Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

NaEPASC: a novel and efficient public auditing scheme for cloud data*

Shuang TAN, Yan JIA

(*School of Computer, National University of Defense Technology, Changsha 410073, China*)

E-mail: tanshuang@nudt.edu.cn; jiayanjy@vip.sina.com

Received Feb. 9, 2014; Revision accepted June 20, 2014; Crosschecked Aug. 11, 2014

Abstract: Cloud computing is deemed the next-generation information technology (IT) platform, in which a data center is crucial for providing a large amount of computing and storage resources for various service applications with high quality guaranteed. However, cloud users no longer possess their data in a local data storage infrastructure, which would result in auditing for the integrity of outsourced data being a challenging problem, especially for users with constrained computing resources. Therefore, how to help the users complete the verification of the integrity of the outsourced data has become a key issue. Public verification is a critical technique to solve this problem, from which the users can resort to a third-party auditor (TPA) to check the integrity of outsourced data. Moreover, an identity-based (ID-based) public key cryptosystem would be an efficient key management scheme for certificate-based public key setting. In this paper, we combine ID-based aggregate signature and public verification to construct the protocol of provable data integrity. With the proposed mechanism, the TPA not only verifies the integrity of outsourced data on behalf of cloud users, but also alleviates the burden of checking tasks with the help of users' identity. Compared to previous research, the proposed scheme greatly reduces the time of auditing a single task on the TPA side. Security analysis and performance evaluation results show the high efficiency and security of the proposed scheme.

Key words: Cloud storage, Public verification, Identity-based aggregate signature
doi:10.1631/jzus.C1400045 **Document code:** A **CLC number:** TP309.2

1 Introduction

Cloud computing has been envisioned as the next-generation information technology (IT) architecture for enterprisers, and quickly received growing attention in the scientific and industrial communities (Mell and Grance, 2009; Hashizume *et al.*, 2013). Also, cloud computing is considered as first among the top 10 important technologies in Gartner's study, and will receive better prospects in successive years (Gartner, 2010). According to a new forecast from International Data Corporation (IDC) (Lokantas and Salonu, 2013), the spending on public IT cloud services in the world will reach 100 billion USD in 2016. Despite the fast development of cloud computing, the growth of cloud service subscribers has still not met expectations (Khan *et al.*, 2013a; 2013b). According to the survey conducted by IDC, most IT executives and CEOs do not prefer to adopting such services due to security and privacy risks (Hochmuth *et al.*, 2013).

New opportunities and challenges are being presented for the deployment of new applications in cloud computing. From the perspective of the users, storing their data to the remote cloud data center in a flexible on-demand manner brings some benefits,

* Project supported by the National Natural Science Foundation of China (Nos. 60933005 and 91124002), the National High-Tech R&D Program (863) of China (Nos. 2010AA012505, 2011AA010702, 2012AA01A401, and 2012AA01A402), the National 242 Foundation (No. 2011A010), and the National Technology Support Foundation (Nos. 2012BAH38B04 and 2012BAH38B06)

such as alleviating the burden of the management of the storage data, providing universal data access with independent geographical locations, and avoiding capital expenditure on hardware, software, and personnel maintenance (Mell and Grance, 2009; Wang *et al.*, 2013). Obviously, it also brings new security threats to the users' outsourced data. If the users want to rent the resources through the cloud platform, their data is being centralized or outsourced to the cloud providers, which will make them lose eventual control over their data. As a result, the correctness of the data stored in the cloud may suffer from some security threats due to the following reasons: (1) Although the cloud infrastructures are much more powerful and reliable than personal computing devices, the data integrity is still facing the broad range of both internal and external threats (Wang *et al.*, 2010). For example, the malicious users may infringe other users' data through mounting cross-VM side-channel attacks (Ristenpart *et al.*, 2009). (2) There are various motivations for the cloud service provider (CSP) to behave unfaithfully towards the cloud users; e.g., the CSP might reclaim the storage space to maximize their profit in the following ways, discarding some data that has been rarely accessed, or hiding the data loss incidents to maintain a reputation (Wang Q *et al.*, 2009; Wang *et al.*, 2010). Consequently, although adopting the cloud platform to deploy the applications is economically attractive for long-term data services, there do not exist any effective mechanisms to protect the integrity and availability of the data (Wang *et al.*, 2013). Thus, it is normal for the users to wonder whether the integrity of their data is protected by an appropriate method in the cloud platform.

Due to users no longer storing their data in a local infrastructure, traditional cryptographic checking methods cannot be directly adopted to the cloud platform (Wang *et al.*, 2010). Considering the high I/O overheads and transmission costs, the cloud user could not download the whole outsourced data from the cloud infrastructures for its integrity verification. Besides, as there are limited computing resources, it is almost impossible for the cloud user to take extra time and effort to verify the correctness of the data in a cloud environment. Thus, if users choose the cloud platform to execute their tasks, they should not need to worry about how to verify the integrity of the data after data retrieval. Making public verification is a good method to evaluate the risk of the users' subscribed cloud services. Users can resort to an independent third-party auditor (TPA) on behalf of themselves to audit the integrity of the outsourced data. The concept of 'public verification' was first proposed by Shacham and Waters (2008), in their system and security model. The mechanism can effectively ensure the integrity of the data stored in remote nodes (Shacham and Waters, 2008). Moreover, some similar work can be found in Wang Q *et al.* (2009), Wang *et al.* (2010), Hao *et al.* (2011), and Zhu *et al.* (2011b). However, most of these schemes (Wang Q *et al.*, 2009; Chen and Curtmola, 2012; Wang *et al.*, 2012; 2013) do not consider the key management when the cloud users start to use this data. Indeed, their mechanism can be used merely in such a situation: one key and one file. If they reuse their public/private key for the different files, the cloud server can deceive them by forging the tag of the data block (Zhu *et al.*, 2011a). Consequently, when the user wants to store multiple data files in the cloud, he/she needs to remember multiple keys for different files. Additionally, if the user loses the key, he/she can no longer perform any integrity testing, except for retrieving the data files from the cloud to regenerate the verification metadata. Thus, the key management of the integrity checking scheme becomes a difficult problem as multiple keys need to be stored at the user side.

In this paper an identity-based public verification protocol (NaEPASC) has been proposed to effectively simplify the key management and alleviate the users' burden. Our work is one of the first few research works to support identity-based public auditing in cloud computing. With the motivation of simplifying certificate management in e-mail systems, identity-based encryption was first proposed by Shamir (1985). A fully functional and effective identity-based encryption scheme through any bilinear map was first proposed by Boneh and Franklin (2001). Since then, much research has been carried out about identity-based encryption/signature schemes (Gentry and Silverberg, 2002; Boneh and Boyen, 2004; Boneh *et al.*, 2005; Waters, 2005; Gentry and Ramzan, 2006). In view of the key role of identity-based encryption/signature and public auditability, we propose an efficient protocol which seamlessly integrates these two components in the design of the protocol. The contributions can be

summarized as follows:

1. We analyze the data storage model in cloud computing and provide an identity-based auditing protocol. The proposed scheme enables an external party to verify the users' outsourced data in the cloud without retrieving the original data files.

2. The proposed scheme has for the first time realized identity-based public verification in cloud computing.

3. We prove the security and evaluate the performance of the proposed scheme through experiments, and make comparison with two state-of-the-art schemes.

2 Related work

Some previous research focused on checking the availability and integrity of the outsourced data. Juels and Kaliski (2007) first presented a proof-of-retrievability (POR) scheme where spot-checking and error-correcting codes are used to ensure the possession and retrievability of data files on remote nodes; however, it can be used only to handle a limited number of queries, and does not support public auditability. Based on Juels and Kaliski (2007)'s model, Shacham and Waters (2008) presented an improved POR scheme which combines bilinear signature (BLS) based homomorphic linear authenticator (HLA) with error-correcting codes to support both public verification and fault-tolerance. Through BLS-based HLA, the proofs can be aggregated into a small authentication value; meanwhile, the public retrievability can be achieved. However, the key management is ignored in this scheme. Chen and Curtmola (2012) first presented the constructions for dynamic POR. They extended the POR model in Shacham and Waters (2008) to support provable updates to stored data files by encoding with error-correcting codes. Similar to Shacham and Waters (2008), their scheme focuses merely on dynamic operations and error-correcting, and the characteristic of user identity is ignored.

Similar to POR, Ateniese *et al.* (2007) considered public auditability in their defined provable data possession (PDP) model to ensure possession of data files in untrusted storage. The scheme uses the RSA-based homomorphic linear authenticator to audit the outsourced data. However, this scheme brings a large amount of computational overhead which would be expensive for the entire file (Wang Q *et al.*, 2009). In their subsequent research, Ateniese *et al.* (2008; 2011) proposed a dynamic version of the previous PDP scheme. However, this protocol does not support fully dynamic data operations; i.e., some basic block operations with limited functionality are allowed, while block insertions are not supported. To solve the above problem, Erway *et al.* (2009) introduced a dynamic PDP scheme with a skip list to enable provable data possession to support fully dynamic data operations. However, they did not provide an efficient method to manage the user's key as in Chen and Curtmola (2012).

In some other related research, Wang C *et al.* (2009) proposed a partially dynamically supported scheme in a distributed scenario with the additional feature of data error localization. Wang Q *et al.* (2009) considered combining BLS-based HLA with Merkle hash tree (MHT) to support both public auditability and full data dynamics. Furthermore, Wang *et al.* (2013) proposed privacy-preserving public auditing for cloud storage, using a random masking technique to prevent the TPA from learning data information and keep the user's data privacy.

3 Preliminaries

In this section we introduce the concept of bilinear maps (Boneh and Franklin, 2001) and some assumptions. Let λ denote the security parameter, which is an implicit input of the algorithm in our scheme.

3.1 Bilinear maps

The proposed protocol is based on a bilinear map, which is often called 'pairing'. Typically, the pairing is built on a supersingular elliptic curve or Abelian variety, such as Weil or Tate pairing. In this subsection, we simply describe some properties of bilinear maps.

Let G_1 and G_2 be two multiplicative cyclic groups of prime order p. A bilinear map e is defined as $e\colon G_1 \times G_1 \to G_2$ with the following properties:

1. Bilinear: $e(aQ, bR) = e(Q, R)^{ab}$ for all $Q, R \in G_1$ and $a, b \in Z_p$.

2. Non-degenerate: $e(Q, R) \neq 1$ for some $Q, R \in G_1$.

3. Computable: we can always find an efficiently computable algorithm to compute map e.

Other useful conclusions include:

1. For some $Q \in G_1, R \in G_2$, if $G_1 = G_2$, $e(Q,R) = e(R,Q)$.

2. For $a \in Z_p$ and $Q \in G_1, R \in G_2$, $e(aQ,R) = e(Q,aR)$.

3.2 Computational assumptions

The security of the proposed scheme is based on the assumed hardness of the computational Diffie-Hellman (CDH) problem in G_1.

Definition 1 (Computational Diffie-Hellman problem, or CDH problem) Given $P, aP, bP \in G_1$ and a bilinear map $e: G_1 \times G_1 \to G_2$, compute abP.

The CDH assumption holds in G_1, if no polynomial time algorithm has an advantage of at least ϵ in solving the CDH problem in G_1, which means it is computationally infeasible to solve the CDH problem in G_1.

Definition 2 (Computational Diffie-Hellman assumption in G_1, or CDH_{G_1} assumption) We say that the (t,ϵ)-CDH_{G_1} assumption holds if no t-time algorithm $\mathcal{A}$ has advantage ϵ in solving the CDH_{G_1} problem.

3.3 Blockless verification

For public verification in cloud storage, the TPA can verify the integrity of the outsourced data on behalf of the users. When taking a third-party authentication in cloud storage, the cloud user cannot reveal his/her data content to the third partner. Thus, the checking scheme should have the following properties:

Let (sk,pk) be the user's private/public key pair. For two blocks $m_1, m_2 \in Z_p$, let σ_1 and σ_2 be the signatures of blocks m_1 and m_2, respectively.

1. Blockless verification: Given σ_1 and σ_2, two random numbers α_1, α_2, and a linear combination $m' = \alpha_1 m_1 + \alpha_2 m_2 \in Z_p$, a verifier can verify the correctness of the linear value m' without knowing block m_1 or m_2.

2. Non-malleability: Given σ_1 and σ_2, two random numbers α_1, α_2, and a linear combination $m' = \alpha_1 m_1 + \alpha_2 m_2 \in Z_p$, if there is no private key sk, no one can generate a valid signature σ' on block m' by linearly combining signatures σ_1 and σ_2.

Blockless verification allows a verifier to check the integrity of all the data stored in the cloud with a linearly combining value. If the linearly combining value is correct, we can believe that the data stored in the cloud is intact. Non-malleability indicates that any attacker who knows the existing signatures cannot forge a valid signature by a linearly combining technique.

4 Problem statement

4.1 System model

In this study, the architecture of the cloud data storage model includes four entities (Fig. 1): user (U), cloud service provider (CSP), third-party auditor (TPA), and private key generator (PKG). Users can customize the cloud service, and have a large number of data files stored in the cloud. CSPs provide storage service or computing service for users with a large amount of computing and storage resources. TPA generally has a rich experience and special capabilities that cloud users do not possess, and is able to assess cloud storage service reliability on behalf of the users (Wang Q *et al.*, 2009). Different from Shacham and Waters (2008), Wang Q *et al.* (2009), and Wang *et al.* (2010; 2012), the mechanism proposed in this study contains an additional new role, the PKG server, which generates the cloud users' private/public key pair according to the requests. With the help of PKG, cloud users can effectively manage their keys.

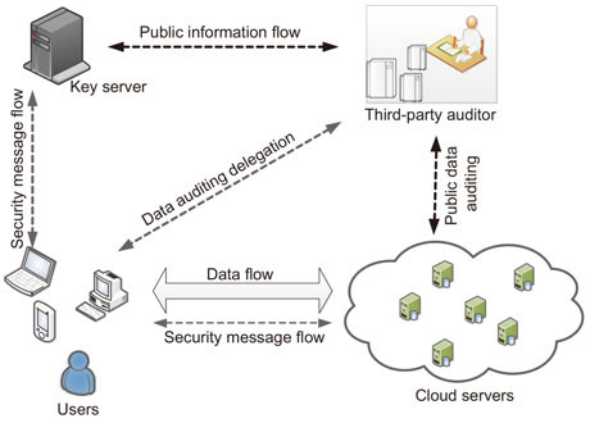

Fig. 1 The architecture of our cloud data storage model

4.2 Design goals

In this paper, we present an identity-based auditing scheme, NaEPASC, to ensure the security

and dependability of cloud data storage through the aforementioned adversary model to achieve the following goals:

1. Correctness: ensuring that the users' data is indeed stored appropriately and kept intact all the time in the cloud;

2. Efficiency: verifying the integrity of data without retrieving the entire data from the cloud server;

3. Dependability: improving the data availability against Byzantine failures, malicious data modification, and server colluding attacks;

4. Lightweight: enabling the user and the TPA to perform storage correctness checks with the minimum overhead.

Note that the proposed design does not consider any additional property, such as error-resiliency and privacy protection. Therefore, if the user wants to support error-resiliency, he/she should first redundantly encode the data files before generating the verification metadata and, if the user does want to protect data privacy, he/she should use random masked technology to protect the data during the challenge phase (Wang *et al.*, 2013).

5 Constructions

In this study, NaEPASC extends identity-based aggregate signatures (IBAS) to achieve blockless verification. We first make a brief review of IBAS.

5.1 Identity-based aggregate signatures

The IBAS scheme often consists of five algorithms: KeyGen, Extraction, Sign, Aggregate, and Verify (Gentry and Silverberg, 2002; Gentry and Ramzan, 2006). KeyGen is a key generation algorithm which uses the PKG to output a suitable key pair. Extraction generates the user's secret value according to the identity by the PKG. Sign is used to sign messages by the user through the private key. Aggregate is used to aggregate a collection of individual signatures into one signature. In the Verify algorithm, the correctness of a message is checked by the verifier with the public key.

Our identity-based auditing scheme for checking the integrity of the data in the cloud is constructed based on IBAS. The differences between NaEPASC and IBAS are as follows: First, NaEPASC is a blockless verification. Thus, the verifier, e.g., TPA, does not need to download all the data from the cloud server to check the integrity of the users' data. Second, considering the large size of the outsourced data, NaEPASC usually randomly samples a constant block of the file to check the integrity of the outsourced data. According to Ateniese *et al.* (2011), if 300 blocks are randomly selected, the verifier can detect the misbehavior of the cloud server with probability 95%. Third, NaEPASC verifies the integrity of the outsourced data by periodically challenging the cloud server. In other words, the verification process of NaEPASC does not stop unless the user decides to drop it.

5.2 Framework of NaEPASC

NaEPASC is also composed of five algorithms: KeyGen, KeyExt, Sign, GenProof, and VerifyProof. KeyGen and KeyExt algorithms are the same as the KeyGen and Extraction in Gentry and Silverberg (2002) and Gentry and Ramzan (2006). Sign is used by the cloud users to generate the verification metadata. GenProof is used to generate a proof of possession of data by the cloud server. VerifyProof is used to check the proof which is returned from the cloud server by the verifier.

NaEPASC is usually executed in two phases: Setup and Challenge.

1. Setup: The PKG initializes the public and secret parameters of the system by executing KeyGen. Then, the cloud user with his/her public identity receives the secret key from the PKG, and preprocesses data file F using Sign to generate the verification metadata. Finally, the cloud user stores data file F and the verification metadata at the cloud server, and deletes its local copies.

2. Challenge: The TPA first sends an auditing challenge request to the cloud server to make sure that the cloud server has truly stored their data files F at the time of the Challenge. As the cloud server receives the request, it aggregates the proof of the stored data files F by executing GenProof, and returns this proof to the TPA. Lastly, the TPA verifies this proof via running VerifyProof.

In this study, we omit some additional properties on data file F. If the user wants to have some other properties such as error-resiliency, he/she can use error-correcting codes to pre-process the data file before executing the Sign (Shacham and Waters, 2008).

5.3 Construction of NaEPASC

In this subsection, the details of NaEPASC are introduced. First, some definitions are presented. G_1 and G_2 refer to cyclic groups of order p, and e is an admissible pairing: $G_1 \times G_1 \to G_2$. Obviously, e is a bilinear map. There are three cryptographic hash functions $H_1, H_2 \colon \{0,1\}^* \to G_1$ and $H_3 \colon \{0,1\}^* \to Z_p$. The details are given as the following.

5.3.1 Setup phase

The PKG runs KeyGen to generate the public and secret parameters. The PKG first chooses an arbitrary generator $P \in G_1$, a random $x \in Z_p$ and sets $Q \leftarrow xP$. The PKG's public key is (P, Q), and its secret is $x \in Z_p$.

The cloud user with identity ID receives the value of xP_j for $j \in \{0,1\}$ from the PKG, where $P_j \leftarrow H_1(\mathrm{ID}, j) \in G_1$. The secret parameter is $\mathrm{sk} = (xP_j)$, and the public parameters of NaEPASC are $\mathrm{pk} = (Q, P)$.

Given a data file F, the user runs Sign to compute signature $\{S_i, T_i\}$ for each block m_i as follows:

1. Choose a random $r_i \in Z_p$ and compute $T_i = r_i P$.

2. Compute two hash values, P_w and c_i: $P_w = H_2(\mathrm{filename}) \in G_1$, $c_i = H_3(\mathrm{ID}, \mathrm{filename}, \mathrm{id}_i) \in Z_p$, where id_i is the index of block m_i, and filename is the name of data file F.

3. Compute $S_i = r_i P_w + c_i x P_0 + m_i x P_1 \in G_1$.

The set of signatures is denoted by $\Phi = \{\sigma_i\} = \{S_i, T_i\}$, $1 \le i \le n$. Finally, the client sends $\{\mathrm{filename}, \Phi\}$ to the cloud server and deletes them from its local storage.

5.3.2 Challenge phase

For simplicity, we assume that the TPA knows the number of blocks and the file name. Before challenging, the TPA needs to generate a challenge request chal. First, the TPA randomly picks c-element subset $I = \{s_1, s_2, \cdots, s_c\}$ of the set {1, 2, ..., n} as the position of the blocks that will be checked in this auditing process. Second, for each $i \in I$, the TPA also chooses a random value $v_i \in Z_q$, where $|q| = |p|/2$. Then, the TPA sends an auditing message $\mathrm{chal} = \{i, v_i\}_{i \in I}$ to the cloud server.

Upon receiving the challenge $\mathrm{chal} = \{(i, v_i)\}_{i \in I}$, the cloud server runs GenProof to generate a response proof of data storage correctness. The cloud server first computes the linear combination of sampled blocks specified in chal: $\mu = \sum_{i \in I} v_i m_i \in Z_p$. Then it calculates two aggregated authenticators $S_n = \sum_{i \in I} v_i S_i \in G_1$ and $T_n = \sum_{i \in I} v_i T_i \in G_1$. Finally, it sends $\{\mu, S_n, T_n\}$ as the response proof of storage correctness to the TPA.

With the response from the cloud server, the TPA runs VerifyProof to verify the response by checking the verification equation:

$$e(S_n, P) = e(T_n, P_w) e\left(\sum_{s_1}^{s_n} c_i v_i P_0 + \mu P_1, Q\right), \quad (1)$$

where $P_j = H_1(\mathrm{ID}, j)$, $P_w = H_2(\mathrm{filename})$, $c_i = H_3(\mathrm{ID}, \mathrm{filename}, \mathrm{id}_i)$.

6 Security analysis

In this section, we discuss the security characteristics of NaEPASC, including its correctness and unforgeability.

Theorem 1 Given a data file F and its block signatures, the TPA is able to correctly check the integrity of F through NaEPASC.

Proof To prove the correctness of NaEPASC, we need only to prove Eq. (1) is true. According to the properties of bilinear maps, the right-hand side of Eq. (1) can be expanded as follows:

$$\begin{aligned}
\mathrm{Right} &= e(T_n, P_w) e\left(\sum_{s_1}^{s_n} c_i v_i P_0 + \mu P_1, Q\right) \\
&= e\left(\sum_{s_1}^{s_n} r_i v_i P, P_w\right) e\left(\sum_{s_1}^{s_c} c_i v_i P_0 + \sum_{s_1}^{s_c} v_i m_i P_1, Q\right) \\
&= e\left(P, \sum_{s_1}^{s_n} r_i v_i P_w\right) e\left(x \sum_{s_1}^{s_c} v_i (c_i P_0 + m_i P_1), P\right) \\
&= e\left(\sum_{s_1}^{s_n} v_i (r_i P_w + c_i x P_0 + x m_i P_1), P\right) \\
&= e\left(\sum_{s_1}^{s_n} v_i S_i, P\right) = e(S_n, P) = \mathrm{Left}.
\end{aligned}$$

Obviously, our NaEPASC is able to correctly check the integrity of data F.

The following theorem shows that the attacker can never give a forged response back to the verifier. In other words, the verification algorithm Verify will omit 'false' except when the prover's proof $\{S_n, T_n, \mu\}$ is correctly computed.

Theorem 2 If the signature scheme is existentially unforgeable, and the CDH problem is difficult in bilinear groups, no adversary can break NaEPASC, except by responding with the correct values $\{S_n, T_n, \mu\}$.

Proof The proof process has some similarities to the IBAS (Gentry and Ramzan, 2006). We show how to construct a t'-time algorithm $\mathcal{B}$ to solve CDH in G_1 with a probability of at least ε'. This contradicts the fact that G_1 is a (t', ε')-CDH group.

Given $X = xP$, P, and P', the goal of $\mathcal{B}$ is to output xP'. Assume that algorithm $\mathcal{A}(t, q_{H_1}, q_{H_2}, q_{H_3}, q_E, q_S, \epsilon)$ breaks our checking scheme. $\mathcal{B}$ arbitrarily interacts with algorithm $\mathcal{A}$ as follows:

Setup: Algorithm $\mathcal{B}$ initializes $\mathcal{A}$ with $Q = X$ as a system overall public key.

Hash queries: $\mathcal{A}$ makes an H_1-query, H_2-query, or H_3-query at any time.

Queries on oracle H_1: To respond to query on H_1 oracle, $\mathcal{B}$ maintains a list L_1 of tuples $\langle \mathrm{ID}, c_1, b_0, b_0', b_1, b_1' \rangle$ as explained below:

1. If ID is in a previous H_1-query, $\mathcal{B}$ recovers (b_0, b_0', b_1, b_1') from its L_1.

2. Otherwise, $\mathcal{B}$ generates a random $c_1 \in \{0, 1\}$, so that $\Pr[c_1 = 0] = 1/(q_\mathrm{E} + 1)$. If $c_1 = 0$ holds, $\mathcal{B}$ generates random $b_0, b_1 \in Z_p$, and sets $b_0' = b_1' = 0$; else, it generates random $b_0, b_0', b_1, b_1' \in Z_p$. $\mathcal{B}$ logs $\langle \mathrm{ID}, c_1, b_0, b_0', b_1, b_1' \rangle$ in its L_1.

3. $\mathcal{B}$ responds with $H_1(\mathrm{ID}, j) = P_j = b_j P + b_j' P'$ for $j \in \{0, 1\}$.

Queries on oracle H_2: To respond to query on H_2 oracle, $\mathcal{B}$ also maintains a list L_2 of tuples $\langle \mathrm{filename}, c_2, \eta \rangle$ related to its previous oracle query responses for consistency. $\mathcal{B}$ responds to $\mathcal{A}$'s H_2-query as follows:

1. If the query filename already appears on the H_2-query, algorithm $\mathcal{B}$ recovers η from its L_2.

2. Otherwise, $\mathcal{B}$ generates a random $c_2 \in \{0, 1\}$ so that $\Pr[c_2 = 0] = 1/2$, which will be determined later.

3. $\mathcal{B}$ generates a random $\eta \in Z_p$, and adds the tuple $\langle \mathrm{filename}, c_2, \eta \rangle$ to the L_2. If $c_2 = 0$, algorithm $\mathcal{B}$ responds with $H(\mathrm{filename}) = P_w = \eta P'$; otherwise, it responds with $H(\mathrm{filename}) = P_w = \eta P$.

Queries on oracle H_3: For $\mathcal{A}$'s H_3-query, $\mathcal{B}$ maintains a list L_3 of tuples $\langle \mathrm{ID}, \mathrm{filename}, \mathrm{id}_i, \upsilon_i \rangle$. When a tuple $\langle \mathrm{ID}, \mathrm{filename}, \mathrm{id}_i \rangle$ is submitted to the H_3 oracle, algorithm $\mathcal{B}$ responds as follows:

1. If the query tuple already appears on list L_3 in some tuple $\langle \mathrm{ID}, \mathrm{filename}, \mathrm{id}_i, \upsilon_i \rangle$, $\mathcal{B}$ responds with $H_3(\mathrm{ID}, \mathrm{filename}, \mathrm{id}_i) = \upsilon_i \in Z_p$.

2. Otherwise, $\mathcal{B}$ runs an H_1-query on $\langle \mathrm{ID} \rangle$ to recover b_0' and b_1' from its L_1. $\mathcal{B}$ generates a random $c_3 \in \{0, 1\}$ so that $\Pr[c_3 = 0] = 1/p$.

(1) If $c_1 = 1, c_2 = 1$, and $c_3 = 0$, $\mathcal{B}$ sets $\upsilon_i = -m_i b_1'/b_0'$.

(2) If $c_1 = 0, c_2 = 0$, or $p_3 = 1$, $\mathcal{B}$ generates a random υ_i.

(3) $\mathcal{B}$ logs $\langle \mathrm{ID}, \mathrm{id}_i, \mathrm{filename}, c_3, \upsilon_i \rangle$ in its L_3.

3. $\mathcal{B}$ responds with $H_3(\mathrm{ID}, \mathrm{id}_i, w) = \upsilon_i$.

Extraction queries: When $\mathcal{A}$ requests the private key associated with the identity ID, $\mathcal{B}$ recovers the corresponding tuple $\langle \mathrm{ID}, c_1, b_0, 0, b_1, 0 \rangle$ from L_1. If $c_1 = 0$, the values $(b_0 Q, b_1 Q)$ are then returned to $\mathcal{A}$ as a private key associated to ID. Otherwise, it outputs 'failure' and halts.

Signature queries: When $\mathcal{A}$ queries a signature on the designed block $\langle m_i, \mathrm{filename}, \mathrm{ID} \rangle$, $\mathcal{B}$ first checks whether $\mathcal{A}$ has previously requested this query or not. If not, $\mathcal{B}$ then generates the signature of block i without private key xP_j, which proceeds as follows:

1. If $c_1 = c_2 = c_3 = 1$, $\mathcal{B}$ aborts.

2. If $c_1 = 0$, $\mathcal{B}$ generates random $r_i \in Z_p$ and outputs signature $\{S_i', T_i'\}$, where $S_i' = \upsilon_i x P_0 + m_i x P_1 + r_i P_w = \upsilon_i b_0 Q + m_i b_1 Q + r P_w$, and $T_i' = r_i P$.

3. If $c_1 = 1$ and $c_2 = 0$, $\mathcal{B}$ generates random $r_i \in Z_p$ and outputs signature $\{S_i', T_i'\}$, where

$$
\begin{aligned}
S_i' &= \upsilon_i x P_0 + m_i s P_1 \\
&\quad + [r_i - (c_i b_0' + m_i b_1') x \eta^{-1}] P_w \\
&= \upsilon_i x (b_0 P + b_0' P') + m_i x (b_1 P + b_1' P') \\
&\quad + [r_i - (c_i b_0' + m_i b_1') x \eta^{-1}] \eta P' \\
&= \upsilon_i b_0 Q + m_i b_1 Q + (c_i b_0' + m_i b_1') x P' + r \eta P' \\
&\quad - (\upsilon_i b_0' + m_i b_1') x P' \\
&= \upsilon_i b_0 Q + m_i b_1 Q + r \eta P', \\
T_i' &= [r_i - (\upsilon_i b_0' + m_i b_1') x \eta^{-1}] P \\
&= r_i P - (\upsilon_i b_0' + m_i b_1') \eta Q.
\end{aligned}
$$

4. If $c_1 = c_2 = 1$ and $c_3 = 0$, $\mathcal{B}$ generates random

$r_i \in Z_p$ and outputs signature $\{S'_i, T'_i\}$, where

$$\begin{aligned} S'_i &= \upsilon_i x P_0 + m_i s P_1 + r_i P_w \qquad (\upsilon_i = -m_i b'_1/b'_0) \\ &= -m_i b'_1/b'_0 x(b_0 P + b'_0 P') + m_i x(b_1 P + b'_1 P') \\ &\quad + r_i \eta P \\ &= -m_i b'_1/b'_0 b_0 Q + m_i b_1 Q + r_i \eta P \\ &= \upsilon_i b_0 Q + m_i b_1 Q + r_i \eta P, \\ T'_i &= r_i \eta P. \end{aligned}$$

Output: Finally, $\mathcal{A}$ halts and at least one of the following cases exists:

1. $\mathcal{A}$ concedes failure, so does $\mathcal{B}$.

2. $\mathcal{A}$ outputs a forgery response $\{\mu', S'_n, T'_n\}$.

If $c_1 = c_2 = c_3 = 1$, $\mathcal{B}$ declares failure and halts. Otherwise, it can solve its instance of CDH_{G_1}. The forgery proof $\{\mu', S'_n, T'_n\}$ must satisfy the following verification equation:

$$e(S'_n, P) = e(T'_n, P_w) e\left(\sum_{s_1}^{s_n} c_i \upsilon_i P_0 + \mu' P_1, Q\right). \tag{2}$$

Additionally, $\mathcal{B}$ would have obtained the right proof $\{\mu, S_n, T_n\}$ from an honest prover. By the correctness of the scheme, we know that the expected response satisfies

$$e(S_n, P) = e(T_n, P_w) e\left(\sum_{s_1}^{s_n} c_i \upsilon_i P_0 + \mu P_1, Q\right). \tag{3}$$

Note that if $\mu' = \mu$, we can obtain

$$e(S'_n - S_n, P) = e(T'_n - T_n, P_w) = e(\eta(T'_n - T_n), P). \tag{4}$$

Obviously, from this equation we cannot compute the value of xP', which contradicts the above assumption. We define $\Delta\mu = \mu - \mu'$, $\Delta S_n = S_n - S'_n$, and $\Delta T_n = T_n - T'_n$. Now, dividing Eq. (2) by Eq. (3), we obtain

$$\begin{aligned} & e(\Delta S_n, P) = e(\Delta T_n, P_w) e\left(\sum_{s_1}^{s_n} c_i \upsilon_i P_0 + \Delta\mu P_1, Q\right) \\ \Rightarrow\ & e(\Delta S_n, P) = e(\Delta T_n, P_w) e\left(\sum_{s_1}^{s_n} c_i \upsilon_i (b_0 P + b'_0 P') \right. \\ & \qquad \left. + \Delta\mu(b_1 P + b'_1 P'), Q\right) \\ \Rightarrow\ & e(\Delta S_n, P) = e(\eta\Delta T_n, P) e\left(\sum_{s_1}^{s_n} c_i \upsilon_i (b_0 P + b'_0 P') \right. \\ & \qquad \left. + \Delta\mu(b_1 P + b'_1 P'), Q\right) \\ \Rightarrow\ & e(\Delta S_n, P) = e(\eta\Delta T_n, P) e\left(\left(\sum_{s_1}^{s_n} c_i \upsilon_i b_0 + b_1 \Delta\mu\right) P \right. \\ & \qquad \left. + \left(\sum_{s_1}^{s_n} c_i \upsilon_i b'_0 + b'_1 \Delta\mu\right) P', Q\right) \\ \Rightarrow\ & e(S_n - S'_n, P) = e(\eta\Delta T_n, P) e\left(\left(\sum_{s_1}^{s_n} c_i \upsilon_i b_0 \right. \right. \\ & \qquad \left. \left. + b_1 \Delta\mu\right) xP + \left(\sum_{s_1}^{s_n} c_i \upsilon_i b'_0 + b'_1 \Delta\mu\right) xP', P\right) \\ \Rightarrow\ & S_n - S'_n = \eta\Delta T_n + \left(\sum_{s_1}^{s_n} c_i \upsilon_i b_0 + b_1 \Delta\mu\right) Q \\ & \qquad + \left(\sum_{s_1}^{s_n} c_i \upsilon_i b'_0 + b'_1 \Delta\mu\right) xP' \\ \Rightarrow\ & xP' = \frac{\Delta S_n - \eta\Delta T_n - (\sum_{s_1}^{s_n} c_i \upsilon_i b_0 + b_1 \Delta\mu) Q}{\sum_{s_1}^{s_n} c_i \upsilon_i b'_0 + b'_1 \Delta\mu}. \end{aligned}$$

So, we find a way to solve the problem of CDH.

7 Performance

In this section, the performance of the proposed identity-based public auditing scheme, NaEPASC, is discussed. Suppose there are c random blocks specified in the challenge request during the Challenge phase. We give the cost of basic cryptographic operations in Table 1. In the following discussion, we will omit the operation in Z_p, such as addition, multiplication, and hash operation, because they are much easier to compute than those basic cryptographic operations in G_1. In this setting environment, we can quantify the cost of server computation, auditor computation, and communication overhead in our NaEPASC. To make comparison with existing research, we have implemented the other two schemes, privacy-preserving public auditing (Wang *et al.*, 2013) and compact POR (Shacham and Waters, 2008; 2013). The security level of these experiments is set to be 80 bits, which means $|\upsilon_i| = 80$ and $|p| = 160$.

Table 1 Notations of cryptographic operations*

Notation	Description
$\mathrm{Hash}^t_{G_1}$	Hash t values into group G_1
$\mathrm{Add}^t_{G_1}$, $\mathrm{Mult}^t_{G_1}$	t additions/multiplications in group G_1
$\mathrm{Mult}^t_{G_1}(l)$, $\mathrm{Exp}^t_{G_1}(l)$	t multiplications/exponentiations a_iP/P^{a_i}, for $P\in G_1, \lvert a_i\rvert = l$
m-$\mathrm{AddMult}^t_{G_1}(l)$, m-$\mathrm{MultExp}^t_{G_1}(l)$	t m-term multiplications/exponentiations $\sum_{i=1}^m a_iP/\prod_{i=1}^m P^{a_i}$
$\mathrm{Pair}^t_{G_1,G_1}$	t pairings $e(U_i,P_i)$, where U_i, $P_i\in G_1$

* Different cryptographic operations in each item of this table have the same computation cost

7.1 Computation cost

We first analyze the cost of server computation. On the cloud server side, the response generated includes a pair of aggregated signatures $\{S_n, T_n\}$ and a linear combination of sampled blocks $\mu = \sum_{i\in I} v_i m_i$. The whole computation cost is about c-$\mathrm{AddMult}^2_{G_1}(|v_i|)$+ $\mathrm{Add}^{c-1}_{Z_p} + \mathrm{Mult}^c_{Z_p}$. Table 2 shows the overall computation cost of these three schemes. The extra cost of NaEPASC on the server side is about c-$\mathrm{MultExp}^1_{G_1}(|v_i|)$ against the total server computation compared to the other two schemes.

Similarly, on the auditor side, the receiving response is $\{S_n, T_n, \mu\}$, and the corresponding computation cost for verifying a proof is about $\mathrm{Pair}^3_{G_1,G_1}$ + $2\mathrm{Mult}_{G_1} + \mathrm{Mult}_{G_2} + \mathrm{Mult}^c_{Z_p} + \mathrm{Add}^{c-1}_{Z_p} + \mathrm{Hash}^3_{G_1}$ + $\mathrm{Hash}^c_{Z_p}$. Considering the relatively expensive c-$\mathrm{MultExp}^t_{G_1}(|v_i|)$ operation, our scheme has better performance than the other two.

7.2 Communication overhead

The communication cost of NaEPASC comprises mainly two parts: auditing request and auditing proof. The TPA first sends auditing message $\{i, v_i\}_{i\in Q}$ to the cloud, and the corresponding communication overhead is about $c(|n| + |p|/2)$ bits. After receiving the auditing request, the cloud computes a proof $\{\mu, S_n, T_n\}$ of the designed files according to Eq. (1), and the corresponding communication overhead is about $3|p|$.

7.3 Experimental results

We run our experiment on a PC with an Intel Core 2 processor running at 2.4 GHz, 2 GB RAM, and a 5400 RPM Western Digital 320 GB serial ATA drive with an 8 MB buffer. Our source code uses the pairing-based cryptography (PBC) library of version 0.4.18. The elliptic curve used in the experiment is an MNT curve, with a base field size of 159 bits and an embedding degree of 6. All the results are the average of 100 trials. In our experiment, the size of the simulation data sets is about 1 GB. According to Ateniese *et al.* (2007), we know that if 1% of all the blocks are corrupted, by randomly selecting 460 blocks, the probability that the TPA successfully detects this misbehavior is greater than 99%. If 300 blocks are selected randomly, the probability is greater than 95%.

Table 3 shows the auditing time of NaEPASC and the other two schemes (Shacham and Waters, 2013; Wang *et al.*, 2013). Due to the shorter operation (i.e., c-$\mathrm{MultExp}^t_{G_1}(|v_i|)$), the auditing of NaEPASC is about 45 times faster than that of the other two schemes in terms of TPA computation time. However, NaEPASC has a larger computation cost on the server side as one more aggregated value needs to be computed. Additionally, if someone wants to obtain higher assurance through increasing the number of sampled blocks, the auditing time of these three mechanisms grows. The increase of the computation time in NaEPASC, however, is much less rapid than that in the other two schemes on the TPA side. Increasing the value of c from 300 to 460, TPA's verification time increases by more than 300 ms for the other two schemes; in contrast, NaEPASC needs only 0.4 ms more. Thus, when the cloud user wants to obtain a higher detection probability, this property of NaEPASC will greatly reduce the burden of the TPA. Finally, the evaluation results show that the communication cost of NaEPASC is almost the same as that of the other two schemes, although NaEPASC supports the additional identity-based guarantee.

8 Conclusions

In this paper, the first identity-based auditing scheme (NaEPASC) is proposed to check the integrity of the data stored in the cloud. NaEPASC adopts an identity-based aggregate signature, to construct real-time and homomorphic verifiable tags, and the TPA is able to audit the correctness on behalf of the users. The proposed scheme not only eliminates the burden on cloud users of tedious and

Table 2 Comparison of computation cost and communication overhead between different data integrity checking schemes

Method	Server computation cost	TPA computation cost	Communication overhead
Wang *et al.* (2013)	c-MultExp$^1_{G_1}(\|v_i\|)+$Exp$^1_{G_1}(\|p\|)$ +Hash$^1_{Z_p}$ + Add$^c_{Z_p}$ + Mult$^{c+1}_{Z_p}$	Pair$^2_{G_1,G_2}$ + Exp$^3_{G_1}(\|p\|)$ + Mult$^1_{G_T}$ + Mult$^1_{G_1}$ +Hash$^c_{G_1}$ + Hash$^1_{Z_p}$ + c-MultExp$^1_{G_1}(\|v_i\|)$	$c\|n\|+(c/2+7)\|p\|$
Shacham and Waters (2008; 2013)	c-MultExp$^1_{G_1}(\|v_i\|)$ + Add$^{c-1}_{Z_p}$ +Mult$^c_{Z_p}$	Pair$^2_{G_1,G_2}$ + Exp$^1_{G_1}(\|p\|)$ + Mult$^1_{G_1}$ + Hash$^c_{G_1}$ + c-MultExp$^1_{G_1}(\|v_i\|)$	$c\|n\|+(c/2+2)\|p\|$
NaEPASC	c-AddMult$^2_{G_1}(\|v_i\|)$ + Add$^{c-1}_{Z_p}$ +Mult$^c_{Z_p}$	Pair$^3_{G_1,G_1}$ + Mult$^2_{G_1}$ + Mult$^1_{G_2}$ + Mult$^c_{Z_p}$ +Add$^{c-1}_{Z_p}$ + Hash$^3_{G_1}$ + Hash$^c_{Z_p}$	$c\|n\|+(c/2+3)\|p\|$

Table 3 Performances under different numbers of sampled blocks c for high assurance checking schemes

Method	TPA computation time (ms)		Server computation time (ms)		Communication overhead (KB)	
	$c=300$	$c=460$	$c=300$	$c=460$	$c=300$	$c=460$
Wang *et al.* (2013)	636.0	963.9	619.4	947.5	4.24	6.43
Shacham and Waters (2008; 2013)	627.9	955.7	615.3	943.4	4.14	6.33
NaEPASC	**14.3**	**14.6**	1230.7	1886.1	4.16	6.34

possibly expensive auditing tasks, but also alleviates the users' fear of losing their keys. Experiment results show that the proposed scheme is efficient and secure in checking the integrity of the data stored in the cloud.

References

Ateniese, G., Burns, R., Curtmola, R., *et al.*, 2007. Provable data possession at untrusted stores. Proc. 14th ACM Conf. on Computer and Communications Security, p.598-609. [doi:10.1145/1315245.1315318]

Ateniese, G., di Pietro, R., Mancini, L.V., *et al.*, 2008. Scalable and efficient provable data possession. Proc. 4th Int. Conf. on Security and Privacy in Communication Networks, Article 9. [doi:10.1145/1460877.1460889]

Ateniese, G., Burns, R., Curtmola, R., *et al.*, 2011. Remote data checking using provable data possession. *ACM Trans. Inform. Syst. Secur.*, **14**(1):1-12. [doi:10.1145/1952982.1952994]

Boneh, D., Boyen, X., 2004. Efficient selective-ID secure identity-based encryption without random oracles. Advances in Cryptology-EUROCRYPT, p.223-238.

Boneh, D., Franklin, M., 2001. Identity-based encryption from the weil pairing. Advances in Cryptology-CRYPTO, p.213-229.

Boneh, D., Boyen, X., Goh, E.J., 2005. Hierarchical identity based encryption with constant size ciphertext. Advances in Cryptology-EUROCRYPT, p.440-456. [doi:10.1007/11426639_26]

Chen, B., Curtmola, R., 2012. Robust dynamic provable data possession. 32nd Int. Conf. on Distributed Computing Systems Workshops, p.515-525. [doi:10. 1109/ICDCSW.2012.57]

Erway, C., Kupcu, A., Papamanthou, C., *et al.*, 2009. Dynamic provable data possession. Proc. 16th ACM Conf. on Computer and Communications Security, p.213-222. [doi:10.1145/1653662.1653688]

Gartner, 2010. Gartner Identifies the Top 10 Strategic Technologies for 2011. Available from http://www.gartner.com/newsroom/id/1454221

Gentry, C., Ramzan, Z., 2006. Identity-based aggregate signatures. Public Key Cryptography, p.257-273.

Gentry, C., Silverberg, A., 2002. Hierarchical ID-based cryptography. Advances in Cryptology-CRYPTO, p.548-566.

Hao, Z., Zhong, S., Yu, N.H., 2011. A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. *IEEE Trans. Knowl. Data Eng.*, **23**(9):1432-1437. [doi:10.1109/TKDE.2011.62]

Hashizume, K., Rosado, D.G., Fernandez-Medina, E., *et al.*, 2013. An analysis of security issues for cloud computing. *J. Internet Serv. Appl.*, **4**:5. [doi:10.1186/1869-0238-4-5]

Hochmuth, P., Richmond, C., Hudson, S., *et al.*, 2013. 2013 U.S. Cloud Security Survey. Technical Report No. 242836, International Data Corporation (IDC), USA. Available from http://www.idc.com/getdoc.jsp?containerId=242836.

Juels, A., Kaliski, B.S.Jr., 2007. Pors: proofs of retrievability for large files. Proc. 14th ACM Conf. on Computer and Communications Security, p.584-597. [doi:10.1145/1315245.1315317]

Khan, A., Kiah, M.L.M., Khan, S.U., *et al.*, 2013a. A study of incremental cryptography for security schemes in mobile cloud computing environments. IEEE Symp. on Wireless Technology and Applications, p.62-67. [doi:10.1109/ISWTA.2013.6688818]

Khan, A., Othman, M., Madani, S.A., *et al.*, 2013b. A survey of mobile cloud computing application models. *IEEE Commun. Surv. Tutor.*, **16**(1):393-413. [doi:10.1109/SURV.2013.062613.00160]

Lokantas, F., Salonu, H.S., 2013. IDC's Cloud Computing and Datacenter Roadshow 2013. Available from http://idc-cema.com/eng/events/50527-idc-s-cloud-computing-and-datacenter-roadshow-2013.

Mell, P., Grance, T., 2009. The NIST Definition of Cloud Computing. Technical Report No. SP 800-145, National Institute of Standards and Techninology (NIST), USA.

Ristenpart, T., Tromer, E., Shacham, H., *et al.*, 2009. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. Proc. 16th ACM Conf. on Computer and Communications Security, p.199-212. [doi:10.1145/1653662.1653687]

Shacham, H., Waters, B., 2008. Compact proofs of retrievability. Advances in Cryptology-ASIACRYPT, p.90-107.

Shacham, H., Waters, B., 2013. Compact proofs of retrievability. *J. Cryptol.*, **26**(3):442-483. [doi:10.1007/s00145-012-9129-2]

Shamir, A., 1985. Identity-based cryptosystems and signature schemes. Advances in Cryptology-ASIACRYPT, p.47-53. [doi:10.1007/3-540-39568-7_5]

Wang, C., Wang, Q., Ren, K., *et al.*, 2009. Ensuring data storage security in cloud computing. 17th Int. Workshop on Quality of Service, p.1-9. [doi:10.1109/IWQoS.2009.5201385]

Wang, C., Wang, Q., Ren, K., *et al.*, 2010. Privacy-preserving public auditing for data storage security in cloud computing. Proc. IEEE INFOCOM, p.1-9. [doi:10.1109/INFCOM.2010.5462173]

Wang, C., Wang, Q., Ren, K., *et al.*, 2012. Toward secure and dependable storage services in cloud computing. *IEEE Trans. Serv. Comput.*, **5**(2):220-232. [doi:10.1109/TSC.2011.24]

Wang, C., Chow, S., Wang, Q., *et al.*, 2013. Privacy-preserving public auditing for secure cloud storage. *IEEE Trans. Comput.*, **62**(2):362-375. [doi:10.1109/TC.2011.245]

Wang, Q., Wang, C., Li, J., *et al.*, 2009. Enabling public verifiability and data dynamics for storage security in cloud computing. Computer Security-ESORICS, p.355-370. [doi:10.1007/978-3-642-04444-1_22]

Waters, B., 2005. Efficient identity-based encryption without random oracles. Advances in Cryptology-EUROCRYPT, p.114-127. [doi:10.1007/11426639_7]

Zhu, Y., Hu, H.X., Ahn, G.J., *et al.*, 2011a. Collaborative integrity verification in hybrid clouds. 7th Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), p.191-200.

Zhu, Y., Wang, H.X., Hu, Z.X., *et al.*, 2011b. Zero-knowledge proofs of retrievability. *Sci. China Inform. Sci.*, **54**(8):1608-1617. [doi:10.1007/s11432-011-4293-9]