCloud. Further experimental results have demonstrated the effectiveness and efficiency of the proposed SecCloud.

1. Introduction

The recent development of cloud computing has shown its potential to reshape the current way that IT hardware is designed and purchased. Among numerous benefits, cloud computing offers customers a more flexible way to obtain computation and storage resources on demand. Rather than owning (and maintaining) a large and expensive IT infrastructure, customers can now rent the necessary resources as soon as, and as long as, they need [1]. Thus, customers cannot only avoid a potentially large up-front investment (which is particularly attractive for small companies and startups), they may also be able to reduce their costs through economies of scale and by paying only for the resources they actually use.

Even though cloud computing is envisioned as a promising service platform for the Next Generation Internet [14], security and privacy are the major challenges which inhibit the cloud computing wide acceptance in practice [31]. Different from the traditional computing model in which users have full control of data storage and computation, cloud computing entails that the managements of physical data and machines are delegated to the cloud service providers while the users only retain some control over the virtual machines. Thus, the correctness of data storage and computation might be compromised due to

* Corresponding author. Tel.: +86 21 34204642.
E-mail address: zfcao@cs.sjtu.edu.cn (Z. Cao).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.04.028

the lack of the control of data security for data owners. In this study, we further classify cloud computing security into two major classes: *Cloud Storage Security* and *Cloud Computation Security*, where the former is referred to ensuring the integrity of outsourced data stored at untrustworthy cloud servers while the latter refers to checking the correctness of the outsourced computation performed by untrustworthy cloud servers.

Most of the current researches on secure cloud computing still focus on the cloud storage security. However, the outsourced computation security receives less attention. For sake of saving computation resources, the cloud servers may not perform the necessary computations but claim to have done so. Additionally, the centralized architectures emphasize the fact that the cloud servers can represent a single point of failure, as proven by the recent meltdown of Google's Gmail systems [25]. Under Byzantine [11] failure or even external attacks, the cloud may perform unreliable computation operations while choosing to hide the computations. This cheating behavior of the cloud servers, if undetected, may render the results useless. Even from the point of accountability, some secure computation mechanisms should be in place to meet the needs of deciding whether the cloud servers or the users should be responsible for it once there is any problem taking place. Note that, it is quite natural for the servers to initially suspect a problem with the customer's software, and vice versa [18].

Generally, due to the limitation of the computation and communication resources, the cloud users cannot afford the cost incurred by result auditing or verification. One promising approach to prevent the cloud users from incurring expensive verification costs is to introduce a trusted auditor who conducts cloud auditing on behalf of the users. Even though public auditability of secure storage in cloud [33,34] has been proposed in the context, public auditability in secure computation receives less attentions. More closely related references are secure remote computation in distributed system [24]. However, few of those proposed schemes target at secure cloud computation. Furthermore, privacy preserving is a critical issue for secure cloud computing while only a few of the existing researches [20,29] have taken it into consideration.

To achieve secure computing auditing in cloud, one straightforward method is to double-check each result. The cloud providers may give the inputs and overall computing result to the auditor, which will follow an identical procedure to compute the result and then compare it with the one provided by the cloud providers. However, these schemes may lead to a waste of I/O and computation resources. Note that the data transferring bottlenecks rank in the top of the ten obstacles which may prevent the overall success of the cloud computing [1]. In [13], a Commitment-Based Sampling (CBS) technique is introduced in the conventional grid computing however it does not take the privacy issue into consideration. In this paper, we introduce a novel technique by integrating CBS with the designated verification technique.

The contributions of this paper can be summarized as follows.

- Firstly, we model the security problems in cloud computing and define the concepts: *uncheatable cloud computation* and *privacy cheating discouragement* in our cloud computing, which are our design goals.
- Secondly, we propose a basic protocol, **SecCloud**, to attain data storage security and computation auditing security as well as privacy cheating discouragement and an advanced protocol to achieve computation and communication efficiency improvement through batch verification.
- Thirdly, we analyze and prove that **SecCloud** achieves our design goals and discuss how to minimize the computation cost by choosing the optimal sampling size.
- Finally, we develop a cloud computing experimental environment **SecHDFS** and implement **SecCloud** as a test bed. Experiment results demonstrate the suitability of the proposed protocol.

The remainder of this paper is organized as follows. A brief review on the related work is given in Section 2. Section 3 describes the system architecture and security problems and presents design goals. Some necessary preliminary knowledge is given in Section 4. We propose an overview of our **SecCloud** in Section 5 and then present an advanced **SecCloud** with performance optimization in Section 6. Section 7 gives out detailed security analysis and discussion. Section 8 introduces the experiment environment **SecHDFS** and implement our **SecCloud** as a test bed. Finally, Section 9 concludes the whole paper.

2. Related work

Security and privacy issues in cloud computing has received extensive attentions recently. Generally speaking, the research work on cloud computing almost falls into the two cases: *cloud storage security* and *cloud computation security*.

Cloud storage security mainly addresses the secure outsourced storage issue. In [2], Ateniese et al. first defined a model for *provable data possession* (PDP), which allowed a client that had data stored at an untrusted server to verify that the server possessed the original data without retrieving it. They utilized RSA-based homomorphic tags for auditing outsourced data, but they did not consider the dynamic data storage. In their later work, Ateniese et al. [3] proposed a partially dynamic version of the PDP scheme using symmetric key cryptography. However, it did not support public auditability. Juels et al. [22] proposed the definition of *proof of retrievability* (PoR), which used spot-checking and error-correcting codes to ensure both possession and retrievability for data file on archive service system. Wang et al. [34] first achieved both public verifiability and dynamic data storage operations employing an Third Party Auditor and improving the proof of retrievability model by using classic Merkle Hash Tree [26] construction for BLS [8] based block tag authentication. Later, they proposed a scheme achieving privacy preserving public verifiability as well as the dynamic data storage operations in [33] by utilizing the public key based homomorphic authenticator and uniquely integrate it with random mask technique. The further work explored the technique of bilinear aggregate signature for TPA can verify data auditing in a complexity of $O(n)$. Erway et al. [15] proposed the first construction of dynamic provable data possession, which extended the PDP model in [2] to achieve provable updating stored data using rank-based authenticated skip lists.

Compared with secure cloud storage, secure cloud computation still receives less attentions. The related work include remote computation audit and verifiable computation. [17] proposed a ringer scheme in distributed computing where the supervisor sent to the participant some pre-computed results without disclosing the corresponding inputs. [27] presented a remote audit mechanism on an existing distributed computing model and provided efficient methods for verifying whether a remote host performed the assigned task. Similar to our prior work [35,16] introduced and formalized a notion of verifiable computation, which allows a weak client to outsource the computation of a function on various dynamically-chosen inputs to the help workers, which return the results as well as proofs that the computation was carried out correctly on the given input value. The primary constraint is that the verification of the proof should require substantially less computational effort than computing the function again from scratch. Further work about verifiable computation could be found in [9,30] which consider the efficiency of the outsourcing computation. The incentive issues of outsourcing computation have been considered in [4] to prevent from cheating.

3. Problem formulation

In this section, we present the system architecture, model formulation and design goals.

3.1. System architecture of cloud computing

As shown in Fig. 1, we consider a general cloud computing model constituted of a number of *cloud servers*, $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_N$, which are under the control of one or multiple *cloud service providers* (**CSP**). These cloud servers process plenty of computation resource and storage resource. **CSP** allocates these resources by means of customized Service Level Agreements [28]. For example, to perform a batch-processing tasks, by employing the existing programming abstraction techniques such as MapReduce [12] and its open-source counterpart Hadoop [5], **CSP** divides such a large task into multiple small sub-tasks and allows them parallelly executed across up to hundreds of cloud servers.

We assume the *cloud user* (**CU**), such as a mobile phone, a laptop, and an apple ipad, which has lower computation resource and smaller storage resource than those of the cloud servers. Most of the communication are through wireless, even if an ordinary computer with limited hours (not 24-h) wired-connecting to the cloud servers. **CU** would submit storage service requests and computation service requests to **CSP** when it demands.

Similar to existing secure storage auditing schemes, we also assume the existence of a number of *verification agencies* (**VAs**), which are chosen and trusted by **CU** and responsible for auditing the cloud services on data storage and computation. **VAs** are expected to have more powerful computation and storage capability to perform the auditing operations than those of **CU**.

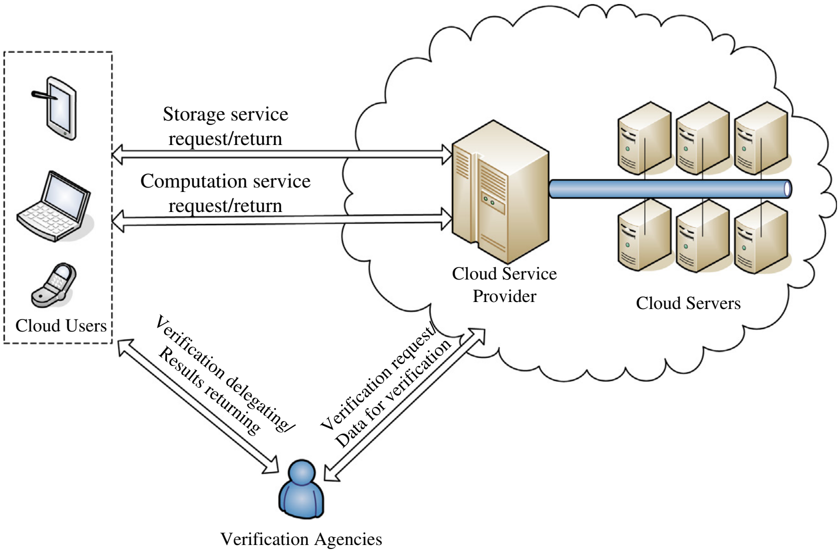

Storage service request/return
Computation service request/return
Cloud Service Provider
Cloud Servers
Cloud Users
Verification delegating/ Results returning
Verification request/ Data for verification
Verification Agencies

Fig. 1. Cloud computing architecture in our protocol.

3.2. Adversarial models

In our assumption, the adversary $\mathcal{A}$ could corrupt a small set of cloud servers and control these servers to launch various cheating attacks as $\mathcal{A}$'s wish. Obviously, according to the different goals, these attacks are summarized as follows.

- *Storage-cheating attack model.* When the attacks towards data storage security in the cloud, for example, the adversary would arbitrarily modify the stored data to compromise the data integrity (malicious case) or reveal the confidential data to purchase interest (interest-purchasing case) or in both of cases. In the malicious case, the compromised cloud servers would simply reply to the cloud users' storage queries with a random number. It is a great challenge for cloud users due to lacking the physical possession of the potentially large size of outsourced data. We assume that if the request data set is X, the honest returned data set is X' and the invalid returned data set is $X - X'$.
- *Computation-cheating attack model.* When the attacks towards data computation security of the cloud, we assume that a complicated computation request, denoted F, is comprised of a set of subtasks $\{f_1, f_2, \ldots, f_n\}$[1] and each of subtasks may involve the input data x_{p_i} located at position p_i in the cloud server. Thus, the expected computation result is $R = \{f_1(x_{p_1}), f_2(x_{p_2}), \ldots, f_n(x_{p_n})\}$ for further computing $F(R)$. The adversary would cheat by the following three ways: The adversary partially computes $f_i(x_{p_i})$ for some is $(1 \leqslant i \leqslant n)$ and returns random numbers for the rest, but claims to have completed all the computation; the adversary sophisticatedly takes other $\hat{x}$ where $\hat{x} \notin X$ and leads to much lower computational cost; even claims to use the correct data x but the original data x is just missing. We denote the set $\{f_i\}$ as F' in which the subtasks f_i are carried out honestly and the set $F - F'$ where the subtasks are not carried out honestly.
- *Privacy-cheating attack model.* When the attacks towards privacy issue of the cloud, which can be viewed as another kind of storage-cheating attack, we assume that the adversary may compromise cloud users' privacy by leaking their confidential data to others, e.g. healthy condition to public or auction price to business competitors, which would lead to serious consequences. To provide data confidentiality, one straightforward approach is to save encrypted the data in the cloud servers. However, such an approach may prevent the regular cloud computation from being further processed.[2] Besides, if the data are stored in a plaintext in the cloud servers, the adversary in the interest-purchasing case may break into and sell/publish the sensitive data to the public. Furthermore, we assume that to sell the sensitive data, the adversary should provide the corresponding proofs to demonstrate the authenticity of the stored data and computing results to convince others.

3.3. Secure cloud computing

3.3.1. Uncheatable cloud computation

To formally define the security model in the cloud computing, we introduce two concepts *Secure Computation Confidence* (**SCC**) and *Secure Storage Confidence* (**SSC**) to indicate the trust level of computation security and storage security, respectively. **SCC** is defined as $|F'|/|F|$ and **SSC** is formalized as $|X'|/|X|$. In both cases, cloud computation or cloud storage is regarded as fully trusted if **SCC** (**SSC**) equals 1. Otherwise, it is semi-trusted.

Definition 1 (*Uncheatable cloud computation*). Let $\Pr[\mathtt{CheatingSuccessfully}]$ be the probability that an adversary with the trust level of **SCC** and **SSC** could successfully cheat without being detected by sampling based verifiers. We denote the computation is uncheatable, if for arbitrary sufficiently small positive number ϵ, there exists a sampling size t such that the following conditions always satisfies:

$$\Pr[\mathtt{Cheating\ Successfully}] = \Pr[SCC, SSC, t] < \epsilon. \tag{1}$$

3.3.2. Privacy-cheating discouragement

To discourage the adversary from leaking the cloud users' sensitive data, we introduce a novel privacy-cheating discouragement model where the adversary wants to illegally sells the cloud users' sensitive data to others. Similar to software sales [21], the software vendor may embed a digital signature in its products to allow the users to authenticate them. Such an authentication could be strictly limited to paying customers rather than the illegitimate users to avoid software piracy. Therefore, it is required that any storage and computation auditing should be authorized by the cloud users. In other words, this could discourage adversaries from leaking users' private data. To achieve this target, we introduce the following definition.

Definition 2 (*Privacy cheating discouragement*). Let $\mathtt{InfoLeak}$ denote the event that valid information is leaked by the adversary. The cloud computing is privacy cheating discouragement, if for a sufficiently small positive number ϵ, the following equation holds in a polynomial time t:

$$\Pr[\mathtt{InfoLeak}] < \epsilon. \tag{2}$$

[1] A research by [24] was also dividing sequential tasks into smaller subtasks, permuting them among participants and then enabling the detection of individual and colluding malicious participants.

[2] Most of the current secure multi-computation (SMC) protocols are at the theoretical level and may be impractical implementation in cloud computing [32].

3.4. Design goals

The proposed protocol is expected to achieve the following security and performance goals:

- *Data storage security*: To make sure that the data are securely stored in cloud, the proposed protocol should enable that **CU** and **VA** could fetch and audit the pre-stored data effectively.
- *Data computation security*: To achieving secure computation, the proposed protocol should ensure that the computation be audited by **CU** and **VA**. Considering the fact that some cloud users suffer from computation and transmission constraints, **VA**'s verification is a promising approach for securing cloud computation.
- *Privacy cheating discouragement*: The proposed scheme should ensure that only designated verification parties (e.g., **CSP** or **VAs**) could verify the stored data or computing results, which can discourage the **CSP** from compromising users' privacy, even if the cloud servers are compromised by the attackers.
- *Efficiency:* The computation and transmission overhead of the auditing should be reduced, as is best to meet the minimum.

4. Preliminaries and notation

4.1. Bilinear pairing

Let $\mathbb{G}_1$ be a cyclic additive group with an operation $(+)$ and $\mathbb{G}_2$ be a cyclic multiplicative group with an operation $(\cdot)$. Both of them have the same prime order q. Let P be a generator of $\mathbb{G}_1$. An efficient admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, with the following properties: (1) for all $P \in \mathbb{G}_1$ and $a,\ b \in \mathbb{Z}_q^*,\ \hat{e}(aP, bP) = \hat{e}(P, P)^{ab}$; (2) Non-degenerate: There exists $P \in \mathbb{G}_1$ such that $\hat{e}(P, P) \neq 1$; and (3) Computable: there is an efficient algorithm to compute $\hat{e}(P, Q)$ for any $P, Q \in \mathbb{G}_1$. Typically, we can implement the bilinear map using Weil or Tate pairing [6]. Most of the identity based cryptographic schemes are achieved by employing this technique.

4.2. BDH problem and BDH assumption

The Bilinear Diffie–Hellman (BDH) problem in $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$ is as follows: Given (P, aP, bP, cP) for some unknown $a,\ b,\ c \in \mathbb{Z}_q^*$, compute $\hat{e}(P, P)^{abc} \in \mathbb{G}_2$. An algorithm $\mathcal{A}$ has advantage ϵ in solving BDH in $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$ if

$$\Pr\left[\mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}\right] \geqslant \epsilon$$

where the probability is over the random choice of $a,\ b,\ c \in Z_q^*$, the random choice of $P \in \mathbb{G}_1$, and the random bits of $\mathcal{A}$.

We denote the BDH assumption that tere exists no algorithm $\mathcal{A}$ has advantage ϵ in solving the BDH problem in any polynomial time, That is,

$$\Pr[\mathtt{BDH}] \leqslant \epsilon.$$

The BDH assumption is a variant of the computational Diffie–Hellman assumption [6].

4.3. Designated verifier signature

It is a special signature scheme [37,23,19] that the designated verifier can take its private key and the signer's public key to verify the signatures, but it is unable to use this signature to convince any other parties that the message is indeed signed by the original signer, even if the verifier is willing to reveal its private key. This is achieved on that the verifier could take advantage of its private key to generate a fake signature and any other parties are unable to distinguish whether these messages are authentic, i.e. whether they have been signed by the users or not.

4.4. Merkle hash tree

Merkle hash tree is a well-known authentication structure proposed by Merkle [26], which is constructed as a binary tree where each leaf of the tree is a hash value of authentic data values. It is often used to efficiently and securely to ensure the authencithy and integrity.

4.5. Sampling technique

In order to obtain data verification, a solution is to use sampling technique. The verifier randomly selects a small number of inputs value; it only double-checks the results of these sample inputs. If the dishonest cloud server computes only one half of the inputs, the probability that it can successfully cheat the supervisor is one out of 2^m, where m is the number of samples. If we make m large enough, e.g. $m = 50$, the cheating is almost impossible. This solution has a very small computational overhead $(\mathcal{O}(m))$ compared with double-check all the computation results.

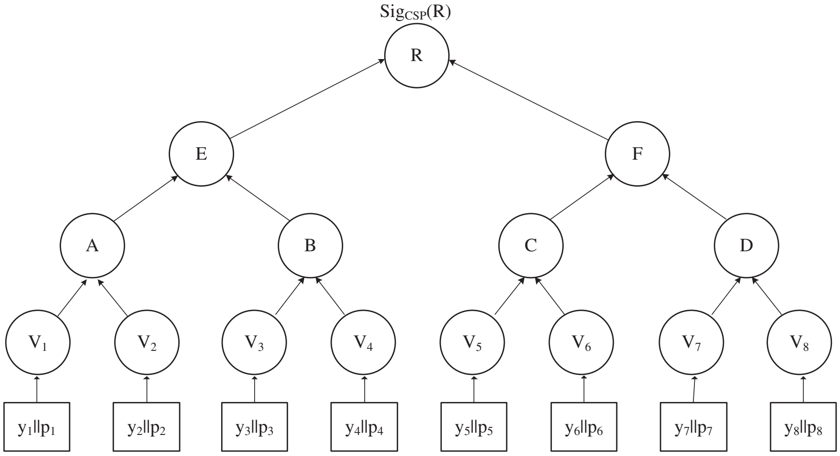


Fig. 2. Construct a Merkle hash tree based commitment scheme.

5. The basic SecCloud protocol

To achieve secure cloud computing, we propose a basic protocol relying on the identity-based cryptography. Our proposed protocol consists of four phases: "*system initialization*", "*secure cloud storage*", "*secure cloud computation*", and "*computation result auditing*". Fig. 1 illustrates an overview of data and service flows in the protocol.

5.1. System initialization

5.1.1. System setup step

The *System Initialization Operator* (**SIO**)[3] generates the system parameters and master secret keys. Specifically, **SIO** selects two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and an admissible pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$; then it chooses cryptographic hash functions $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$[4], $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*, H_3 : \mathbb{G}_2 \rightarrow \mathbb{Z}_q$. After setting the system parameters, **SIO** picks up a random number $s \in \mathbb{Z}_q^*$ as its master secret key and chooses an arbitrary generator $P \in \mathbb{G}_1$ and sets its public key as $P_{pub} = s \cdot P$. Finally, the system parameters are $params = (\mathbb{G}_1, \mathbb{G}_2, q, \hat{e}, P, P_{pub}, H, H_1, H_2, H_3)$. The system's secret key s is safely kept.

5.1.2. User registration step

When a cloud user applies for the cloud services, it first needs to register to **SIO**. The cloud user submits its identity *ID* to **SIO** and receives system parameters *params* and a secret key sk_{ID} from **SIO** in a secure way, such as TLS or SSL. Specifically, the procedure is as follows:

$$sk_{ID} = s \cdot Q_{ID} \tag{3}$$

where $Q_{ID} = H_1(ID)$. Note that system setup and registration step could be performed off-line.

5.2. Secure cloud storage

Our secure cloud storage protocol includes the following four steps:

5.2.1. Storage space applying step

Before the data are transmitted to the cloud, the cloud user first applies the storing space for its messages $M = \{m_1, m_2, \ldots, m_n\} \in \mathbb{Z}_q$. The cloud service provider allocates these space and returns the space index set $I = \{i_1, i_2, \ldots, i_n\}$ to **CU**.

5.2.2. Data signing step

To enable the storage data auditing, the cloud user needs to sign each transmission block m_i to generate an authenticated information in Algorithm 1. To preserve user's privacy in our model, **CU** makes a little modification of the traditional signature scheme according to the designated verifier signature. **CU** also chooses a trusted verification agency and then does Algorithm 1 to sign each block.

[3] In reality, the government or other trusted third party (**TTP**) plays the role of **SIO**. Since the system initialization step could be performed off-line, it will not bring heavy burden to these organization. According to [36], the key escrow problems have been considered in identity based signature schemes to prevent SIO's abuse.

[4] For simplicity, *H* is only used for constructing the Merkle hash tree.

Algorithm 1. Data signing algorithm.

```
Data: Input $n$ un-signed data $D = \{m_1, m_2, ..., m_n\}$ and space index
      $I = \{i_1, i_2, ..., i_n\}$;
Result: Output $n$ signatures on these data
Let $\Phi = \emptyset$;
for $i = 1$ to $n$ do
    Random choose a numbers $r_i \in \mathbb{Z}_q^*$;
    Compute signature part I : $U_i = r_i \cdot Q_{ID}$;
    Use a hash function $h_i = H_2(U_i \| m_i \| i_i)$;
    Compute signature part II: $V_i = (r_i + h_i) \cdot sk_{ID}$;
    Generate their designated verifier signatures $\Sigma_i = \hat{e}(V_i, Q_{CS})$ and
    $\Sigma_i' = \hat{e}(V_i, Q_{VA})$ for cloud server and verification agency;
    Denote $\sigma_i = (U_i, \Sigma_i, \Sigma_i')$;
    $\Phi = \Phi \cup \{\sigma_i\}$;
end
return $\Phi$;
```

5.2.3. Data encapsulation step

To guarantee the secure data transmissions, after identifying the cloud server or the verification agency, the cloud user pre-computes the session key $key_{ID,CS}$ or $key_{ID,VA}$ as following:

$$key_{ID,CS} = H_3(\hat{e}(sk_{ID}, Q_{CS})) \text{ or } key_{ID,VA} = H_3(\hat{e}(sk_{ID}, Q_{VA}))$$

At the cloud side, the cloud server could also compute the session key

$$key_{CS,ID} = H_3(\hat{e}(sk_{CS}, Q_{ID}))$$

Here, a pair of symmetrical keys between **CU** and **CSP** are shared, since

$$\begin{aligned} key_{CS,ID} &= H_3(\hat{e}(sk_{CS}, Q_{ID})) = H_3(\hat{e}(s \cdot Q_{CS}, Q_{ID})) = H_3(\hat{e}(Q_{CS}, s \cdot Q_{ID})) \quad (\because bilinearity\ of\ \hat{e}) = H_3(\hat{e}(Q_{CS}, sk_{ID})) \\ &= H_3(\hat{e}(sk_{ID}, Q_{CS})) = key_{ID,CS} \end{aligned} \tag{4}$$

Due to the hardness of Bilinear Diffie–Hellman (BDH) problem, the session key is secure against the external attacks. Finally, the user sends the data and corresponding signature pairs $\{D, \Phi\}$ encrypted by this session key to **CSP**.

5.2.4. Data receiving step

After receiving the pactkets, **CSP** first decrypts them by its own session key to obtain the data-signature pairs $\{D, \Phi\}$ and checks the signatures by verifying Eq. (5) using its secret key sk_{CS}:

$$\Sigma_i = \hat{e}(U_i + H_2(U_i \| m_i \| i_i) Q_{ID}, sk_{CS}) \tag{5}$$

If Eq. (5) holds, **CSP** is convinced that users' data are securely transferred without malicious tampering. Then **CSP** arranges storing these data in the space $\{I\}$ and returns an acknowledge to cloud user, which makes cloud user delete $\{D, \Phi\}$ from local storage as general cloud computing model. Otherwise, a "Complaint Message" is broadcast to alert a invalid data.

Note that, privacy cheating is discouraged in our proposed protocol since that only **CSP** and designated **VA** could verify the results and the positions of data storage whereas any other parties could not check it since they do not have the corresponding private keys.

5.3. Secure cloud computation

Our secure cloud computation protocol is based on a commitment scheme using Merkle hash tree, which includes the following two steps:

5.3.1. Computation request step

When a cloud user submits a computation service request, which could be viewed as a set of functions $F = \{f_1, f_2, \ldots, f_n\}$ and the positions index of data blocks $P = \{p_1, p_2, \ldots, p_n\}$. The functions $f_i \in F$ can be considered as basic functions such as data sum and average, and other complicated computations based on these basic functions and on the data which are stored in the position $p_i \in P$.

5.3.2. Commitment generation step

When **CSP** receives the computation requests $\{F,P\}$, **CSP** divides such a large task into multiple small sub-tasks and allows them executed parallelly on the different cloud servers based on the data positions. Each cloud server first finds the data in the position p_i, computes the function as $y_i = f_i(x_{p_i})$ honestly and returns to **CSP**. The later constructs a Merkle hash tree with n-leaves, each value of which V_i can be calculated by

$$\Omega(V_i) = H(y_i \| p_i),$$

and builds from bottom to top. For each internal node V, its value $\Omega(V)$ could be derived from its children node value as follows:

$$\Omega(V) = H(\Omega(V_{LeftChild}) \| \Omega(V_{RightChild})) \tag{6}$$

where $V_{LeftChild}$ and $V_{RightChild}$ denotes V's two child nodes. Fig. 2 shows an example to construct a Merkle hash tree with data and computation results. We denote R as the root value of the Merkle hash tree. The cloud service provider signs the root R and obtains the signature $Sig_{CS}(R)$. Finally, the cloud service provider returns the results $Y = \{y_i | (1 \leqslant i \leqslant n)\}$, the commitment value R, and its signature $Sig_{CS}(R)$ to the cloud user.

5.4. Computation result auditing

To achieve the secure cloud computation, **CUs** need to perform result verification. In our scheme, the result verification also includes the storage verification to ensure that the data are not only stored at the correct positions but also indeed taken into computation.

Due to the resource limitation, an ordinary cloud user could just afford the commitment verification. More specifically, the cloud user re-builds the Merkle hash tree, compares the root value, and verifies the signature. Furthermore, the cloud user needs to delegate the verification right to its trusted **VA** for further verification if necessary. More specifically, the cloud user proceeds as follows: it sends $\{F,P,Y,R,Sig_{CS}(R)\}$ as well as a *warrant* include the identity of the delegatee right and the expired time to **VA** for auditing.

Note that, **VA** not only checks if the data has been appropriately stored at the cloud server but also ensures if the computation process has been performed correctly. In our assumption, even though **VA** is expected to have more powerful computational and transmission capability than the cloud users, it is not cost-effective to fetch each of the data block in P and re-compute each $f_i(\cdot)$ to check if the cloud storage and computation is well performed. Thus, the probabilistic sampling technique is adopted here to reduce the overall verification cost. The detailed protocol includes the following steps:

5.4.1. Auditing challenge step

When **VA** starts to audit the results, it first picks a random sampling index set $C = \{c_1, c_2, \ldots, c_t\}$ from the domain $\{1,2,\ldots,n\}$. Then **VA** sends these challenge requests to **CSP** as well as the warrant.

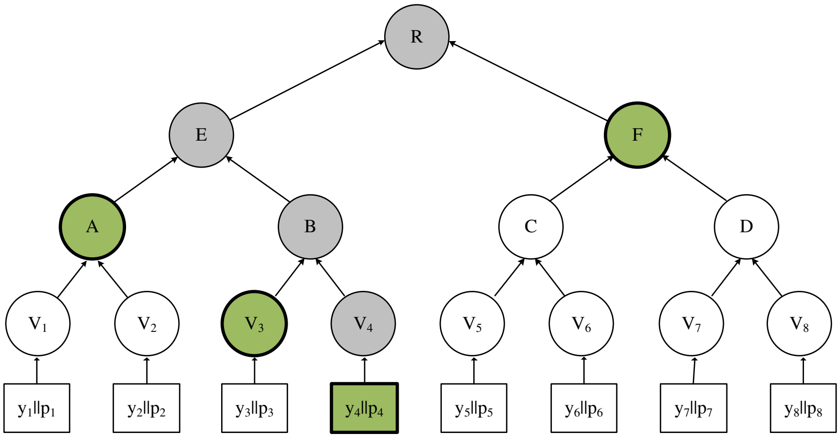


Fig. 3. Re-construct a Merkle hash tree for example. The challenge on $f_4(m_4)$ needs to find a path ϕ_4 including the vertices $\{V_4,B,E,R\}$. We have shown them in dark. To perform this computation, each node's sibling vertex are required to compute the root R. Thus, $\psi_4 = \{V_3,A,F\}$. **CSP** returns the data m_4, its signature Σ_4' and U_4, and $\psi_4 = \{V_3,A,F\}$ to the challenger **VA**. We show them in green. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.4.2. Auditing response step

When **CSP** receives these auditing challenges, it first verifies the warrant to check whether it is expired. After that, for each sampling index c_i, **CSP** finds the stored data m_{c_i} and its signatures Σ'_{c_i} and U_{c_i} which are used to audit secure storage in the cloud. **CSP** finds the path ϕ_{c_i} in the Merkle hash tree of auditing computation from the leaf value c_i to the root R. For each node on this path ϕ_{c_i}, **CSP** collects the sibling nodes in the set ψ_{c_i}. Finally, **CSP** returns the $\{m_{c_i}, U_{c_i}, \Sigma'_{c_i}, \psi_{c_i}, R\}$ (see Fig. 3).

5.4.3. Response verification step

When **VA** gets the results from the cloud, **VA** verifies the storage correctness and the computation correctness as shown in Algorithm 2.

Firstly, **VA** checks whether the cloud servers use the data in the request positions, not other positions. **VA** verifies the signature Σ'_{c_i} using the algorithm

$$\texttt{VerifySigByVA}(U_{c_i}, \Sigma'_{c_i}, m_{c_i}, p_{c_i}, sk_{VA}),$$

which returns true if the following equation is satisfying:

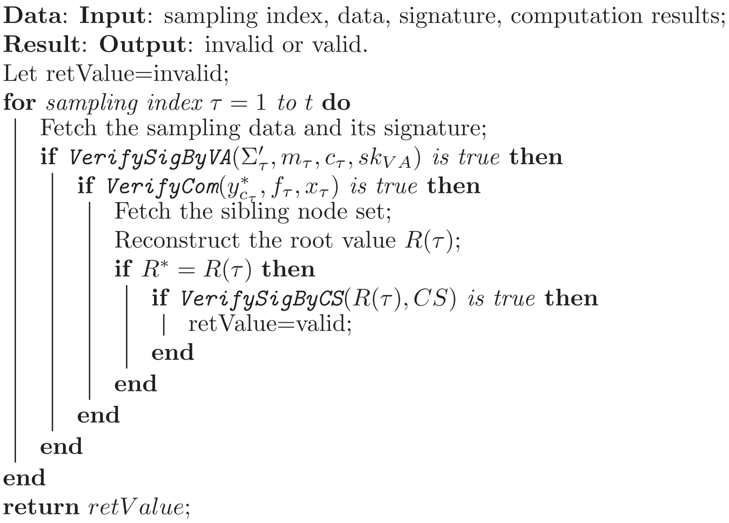
$$\Sigma'_{c_i} = \hat{e}(U_{c_i} + H_2(U_{c_i} \| m_{c_i} \| p_{c_i})Q_{ID}, sk_{VA}). \tag{7}$$

If the signature Σ'_{c_i} is correct, **VA** is convinced that the cloud servers use the data on the correct positions. Otherwise, the cloud servers' cheating behaves are detected.

Secondly, **VA** checks the correctness of the result $y^*_{c_i}$ using algorithm

$$\texttt{VerifyCom}\left(y^*_{c_i}, f_{c_i}, m_{c_i}\right),$$

Algorithm 2. Sampling based cloud computation auditing protocol.



```
Data: Input: sampling index, data, signature, computation results;
Result: Output: invalid or valid.
Let retValue=invalid;
for sampling index $\tau = 1$ to $t$ do
    Fetch the sampling data and its signature;
    if $VerifySigByVA(\Sigma'_\tau, m_\tau, c_\tau, sk_{VA})$ is true then
        if $VerifyCom(y^*_{c_\tau}, f_\tau, x_\tau)$ is true then
            Fetch the sibling node set;
            Reconstruct the root value $R(\tau)$;
            if $R^* = R(\tau)$ then
                if $VerifySigByCS(R(\tau), CS)$ is true then
                    retValue=valid;
                end
            end
        end
    end
end
return $retValue$;
```

which returns true if the following equation is satisfying:

$$y^*_{c_i} = f_{c_i}(m_{c_i}).$$

The servers' cheatings are caught once the algorithm $\texttt{VerifyCom}\left(y^*_{c_i}, f_{c_i}, m_{c_i}\right)$ returns false.

Finally, **VA** uses the commit information to ensure that each y_{c_i} is used at the beginning of building the Merkle hash tree: **VA** uses the correct $y^*_{c_i} \| p_{c_i}$ as one leaf and sibling node set to reconstruct the root R^* of the tree by Eq. (6). Only if $R^* = R$, **VA** can trust that all of the computation $\{f_{c_i}(m_{c_i})\}$ had been completed before the Merkle hash tree was built.

5.4.4. Auditing return step

VA returns true to **CU** if the cloud server's cheating and attacker's behaves are not caught in the all challenges. **CU** is convinced that the cloud servers do not cheated and behave normally with a high probability. Otherwise, **VA** returns false and **CU** may drop the computing results and report to **CSP** a cheating alarm.

6. The advanced protocol with batch verification

Considering the fact that the major communication and computation overhead comes from verification of the signatures, we introduce an advanced protocol to further reduce the computational and communication overhead in this section.

The idea comes from aggregate signatures with batch verification. According to identity based aggregate signatures, our scheme can achieve almost constant computation overhead for **VA** in the data verification as well as **CSP**. **VA** can concurrently handle the multiple verification requests not only from one cloud user but also from different cloud users. Assume that a set of users $\{ID_i | 1 \leqslant i \leqslant k\}$, each of which generates signature $\{\sigma_{ij} | 1 \leqslant j \leqslant n_i\}$ on message $\{m_{ij} | 1 \leqslant j \leqslant n_i\}$. **VA** does as follows: $\Sigma'_A = \prod_{i=1}^{k}\prod_{j=1}^{n_i}\Sigma'_{ij}$, $U_A = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(U_{ij} + H_2(U_{ij}\|m_{ij})Q_{ID_i})$. Then **VA** uses its secret key to verify

$$\hat{e}(U_A, sk_{VA}) \overset{?}{=} \Sigma'_A \tag{8}$$

The correctness is as follows:

$$\begin{aligned}\Sigma'_A &= \prod_{i=1}^{k}\prod_{j=1}^{n_i}\hat{e}(V_{ij}, Q_{VA}) = \hat{e}\left(\sum_{i=1}^{k}\sum_{j=1}^{n_i}V_{ij}, Q_{VA}\right) = \hat{e}\left(\sum_{i=1}^{k}\sum_{j=1}^{n_i}(r_{ij} + H_2(U_{ij}\|m_{ij}))sk_{ID_i}, Q_{VA}\right)\\ &= \hat{e}\left(\sum_{i=1}^{k}\sum_{j=1}^{n_i}(U_{ij} + H_2(U_{ij}\|m_{ij})Q_{ID_i}), sk_{VA}\right) = \hat{e}(U_A, sk_{VA}).\end{aligned} \tag{9}$$

The computational complexity of our scheme is analyzed as follows. Note that the signature combination Σ'_A and U_A can be performed incrementally and the computational cost are almost measured by the expensive pairing operations $\hat{e}$. It is obvious that given ξ unauthenticated signatures, in such batch verification, the major computational cost is bounded by 2 pairings while it costs 2ξ pairings by individual verification, which is a significant improvement on computational efficiency. If the verification succeeds for all the sample set C, the verifier is convinced that the cloud has not cheated as a high probability.

7. Security analysis

7.1. Uncheatability analysis

To analyze uncheatability of our computation, we evaluate the sampling performance of **SecCloud** in terms of the number of sampling blocks to be retrieved.

Theorem 1. *According to Definition 1, our protocol is uncheatable.*

Proof. We first define **FCS** as the event that the adversary could successfully cheat by guessing the results of f or behave normally. For simplicity, let $|f|$ be the result range of f. Thus, the probability that an adversary could randomly guess the correct result of $f(x)$ is $\frac{1}{|f|}$ without any auxiliary information if f is uniform distribution. Besides, we adapt the concept of **SCC** in our definition to qualify the probability of the adversary's attacks. Therefore, the probability that the adversary could successfully cheat a t-time sampling scheme without being detected is as follows:

$$\Pr[\texttt{FCS}] = \left(SCC + (1 - SCC)\frac{1}{|f|}\right)^t. \tag{10}$$

Since in the most cases, $|f|$ would be quite large, that is, $|f| > 1$, $\Pr[\texttt{FCS}]$ approaches to 0 when the sampling size t is large. Thus, it must exist some t, where $t \geqslant \left\lceil \lg\frac{\epsilon}{2} / \lg\left(SCC + (1 - SCC)\frac{1}{|f|}\right)\right\rceil$, such that

$$\Pr[\texttt{FCS}] < \frac{\epsilon}{2}. \tag{11}$$

We also define **PCS** is the event that the adversary could successfully cheat by using the data on the incorrect positions. In this case, the adversary needs to forge the signature to pass the verification. We take the concept of **SSC** to qualify adversary's attacks. Therefore, in a t-time sampling scheme, the probability the adversary can successfully using the data on the incorrect positions is:

$$\Pr[\texttt{PCS}] = (SSC + (1 - SSC)\Pr[\texttt{SigForge}])^t. \tag{12}$$

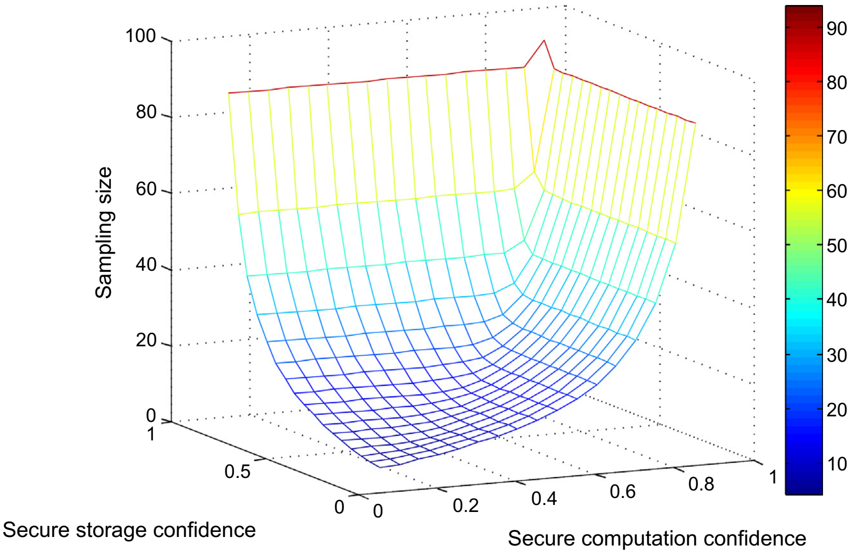


Fig. 4. Required sample size to achieve uncheatable computation given $\epsilon = 0.0001$.

where Pr[$\mathtt{SigForge}$] is the probability that the cloud could forge such a digital signatures. Since the signature scheme is proved secure based on the assumption of BDH, that is, Pr[$\mathtt{SigForge}$] $\leqslant$ Pr[$\mathtt{BDH}$], Pr[$\mathtt{SigForge}$] is a small value in our model. Pr[$\mathtt{PCS}$] approaches to 0 when the sampling size is large. Thus, it must exist some t, where $t \geqslant \lceil \lg\frac{\epsilon}{2} / \lg(SCC + (1 - SCC)\text{Pr}[\mathtt{SigForge}]) \rceil$, such that

$$\Pr[\mathtt{PCS}] < \frac{\epsilon}{2}. \quad (13)$$

From our Definition 1, we can define the adversary cheats successfully if the event **FCS** or event **PCS** takes place. For simplicity, we assume that the event **FCS** and **PCS** are independent. Thus, the probability that the adversary can cheat successfully is evaluated as follows:

$$\Pr[\mathtt{Cheating\ Successfully}] = \Pr[\mathtt{FCS} \cup \mathtt{PCS}] \leqslant \Pr[\mathtt{FCS}] + \Pr[\mathtt{PCS}] < \epsilon. \quad (14)$$

To get more accurate results, we evaluate the minimal sampling size by numerical method in MATLAB. Given $\epsilon = 0.0001$, Fig. 4 shows the various minimal sampling size$t$ for different **SCC** and **SSC**.

When we consider an extreme situation that the cloud server has computing with half of **SCC** and half of **SSC** in the task, the range is $|f| = 2$ since the results are $\{0, 1\}$ for simplicity, we need at least 33 samples to ensure the probability of successful cheating to be below $\epsilon = 0.0001$. When $|f|$ is large enough, (i.e., $|f| \to \infty$, it is almost impossible to make a correct guess on $f(x)$ without computing it), we only need 15 samples.

Therefore, the adversary could successfully cheat with a neglect probability. That means that our protocol is uncheatable. □

7.2. Privacy discouragement analysis

To analyze our privacy level, we compare the probability with cryptographic assumptions in Section 4.

Theorem 2. *According to Definition 2, our protocol could achieve privacy cheating discouragement.*

Proof. In our protocol, the designated verifier signature scheme is proved secure based on the assumption of BDH. That means no adversary could forge such a signature in the polynomial time unless it could solve BDH problem, that is,

$$\Pr[\mathtt{SigForge}] \leqslant \Pr[\mathtt{BDH}].$$

On one hand, the cloud servers can confirm the users' identity by verifying the signature since it is hard for any other users to forge these signature. On the other hand, even the cloud servers are compromised by attackers, the data and the signatures the adversary obtains cannot convince others since it needs the secret keys to verify the signature and the cloud server can also generate the signature of themselves.

Thus, the only way for successful cheating is that the adversary forges the cloud users' signatures, which leads to conflict with the BDH assumption in the polynomial time. We assume that the probability of solving BDH problem is ϵ for any polynomial time algorithms. From the discussion above, in the polynomial time, we have:

Table 1
Cryptographic operation's execution time.

	Descriptions	Execution time (ms)
T_{pmul}	Time for one point multiplication	20.63
T_{pair}	Time for one pairing operation	26.75

Table 2
Comparison of related work.

Scheme	Verification cost
ESORICS09 [34]	$2K \cdot T_{Pair} + (S+1)K \cdot T_{Mul}$
INFOCOM10 [33]	$(K+1) \cdot T_{Pair} + (S+1)K \cdot T_{Mul}$
Our scheme	$2T_{Pair} + K \cdot T_{Mul}$

$$\Pr[\mathtt{InfoLeak}] \leqslant \Pr[\mathtt{SigForge}] \leqslant \Pr[\mathtt{BDH}] \leqslant \epsilon. \tag{15}$$

Therefore, from Definition 2, our protocol is privacy cheating discouragement with a neglect probability. □

7.3. Optimal sample size to minimize total cost

To generally estimate the total cost for our sampling algorithm in the protocol, we assume that it can be divided into three kinds of cost. Let C_{trans} and C_{comp} be the transmission cost and the computational cost for each sampling message-signature pairs, respectively. We also define the cost C_{cheat} that is caused by the undetected cheating attacks. The total cost C_{total} for a sample set of t is as follows:

$$C_{total} = a_1 \cdot t \cdot C_{trans} + a_2 \cdot C_{comp} + a_3 \cdot C_{cheat} \cdot q^t \tag{16}$$

where q is the probability of cheating successful and a_1, a_2, and a_3 are coefficients for these costs, respectively. The following theorem gives the optimal sample size t to achieve a minimal total cost.

Theorem 3. *Given the transmission cost C_{trans}, computational cost C_{comp} and successfully cheating cost C_{cheat}, and the probability of cheating successful q, the optimal sample set t for achieving the minimal cost is*

$$t = \left\lceil \ln\left(-\frac{a_1 \cdot C_{trans}}{a_3 \cdot C_{cheat} \cdot \ln q}\right) \Big/ \ln q \right\rceil. \tag{17}$$

Proof. Since $C_{total} = a_1 \cdot t \cdot C_{trans} + a_2 \cdot C_{comp} + a_3 \cdot C_{cheat} \cdot q^t$, to minimize the total cost C_{total}, we have

$$\frac{dC}{dt} = a_1 \cdot C_{trans} + a_3 \cdot C_{cheat} q^t \ln q. \tag{18}$$

It is easy to check that the derivative is 0 when $t = \ln\left(-\frac{a_1 \cdot C_{trans}}{a_3 \cdot C_{cheat} \cdot \ln q}\right) \Big/ \ln q$. Note that, t must be an integer in practice.[5] □

8. Performance analysis

We evaluate the performance of **SecCloud** in two aspects: the cryptographic operation costs and the effectiveness and efficiency of **SecCloud** in our developed test bed.

8.1. Cryptographic overhead evaluation

We use the JPBC library [10] for doing pairing and large number operations to implement some of our cryptographic protocols. We experiment to record the delay of cryptographic operations according to typical security parameters (see Table 1).

Compared with related work as in [34,33], we consider the verification cost in the protocol which is most expensive cost operation at both the cloud server side and the verifier side. To a large extent, the computation costs are measured by pairing time (T_{Pair}) and point multiplication time (T_{Mul}). Table 2 shows the comparison of various data auditing scheme where K represents the number of cloud users and S represents the concurrent signatures. It is obviously that our protocol is much more

[5] In order to optimize the cost in practice, we need the detail cost information such as C_{trans}, C_{comp}, C_{cheat}, a_1, a_2, and a_3. We evaluate them through a history learning process.

Table 3
Comparison of various signature scheme.

Scheme	Individual verify	Batch verify
RSA	$\eta \cdot T_{RSA}$	*NA*
ECDSA	$\eta \cdot T_{ECDSA}$	*NA*
BGLS [7]	$2\eta \cdot T_{pair}$	$(\eta + 1) \cdot T_{pair}$
Our scheme	$2\eta \cdot T_{pair}$	$2 \cdot T_{pair}$

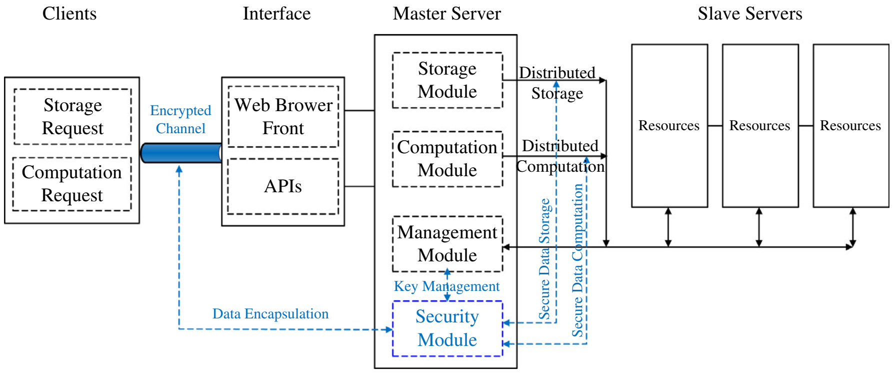


Fig. 5. A cloud computing experiment environment: SecHDFS.

efficient than the previous ones since pairing times are constant in **SecCloud** while in their schemes pairing times are linear with the number of concurrent signatures.

To further demonstrate the suitability of the proposed scheme, we compare the computational cost for different signature schemes in Table 3, which need to handle a batch signatures with a size of η.

8.2. Experiment environment

We first give a brief description on the traditional distributed file systems (DFS) in cloud and Google's Hadoop Distributed File System (HDFS) [5]. Then we introduce our developed test bed: **SecHDFS** which integrates security module to HDFS. Finally, we implement **SecCloud** into this test bed.

8.2.1. A brief introduction to traditional DFS

Traditional DFS systems such as Google HDFS are wildly accepted and used in Google, IBM, Yahoo and other 30+ enterprisers to store and manage their large number of data in cloud. As a typical DFS in cloud, the cloud master contains three modules, which are interface module, storage module, and management module. For adapting to different client platforms including personal computer operating systems, cell phone operating systems, and PDA operating systems, interface module provides web browser front and APIs for both general clients and specific programmers. The storage module is in charge of maintaining virtual file system organization and splitting files into blocks which will be uploaded to selected slave servers. The management module is in charge of monitoring the storage resources as well as the slave servers information such as heart detecting, sparse space on disks and transmission speed. In addition, HDFS improves DFS by adding its computation module Hadoop in the file system. The computation module is in charge of adopting computation requests provided from users and distributing the computation and related data to slave servers.[6]

8.2.2. Our developed SecHDFS

To add security feature into the current HDFS, we develop a secure Hadoop distributed file system **SecHDFS**, as a practical test bed shown in Fig. 5. Furthermore, as illustrated in blue color in Fig. 5, we adds security module to HDFS including confidentiality by encrypted channels for data uploading, batch verification mechanism, and privacy cheating discouragement by designated verifier digital signature.

Our test bed consists of four computers with Intel Core processor i5-760 running at 2.8 GHz with 4 GB RAM memory. One of the four computers plays the role of the master server in the cloud computing, which allocates storage space and storage data index for the rest three slave servers. Cloud users upload their storage requests and computation requests through a wired or wireless communication. To be compatible with HDFS, we define the block size 64 MB by splitting each big file into several blocks, which have a length of 60 MB file contents plus 4 MB security head. The security head includes designated

[6] For the more detail of HDFS, we can refer [5].

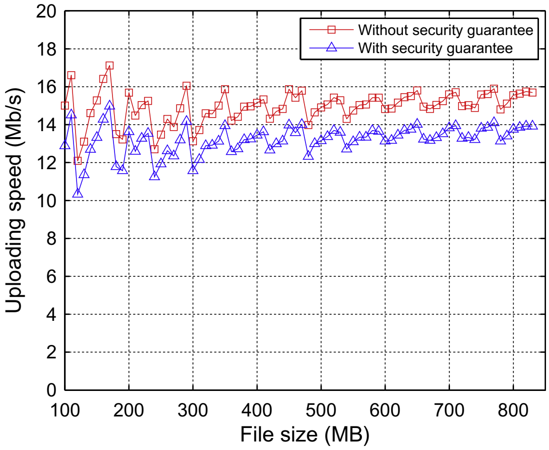
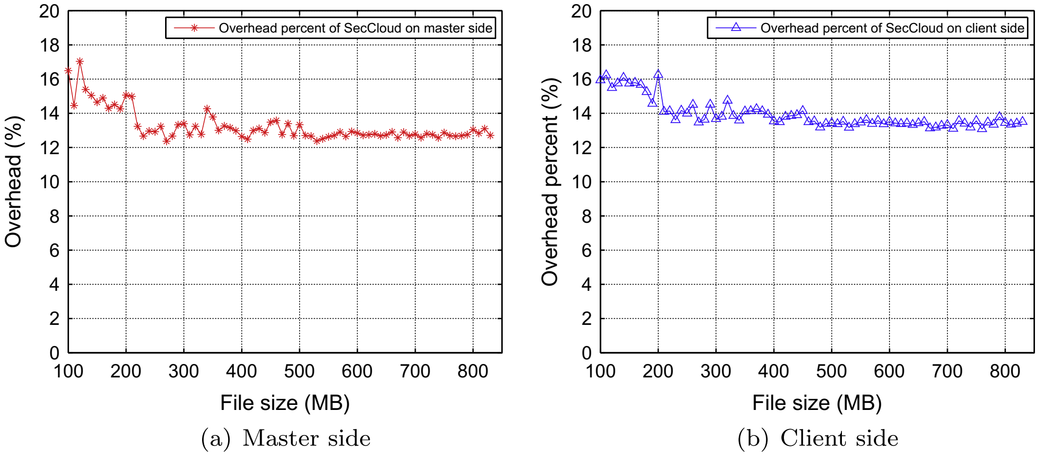


Fig. 6. The impact traffic load on system performance.



(a) Master side

(b) Client side

Fig. 7. The impact traffic load on the master side and client side.

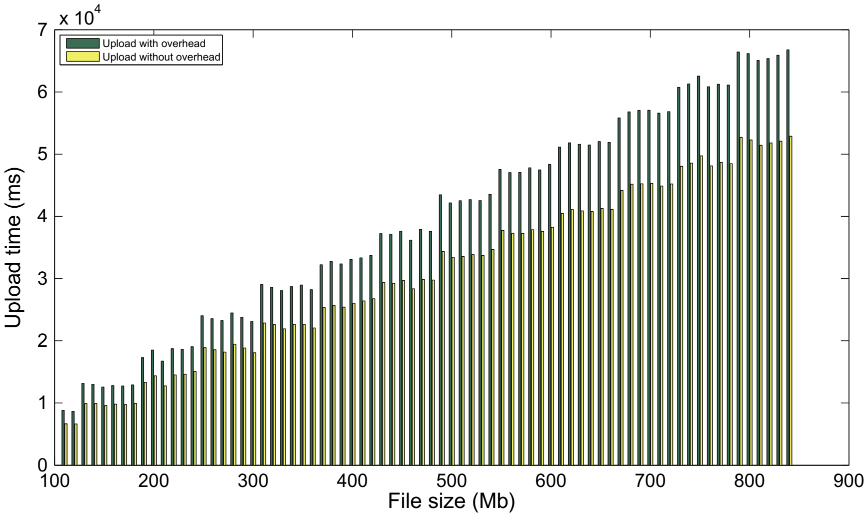


Fig. 8. Total overhead of protocols with/without security guarantee.

verifier signatures, symmetric key parameters, hash values and other security parameters, which are defined in the **SecCloud**. After that, we start our experiments by uploading data to cloud servers from cloud users with cryptographic techniques. After the master server verifies the signature of each block, the storage module maintains a virtual file system organization by Extensible Markup Language (XML) and distributes each block to slave servers at random.

8.3. Experiment results

We begin our experiments by observing the system performance under various traffic load initializing at 100 MB and gradually increasing to 850 MB. We also consider the impact of traffic load brought by security overhead.

8.3.1. Impact of traffic load I

The impact of traffic load to system performance is measured by the uploading speed shown in Fig. 6. We can find that the uploading speeds with security guarantee in **SecCloud** are very close to that of the original protocol without any security addons. For example, for the large files (over 500 MB), the uploading speeds are in the range of 13–15 Mb/s, which are almost 2 Mb/s lower than that of the original protocols; for the small files, the speeds become a little unstable at initial step due to the session establishment delay.

8.3.2. Impact of traffic load II

Security overhead leads to the system delay both in the cloud user's side (signature generation and symmetric encryption) and in the cloud server's side (signature verification and symmetric decryption) in our protocol. Thus, we define the percent of security overhead by the time of security operations dividing the time of file transmissions. Fig. 7 shows the relationship in **SecCloud** between the uploading file size and the percents of the security overhead in the master server's side and in the client's side, respectively. It is observed that after a period of dithering, the curves become stable, where the percents of security overhead are no more than 14%. We also find that for large files (over 500 MB) security overhead would become to more stable since the transmission time is major and session establishing time is minor.

8.3.3. Overall overhead comparison

To consider the overall overhead, we record the total time of uploading files in two cases for comparison. Fig. 8 shows the total uploading time comparison between the original protocol and **SecCloud**. It is observed that **SecCloud** only has a slightly more time than the original protocol (between 18% in the best case and 32% in the worst case). Thus, the increased time is not significantly reduce the system performance.

In summary, the experiment results demonstrate that **SecCloud** is indeed a viable, lightweight solution for secure data storage and computation in the cloud computing.

9. Conclusions

In this paper, we have proposed, **SecCloud**, a privacy-cheating discouragement and secure-computation auditing protocol for data security in the cloud. To the best of our knowledge, it is the first work that jointly considers both of data storage security and computation auditing security in the cloud. We have defined the concepts of uncheatable cloud computation and privacy-cheating discouragement and proposed **SecCloud** to achieve the security goals. To improve the efficiency, different users' requests can be concurrently handled through the batch verification. By the extensive security analysis and performance simulation in our developed **SecHDFS**, it is showed that our protocol is effective and efficient for achieving a secure cloud computing. In our future work, we continue to consider some detailed computation such as linear program computation and data mining and formalize these security models in the cloud computing. In addition, we also focus on the privacy preserving issues in the above computation. Furthermore, we plan to implement them in the real cloud platform such as EC2 and OpenStack.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61033014 and 61161140320) and the National 973 Program (Grant No. 2012CB723401). We would like to thank anonymous reviewers who helped us in giving comments to this paper.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al, A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, Provable data possession at untrusted stores, in: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07), Alexandria, Virginia, USA, October 28–31, 2007.

[3] G. Ateniese, R. Di Pietro, L. Mancini, G. Tsudik, Scalable and efficient provable data possession, in: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, Istanbul, Turkey, September 22–26, 2008.

[4] M. Belenkiy, M. Chase, C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, Incentivizing outsourced computation, in: Proceedings of the 3rd International Workshop on Economics of Networked Systems, Seattle, WA, USA, August 17–22, 2008.

[5] A. Bialecki, M. Cafarella, D. Cutting, O. OMalley, Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware. <http://lucene.apache.org/hadoop>.

[6] D. Boneh, M. Franklin, Identity-based encryption from the Weil pairing, SIAM Journal on Computing 32 (3) (2003) 586–615.

[7] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in: International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2003), Warsaw, Poland, May 4–8, 2003.

[8] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing, Journal of Cryptology 17 (4) (2004) 297–319.

[9] R. Canetti, B. Riva, G. Rothblum, Verifiable computation with two or more clouds, in: Workshop on Cryptography and Security in Clouds, Zurich, Switzerland, March 15–16, 2011.

[10] A.D. Caro, jPBC-Java Pairing-based Cryptography Library (Technique Report). <http://gas.dia.unisa.it/projects/jpbc/> (28.12.10).

[11] M. Castro, B. Liskov, Practical byzantine fault tolerance and proactive recovery, ACM Transaction on Computer Systems 20 (4) (2002) 398–461.

[12] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of The ACM 51 (1) (2008) 107–113.

[13] W. Du, J. Jia, M. Mangal, M. Murugesan, Uncheatable grid computing, in: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), Hachioji, Tokyo, Japan, March 24–26, 2004.

[14] Q. Duan, Y. Yan, A.V. Vasilakos, A survey on service-oriented network virtualization toward convergence of networking and cloud computing, IEEE Transactions on Network and Service Management 9 (4) (2012) 373–392.

[15] C. Erway, A. Kupcu, C. Papamanthou, R. Tamassia, Dynamic provable data possession, in: Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09), Chicago, Illinois, USA, November 9–13, 2009.

[16] R. Gennaro, C. Gentry, B. Parno, Non-interactive verifiable computing: Outsourcing computation to untrusted workers, in: 30th International Cryptology Conference (CYPTO 2010), Santa Barbara, California, USA, August 15–19, 2010.

[17] P. Golle, I. Mironov, Uncheatable distributed computations, in: The Cryptographers' Track at RSA Conference 2001, San Francisco, CA, USA, April 8–12, 2001.

[18] A. Haeberlen, A case for the accountable cloud, in: 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, Big Sky Resort, Big Sky, MT, October 10–11, 2009.

[19] Q. Huang, G. Yang, D.S. Wong, W. Susilo, Efficient strong designated verifier signature schemes without random oracle or with non-delegatability, International Journal of Information Security 10 (6) (2011) 373–385.

[20] W. Itani, A.I. Kayssi, A. Chehab, Privacy as a service: privacy-aware data storage and processing in cloud computing architectures, in: Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2009), Chengdu, China, December 12–14, 2009.

[21] M. Jakobsson, K. Sako, R. Impagliazzo, Designated verifier proofs and their applications, in: International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 1996), Zaragoza, Spain, May 12–16, 1996.

[22] A. Juels, B. Kaliski Jr., PORs: proofs of retrievability for large files, in: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07), Alexandria, Virginia, USA, October 28–31, 2007.

[23] B. Kang, C. Boyd, E. Dawson, A novel identity-based strong designated verifier signature scheme, Journal of Systems and Software 82 (2) (2009) 270–273.

[24] G. Karame, M. Strasser, S. Capkun, Secure remote execution of sequential computations, in: 11th International Conference on Information and Communications Security (ICICS'09), Beijing, China, December 14–17, 2009.

[25] A. Marinos, G. Briscoe, Community cloud computing, in: Proceedings of Cloud Computing: First International Conference (CloudCom 2009), Beijing, China, December 1–4, 2009.

[26] R. Merkle, Protocols for public key cryptosystems, in: IEEE Symposium on Security and Privacy, Oakland, California, USA, April, 1980.

[27] F. Monrose, P. Wyckoff, A. Rubin, Distributed execution with remote audit, in: Proceedings of the Network and Distributed Systems Security Symposium (NDSS), San Diego, California, USA, 1999.

[28] P. Patel, A. Ranabahu, A. Sheth, Service level agreement in cloud computing, in: Cloud Workshops at OOPSLA09, Orlando, Florida, USA, October 25–29, 2009.

[29] S. Pearson, Y. Shen, M. Mowbray, A privacy manager for cloud computing, in: First International Conference (CloudCom 2009), Beijing, China, December 1–4, 2009.

[30] A. Sadeghi, T. Schneider, M. Winandy, Token-based cloud computing: Secure outsourcing of data and arbitrary computations with lower latency, in: Trust and Trustworthy Computing, Berlin, Germany, June 21–23, 2010.

[31] H. Takabi, J. Joshi, G. Ahn, Security and privacy challenges in cloud computing environments, IEEE Security & Privacy 8 (6) (2010) 24–31.

[32] C. Wang, K. Ren, J. Wang, Secure and practical outsourcing of linear programming in cloud computing, in: 30th IEEE Conference on Computer Communications (INFOCOM 2011), Shanghai, China, April 11–15, 2011.

[33] C. Wang, Q. Wang, K. Ren, W. Lou, Privacy-preserving public auditing for data storage security in cloud computing, in: 29th IEEE Conference on Computer Communications (INFOCOM'10), San Diego, California, USA, March 14–19, 2010.

[34] Q. Wang, C. Wang, J. Li, K. Ren, W. Lou, Enabling public verifiability and data dynamics for storage security in cloud computing, in: 14th European Symposium on Research in Computer Security (ESORICS'09), Saint Malo, France, September 21–23, 2009.

[35] L. Wei, H. Zhu, Z. Cao, W. Jia, A. Vasilakos, Seccloud: bridging secure storage and computation in cloud, in: 30th International Conference on Distributed Computing Systems Workshops (IEEE ICDCSW 2010), Genova, Italy, June 21–25, 2010.

[36] T. Yuen, W. Susilo, Y. Mu, How to construct identity-based signatures without the key escrow problem, International Journal of Information Security 9 (4) (2010) 297–311.

[37] J. Zhang, J. Mao, A novel ID-based designated verifier signature scheme, Information Sciences 178 (3) (2008) 766–773.