This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

Blockchain-assisted Public-key Encryption with Keyword Search against Keyword Guessing Attacks for Cloud Storage

Yuan Zhang, *Student Member, IEEE,* Chunxiang Xu, *Member, IEEE,* Jianbing Ni, *Member, IEEE,* Hongwei Li, *Senior Member, IEEE,* and Xuemin (Sherman) Shen, *Fellow, IEEE*

Abstract—Cloud storage enables users to outsource data to storage servers and retrieve target data efficiently. Some of the outsourced data are very sensitive and should be prevented for any leakage. Generally, if users conventionally encrypt the data, searching is impeded. Public-key encryption with keyword search (PEKS) resolves this tension. Whereas, it is vulnerable to keyword guessing attacks (KGA), since keywords are low-entropy. In this paper, we present a secure PEKS scheme called SEPSE against KGA, where users encrypt keywords with the aid of dedicated key servers via a threshold and oblivious way. SEPSE supports key renewal to periodically replace an existing key with a new one on each key server to thwart the key compromise. Furthermore, SEPSE can efficiently resist online KGA, where each keyword request made by a user is integrated into a transaction on a public blockchain (e.g., Ethereum), which allows key servers to learn the number of keyword requests made by the user without requiring a synchronization between them for per-user rate limiting. Security analysis and performance evaluation demonstrate that SEPSE provides a stronger security guarantee compared with existing schemes, at the expense of acceptable computational costs.

Index Terms—Cloud storage, public-key encryption with keyword search, keyword guessing attacks, key renewal, blockchain.

1 Introduction

With cloud storage services, users can outsource their data to the remote storage server and flexibly access them via the Internet [1], [2], [3]. Such services also provide users an efficient way to send their data to others. Typical applications include store-and-forward systems, such as cloud-based email systems [4], where multiple users (called senders) are willing to send data containing a small number of keywords to one user (called receiver). Senders are able to outsource the data as well as keywords to the storage server, and the receiver can retrieve target data from the storage server through searching by keywords. This can free senders and the receiver from heavy local storage costs, and allows the receiver to access the target data on other devices (e.g., smartphones) at a later point in time [5], [6].

While users (hereinafter, the senders and receiver are collectively referred to as "users") enjoy great benefits from the cloud storage services, critical security concerns in data outsourcing have been raised seriously. One of the most important security issues is data confidentiality. From the perspective of users, contents of outsourced data are very sensitive, and should not be leaked for preserving privacy [7], [8], [9]. Therefore, senders always encrypt the data before outsourcing. This can be achieved by utilizing conventional encryption, but it makes efficient searches over ciphertexts by keyword impossible [10].

Public-key encryption with keyword search (PEKS) is a cryptographic primitive that addresses the above problem [11], [12]. In PEKS, both the data and the corresponding keywords are encrypted under the receiver's public key, and the ciphertexts are outsourced to the storage server (i.e., cloud server). The receiver can generate a trapdoor on a keyword by using her/his secret key and the storage server can test whether the ciphertext of keyword matches the trapdoor for data retrieval [13]. However, PEKS is subject to an inherent security limitation: vulnerability against off-line keyword guessing attacks (KGA). Specifically, given a trapdoor, an adversary encrypts all keyword candidates by using the receiver's public key and identifies the ciphertext which matches the targeted trapdoor, this enables the adversary to recover the keyword hidden in the trapdoor to violate the users' privacy. Such attacks are based on the observation that keywords are always selected from a small space and receivers usually utilize well-known keywords for searches of files [14]. We stress that vulnerability of PEKS against off-line KGA is a major hindrance towards the broad adoption of PEKS, since searched data (e.g., emails with sensitive keywords) is considered as being highly secret by many individuals and organizations.

Recently, a practical PEKS scheme with resistance against off-line KGA was proposed [15]. The keyword to be encrypted is derived from a key server that is separate from the storage server. Such a server-derived keyword is generated by an oblivious protocol executed between a user and the key server, which enables the user to receive the server-derived keyword without leaking any information on the

- *Y. Zhang, C. Xu, and H. Li are with the Center for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, China (e-mail: ZY_LoYe@126.com; chxxu@uestc.edu.cn; hongweili@uestc.edu.cn).*
- *J. Ni and X. Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Canada (e-mail: jianbing.ni@uwaterloo.ca; sshen@uwaterloo.ca).*
- *Y. Zhang is also with the Department of Electrical and Computer Engineering, University of Waterloo, Canada.*

The corresponding authors are Chunxiang Xu and Yuan Zhang.

2168-7161 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

keyword to the key server. Adversaries, who cannot compromise the key server, cannot generate ciphertexts of keywords in an off-line manner. As such, PEKS with resistance against off-line KGA is achieved through the key server's assistance. Actually, introducing a key server to resist such "brute-force attacks" has already shown great advantages in analogous topics, e.g., encrypted data deduplication [16], [17], [18].

Despite the great advantages of server-aided PEKS, such the mechanism bears a strong assumption that the key server is reliable. At this point, the key server becomes a single point of failure. However, compromising the key server is feasible and practical, because an adversary (e.g., an adversarial storage server) can perpetually attempt to break into the target key server over a long period of time (i.e., the entire life-time of the storage server). Once he succeeds, the security of affected keywords cannot be ensured. To address this problem, the generation of server-derived keyword can be distributed over multiple key servers via a threshold secret sharing protocol [19] such that compromising any single key server would not break the security of the scheme. Nonetheless, this is still vulnerable to the key compromise, where an adversary collects a threshold number of key servers' secret shares over the entire life-time of the storage server. Thus, for long-lived data, the protection provided by secret sharing could be insufficient [20]. Furthermore, the server-aided PEKS [15] is also confronted with online KGA, where the adversarial cloud server is able to impersonate a sender to access the key server such that it obtains enough server-derived keywords to perform off-line KGA.

In this paper, we present a secure and efficient PEKS scheme called SEPSE to thwart both off-line and online KGA for secure cloud storage. In SEPSE, multiple key servers are employed to help users in encrypting keywords to resist off-line KGA without the single-point-of-failure problem. A secret used to protect keywords is shared among these key servers in a threshold way and users derive keywords from the key servers via an oblivious protocol. SEPSE supports efficient key renewal on key servers, i.e., each key server renews its secret periodically (a period is a fixed and predetermined time-interval and is called an epoch), and it would not impact on searching existing ciphertexts under the "old" secret shares. Moreover, in regards to the case that the number of key servers is large while the threshold is small, SEPSE utilizes a blockchain-assisted rate-limiting mechanism to resist online KGA, where the number of servers-derived keywords requested by a user within an epoch is limited by the key servers. Specifically, the contributions of this paper are three-folded:

- We propose SEPSE to resist off-line KGA without the single-point-of-failure problem, where multiple dedicated key servers are employed to assist users in encrypting keywords without learning any information about the keywords. SEPSE supports key renewal on key servers so as to fight against the key compromise. Compared with existing schemes [15], [21], SEPSE provides a stronger security guarantee.
- We present a blockchain-assisted rate-limiting mechanism and integrate it into SEPSE to resist online KGA, where each request of servers-derived keyword made by the user is integrated into a transaction on a public blockchain (e.g., Ethereum [22]), the key servers are able to check the number of servers-derived keyword requests of a user by checking the number of transactions created by her/him, and stop responding after a bound is reached. The proposed mechanism does not require the synchronization between key servers, and thereby is highly efficient in terms of communication costs. Since the blockchain provides a distributed and publicly verifiable way to conduct transactions, the recorded number of servers-derived keyword requests of each user cannot be tampered with. The blockchain-assisted rate-limiting mechanism is economical and favorable in the case that the number of all key servers is large but the threshold is small.
- We prove the security of SEPSE against keyword guessing adversaries (i.e., adversarial storage server), compromised key server(s), and malicious users. We provide a comprehensive performance evaluation, which demonstrates that SEPSE is efficient. Specifically, the key renewal on the key server takes less than 0.5 seconds in the case of 10 key servers, and generating one servers-derived keyword on the user side only takes within 48 milliseconds in the case of 10 key servers.

The remainder of this paper is organized as follows. The related work is reviewed in Section 2. The system model, adversary model, and design goals are defined in Section 3. Preliminaries are presented in Section 4 and the construction of SEPSE is presented in Section 5. Security proofs are provided in Section 6 and the performance is evaluated in Section 7. Finally, the conclusion and future work are provided in Section 8.

2 Related Work

The searchable encryption technique plays an important role in data outsourcing systems, such as cloud storage, where a user can submit a search query to a storage server (e.g., cloud server), and the server is able to respond with the corresponding data without learning anything more about the data content than the search result. The problem of searching on encrypted data is first introduced by Song et al. [23], in which the first scheme for searches on encrypted data was proposed. However, the scheme in [23] requires the server to linearly scan word-by-word of each file, which makes the scheme inefficient. Chang et al. [24] presented a searchable encryption scheme with enhanced security. To improve the efficiency, Goh [25] presented an index-based searchable encryption scheme. The security model of searchable encryption was first defined by Curtmola et al. [26], which requires a secure searchable encryption scheme to leak no more than the search pattern (i.e., whether a search query is repeated) and the access pattern (i.e., pointers to ciphertexts that satisfy search query). Subsequently, many searchable encryption schemes were proposed with different features [27], [28], [29], [30], [31], [32]. These schemes are based on symmetric cryptosystems and are typically applied in the scenario that a user outsources her/his data

2168-7161 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

to the storage server, and later she/he searches the data by keywords.

Boneh et al. [11] investigated the problem of searching on encrypted data under public-key cryptosystems. Compared with its symmetric counterpart, PEKS is less efficient but more expressive. Since ciphertexts in PEKS are protected under the public key of a user who searches the data from the storage server, a PEKS scheme is typically used in store-and-forward systems, e.g., cloud-based email systems. Following the Boneh et al.'s work [11], several PEKS variants [12], [33], [34], [35], [36], [37] with different features were proposed. However, Byun et al. [14] observed that keywords are selected from small space and users usually utilize well-known keywords for searches of files, this increases the advantage of off-line KGA significantly. Actually, the vulnerability of PEKS against off-line KGA has become a critical security issue in practice.

Prior works on resisting off-line KGA in PEKS can mainly be classified into four categories.

- PEKS with an authorized tester to resist off-line KGA. In this mechanism, testing matching between a ciphertext and a trapdoor only can be performed by an authority (i.e., authenticated cloud storage server) [38]. This protects keywords from KGA performed by outside adversaries. However, such the mechanism cannot resist malicious cloud storage servers, since the authenticated server can test whether a ciphertext matches a trapdoor without any limitation [39].
- PEKS built on emerging cryptographic primitives to resist off-line KGA. In [40], Sun et al. presented a PEKS scheme with resistance against off-line KGA by using indistinguishability obfuscation ($i\mathcal{O}$) [41], [42]. In this scheme, ciphertexts are generated by a signcryption scheme, and the cloud storage server cannot generate a legitimate ciphertext to test whether it matches a received trapdoor. To ensure the security, the keys used to unsigncryption are embedded into an obfuscated program generated by the receiver. Nevertheless, current $i\mathcal{O}$ constructions are inefficient to generate and execute, which makes the scheme presented in [40] inefficient.
- PEKS with fuzzy keyword search to resist off-line KGA [21]. In such the mechanism, a keyword is related to an exact keyword search trapdoor and a fuzzy keyword search trapdoor. A fuzzy keyword trapdoor corresponds to two or more keywords. The cloud server only has the fuzzy keyword search trapdoor to retrieve data. As a result, a malicious cloud storage server would not learn the exact keyword. Nonetheless, the malicious cloud storage server still can narrow down the space of the target keyword, and thereby the keyword privacy is not well protected in such the mechanism. Furthermore, it requires the receiver to filter out the non-matching ciphertexts received from the cloud storage server, which incurs heavy communication and computation costs on the receiver side.
- PEKS constructed on a new framework to resist off-line KGA. An independent trust party is employed to assist users in generating and searching ciphertexts. This mechanism is inspired by the Bellare et al.'s scheme [16] that resists brute-force attacks for secure deduplication, and was first proposed by Chen et al. [15]. Compared with the above three categories of solutions, this mechanism balances the trade-off between efficiency and security, and is more practical. Nevertheless, the Chen et al.'s scheme [15] bears a quite strong assumption that the key server is reliable and it well maintains its secret over the entire lifetime of the storage server. Furthermore, the scheme is also vulnerable to online KGA, where the adversary (e.g., the malicious storage server) impersonates a valid sender to access the key server for performing KGA.

Recently works [43], [44], [45], [46], [47] have shown great potentials for enhancing cloud storage services in terms of security and privacy from blockchains. Due to the public verifiability of public blockchains, all transactions created by a user on a blockchain can be tracked and cannot be modified and forged. Therefore, a public log can be constructed by using public blockchains to keep track of what happens to the outsourced data [5]. The log is inherently resistant to modification or forgery, which can be utilized to thwart online KGA.

To protect PEKS against off-line KGA without the single-point-of-failure problem, we propose SEPSE, where multiple key servers are employed to jointly help users in producing keywords without learning any information about the keywords. SEPSE supports key renewal on the key servers to resist the key compromise on key servers. Furthermore, SEPSE resists online KGA by a blockchain-assisted rate-limiting mechanism, where the number of servers-derived keyword requests for each user is recorded to the blockchain, and the key servers stop responding after the bound is reached. Consequently, SEPSE achieves a stronger security guarantee than existing schemes [11], [12], [15].

3 Problem Statement

3.1 System Model

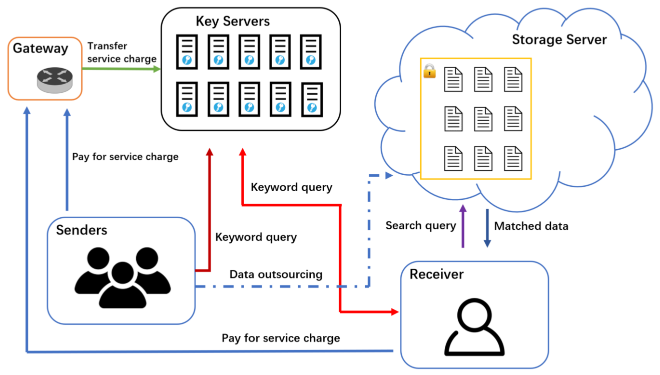


Fig. 1. System model

Fig. 1 depicts the system model. Five entities are involved in SEPSE.

- Senders: Senders generate data files which contain a small number of keywords, and send the data files as well as the keywords to a target receiver *securely* by encrypting the data files and the selected keywords with the receiver's public key. The ciphertexts are outsourced to the storage server.
- Receiver: The receiver is the data (i.e., encrypted data files) owner, she/he receives the encrypted data from the storage server, and decrypts them locally. Furthermore, she/he would search her/his data stored on the storage server by keywords at a later point in time.
- Key servers: The key servers are employed to assist senders and receivers in generating the keyword to be encrypted, which is able to resist KGA.
- Storage server: The storage server (i.e., cloud server) receives encrypted data as well as the encrypted keywords. Later, it provides the receiver with an efficient and secure way to search the ciphertexts by keywords, and forwards the target ciphertexts to the receiver.
- Gateway: The gateway is introduced to assist users in forwarding the request of keyword generation to key servers, and assist the key servers in collecting service charge from the users.

SEPSE consists of five algorithms, **Setup**, **PEKS**, **Trapdoor**, **Test**, and **KeyRenew**.

- **Setup**. In this algorithm, necessary parameters utilized in the subsequent algorithms are generated.
- **PEKS**. In this algorithm, the sender generates a servers-derived keyword with the aid of key servers, and encrypts the servers-derived keyword using the receiver's public key.
- **Trapdoor**. This algorithm enables the receiver to generate a servers-derived keyword with the aid of a random subset of all key servers, and to generate a corresponding search token (i.e., trapdoor).
- **Test**. This algorithm enables the storage server to test whether a ciphertext of servers-derived keyword matches a given search token.
- **KeyRenew**. This algorithm enables each key server to update its secret share without changing the secret shared among all key servers.

Time in SEPSE is divided into fixed intervals of predetermined length called epochs. **Setup** needs to be executed only once over the entire life-time, **KeyRenew** needs to be executed only once in an epoch, and **PEKS**, **Trapdoor**, and **Test** can be executed multiple times in an epoch.

3.2 Adversary Model

In a servers-aided PEKS scheme, threats are mainly from four angles: adversarial storage server, compromised key server(s), malicious sender, and malicious gateway.

- Adversarial storage server. The adversarial storage server would perform KGA to break the confidentiality of the outsourced keywords.
- Compromised key server(s). An adversary may compromise the key servers, where he may break into each key server multiple times, to retrieve the outsourced keywords. Here, we assume that the number of key servers that can be compromised by the adversary within one epoch is less than the threshold, a key server can be compromised until the end of the current epoch, and compromised key servers are allowed to deviate from the scheme arbitrarily as the adversary needed. When a new epoch begins, the key servers that are compromised by the adversary in the previous epoch are "released" by the adversary [20].
- Malicious sender. Any adversary can become a sender to query the servers-derived keywords and perform KGA to retrieve the outsourced keywords.
- Malicious gateway. The malicious gateway may collude with an online keyword guessing adversary (will be detailed in Section 5.1) to break the security.

To prove the security of SEPSE under the above adversary model, we follow the security notions of semantic-security against chosen keyword guessing attacks (SS-CKGA) and indistinguishability against chosen keyword attacks (IND-CKA) [11], [15]. Details will be provided in Section 6.

3.3 Design Goals

To design SEPSE, two challenges should be addressed:

1) How to resist KGA without the single-point-of-failure problem. Since the keywords are inherently low-entropy, when the receiver searches data from the storage server by keyword, the storage server can enumerate all possible keywords and test them one by one to retrieve the target keyword. Although the existing scheme [15] can resist such attacks, it faces the single-point-of-failure problem, and its security solely relies on the reliability of the key server.
2) How to periodically renew the secret shares on the key servers. In reality, an active adversary may break into a subset of key servers multiple times over the entire life-time of the storage server. As such, periodical renewal of secret share on each key server is an affordable and effective solution to resist the active adversary.

With the above model, SEPSE should accomplish the following objects.

- Functionality: The storage server, who only holds the public parameters, can test that whether a trapdoor matches a ciphertext; Such the search operation over encrypted data should not be affected or stopped after the key renewal on key servers.
- Security: The storage server cannot violate the privacy of receiver/sender by guessing keywords; Compromising one or more key servers cannot break the security; Each key server can periodically renew its secret share so as to resist the key compromise.
- Efficiency: The computation and communication overhead on the sender/receiver side should be as soon as possible; The key renewal phase should solely require the participation of key servers.

4 Preliminaries

4.1 Notation

Suppose i and j are positive integers ($i < j$), $[1, i]$ denotes the set $\{1, 2, ..., i\}$ and $i{+}{+}$ denotes the operation of $i = i + 1$. Let S be a finite set, the number of its elements is denoted by $|S|$. For two bit-strings a and b, their concatenation is denoted by $a||b$.

4.2 Threshold Cryptography

A (t, n)-threshold cryptosystem enables n players to share a secret and each of them to hold a *secret share*. Any t players can pool their shares and complete certain cryptographic operations (e.g., signature generation, encryption/decryption, recovering the secret), but no coalition of fewer than t players can get any information on the secret from their collective shares. The first threshold cryptosystem was presented by Shamir [19].

4.3 Bilinear Maps

Suppose G is an additive group with a prime order p and G_T is a multiplicative group with the same order. $e\colon G \times G \to G_T$ is a bilinear map if it has three properties:

- $e(uX, vY) = e(X, Y)^{uv}, \forall X, Y \in G,\ u, v \in Z_p^*$.
- $\forall X, Y \in G$ and $X \neq Y$, $e(X, Y) \neq 1$.
- e can be calculated efficiently.

Computational Diffe-Hellman (CDH) problem in G: given G and one of its generators P, for any unknown $a, b \in Z_p^*$, given aP and bP to compute abP.

4.4 Blockchain

A blockchain is a protocol executed by multiple participants to enable them to achieve a consensus on newly generated data components. Each data component is called *block*. The data structure of a blockchain is a hash chain which is formed by all blocks and is secured using a cryptographic hash function. Consensus algorithms play a key role in constructing blockchains. Blockchains can be constructed on different consensus algorithms, e.g., proof of work (PoW) [48], proof of stake (PoS) [49], [50], etc. PoW-based blockchains are the most widely-used ones in reality. In a PoW-based blockchain, each block typically contains two categories of data fields as illustrated in Fig. 2. The first one is called block header which mainly contains the following items.

- PreBlockHash. It is the hash value of the last block and serves as a pointer to form the chain.
- Time. It is a timestamp and indicates when the block is appended.
- Nonce. It is a solution of PoW puzzle.
- MerkleRoot. It is a root value derived from all transactions in the current block via Merkle hash tree [51].

The second one is called transaction data which contain all transactions in the current block.

Blockchains can be broken down into two categories. The first one is private blockchains (includes consortium blockchains), in which the participants should be authorized by an authority or a set of authorities. The second one is public blockchains, in which anyone can participate in or leave the system without the delegation of any authority. A block can be appended to the blockchain, only if it is accepted by a majority of participants [52], [53].

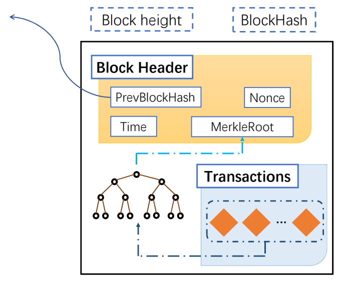


Fig. 2. A simplified graphical block on blockchain

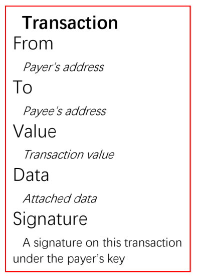


Fig. 3. A graphical transaction in Ethereum

A successful application of public blockchains is blockchain-based currencies, e.g., Bitcoin [48] and Ethereum [22], where the public blockchain serves as a publicly verifiable and distributed ledger to securely record transactions between participants. The ledger is secure against modification of appended blocks. The participants who verify blocks and maintain the blockchain are called *miners*. In this paper, SEPSE is constructed on Ethereum, since Ethereum is more expressive than other public blockchains.

An Ethereum transaction mainly includes five fields as shown in Fig. 3. The first two fields indicate the source account and the destination account. In the “Value” field, the payer who creates the transaction can set how many Ethers she/he transfers. Moreover, the payer is able to set the “attached data” as any binary string she/he generates. In other words, the data value can be considered as the attached data in traditional electronic transactions of banks. We stress that the size of the data is not limited in Ethereum and the data value cannot be modified or forged once the corresponding transaction is confirmed on the blockchain. This allows us to design the blockchain-assisted rate-limiting mechanism. More technical details can be found in [22].

5 The Proposed SEPSE

5.1 Overview of SEPSE

In SEPSE, multiple key servers are employed to jointly help users in encrypting keywords to resist KGA, where the ability of keyword generation is distributed to these key servers, such that any single key server’s misbehavior cannot break the security. Furthermore, a secret is shared among the key servers in a distributed and threshold way, (i.e., (t, n)-threshold distributed secret sharing). As a consequence, sharing the secret does not require a trusted dealer, which frees SEPSE from the single-point-of-failure problem.

SEPSE utilizes an oblivious protocol [54], [55] which is executed between users and the key servers to generate servers-derived keywords. Unlike current schemes [15], [16], SEPSE does not rely on RSA, and is based on a $(t, n)-$threshold blind BLS signature [55], [56], since the BLS signature is much shorter and is faster to compute by the key servers than the RSA counterpart.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

To protect SEPSE from compromising key servers' secret shares, time in SEPSE is divided into fixed and predetermined time-intervals called epochs. At the end of an epoch, each key server is required to renew its secret share. The secret share renewal should not change the secret shared among all key servers, such that for the same keyword, the key servers would produce the same servers-derived keyword. It ensures the functionality of PEKS.

There is still a subtle security issue. Since anyone can become a sender in PEKS, an adversary (e.g., the adversarial cloud server) is able to try different keywords by requesting the corresponding servers-derived keywords. The adversary utilizes different identities of senders to perform this online attack[1], and then obtains all servers-derived keywords. Finally, the adversary can perform off-line KGA to recover the keywords. We call such attacks online keyword guessing attacks (online KGA).

To thwart online KGA, we employ a rate-limiting mechanism: the number of servers-derived keyword requests made by a user in an epoch is limited by key servers. Specifically, a bound ρ is determined with the security parameter, and key servers stop responding after the bound ρ is reached. To improve the communication efficiency on the user side, a gateway is utilized to help users in forwarding the servers-derived keyword queries. The servers-derived keyword request made by a user is first submitted to the gateway who then transfers to all key servers.

We further consider the case that the number of all key servers n is large but the threshold t is small (compared with n). In this case, submitting the request of the servers-derived keyword to n key servers incurs a heavy communication burden for users. We stress that requiring the user to submit the servers-derived keywords to only t key servers is confronted with a malicious gateway or malicious users. Since the malicious gateway may omit the number of servers-derived keyword queries made by an adversary and always allows him to request servers-derived keywords, and a malicious user can select different t key servers for different queries to perform online KGA.

To resist the malicious gateway and malicious users, the key technique used here is the public blockchain, e.g., Ethereum [22], where each servers-derived keyword query made by a user is converted to a transaction on the Ethereum blockchain, and the total number of keyword queries for a user in each epoch can be verified by checking the number that the user creates transactions in the epoch. This yields our blockchain-assisted rate-limiting mechanism which we prove that breaking the security by performing online KGA as hard as forking the Ethereum blockchain. As a result, integrating this mechanism into SEPSE can reduce the communication overhead significantly while achieving the same security guarantee.

5.2 Construction of SEPSE

A sender $\mathcal{S}$, a set of key servers $\{\mathcal{KS}_1, \mathcal{KS}_2, ..., \mathcal{KS}_n\}$, a cloud storage server $\mathcal{C}$, a receiver $\mathcal{R}$, and a manager of the gateway $\mathcal{D}$ are involved in SEPSE.

1. It is easy to detect in reality if the adversary is trying to request servers-derived keywords very frequently within a short period of time in reality.

Setup.

- With the security parameter ℓ, system parameters $\{p, P, G, G_T, e, h, H, H_1, H_2, F, \rho, t\}$ are generated, in which $e : G \times G \to G_T$ is a bilinear map, and P is the generator of G, $h : G \to Z_p$, $H, H_1 : \{0,1\}^* \to G$, and $H_2 : G_T \to \{0,1\}^{\lg p}$ are secure hash functions, $F : Z_p \times \{0,1\}^* \to \{0,1\}^*$ is a pseudorandom function, ρ is a maximum number that each sender and the receiver can require the keyword in each epoch, and t is the threshold number.
- For the receiver $\mathcal{R}$, her/his secret key is α randomly choosen from Z_p^* and the corresponding public key is $Q_{\mathcal{R}} = \alpha P \in G$.
- For $i = \{1, 2, \cdots, n\}$, $\mathcal{KS}_i$ randomly picks $a_{i0} \in Z_p^*$ and a polynomial $f_i(x) \in Z_p$ whose degree is $t-1$, s.t. $f_i(0) = a_{i0}$, where $f_i(x) = a_{i0} + a_{i1}x + ... + a_{i,t-1}x^{t-1}$.
- $\mathcal{KS}_i$ computes $a_{i0}P$ and $a_{i\epsilon}P (\epsilon = \{1, 2, \cdots, t-1\})$, and publishes them. $\mathcal{KS}_i$ computes $f_i(j)$ and sends it to $\mathcal{KS}_j$ for $j = 1, 2, ..., n$; $j \neq i$ via a secure channel.
- $\mathcal{KS}_i$ extracts $f_j(i)$ and checks if by verifying $f_j(i)P = \sum_{\gamma=0}^{t-1} i^\gamma \cdot a_{j\gamma}P$. If the checking passes, it accepts $f_j(i)$.
- $\mathcal{KS}_i$'s secret share is $s_i = \sum_{\gamma=1}^{n} f_\gamma(i)$, its public share is $Q_i = s_i P$. The secret key shared among $\{\mathcal{KS}_1, \mathcal{KS}_2, ..., \mathcal{KS}_n\}$ is $s = \sum_{i=1}^{n} a_{i0}$, and the corresponding public key is $Q = \sum_{i=1}^{n} a_{i0}P$.
- The gateway $\mathcal{D}$ and $\mathcal{KS}_i$ maintain a log file to count the number of keywords requested by $\mathcal{S}$ (denoted by $\rho_{\mathcal{S}}$) and $\mathcal{R}$ (denoted by $\rho_{\mathcal{R}}$). Initially, $\rho_{\mathcal{S}} = 0$ and $\rho_{\mathcal{R}} = 0$.

PEKS. Given a keyword w, $\mathcal{S}$ computes the ciphertext as follows.

- $\mathcal{S}$ randomly chooses $r \in Z_p^*$, computes $w^{'} = rH(w)$.
- $\mathcal{S}$ transfers a service charge and sends $w^{'}$ to $\mathcal{D}$.
- $\mathcal{D}$ checks whether $\rho_{\mathcal{S}} < \rho$. If yes, it transfers the corresponding service charge to $\mathcal{KS}_k$ for each $k \in \{1, 2, \cdots, n\}$, sets $\rho_{\mathcal{S}}{+}{+}$, and informs $\mathcal{S}$; Otherwise, it rejects.
- $\mathcal{S}$ sends $w^{'}$ to $\mathcal{KS}_k$ $(k = 1, 2, \cdots, n)$.
- After receiving the service charge, $\mathcal{KS}_k$ $(k = 1, 2, \cdots, n)$ checks whether $\rho_{\mathcal{S}} < \rho$. If yes, it generates a signature σ_k on $w^{'}$ by using the secret share s_k as $\sigma_k = s_k w^{'}$, sets $\rho_{\mathcal{S}} + +$, and sends σ_k to $\mathcal{S}$; Otherwise, it aborts.
- $\mathcal{S}$ verifies σ_k by checking

$$e(\sigma_k, P) \overset{?}{=} e(w^{'}, Q_k).$$

  If the checking fails, $\mathcal{S}$ rejects σ_k; Otherwise, $\mathcal{S}$ stores σ_k locally.
- After receiving t valid signatures (these t signatures are denoted by $\{\sigma_{i_1}, \sigma_{i_2}, \cdots, \sigma_{i_t}\}$ and their indexes form a set $T = \{i_1, i_2, \cdots, i_t\}$), $\mathcal{S}$ computes

$$\sigma_w = r^{-1} \sum_{k=i_1}^{i_t} \omega_k \sigma_k, \qquad (1)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

where $\omega_k = \prod_{\substack{i_1 \le j \le i_t \\ j \ne k, j \in T}} \frac{j}{j-k}$. $\mathcal{S}$ verifies the correctness of σ_w by checking

$$e(\sigma_w, P) \overset{?}{=} e(H(w), Q).$$

- $\mathcal{S}$ computes $sd_w = F(h(\sigma_w), w)$ as the servers-derived keyword of w, randomly chooses $\chi \in Z_p^*$, computes $\tau = e(H_1(sd_w), \chi Q_{\mathcal{R}})$ and $C_{sd_w} = (\chi P, H_2(\tau))$. Finally, $\mathcal{S}$ sends C_{sd_w} to $\mathcal{C}$.

Trapdoor. Given a keyword w, the trapdoor td_w used to retrieve data is generated by $\mathcal{R}$ as follows.

- $\mathcal{R}$ interacts with the key servers to generate the servers-derived keyword of w (i.e., sd_w), this process is the same as the one performed by $\mathcal{S}$ in **PEKS**, where $\mathcal{R}$ plays the role of $\mathcal{U}$.
- $\mathcal{R}$ computes $td_w = \alpha H_1(sd_w)$.

Finally, $\mathcal{R}$ sends td_w to $\mathcal{C}$.

Test. $\mathcal{C}$ takes as input $C_{sd_w} = (A, B) = (\chi P, H_2(\tau))$ and td_w, and checks $H_2(e(td_w, A)) \overset{?}{=} B$. If the equation holds, $\mathcal{C}$ outputs *True*; Otherwise it outputs *False*.

KeyRenew. For each key server $\mathcal{KS}_i$ ($i \in [1, n]$), it renews its secret share as follows.

- $\mathcal{KS}_i$ randomly selects a polynomial $g_i(x)$ over Z_p with degree at most $t-1$ s.t. $g_i(0) = 0$, in which $g_i(x) = b_{i1}x + b_{i2}x^2 + ... + b_{i,t-1}x^{t-1}$.
- $\mathcal{KS}_i$ computes $b_{i\epsilon}P, \epsilon = \{1, 2, ..., t-1\}$ and publishes it. $\mathcal{KS}_i$ sends $g_i(j) \mod p$ to $\mathcal{KS}_j$ for $j = 1, 2, ..., n; j \ne i$ via a secure channel.
- $\mathcal{KS}_i$ extracts $g_j(i)$ and checks $g_j(i)P \overset{?}{=} \sum_{\gamma=1}^{t-1} i^\gamma b_{j\gamma}P$. If the checking succeeds, $\mathcal{KS}_i$ accepts $g_j(i)$.
- $\mathcal{KS}_i$ computes a new secret share $s_i^{'}$ as

$$s_i^{'} = s_i + \sum_{j=1}^{n} g_j(i).$$

Note that the secret s has the form $s = \sum_{i=1}^n a_{i,0} = \sum_{i=1}^n f_i(0)$. Assume the renewed secret distributed to all key servers is $s^{'}$, it has the form $s^{'} = \sum_{i=1}^n f_i^{'}(0)$. Since $f_i^{'}(x) = f_i(x) + g_i(x)$, we have $s^{'} = \sum_{i=1}^n f_i^{'}(0) = \sum_{i=1}^n f_i(0) + g_i(0)$. Because $g_i(0) = 0$, we further have $s^{'} = \sum_{i=1}^n f_i(0) = s$. Therefore, the renewal of secret shares would not change the secret s shared among all key servers. This ensures the proactive security of SEPSE: an adversary who breaks into multiple key servers cannot break the confidentiality of keywords by performing KGA.

5.3 Improvement on the Communication Efficiency

With the establishment of SEPSE, the total number of key servers n may be large while the threshold t may be small. The rate-limiting mechanism that requires a servers-derived keyword query to be submitted to *all* key servers can be tedious and inefficient. We assume that there are dozens of key servers and only a couple of them need to assist users in generating servers-derived keywords, e.g., $n = 30, t = 3$, it is more advantageous for both users and the gateway to only submit a servers-derived keyword request to t key servers rather than all of them. However, a malicious sender who compromises the gateway can request different servers-derived keywords from different t key servers. If the sender's request is only submitted to t key servers rather than all of them, in the extreme case, he can obtain up to $\lfloor \frac{n}{t} \rfloor \cdot \rho$ servers-derived keywords in an epoch[2], which significantly increases the success probability of online KGA. Therefore, it is necessary to achieve the synchronization on the number of servers-derived keyword requests made by a user among all key servers without requiring the user to submit each request to all key servers. A straightforward method is to require each key server to broadcast each received request to all others. However, it requires key servers to interact with each other to generate one servers-derived keyword and causes a heavy communication burden for them. Actually, it is very challenging to achieve such the synchronization among all key servers without introducing heavy communication costs when the gateway is compromised by an adversary (i.e., a malicious sender). Therefore, we further propose a blockchain-assisted rate-limiting mechanism and integrate it into SEPSE to resist online KGA with high communication efficiency.

In **Setup**, the system parameters are the same as those in Section 5.2, with one difference. The sender $\mathcal{S}$, the receiver $\mathcal{R}$, and the manager of gateway $\mathcal{D}$ each create their own accounts in Ethereum, where their accounts are denoted by $A_{\mathcal{S}}$, $A_{\mathcal{R}}$, and $A_{\mathcal{D}}$, respectively. Only $\mathcal{D}$ is required to maintain $\rho_{\mathcal{S}}$ and $\rho_{\mathcal{R}}$.

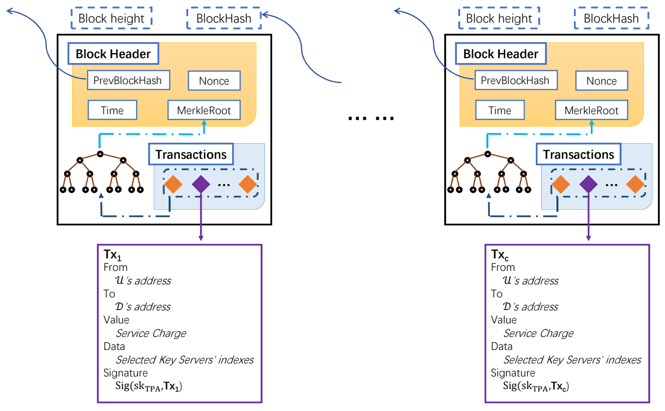


Fig. 4. The transaction of transferring service charge

In **PEKS**, given a keyword w, $\mathcal{S}$ generates the servers-derived keyword sd_w as the same as that in Section 5.2, with the following differences.

- After computing $w^{'} = rH(w)$, $\mathcal{S}$ randomly picks a subset T of set $\{1, \cdots, n\}$, in which $|T| = t$ is the threshold. Support the selected key servers are $\{\mathcal{KS}_{i_1}, \mathcal{KS}_{i_2}, ..., \mathcal{KS}_{i_t}\}$, i.e., $T = \{i_1, i_2, \cdots, i_t\}$. For each $k \in T$, $\mathcal{S}$ computes $\omega_k = \prod_{\substack{i_1 \le j \le i_t \\ j \ne k, j \in T}} \frac{j}{j-k}$.
- $\mathcal{S}$ creates a transaction shown in Fig. 4, where $\mathcal{S}$ plays the role of $\mathcal{U}$ to transfer the service charge to $\mathcal{D}$'s account (i.e., the service charge is transferred from $A_{\mathcal{S}}$ to $A_{\mathcal{D}}$), and the selected key servers' indexes are attached as the transaction information.

2. $\lfloor \frac{n}{t} \rfloor$ denotes the largest integer smaller than $\frac{n}{t}$.

- $\mathcal{D}$ checks whether $\rho_{\mathcal{S}} < \rho$, if yes, $\mathcal{D}$ transfers the corresponding service charge to $\mathcal{KS}_k$ for each $k \in T$, sets $\rho_{\mathcal{S}} ++$, and informs $\mathcal{S}$; Otherwise, $\mathcal{D}$ aborts.
- $\mathcal{S}$ only sends $w^{'}$ to $\mathcal{KS}_{i_1}, \mathcal{KS}_{i_2}, \cdots, \mathcal{KS}_{i_t}$.
- After receiving the service charge, $\mathcal{KS}_k$ $(k = i_1, i_2, ..., i_t)$ first obtains the account information of $A_{\mathcal{S}}$ from Ethereum blockchain and obtains $\rho_{\mathcal{S}}$ by extracting the number of transactions that $A_{\mathcal{S}}$ creates (transferring service charge from $A_{\mathcal{S}}$ to $A_{\mathcal{D}}$). $\mathcal{KS}_k$ $(k = i_1, i_2, ..., i_t)$ checks whether $\rho_{\mathcal{S}} < \rho$. If the checking fails, it aborts; Otherwise, it generates σ_k on $w^{'}$ as $\sigma_k = s_k w^{'}$, and sends σ_k to $\mathcal{S}$.

In **Trapdoor**, given a keyword w, $\mathcal{R}$ first generates the servers-derived keyword sd_w with the aid of *only* t key servers. This process is the same as the one performed by $\mathcal{S}$ described above, where $\mathcal{R}$ plays the role of $\mathcal{U}$ (i.e., a service charge is transferred from $A_{\mathcal{R}}$ to $A_{\mathcal{D}}$). With sd_w, $\mathcal{R}$ generates the corresponding trapdoor td_w as the same as that in Section 5.2.

Test and **KeyRenew** are the same as those in Section 5.2, for the sake of brevity, we would not repeat them.

Efficiency improvement. As shown before, the blockchain-assisted rate-limiting mechanism allows users to only communicate with t of key servers rather than all of them without sacrificing the security. This is because integrating each servers-derived keyword query into a transaction on the Ethereum blockchain enables each key server to obtain the total number of queries made by each user in one epoch without interacting with the user. We must stress again that the blockchain-assisted rate-limiting mechanism is favorable in the case that n is large but t is small. Since creating a transaction in Ethereum takes slight communication and computation costs, if n is *slight more than* t, the communication efficiency on users cannot be improved significantly.

5.4 Correctness

To prove the correctness of SEPSE, we need prove two parts.

First, for the different choice of key servers, the same keyword has the same servers-derived keyword. This depends on the correctness of Eq. 1. In particular, for the sake of brevity, we assume that t valid signatures generated by t key servers are $\{\sigma_1, \sigma_2, \cdots, \sigma_t\}$ and σ_w has the form

$$\begin{aligned}\sigma_w &= r^{-1}\sum_{i=1}^{t}\omega_i\sigma_i \\ &= r^{-1}(\sum_{i=1}^{t} s_i \prod_{1\le j\le t, j\ne i}\frac{j}{j-i})\cdot w' \qquad (2)\\ &= r^{-1}\cdot r\cdot s\cdot H(w) \qquad (3)\\ &= s\cdot H(w),\end{aligned}$$

where the derivation from Eq. 2 to Eq. 3 is based on Lagrange interpolation.

Second, for the different epoches, the same keyword has the same servers-derived keyword. To prove this, we prove that renewing each key server's secret (i.e., $\{s_1, s_2, ..., s_n\}$ are renewed as $\{s_1^{'}, s_2^{'}, ..., s_n^{'}\}$) would not change servers-derived keyword. Given a keyword w, its servers-derived keyword has the form $sd_w = F(h(\sigma_w), w)$, where $\sigma_w = s\cdot H(w)$, and s is the secret distributed to all key servers. Since the renewal of each key server's secret share would not change s, the correctness of SEPSE is ensured.

5.5 Remark and Further Discussion

Although we adopt the Boneh et al.'s PEKS scheme [11] as the underlying scheme of SEPSE, the mechanism that keywords are derived from multiple key servers is also well compatible with other PEKS schemes, such as [12], [33], to resist off-line KGA with retention the characteristics of underlying schemes. In Section 5.3, the rate-limiting mechanism is built on the Ethereum blockchain, in which the security of the rate-limiting mechanism is based on the public verifiability property of Ethereum: the number of transactions created by a user cannot be modified or forged. It is possible to utilize other public blockchains to construct the rate-limiting mechanism, since public verifiability is an inherent property of any secure public blockchain. However, for a public blockchain system, the more participants in it, the stronger the security guarantee it can provide [3]. As Ethereum is one of the most well-established and widely-used public blockchains in reality, we recommend using the Ethereum blockchain, which balances the tradeoff between security and efficiency.

We also stress that the blockchain-assisted rate-limiting mechanism does not require the key servers, gateway, and users to become a full node in Ethereum, because a light wallet of Ethereum has been issued[3] such that one can create transactions without maintaining the Ethereum blockchain. Moreover, the account information, as well as block content in Ethereum, is being released by multiple sites, platforms, and supernodes of Ethereum, such as Etherscan[4] and Etherchain[5]. It enables the key servers to efficiently extract the account information of senders and the receiver without bearing heavy computation and communication costs.

Note that the key observation of the success of KGA is that keywords are always selected from small space and users usually utilize well-known keywords for searches of data. In other words, the number of keywords that a user utilizes would not be too large, and most of the keywords can be predetermined. As such, a user can request her/his commonly used keywords from key servers when the keywords are determined. Furthermore, as discussed before, the same keyword would yield the same servers-derived keyword. Therefore, servers-derived keywords can be reused subsequently. Therefore, the generation of servers-derived keywords can be considered as a one-time operation.

6 Security Analysis

The security of SEPSE is analyzed from four aspects.

6.1 Malicious Storage Server

The malicious storage server may perform both off-line and online KGA to violate the confidentiality of ciphertext.

Intuitively, a keyword w is first protected under its signature σ_w generated by the key servers, while these key

3. https://github.com/ethereum/mist/releases
4. https://etherscan.io/blocks
5. https://www.etherchain.org/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

servers can not learn any information about w. The sender then encrypts the protected keyword (i.e., sd_w) and the receiver generates the trapdoor td_w based on sd_w. Furthermore, in each epoch the number of keyword queries for each user is limited. Therefore the storage server cannot enumerate all servers-derived keyword candidates to perform KGA without the secret shared among all key servers.

Formally, we follow the security notion of semantic-security against chosen keyword guessing attack (SS-CKGA) [15].

Definition 1. The SS-CKGA game is defined by an interactive game between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$ as follows.

Setup. The environment initializes SEPSE, and sends $pk_{\mathcal{R}}, sk_{\mathcal{R}}, Q$ to $\mathcal{B}$ and sends $pk_{\mathcal{R}}, Q$ to $\mathcal{A}$.

Query-1. $\mathcal{A}$ can adaptively request the trapdoor/ciphertext of any keyword w from $\mathcal{B}$.

Challenge. $\mathcal{A}$ submits two keywords w_0, w_1 to $\mathcal{B}$, where w_0, w_1 have not been requested before. $\mathcal{B}$ randomly selects $b \in \{0, 1\}$ and generates sd_{w_b}, $C_{sd_{w_b}}$, and td_{w_b}. $\mathcal{B}$ sends $C_{sd_{w_b}}$ and td_{w_b} to $\mathcal{A}$.

Query-2. $\mathcal{A}$ continues the request on the trapdoor/ciphertext of any keyword except of w_0 and w_1.

Output. $\mathcal{A}$ outputs a guess $b^{'} \in \{0, 1\}$, if $b^{'} = b$, $\mathcal{A}$ wins.

The advantage that $\mathcal{A}$ wins this game is defined as $\mathsf{Adv}_{\mathcal{A}}^{SS-CKGA}(\ell) = \Pr[b^{'} = b] - 1/2$.

SEPSE is SS-CKGA secure due to the following theorems.

Theorem 1. If a PPT adversary $\mathcal{A}$ can break the SS-CKGA security of SEPSE, there is a PPT adversary $\mathcal{B}$ who can break the security of underlying signature (i.e., the BLS signature [57]).

Proof: This proof is identical to the proof of SS-CKGA security in Section III-C of [15]. Here we would omit it. □

Theorem 2. Assuming the Shamir's secret sharing scheme [19] is secure, the distributed way to generate the signature of a given keyword in SEPSE would not reduce the SS-CKGA security.

Proof: In SEPSE, the signature of w is generated by multiple key servers, in which the key servers share s using the Shamir's secret sharing scheme, and sign w using s. This is different from the server-aided PEKS. However, from the view of $\mathcal{A}$, there is no difference between the distributed way (i.e., the case of multiple key servers) and the centralized way (i.e., the case of single key server). Since in *Query-1* and *Query-2*, when $\mathcal{A}$ submits a ciphertext or/and trapdoor query to $\mathcal{B}$, $\mathcal{B}$ interacts with the signing oracle, and all following operations are based this oracle's output. In *Challenge*, $\mathcal{B}$ cannot interact with the singing oracle, and randomly chooses an element as the signature of w_b to generate the challenge ciphertext and trapdoor. Actually, all the above processes are transparent to $\mathcal{A}$, and it would not impact the advantage that $\mathcal{A}$ wins the SS-CKGA game. □

6.2 Compromised Key Server(s)

A compromised key server is the strongest adversary that attempts to extract any information on the user's private input (i.e., keyword). Formally, such the security requirement is captured by the security notion of indistinguishability against chosen keyword attack (IND-CKA) [11].

Definition 2. The IND-CKA game is defined as follows.

Setup. With the security parameter ℓ, the challenger generates the secret and public parameter of a key server (without loss of generality, $\mathcal{KS}_1$), that is, s_1 and Q_1, and sends them to the adversary.

Challenge. The adversary sends two keywords w_0 and w_1 to the challenger. The challenger randomly chooses $b \in \{0, 1\}$, and queries the signature σ_1.

Output. Finally, the adversary outputs its guess $b^{'} \in \{0, 1\}$ on b. If $b^{'} = b$, the adversary wins.

The advantage of adversary that wins the IND-CKA game is defined as $\mathsf{Adv}_{\mathcal{A}}^{IND-CKA}(\ell) = \Pr[b^{'} = b] - 1/2$.

SEPSE is IND-CKA secure because of the theorem as follows.

Theorem 3. If there is a PPT adversary $\mathcal{A}$ who can break the IND-CKA security of SEPSE, then there is a PPT adversary $\mathcal{B}$ who can break the blindness of the threshold blind signature scheme [55].

Proof: Without loss of generality, we assume $\mathcal{KS}_1$ as the adversary $\mathcal{A}$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine, and is transparent to the challenger.

In *Setup*, $\mathcal{B}$ receives s_1 and Q_1 from the challenger and forwards them to $\mathcal{A}$. In *Challenge*, $\mathcal{B}$ receives w_0 and w_1 from $\mathcal{A}$, and forwards them to the challenger. The challenger random picks $b \in \{0, 1\}$ and $r \in Z_p$, computes $w_b^{'} = rH(w_b)$, and sends $w_b^{'}$ to $\mathcal{B}$. $\mathcal{B}$ forwards $w_b^{'}$ to $\mathcal{A}$, computes $\sigma_1 = s_1 w_b^{'}$, and sends σ_1 to the challenger. In *Output*, $\mathcal{A}$ sends a guess $b^{'}$ to $\mathcal{B}$, and $\mathcal{B}$ outputs $b^{'}$ as his guess on b.

In the above process, from the view of challenger, $\mathcal{B}$ is an adversary who breaks the blindness of the scheme [55]. From the view of $\mathcal{A}$, $\mathcal{B}$ is the challenger in the IND-CKA game against SEPSE. Furthermore, the above process is indistinguishable from the IND-CKA game. Therefore, the advantage that $\mathcal{B}$ breaks the blindness of the signature [55] is greater than or equal to $\mathsf{Adv}_{\mathcal{A}}^{IND-CKA}(\ell)$. □

In SEPSE, even if some key servers' secrets are leaked, the security of SEPSE is still ensured, since each key server's secret is renewed at the end of each epoch. We prove that an adversary, who collects t (the threshold number of) secret shares from different epochs, cannot reconstruct the secret s shared among all key servers as follows.

For the sake of brevity, we assume the adversary collects t shares of s from t different key servers in two successive epochs. These shares are denoted by $\{s_1, s_2, \cdots, s_\pi, s^{'}_{\pi+1}, s^{'}_{\pi+2}, \cdots, s^{'}_t\}$, where $1 < \pi < t$, $\{s_1, s_2, \cdots, s_\pi\}$ are shares of s in the first epoch and $\{s^{'}_{\pi+1}, s^{'}_{\pi+2}, \cdots, s^{'}_t\}$ are shares of s in the second epoch.

Recall that

$$\begin{aligned} s &= \sum_{i=1}^{t} \omega_i s_i = \sum_{i=1}^{t} \omega_i s_i^{'} = \sum_{i=1}^{t} \Big(\prod_{\substack{1 \leq k \leq t \\ k \neq i}} \frac{k}{k-i} \Big) \Big(\sum_{j=1}^{n} f_j(i) \Big) \\ &= \sum_{i=1}^{t} \Big(\prod_{\substack{1 \leq k \leq t \\ k \neq i}} \frac{k}{k-i} \Big) \Big(\sum_{j=1}^{n} (f_j(i) + g_j(i)) \Big), \end{aligned} \tag{4}$$

where n is the number of all key servers.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

With $\{s_1, s_2, \cdots, s_\pi, s'_{\pi+1}, s'_{\pi+2}, \cdots, s'_t\}$ and the above equations, the adversary can compute

$$\begin{aligned} & \sum_{i=1}^{\pi} \omega_i s_i + \sum_{i=\pi}^{t} \omega_i s'_i && (5) \\ = & \sum_{i=1}^{\pi} \Big(\prod_{\substack{1 \le k \le \pi \\ k \neq i}} \frac{k}{k-i} \Big) \Big(\sum_{j=1}^{n} f_j(i) \Big) \\ + & \sum_{i=\pi}^{t} \Big(\prod_{\substack{\pi \le k \le t \\ k \neq i}} \frac{k}{k-i} \Big) \Big(\sum_{j=1}^{n} f_j(i) + \sum_{j=1}^{n} g_j(i) \Big) && (6) \\ = & \; s + \sum_{i=\pi}^{t} \Big(\prod_{\substack{\pi \le k \le t \\ k \neq i}} \frac{k}{k-i} \Big) \Big(\sum_{j=1}^{n} g_j(i) \Big). && (7) \end{aligned}$$

Therefore, we have

$$s = \sum_{i=1}^{\pi} \omega_i s_i + \sum_{i=\pi}^{t} \omega_i s'_i - \sum_{i=\pi}^{t} \Big(\prod_{\substack{\pi \le k \le t \\ k \neq i}} \frac{k}{k-i} \Big) \Big(\sum_{j=1}^{n} g_j(i) \Big).$$

To recover s, the adversary needs to compute

$$\sum_{i=\pi}^{t} \Big(\prod_{\substack{\pi \le k \le t \\ k \neq i}} \frac{k}{k-i} \Big) \Big(\sum_{j=1}^{n} g_j(i) \Big). \quad (8)$$

Since the adversary cannot compromise n key servers in an epoch, he cannot collect $g_j(i)$ for $j = 1, 2, \cdots, n$. Therefore, he cannot compute Eq. 8 and cannot recover s.

6.3 Malicious Sender

For a malicious sender, he may forge a servers-derived keyword. Once it succeeds, he also can perform off-line KGA. In other words, the servers-derived keyword cannot be generated by anyone without a threshold number of key servers' help. In fact, given a keyword w, its signature generated by the key servers has the form $\sigma_w = sH(w)$. Essentially, it is the BLS signature [57], and is existentially unforgeable. If an adversary can generate the servers-derived keyword without interacting with key servers, the security of the BLS signature is broken. Actually, the attack performed by a malicious sender can be considered as the one performed by the malicious storage server who controls a registered sender, since attacks performed by the malicious sender can also be performed by the malicious storage server.

6.4 Malicious Gateway

The only attack a malicious gateway can launch is to collude with an adversary to stop counting the number of servers-derived keyword queries made by the adversary. This enables the adversary to perform online KGA.

In the basic scheme (SEPSE without blockchain described in Section 5.2), each servers-derived keyword query made by the user would be forwarded to all key servers. In other words, deriving a keyword requires all key servers' participation. Therefore, the gateway cannot contribute to online KGA.

In the SEPSE with blockchain-assisted rate-limiting mechanism described in Section 5.3, deriving a keyword only requires any t key servers' participation. If the gateway does not honestly follow the prescribed scheme, the number of servers-derived keyword queries made by the adversary can be increased significantly, which enables the adversary to break the security of SEPSE by performing online KGA. However, note that each query made by the user is integrated into a transaction in Ethereum, the number that each user queries the key servers with keywords can be extracted by the key servers efficiently. Due to the security of Ethereum (i.e., the security of public blockchain), it is computationally infeasible to cover up or delete existing transactions that have been recorded into the blockchain. Furthermore, it is also computationally infeasible to modify the number of transactions created by the adversary. Hence, the security of SEPSE cannot be broken by the malicious gateway.

A comparison between SEPSE and existing schemes (e.g., the Boneh et al.'s scheme [11], the Bellare et al.'s scheme [12], and the Chen et al.'s scheme [15]) in terms of security is provided in Table 1, where "Y" denotes that the security is achieved, "N" denotes that the security is not achieved, and "$\perp$" denotes the scheme does not focus on the property. From Table 1, we can observe that SEPSE provides a stronger security guarantee than those in [11], [12], [15].

7 Performance Evaluation

The performance of SEPSE is evaluated via experiments. All experiments and implemented by using C language and MIRACL Library, and run on a computer with Window 10 system, an Intel Core 2 i5 CPU and 8GB DDR 3 of RAM. The security level is selected to be 80 bits, it corresponds to the MNT curve[6].

7.1 Communication Overhead

In the **Setup** algorithm, key servers need to share the secret, and in the **KeyRenew** algorithm, key servers need to renew their secret shares. These two processes require each key server to interact with each other. **Setup** only needs to be run once during the life-time of SEPSE and **KeyRenew** only needs to be run once during an epoch. These costs could be amortized over lots of subsequent requests on servers-derived keywords. Here, we do not analyze the communication overhead on each key server side.

In the generation of servers-derived keyword, the blockchain is introduced to thwart online KGA and reduce the communication overhead on the user side. Fig. 5 shows the efficiency improvement on the communication overhead brought by the blockchain.

7.2 Computational Overhead

In Table 2, basic cryptographic operations are estimated. We set the number of all key servers $n = 30$ for simplicity.

Compared with the Boneh et al.'s scheme [11], SEPSE does not cause additional costs on the storage server.

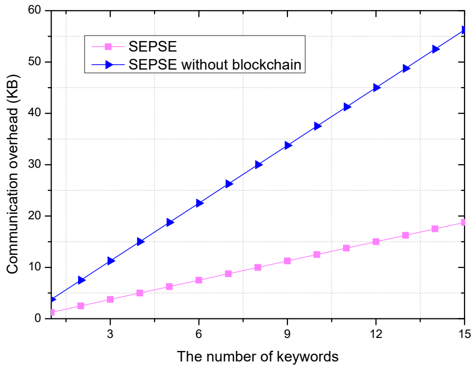
6. Actually, MNT curve is used to construct Type-3 pairings, i.e., $e : G_2 \times G_1 \to G_T$, where G_1 and G_2 are additive groups. Since Type-3 pairings are more efficient than Type-1 pairings when we choose the same security level. In this section, we utilize Type-3 pairing to analyze the performance. This would not affect the feasibility and security of SEPSE.

2168-7161 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

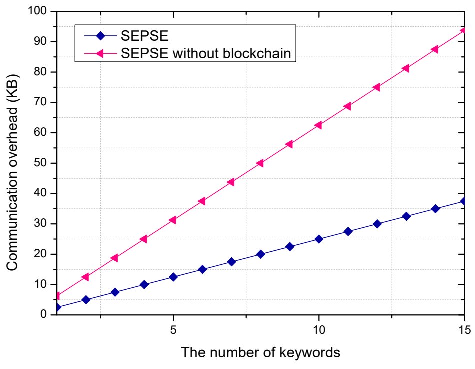
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

TABLE 1
Security comparison

	Scheme in [11]	Scheme in [12]	Scheme in [15]	SEPSE
SS-CKGA	N	N	Y	Y
Resistance against compromised key server(s)	⊥	⊥	N	Y
Resistance against the key compromise	⊥	⊥	N	Y



(a) $t = 10, n = 30$



(b) $t = 20, n = 50$

Fig. 5. Communication overhead

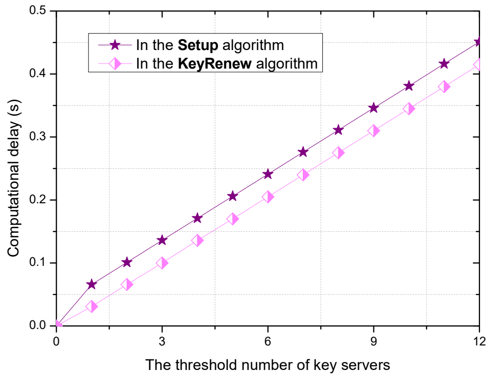
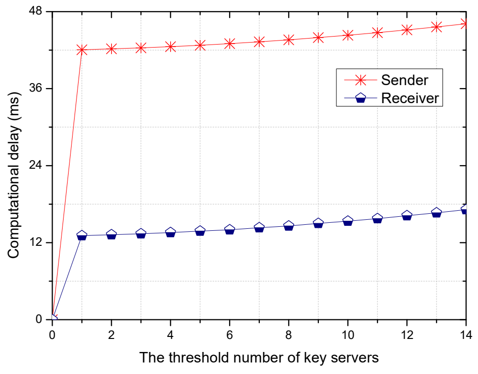
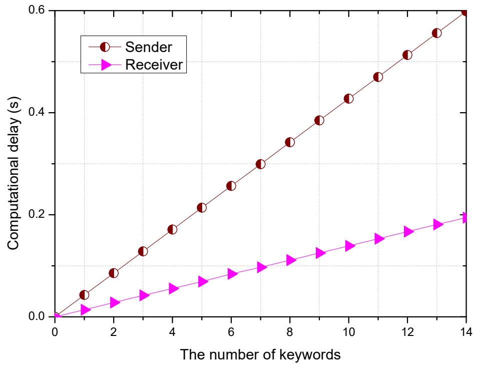


Fig. 6. Computational delay of a key server



(a) Computational delay on different threshold number of key servers



(b) Computational delay on different number of keywords ($t = 5$)

Fig. 7. Computational delay on the user side

TABLE 2
Basic cryptographic operations

Symbol	Operation
$Hash_G$	mapping a value to G
$Mult_G$	multiplication in G
Add_G	addition in G
$Mult_{Z_p}$	multiplication in Z_p
$Pair_{G_T}$	calculating $e(X, Y)$
C_F	calculating $F(\cdot)$
Add_{Z_p}	addition in Z_p
$Hash_{Z_p}$	mapping a value to Z_p
Exp_{Z_p}	exponent operation in Z_p

7.2.1 Computational Overhead on Key Server Side

In the **Setup** algorithm, a key server's computational costs are $(nt+n-1)\cdot Mult_G + t(n-1)\cdot Exp_{Z_p} + t(n-1)\cdot Mult_{Z_p} + n\cdot Add_{Z_p}$. In the **PEKS** algorithm, the key server's computational costs are $Mult_G$. In the **KeyRenew** algorithm, the key server's computational costs are $(nt-1)\cdot Mult_G + (t-1)(n-1)\cdot Exp_{Z_p} + (t-1)(n-1)\cdot Mult_{Z_p} + (n+1)\cdot Add_{Z_p}$. We show the computational delay of the key server in Fig. 6. The computational delay on the key server is within 0.5 seconds if $t = 10$.

7.2.2 Computational Overhead on User Side

In SEPSE, the sender needs to generate a servers-derived keyword and ciphertexts. The computational costs on the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

sender side are $3Pair_{G_T} + 5Mult_G + t \cdot Add_G + 2Hash_G + t(2t-1) \cdot Mult_{Z_p} + t(t-1) \cdot Add_{Z_p} + 2Hash_{Z_p} + C_F$. The receiver needs to generate a servers-derived keyword and a trapdoor. The computational costs on the receiver side are $2Pair_{G_T} + 4Mult_G + t \cdot Add_G + Hash_G + t(2t-1) \cdot Mult_{Z_p} + t(t-1) \cdot Add_{Z_p} + Hash_{Z_p} + C_F$. Fig. 7 shows the computational delay on the sender and receiver sides of SEPSE.

7.3 Simulations on the Ethereum Blockchain

We conduct transactions over Ethereum to demonstrate the practicality of the blockchain-assisted rate-limiting mechanism. In a blockchain system, a block and its transactions are deemed to be confirmed if φ successive blocks are mined following it. In Ethereum, we recommend $\varphi \geqslant 12$. As of Feb. 2019, the time to confirm a transaction in Ethereum is about 3.3 minutes.

We also measure the monetary costs of the blockchain-assisted rate-limiting mechanism. The mechanism does not rely on smart contracts. Hence, the only monetary costs are transaction fee in Ethereum. As of Feb. 2019, conducting a transaction averagely takes within 5 US cents[7]. This can be acceptable in respect of the value of the data protected by SEPSE.

As discussed before, the generation of servers-derived keywords is a one-time operation. Both the senders and receiver can reuse the servers-derived keywords. Therefore, it only requires the user to conduct one transaction in Ethereum for a keyword. The costs to generate servers-derived keywords can be amortized over plenty of future encryption or trapdoor generation operations. Recall the communication improvement (shown in Fig. 5) brought by the blockchain-assisted rate-limiting mechanism in the case that n (i.e., the number of key servers) is large but t (i.e., the threshold) is small, the latency and monetary costs to conduct transactions for rate-limiting can be acceptable in reality.

According to the performance evaluation, we can observe that communication costs between the key server and the user in the Chen et al.'s scheme [15] is constant and is less than that in SEPSE, due to the fact that SEPSE employs multiple key servers to address the single-point-of-failure problem. Furthermore, compared with the Boneh et al.'s scheme [11], users need to generate servers-derived keywords, which brings slight computational overhead. However, the security is surely enhanced at the expensive of this slight computational overhead.

8 Conclusion

In this paper, a secure and efficient PEKS scheme called SEPSE has been presented to resist off-line KGA, where multiple key servers are employed to help in encrypting keywords so as to free SEPSE from the single-point-of-failure problem. SEPSE supports key renewal on each key server, where key servers' secret shares are periodically renewed so as to thwart the key compromise. SEPSE can thwart online KGA by utilizing the proposed blockchain-assisted rate-limiting mechanism, where the number of servers-derived keyword requests for each user is limited in each epoch. We have proved the security of SEPSE and conducted the efficiency evaluation. It has been demonstrated that SEPSE is efficient in respect of communication and computation costs. For the future work, we will study the potentials for enhancing the security, efficiency, and functionality of data outsourcing systems.

7. https://www.coindesk.com/price/ethereum

Acknowledgement

This work is supported by in part by the National Key R&D Program of China under Grant 2017YFB0802000 and Grant 2017YFB0802300, in part by the National Natural Science Foundation of China under Grants 61872060 and 61370203, and in part by the China Scholarship Council. We would like to thank the editor and anonymous reviewers for their valuable comments. The first author would like to thank the colleagues from BBCR lab, University of Waterloo, for their valuable discussions.

References

[1] X. Liu, R. Deng, K. R. Choo, and Y. Yang, "Privacy-preserving outsourced support vector machine design for secure drug discovery," *IEEE Trans. Cloud Computing*, accepted 2018, to appear, doi. 10.1109/TCC.2018.2799219.

[2] A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage," *IEEE Trans. Cloud Computing*, accepted 2018, to appear, doi: 10.1109/TCC.2018.2851256.

[3] Y. Zhang, C. Xu, X. Lin, and X. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Transactions on Cloud Computing*, accepted 2019, to appear, doi: 10.1109/TCC.2019.2908400.

[4] N. Borenstein and J. Blake, "Cloud computing standards: Where's the beef?" *IEEE Internet Computing*, vol. 15, no. 3, pp. 74–78, 2011.

[5] Y. Zhang, X. Lin, and C. Xu, "Blockchain-based secure data provenance for cloud storage," in *Proc. ICICS*, 2018, pp. 3–19.

[6] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Trans. Information Forensics and Security*, vol. 14, no. 4, pp. 870–885, 2019.

[7] Y. Yang, X. Liu, X. Zheng, C. Rong, and W. Guo, "Efficient traceable authorization search system for secure cloud storage," *IEEE Trans. Cloud Computing*, accepted 2018, to appear, doi. 10.1109/TCC.2018.2820714.

[8] H. Ren, H. Li, Y. Dai, K. Yang, and X. Lin, "Querying in internet of things with privacy preserving: Challenges, solutions and opportunities," *IEEE Network*, vol. 32, no. 6, pp. 144–151, 2018.

[9] H. Yang, X. Wang, C. Yang, X. Cong, and Y. Zhang, "Securing content-centric networks with content-based encryption," *Journal of Network and Computer Applications*, vol. 128, pp. 21–32, 2019.

[10] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–51, 2014, article. 18.

[11] D. Boneh, G. Crescemzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYPT*, 2004, pp. 506–522.

[12] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. of CRYPTO*, 2007, pp. 535–552.

[13] X. Zhang, C. Xu, H. Wang, Y. Zhang, and S. Wang, "FS-PEKS: Lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial internet of things," *IEEE Trans. Dependable and Secure Computing*, accepted 2019, to appear, doi: 10.1109/TDSC.2019.2914117.

[14] J. Byun, H. Rhee, H. Park, and D. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *Proc. of SDM*, 2006, pp. 75–83.

[15] R. Chen, Y. Mu, G. Yang, F. Guo, X. Huang, X. Wang, and Y. Wang, "Server-aided public key encryption with keyword search," *IEEE Trans. Information Forensics and Security*, vol. 11, no. 12, pp. 2833–2842, 2016.

2168-7161 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

[16] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage," in *Proc. of USENIX Security Symposium*, 2013, pp. 179–194.

[17] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. Shen, "Providing task allocation and secure deduplication for mobile crowdsensing via fog computing," *IEEE Trans. Dependable and Secure Computing*, to appear, doi: 10.1109/TDSC.2018.2791432.

[18] Y. Zhang, C. Xu, H. Li, K. Yang, J. Zhou, and X. Lin, "HealthDep: An efficient and secure deduplication scheme for cloud-assisted ehealth systems," *IEEE Trans. Industrial Informatics*, vol. 14, no. 9, pp. 4101–4112, 2018.

[19] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[20] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Proc. of CRYPTO*, 1995, pp. 339–352.

[21] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack," *IEEE Trans. Computers*, vol. 62, no. 11, pp. 2266–2277, 2013.

[22] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[23] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE S&P*, 2000, pp. 44–55.

[24] Y. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS*, vol. 5, 2005, pp. 442–455.

[25] E. Goh, "Secure indexes," Cryptology ePrint Archive, report 2003/216, 2003.

[26] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. of ACM CCS*, 2006, pp. 79–88.

[27] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. of ACM CCS*, 2012, pp. 965–976.

[28] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proc. of ACM CCS*, 2014, pp. 310–320.

[29] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. of CRYPTO*, 2013, pp. 353–373.

[30] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. Shen, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Trans. Dependable and Secure Computing*, vol. 13, no. 3, pp. 312–325, 2016.

[31] C. Chen, X. Zhu, P. Shen, J. Hu, S. Guo, Z. Tari, and A. Y. Zomaya, "An efficient privacy-preserving ranked keyword search method," *IEEE Trans. Parallel and Distributed Systems*, vol. 27, no. 4, pp. 951–963, 2016.

[32] H. Li, Y. Yang, Y. Dai, J. Bai, S. Yu, and Y. Xiang, "Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data," *IEEE Trans. Cloud Computing*, accepted 2017, to appear, doi. 10.1109/TCC.2017.2769645.

[33] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions," in *Proc. of CRYPTO*, 2005, pp. 205–222.

[34] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Trapdoor security in a searchable public-key encryption scheme with a designated tester," *Journal of Systems and Software*, vol. 83, no. 5, pp. 763–771, 2010.

[35] Y. Yu, J. Ni, H. Yang, Y. Mu, and W. Susilo, "Efficient public key encryption with revocable keyword search," *Security and Communication Networks*, vol. 7, no. 2, pp. 466–472, 2014.

[36] K. Yang, K. Zhang, X. Jia, M. A. Hasan, and X. Shen, "Privacy-preserving attribute-keyword based data publish-subscribe service on cloud platforms," *Information Sciences*, vol. 387, pp. 116–131, 2017.

[37] X. Zhang, Y. Tang, H. Wang, C. Xu, Y. Miao, and H. Cheng, "Lattice-based proxy-oriented identity-based encryption with keyword search for cloud storage," *Information Sciences*, accepted 2019, to appear, doi: https://doi.org/10.1016/j.ins.2019.04.051.

[38] Y. Chen, "Speks: Secure server-designation public key encryption with keyword search against keyword guessing attacks," *The Computer Journal*, vol. 58, no. 4, pp. 922–933, 2014.

[39] J. Ni, Y. Yu, Q. Xia, and L. Niu, "Cryptanalysis of two searchable public key encryption schemes with a designated tester," *Journal of Information & Computational Science*, vol. 9, no. 16, pp. 4819–4825, 2012.

[40] L. Sun, C. Xu, M. Zhang, K. Chen, and H. Li, "Secure searchable public key encryption against insider keyword guessing attacks from indistinguishability obfuscation," *Science China Information Sciences*, vol. 61, no. 3, p. 038106, 2018.

[41] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," *SIAM Journal on Computing*, vol. 45, no. 3, pp. 882–929, 2016.

[42] Y. Zhang, C. Xu, X. Liang, H. Li, Y. Mu, and X. Zhang, "Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 3, pp. 676–688, 2017.

[43] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE S&P*, 2016, pp. 839–858.

[44] R. Goyal and V. Goyal, "Overcoming cryptographic impossibility results using blockchains," in *Proc. TCC*, 2017, pp. 529–561.

[45] C. Xu, K. Wang, P. Li, S. Guo, J. Luo, B. Ye, and M. Guo, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE Trans. Parallel and Distributed Systems*, vol. 30, no. 4, pp. 870–882, 2019.

[46] C. Xu, K. Wang, G. Xu, P. Li, S. Guo, and J. Luo, "Making big data open in collaborative edges: A blockchain-based framework with reduced resource requirements," in *Proc. IEEE ICC*, 2018, pp. 1–6.

[47] H. Li, K. Wang, T. Miyazaki, C. Xu, S. Guo, and Y. Sun, "Trust-enhanced content delivery in blockchain-based information-centric networking," *IEEE Network*, accepted 2019, to appear, doi: 10.1109/MNET.2019.1800299.

[48] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." https://bitcoin.org/bitcoin.pdf.

[49] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. CRYPTO*, 2017, pp. 357–388.

[50] D. Liu, A. Alahmadi, J. Ni, X. Lin, and X. Shen, "Anonymous reputation system for IIoT-enabled retail marketing atop pos blockchain," *IEEE Trans. Industrial Informatics*, accepted 2019, to appear, doi: 10.1109/TII.2019.2898900.

[51] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. of IEEE S&P*, 1980, pp. 122–134.

[52] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J. Liu, Y. Xiang, and R. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel and Distributed Systems*, accepted 2018, to appear, doi: 10.1109/TPDS.2018.2881735.

[53] J. Weng, J. Weng, and W. Luo, "Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive," Cryptology ePrint Archive, report 2018/679, 2018.

[54] D. Chaum, "Blind signatures for untraceable payments," in *Proc. of CRYPTO*, 1983, pp. 199–203.

[55] D. L. Vo, F. Zhang, and K. Kim, "A new threshold blind signature scheme from pairings," in *Proc. of IEICE SCIS*, 2003, pp. 1–6.

[56] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Proc. PKC*, 2003, pp. 31–46.

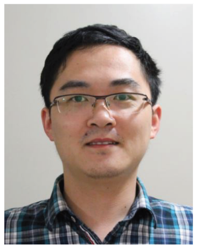
[57] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proc. of ASIACRYPT*, 2001, pp. 514–532.

Yuan Zhang (S'16) received his B.Sc. degree in information security from University of Electronic Science Technology of China (UESTC), Chengdu, China, in 2013. He is currently a Ph.D. candidate in School of Computer Science and Engineering at UESTC, and is a visiting Ph.D. student in BBCR Lab, Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interests are applied cryptography, data security, and blockchain technology. He is a student member of IEEE.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2019.2923222, IEEE Transactions on Cloud Computing

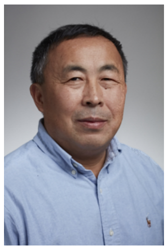
Chunxiang Xu (M'06) received the B.Sc. and M.Sc. degrees in applied mathematics from Xidian University, Xi'an, China, in 1985 and 2004, respectively, and the Ph.D. degree in cryptography from Xidian University in 2004.

She is currently a Professor with the Center for Cyber Security, the School of Computer Science and Engineering, UESTC. Her research interests include information security, cloud computing security, and cryptography. She is a member of IEEE.



Jianbing Ni (M'18) received the Ph.D. degree in Electrical and Computer Engineering from University of Waterloo, Waterloo, Canada, in 2018, and received the B.E. degree and the M.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2011 and 2014, respectively. He is currently a postdoctoral research fellow at the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research interests are applied cryptography and network security, with current focus on cloud computing, smart grid, mobile crowdsensing and Internet of Things.

Hongwei Li (M'12–SM'18) is currently the Head and a Professor at Department of Information Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China. He received the Ph.D. degree from University of Electronic Science and Technology of China in June 2008. He worked as a Postdoctoral Fellow at the University of Waterloo from October 2011 to October 2012 under the supervision of Prof. Sherman Shen. His research interests include network security and applied cryptography. His research is supported by National Science Foundation of China, and Ministry of Science and Technology of China, and Ministry of Industry and Information Technology, and China Unicom. Dr. Li has published more than 80 technical papers. He is the sole author of a book, Enabling Secure and Privacy Preserving Communications in Smart Grids (Springer, 2014). Dr. Li serves as the Associate Editor of IEEE Internet of Things Journal, and Peer-to-Peer Networking and Applications, the Guest Editor of IEEE Network and IEEE Internet of Things Journal. He also serves/served the technical symposium co-chair of ACM TUR-C 2019, IEEE ICCC 2016, IEEE GLOBECOM 2015 and IEEE BigDataService 2015, and many technical program committees for international conferences, such as IEEE INFOCOM, IEEE ICC, IEEE GLOBECOM, IEEE WCNC, IEEE SmartGridComm, BODYNETS and IEEE DASC. He won the Best Paper Award from IEEE MASS 2018 and IEEE HELTHCOM 2015. He is the Senior Member of IEEE and the Distinguished Lecturer of IEEE Vehicular Technology Society.



Xuemin (Sherman) Shen (M'97–SM'02–F'09) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research focuses on resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He is a registered Professional Engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, and a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society.

Dr. Shen received the R.A. Fessenden Award in 2019 from IEEE, Canada, the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Joseph LoCicero Award in 2015 and the Education Award in 2017 from the IEEE Communications Society. He has also received the Excellent Graduate Supervision Award in 2006 and the Outstanding Performance Award 5 times from the University of Waterloo and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He served as the Technical Program Committee Chair/Co-Chair for the IEEE Globecom'16, the IEEE Infocom'14, the IEEE VTC'10 Fall, the IEEE Globecom'07, the Symposia Chair for the IEEE ICC'10, the Tutorial Chair for the IEEE VTC'11 Spring, the Chair for the IEEE Communications Society Technical Committee on Wireless Communications, and P2P Communications and Networking. He is the Editor-in-Chief of the IEEE INTERNET OF THINGS JOURNAL and the Vice President on Publications of the IEEE Communications Society.

2168-7161 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.