IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received September 26, 2017, accepted October 26, 2017, date of publication November 8, 2017, date of current version November 28, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2771301

Fast and Parallel Keyword Search Over Public-Key Ciphertexts for Cloud-Assisted IoT

PENG XU[1,2], (Member, IEEE), XIAOLAN TANG[1], WEI WANG[3], (Member, IEEE), HAI JIN[1], (Senior Member, IEEE), AND LAURENCE T. YANG[3,4], (Senior Member, IEEE)

[1]Services Computing Technology and System Laboratory, Cluster and Grid Computing Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[2]Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen 518057, China
[3]Cyber-Physical-Social Systems Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[4]Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada

Corresponding author: Peng Xu (xupeng@mail.hust.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61472156, in part by the National Program on Key Basic Research Project (973 Program) under Grant 2014CB340600, and in part by the Shenzhen Fundamental Research Program under Grant JCYJ20170413114215614.

ABSTRACT Cloud-assisted Internet of Things (IoT) is a popular system model to merge the advantages of both the cloud and IoT. In this model, IoT collects the real-world data, and the cloud maximizes the value of these data by sharing and analyzing them. Due to the sensitivity of the collected data, maintaining the security of these data is one of the main requirements in practice. Searchable public-key encryption is a fundamental tool to achieve secure delegated keyword search over ciphertexts in the cloud. To accelerate the search performance, Xu *et al.* propose a new concept of searchable public-key ciphertexts with hidden structures (SPCHSs), and it constructs a SPCHS instance to achieve search complexity that is sublinear with the total number of ciphertexts rather than the linear complexity as in the traditional works. However, this paper cannot achieve the parallel keyword search due to its inherent limitations. Clearly, the aforementioned instance is impractical. To address this problem, we propose a new instance of SPCHS to achieve fast and parallel keyword search over public-key ciphertexts. In contrast to the work by Xu *et al.*, a new type of hidden relationship among searchable ciphertexts is constructed by the new instance, where every searchable ciphertext has a hidden relationship with a common and public parameter. Upon receiving a keyword search trapdoor, one can disclose all corresponding relationships in parallel and then find all matching ciphertexts. Hence, the new relationship allows a keyword search task to be performed in parallel. In addition, due to the limited capability of IoT, the new instance achieves a more efficient encryption algorithm to save time and communication cost.

INDEX TERMS Cloud-assisted Internet-of-Things, searchable public-key ciphertexts, hidden relationship, semantic security, parallel search.

I. INTRODUCTION

Internet of Things (IoT) has been a hot word in scientific and technological fields. IoT has experienced remarkable development not only in terms of home and consumer equipment development but also in manufacturing, logistics, mining, oil, utilities and agriculture and in other large assets of industry. Moreover, with the assistance of the cloud, IoT can leverage the unlimited capability of the cloud in terms of both storage and computation to maximize its effectiveness. However, the cloud-assisted IoT model also brings a variety of security and privacy problems. In this model, the IoT devices generally collect a considerable amount of sensitive data and send them to the cloud to be shared and analyzed. Therefore, protecting the security of IoT data in the cloud is a critical requirement.

Searchable public-key encryption (SPKE) [1] is a fundamental tool to achieve secure delegated keyword search over ciphertexts in the cloud. When applying SPKE in the cloud-assisted IoT model, all IoT devices can generate searchable ciphertexts for their data in a public-key setting and upload them to the cloud; then, the owner of these devices can delegate a secure keyword search over these ciphertexts to retrieve

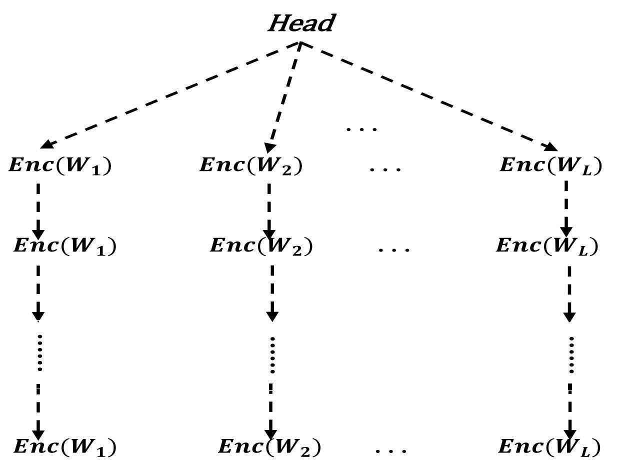

FIGURE 1. The Hidden Relationship Constructed by XW'15 [2]. (The arrows denote the hidden relations. $Enc(W_i)$ denotes the searchable ciphertext of keyword W_i.)

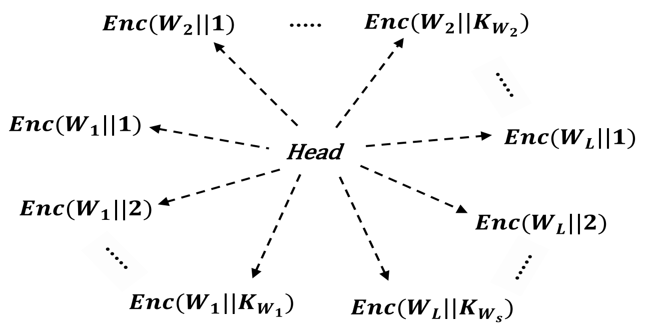

FIGURE 2. Our New Hidden Relationship. (The dashed arrows denote the hidden relations. $Enc(W_i||K_{W_i})$ denotes the searchable ciphertext of keyword W_i and its current counter value K_{W_i}.)

the intended data. However, the early works of SPKE have a high search complexity, which is linear with the total number of ciphertexts. To address this problem, [2] proposes the concept of searchable public-key ciphertexts with hidden structures (SPCHS), and it constructs a SPCHS instance (called XW'15 in this paper) to achieve search complexity that is linear with the number of matching ciphertexts rather than with the number of all ciphertexts.

In XW'15, all keyword-searchable ciphertexts are organized by some hidden relations, as shown in Fig. 1. In XW'15, all ciphertexts of the same keyword construct a hidden chain relationship, and additionally, a hidden relation exists from a common and public parameter *Head* to the first ciphertext of each chain. Upon receiving a keyword search trapdoor, the server seeks out the first matching ciphertext via the corresponding relation from the *Head*. Then, another relation can be disclosed via the found ciphertext, and it guides the server to seek out the next matching ciphertext. By proceeding in this way, all matching ciphertexts can be found.

Although XW'15 greatly reduces the search complexity compared with the traditional works, it also has some significant limitations in practice. Due to the characteristic of the hidden chain relationship, all matching ciphertexts can only be found in series. Thus, it is impractical to employ XW'15 in the scenario of big data. Hence, we are interested in proposing a new instance of SPCHS to achieve fast and parallel keyword search.

A. OUR IDEA AND ITS CLASSICAL APPLICATION

As mentioned above, the hidden chain relationship among searchable ciphertexts is the main obstacle to achieve the parallel search. Hence, we attempt to construct a new type of hidden relationship such that the new one allows the server to disclose all corresponding relations in parallel when receiving a keyword search trapdoor. Fig. 2 shows an example of the new hidden relationship. In this relationship, each keyword has a private counter to record the number of generated ciphertexts of the keyword. Each ciphertext is generated by taking a keyword and its current counter value as inputs. Through this method, all ciphertexts will have a hidden relation from a common and public parameter *Head*. Upon receiving the keyword search trapdoor of a keyword, one can independently disclose all corresponding relations by iterating all possible values of the keyword's counter, where each disclosed relation points to a matching ciphertext. Since the above iteration process can be achieved in parallel, a complete search task can also be achieved in parallel.

When applying our above idea in the cloud-assisted IoT scenario, the owner of IoT devices sets up the public and private keys and stores the public key in all his devices. An IoT device sets up its public parameter *Head* and uploads the parameter to the cloud. To generate the keyword-searchable ciphertext of keyword W for file F, the device initializes the counter as $K_W = 1$ if it is the first time to generate the ciphertext of W, and it takes W and K_W as inputs to generate the ciphertext C_{W,K_W}. Finally, he uploads $C_{W,K_W}||C_F$ to the cloud, where C_F denotes the public-key ciphertext of file F via any traditional public-key encryption schemes, such as RSA and so on. To delegate the keyword search task of keyword W to the cloud, the owner takes his private key and W as inputs and generates a keyword search trapdoor. Upon receiving the trapdoor, the cloud computes a number of indices by iterating all possible counter values in parallel and finds all matching ciphertexts such as C_{W,K_W} by those indices. Finally, the cloud returns the corresponding encrypted files such as C_F to the receiver. Since the cloud can complete a keyword search task in parallel, our new idea enables the time cost of the search to be considerably more practical compared to the performance of previous works.

B. OUR CONTRIBUTIONS

According to our above idea, we construct a new SPCHS instance. Compared with the instance XW'15 proposed in [2], our new instance generates keyword-searchable ciphertexts that have the same hidden relationship, as shown in Fig. 2, and it allows the one with a keyword search trapdoor to search a keyword over the generated ciphertexts in parallel. Hence, it is considerably more practical than XW'15. For example, when applying the new instance in the cloud-assisted IoT scenario, all IoT data are stored as ciphertexts in the cloud to keep their confidentiality. Upon receiving a keyword search task from the data owner, the cloud can find the matching ciphertexts in parallel. In addition, the new instance has a more efficient encryption algorithm to save the time and communication costs of IoT devices in practice.

In addition, the new instance is provably secure based on the computational bilinear Diffie-Hellman (CBDH) assumption [1] in the random oracle (RO) model. Hence, it has the same security as XW'15. In other words, without any keyword search trapdoor, no one knows the semantic information of keywords encrypted by those searchable ciphertexts, and no hidden relationship among those searchable ciphertexts is leaked. With a keyword search trapdoor, only the corresponding hidden relationship is disclosed, and the matching ciphertexts can be found.

Finally, we experimentally test the time cost of our new instance in terms of generating ciphertexts and searching keywords, and we evaluate the communication cost to transfer ciphertexts. The results show that our new instance is more practical than XW'15.

C. ORGANIZATION

Before constructing our new instance, Section II reviews the concepts of SPCHS and its semantic security. Our new instance is proposed in Section III. Section IV applies our new instance in the cloud-assisted IoT model. Section V experimentally shows the advantages of our new instance. Other related works are introduced in Section VI. Section VII concludes this paper.

II. REVIEWING THE CONCEPTS OF SPCHS AND ITS SEMANTIC SECURITY

In this section, we will briefly review the concepts of SPCHS and its semantic security. For more formal details, readers can refer to [2]. The concept of SPCHS consists of five algorithms, which are as follows:

- Algorithm **SystemSetup** takes some parameters, such as the security parameter, to generate a pair of master public and private keys, where the master public key is published, and the master private key must be secretly stored by the one who runs this algorithm, such as a receiver.
- Algorithm **StructureInitialization** takes the master public key as input to initialize a hidden relationship. This relationship consists of a pair of public and private parts. The public part includes the parameter *Head*, which is shown in Fig. 2. The private part includes some secret information, such as the counter values of keywords in our idea. In addition, the private part must be stored by the one who runs this algorithm, such as a sender.
- Algorithm **StructuredEncryption** takes the master public key and the private part of a hidden relationship as inputs to generate a keyword-searchable ciphertext for a keyword. Consequently, the generated ciphertext of the keyword contains the intended hidden relationship with some previously generated ciphertexts. This hidden relationship will accelerate the search performance. In practice, this algorithm is completed by a sender.
- Algorithm **Trapdoor** takes the master private key as input to generate the keyword search trapdoor for a given keyword. This algorithm is completed by the receiver who generates the master private key. In practice, the generated trapdoor allows a server to securely search the keyword over the keyword-searchable ciphertexts generated by algorithm **StructuredEncryption**.
- Algorithm **StructuredSearch** takes a keyword search trapdoor as input to find all matching ciphertexts. Clearly, this algorithm is completed by a server.

In addition to the above algorithms, the concept of SPCHS also defines its consistency. The consistency defines the correctness of SPCHS. It means that with a keyword search trapdoor, only the matching ciphertexts can be found by algorithm **StructuredSearch**.

The semantic security of SPCHS is called the semantic security for both keywords and the hidden relationship under chosen keyword and relationship attacks (SS-CKRA). It models an adaptive attack game on SPCHS, and then it defines that a SPCHS instance is SS-CKRA secure if no one can win the game with a non-negligible advantage.

The adaptive attack game on SPCHS consists of the following five phases:

- The **setup** phase is completed by a challenger who will challenge the capability of an adversary to compromise a SPCHS instance. In this phase, the master public key of SPCHS is generated and published to the adversary.
- The **query 1** phase is launched by the adversary. He will adaptively chose some keywords and hidden relationship, and he will query the corresponding keyword search trapdoors and private parts. The challenger will respond to these queries if their responses cannot allow the adversary to trivially win the game.
- The **challenge** phase allows the adversary to choose two pairs of keyword and hidden relationship as his attack targets. The challenge will generate the challenging keyword-searchable ciphertext for one of the pairs.
- The **query 2** phase is the same as the **query 1** phase.
- The **guess** phase is the final phase. In this phase, the adversary will guess which of the two pairs chosen by the adversary in the **challenge** phase is used to generate the challenging keyword-searchable ciphertext.

If the adversary guesses the correct result, then the adversary wins the above game. Suppose that the probability of the adversary winning the game is $Pr[Win]$. The advantage of the adversary winning the game is defined as $Adv_{\text{SPCHS},\mathcal{A}}^{\text{SS-CKRA}} = Pr[Win] - \frac{1}{2}$.

III. OUR NEW SPCHS INSTANCE

In this section, we will construct our new SPCHS instance and prove its consistency and semantic security. Compared with the previous instance XW'15, our new SPCHS instance has the following main differences:

- The private part of a hidden relationship records the current counter value of each already used keyword. In contrast, XW'15 stores the current pointer of each keyword in the private part of a hidden relationship.
- Every generated ciphertext has a direct relationship with the public part of a hidden relationship. In contrast, XW'15 only allows the first ciphertext of a keyword to have a direct relationship with the public part, and the other ciphertexts of the keyword have the relationship only with their formerly generated one.
- The search to find a matching ciphertext for a keyword is independent of the last found and matching ciphertext. Conversely, XW'15 has the capability to find a matching ciphertext due to the last found and matching ciphertext.

A. CONSTRUCTING OUR INSTANCE

Let $\gamma \xleftarrow{\$} \Re$ indicate randomly selecting an element γ from $\Re$. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative groups of prime order q. Let g be a generator of $\mathbb{G}_1$. A bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is defined as an efficiently computable and non-degenerate function with the following properties:

- Efficient: Given any elements g and $h \in \mathbb{G}_1$, there is a polynomial-time algorithm to compute $\hat{e}(g, h) \in \mathbb{G}_2$.
- Bilinear: For any integers $a, b \in \mathbb{Z}_q^*$, $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$ holds.
- Non-degenerate: If g is a generator of $\mathbb{G}_1$, then $\hat{e}(g, g)$ is a generator of $\mathbb{G}_2$.

Let $\mathbf{BGen}(1^k)$ be an efficient bilinear map generator with a security parameter 1^k as input and probabilistically output $(q, \mathbb{G}_1, \mathbb{G}_2, g, \hat{e})$. Let $\mathcal{W} = \{0, 1\}^*$ be a keyword space. Our new SPCHS instance is constructed as follows:

- Algorithm $\mathbf{SystemSetup}(1^k, \mathcal{W})$: Given a security parameter 1^k and the keyword space $\mathcal{W}$, this algorithm runs $\mathbf{BGen}(1^k)$ to generate parameters $(q, \mathbb{G}_1, \mathbb{G}_2, g, \hat{e})$, selects $s \xleftarrow{\$} \mathbb{Z}_q^*$, sets $p = g^s$, chooses two cryptographic hash functions $\mathbf{H}_1 : \{0, 1\}^* \rightarrow \mathbb{G}_1$ and $\mathbf{H}_2 : \mathbb{G}_2 \rightarrow \{0, 1\}^n$, and outputs a pair of master public and private keys $(\mathbf{PK}, \mathbf{SK})$, where $\mathbf{PK} = (q, \mathbb{G}_1, \mathbb{G}_2, g, \hat{e}, p, \mathbf{H}_1, \mathbf{H}_2)$, $\mathbf{SK} = s$, and $n \in \mathbb{N}$.
- **StructureInitialization**$(\mathbf{PK})$: Given $\mathbf{PK}$, this algorithm selects $u \xleftarrow{\$} \mathbb{Z}_q^*$ and initializes a hidden structure by outputting a pair of public and private parts $(\mathbf{Pub}, \mathbf{Pri})$, where $\mathbf{Pub} = g^u$ and $\mathbf{Pri}$ contains u. Note that $\mathbf{Pri}$ here is a variable list formed as $(u, \{(W, K_W) | W \in \mathcal{W} \text{ and } K_W \in \mathbb{N}\})$, where K_W is a counter value that records the number of generated ciphertexts of keyword W.
- **StructuredEncryption**$(\mathbf{PK}, W, \mathbf{Pri})$: Given $\mathbf{PK}$, a keyword W and the private part of a hidden relationship $\mathbf{Pri}$, this algorithm performs the following steps:

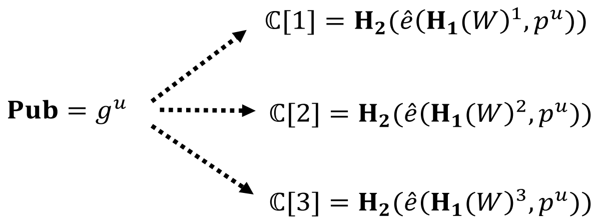
$\mathbf{Pub} = g^u$
$\mathbb{C}[1] = \mathbf{H_2}(\hat{e}(\mathbf{H_1}(W)^1, p^u))$
$\mathbb{C}[2] = \mathbf{H_2}(\hat{e}(\mathbf{H_1}(W)^2, p^u))$
$\mathbb{C}[3] = \mathbf{H_2}(\hat{e}(\mathbf{H_1}(W)^3, p^u))$

FIGURE 3. Example: The Three Generated Ciphertexts of Keyword *W* and Their Hidden Relationship.

  1) Retrieve record (W, K_W) by W from $\mathbf{Pri}$;
  2) If this record does not exist, set $K_W = 1$ and insert (W, K_W) into $\mathbf{Pri}$;
  3) Otherwise, set $K_W = K_W + 1$ and update $\mathbf{Pri}$;
  4) Output the keyword-searchable ciphertext $C = \mathbf{H}_2(\hat{e}(\mathbf{H}_1(W)^{K_W}, p^u))$.
- **Trapdoor**$(\mathbf{SK}, W)$: Given $\mathbf{SK}$ and a keyword W, this algorithm outputs the corresponding keyword search trapdoor $T_W = \mathbf{H}_1(W)^s$.
- **StructuredSearch**$(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_W)$: Given $\mathbf{PK}$, the public part $\mathbf{Pub}$ of a hidden relationship, all generated keyword-searchable ciphertexts $\mathbb{C}$ (let $\mathbb{C}[i]$ denote the i-th ciphertext of $\mathbb{C}$) and a keyword search trapdoor T_W, suppose that there are N threads in parallel to search a keyword; this algorithm runs those threads in parallel and outputs all matching ciphertexts (denoted by $\mathbb{C}'$) when all threads stop. All threads run the same program. Without loss of generality, let us take thread j as an example. Let $t = 0$. Thread j performs the following steps:
  1) Set $M = j + N * t$ and compute $T_{W,M} = T_W^M$;
  2) Compute $C' = \mathbf{H}_2(\hat{e}(\mathbf{Pub}, T_{W,M}))$;
  3) Seek a ciphertext $\mathbb{C}[i] \in \mathbb{C}$ having $\mathbb{C}[i] = C'$;
  4) If this ciphertext exists, add $\mathbb{C}[i]$ into $\mathbb{C}'$, set $t = t + 1$ and go to step 1);
  5) Otherwise, stop;

Fig. 3 and Table 1 show an example of our new SPCHS instance. In this instance, we suppose that a sender has initialized a hidden relationship by algorithm **StructureInitialization** and has generated three ciphertexts for keyword W by algorithm **StructuredEncryption**. Fig. 3 shows those generated ciphertexts and their hidden relationship with the public part. Table 1 shows the details to search keyword W in parallel. Clearly, algorithm **StructuredSearch** can find all matching ciphertexts when completing all parallel threads.

B. PROVING THE CONSISTENCY

Here, we will prove that our new SPCHS instance is correct or consistent. The consistency of SPCHS means that all matching ciphertexts can be found by algorithm **StructuredSearch**. In practice, there are many senders, and each sender will initialize a public part for his hidden relationship. Hence, we only need to prove that our new SPCHS instance is consistent for one sender's hidden relationship.

TABLE 1. Example: searching keyword *W* by 2 threads.

Step	Thread #1	Thread #2
1	$M = 1$	Set $M = 2$
2	$T_{W,1} = \mathbf{H}_1(W)^s$	$T_{W,2} = \mathbf{H}_1(W)^{2s}$
3	$C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^s))$	$C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^{2s}))$
4	$\mathbb{C}[1]$ is found	$\mathbb{C}_2$ is found
5	Add $\mathbb{C}[1]$ into $\mathbb{C}'$	Add $\mathbb{C}[2]$ into $\mathbb{C}'$
6	$M = 3$	$M = 4$
7	$T_{W,3} = \mathbf{H}_1(W)^{3s}$	$T_{W,4} = \mathbf{H}_1(W)^{4s}$
8	$C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^{3s}))$	$C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^{4s}))$
9	$\mathbb{C}[3]$ is found	No ciphertext is found
10	Add $\mathbb{C}[3]$ into $\mathbb{C}'$	Stop
11	$M = 5$	
12	$T_{W,5} = \mathbf{H}_1(W)^{5s}$	
13	$C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^{5s}))$	
14	No ciphertext is found	
15	Stop	

Referring to Table 1, we can find that our new SPCHS instance is consistent in some sense. For example, in step 3 of thread #1, we have $C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^s))$. According to the bilinearity of the bilinear map, we have $C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^s)) = \mathbf{H}_2(\hat{e}(\mathbf{H}_1(W),\ p^u)) = \mathbb{C}[1]$. Hence, the first matching ciphertext of keyword W can be found. Using the same method, all matching ciphertexts of keyword W can be found by the bilinearity of the bilinear map. The formal proof of the consistency of our new SPCHS instance is as follows.

Theorem 1: Suppose that functions $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *are collision free, except with a negligible probability in the security parameter* k*. Our new SPCHS instance is consistent, also except with a negligible probability in the security parameter* k.

Proof: Without loss of generality, it is sufficient to prove that given the keyword search trapdoor $T_W = \mathbf{H}_1(W)^s$ of keyword W and a hidden structure's public part $\mathbf{Pub} = g^u$, algorithm **StructuredSearch**$(\mathbf{PK},\ \mathbf{Pub},\ \mathbb{C},\ T_W)$ can find all ciphertexts of keyword W with the hidden relationship **Pub**.

First, we prove that any ciphertext $\mathbb{C}[i]$ of keyword W with the hidden relationship **Pub** will be found by algorithm **StructuredSearch**$(\mathbf{PK},\ \mathbf{Pub},\ \mathbb{C},\ T_W)$. According to algorithm **StructuredEncryption**, we have $\mathbb{C}[i] = \mathbf{H}_2(\hat{e}(\mathbf{H}_1(W)^{K_W},\ p^u))$. According to algorithm **StructuredSearch**, there is a thread that has its step 1) with the condition $M = K_W$, and the following step 2) has $C' = \mathbf{H}_2(\hat{e}(\mathbf{Pub},\ T_{W,M})) = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^{K_W * s}))$. According to the bilinearity of bilinear map $\hat{e}$, we clearly have $\mathbb{C}[i] = C'$. Hence, ciphertext $\mathbb{C}[i]$ can be found by the thread as one of the outputs of algorithm **StructuredSearch**.

Second, using the same method, we have that all ciphertexts of keyword W with the hidden relationship **Pub** will be found by algorithm **StructuredSearch**$(\mathbf{PK},\ \mathbf{Pub},\ \mathbb{C},\ T_W)$.

Third, we prove that any other ciphertext $\mathbb{C}[j]$ of a different keyword or hidden relationship cannot be found by algorithm **StructuredSearch**$(\mathbf{PK},\ \mathbf{Pub},\ \mathbb{C},\ T_W)$, except with a negligible probability in the security parameter k. Without loss of generality, suppose that ciphertext $\mathbb{C}[j]$ is of keyword W' and with the hidden relationship $\mathbf{Pub} = g^u$, where $W' \neq W$. According to algorithm **StructuredEncryption**, we have $\mathbb{C}[j] = \mathbf{H}_2(\hat{e}(\mathbf{H}_1(W')^{K_{W'}},\ p^u))$. Ciphertext $\mathbb{C}[j]$ can be found by algorithm **StructuredSearch**$(\mathbf{PK},\ \mathbf{Pub},\ \mathbb{C},\ T_W)$ if there is a value of C' that has $C' = \mathbb{C}[j]$ when completing this algorithm. According to this algorithm, it is typical to assume that $C' = \mathbf{H}_2(\hat{e}(g^u,\ \mathbf{H}_1(W)^{M*s}))$. Hence, the condition $C' = \mathbb{C}[j]$ holds if $\mathbf{H}_1(W)^M = \mathbf{H}_1(W')^{K_{W'}}$ is true or function $\mathbf{H}_2$ is not collision free. Since functions $\mathbf{H}_1$ and $\mathbf{H}_2$ are supposed to be collision free, except with a negligible probability in the security parameter k, ciphertext $\mathbb{C}[j]$ cannot be found by **StructuredSearch**$(\mathbf{PK},\ \mathbf{Pub},\ \mathbb{C},\ T_W)$, except with a negligible probability in the security parameter k.

To summarize, all matching ciphertexts can be found by our new SPCHS instance. Additionally, if functions $\mathbf{H}_1$ and $\mathbf{H}_2$ are collision free, then any non-matching ciphertext will not be found. Hence, our new SPCHS instance has consistency. ■

C. PROVING THE SEMANTIC SECURITY

The SS-CKRA security of our new SPCHS instance relies on the CBDH assumption [1]. This means that if the CBDH assumption holds or the corresponding CBDH problem cannot be efficiently solved in practice, our new SPCHS instance is SS-CKRA secure. To prove the SS-CKRA security, we will prove that if there is an adversary that can break the SS-CKRA security, we can leverage the adversary to solve the CBDH problem. Before presenting the proof, we first review the CDBH assumption.

Definition 1 (The CBDH Assumption): The CBDH problem in $\mathbf{BGen}(1^k) = (q,\ \mathbb{G}_1,\ \mathbb{G}_2,\ g,\ \hat{e})$ *is defined as to solve* $\hat{e}(g,\ g)^{abc}$ *for any given* $(g^a,\ g^b,\ g^c)$*. Let* $Adv_{\mathcal{B}}^{CBDH}(1^k)$ *be the probability of a probabilistically polynomial-time (PPT) algorithm* $\mathcal{B}$ *to solve the CBDH problem. We say that the CBDH assumption holds if the advantage* $Adv_{\mathcal{B}}^{CBDH}(1^k)$ *is negligible in the security parameter* k.

The SS-CKRA security of our new SPCHS instance is proven by the following theorem. Since there is no PPT algorithm that can solve the CBDH problem with a non-negligible probability, Theorem 2 means that no PPT adversary can break the SS-CKRA security of our new SPCHS instance in practice.

Theorem 2: Let the hash functions $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *be modeled as the random oracles* $\mathcal{Q}_{\mathbf{H}_1}(\cdot)$ *and* $\mathcal{Q}_{\mathbf{H}_2}(\cdot)$*, respectively. Suppose that there are a total of N hidden relationship in practice and that a PPT adversary* $\mathcal{A}$ *has an advantage of* $Adv_{SPCHS,\mathcal{A}}^{SS\text{-}CKRA}$ *to break our new SPCHS instance in the SS-CKRA game, in which* $\mathcal{A}$ *makes at most* q_1 *queries to oracle* $\mathcal{Q}_{\mathbf{H}_1}(\cdot)$*, at most* q_2 *queries to oracle* $\mathcal{Q}_{\mathbf{H}_2}(\cdot)$*, at most* q_p *queries to oracle* $\mathcal{Q}_{\mathbf{Pri}}$*, at most* q_t *queries to oracle* $\mathcal{Q}_{\mathbf{Trap}}(\cdot)$ *and at most* q_e *queries to oracle* $\mathcal{Q}_{\mathbf{Enc}}(\cdot)$*. Then, there is a PPT algorithm* $\mathcal{B}$ *that solves the CBDH problem in* $\mathbf{BGen}(1^k)$ *with probability*

$$Adv_{\mathcal{B}}^{CBDH} > \frac{256}{(e * q_t * q_p)^4 * (q_2 + q_e) * (q_e + 2)} Adv_{SPCHS,\mathcal{A}}^{SS\text{-}CKRA},$$

where e is the base of the natural logarithm.

Proof: In this proof, we will construct algorithm **B**, which will leverage the adversary $\mathcal{A}$ to solve the CDBH problem in $\mathbf{BGen}(1^k)$. Hence, algorithm $\mathcal{B}$ will simulate and play the SS-CKRA game with adversary $\mathcal{A}$ according to the CDBH problem. This game consists of five phases:

- In the **setup** phase, algorithm $\mathcal{B}$ will simulate the master public key and all public parts of the hidden relationship.
- In the **query 1** and **2** phases, algorithm $\mathcal{B}$ will simulate the responses of the queries from adversary $\mathcal{A}$.
- In the **challenge** phase, algorithm $\mathcal{B}$ will simulate a challenge ciphertext for the challenge targets chosen by adversary $\mathcal{A}$.
- In the **guess** phase, algorithm $\mathcal{B}$ will attempt to solve the CDBH problem according to adversary $\mathcal{A}$'s queries in the **query 1 and 2** phases.

Let $Coin \overset{\sigma}{\leftarrow} \{0, 1\}$ denote the operation that selects $Coin \in \{0, 1\}$ with probability $Pr[Coin = 1] = \sigma$. The specified value of σ will be decided later. Algorithm $\mathcal{B}$ simulates and plays the SS-CKRA game with adversary $\mathcal{A}$ as follows.

- **Setup** phase: Given the keyword space $\mathcal{W}$ and the public parameters $(q, \mathbb{G}_1, \mathbb{G}_2, g, \hat{e}, g^a, g^b, g^c)$ of the CBDH problem, algorithm $\mathcal{B}$ performs the following steps:
  1) Initialize four empty lists $\mathbf{KList} \subseteq \mathcal{W} \times \mathbb{G}_1 \times \mathbb{Z}_q^*$, $\mathbf{SList} \subseteq \mathbb{G}_1 \times \mathbb{Z}_q^* \times \{0, 1\}$, $\mathbf{H_1List} \subseteq \mathcal{W} \times \mathbb{G}_1 \times \mathbb{Z}_q^* \times \{0, 1\}$ and $\mathbf{H_2List} \subseteq \mathbb{G}_2 \times \{0, 1\}^n$, where $n \in \mathbb{N}$;
  2) Set the master public key $\mathbf{PK} = (q, \mathbb{G}_1, \mathbb{G}_2, g, \hat{e}, p = g^a)$;
  3) Initialize N hidden relationship through the following steps for $i \in [1, N]$:
     a) Select $u_i \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ and $Coin_i \overset{\sigma}{\leftarrow} \{0, 1\}$;
     b) If $Coin_i = 1$, compute $\mathbf{Pub}_i = g^{b*u_i}$;
     c) Otherwise, compute $\mathbf{Pub_i} = g^{u_i}$;
  4) Set $\mathbf{PSet} = \{\mathbf{Pub}_i | i \in [1, N]\}$ and $\mathbf{SList} = \{\mathbf{Pub}_i, u_i, \mathbf{Coin_i} | i \in [1, N]\}$;
  5) Send **PK** and **PSet** to adversary $\mathcal{A}$.
- **Query 1** phase: Adversary $\mathcal{A}$ adaptively issues the following queries multiple times.
  - Hash Query $\mathcal{Q}_{\mathbf{H}_1}(W)$: In each query, adversary $\mathcal{A}$ can issue any keyword $W \in \mathcal{W}$ (suppose that each keyword can only be issued one time) to algorithm $\mathcal{B}$; $\mathcal{B}$ responds as follows:
    1) Select $x \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ and $Coin \overset{\sigma}{\leftarrow} \{0, 1\}$;
    2) If $Coin = 1$, add $(W, g^{c*x}, x, Coin)$ into $\mathbf{H_1List}$ and output g^{cx};
    3) Otherwise, add $(W, g^x, x, Coin)$ into $\mathbf{H_1List}$ and output g^x.
  - Hash Query $\mathcal{Q}_{\mathbf{H}_2}(Y)$: When adversary $\mathcal{A}$ issues a value $Y \in \mathbb{G}_2$ (suppose that each value $Y \in \mathbb{G}_2$ can only be issued one time) to this hash query, algorithm $\mathcal{B}$ will select a random value $V \in \{0, 1\}^n$, take the value as its response, and finally add (Y, V) to $\mathbf{H_2List}$.
  - Trapdoor Query $\mathcal{Q}_{\mathbf{Trap}}(W)$: To respond to the issue $W \in \mathcal{W}$ from adversary $\mathcal{A}$, algorithm $\mathcal{B}$ performs the following steps:
    1) If $(W, *, *, *) \notin \mathbf{H_1List}$, query $\mathcal{Q}_{\mathbf{H}_1}(W)$;
    2) Retrieve $(W, X, x, Coin)$ via keyword W from $\mathbf{H_1List}$;
    3) If $Coin = 0$, output g^{a*x};
    4) Otherwise, abort and output $\perp$.

    Note that if $Coin = 0$, algorithm $\mathcal{B}$ can send the correct keyword search trapdoor of keyword W to adversary $\mathcal{A}$.
  - Privacy Query $\mathcal{Q}_{\mathbf{Pri}}(\mathbf{Pub})$: When adversary $\mathcal{A}$ issues the public part $\mathbf{Pub} \in \mathbf{PSet}$ of a hidden relationship to query the corresponding private part, algorithm $\mathcal{B}$ performs the following steps:
    1) Retrieve $(\mathbf{Pub}, u, Coin)$ via **Pub** from **SList**;
    2) If $Coin = 0$, output u;
    3) Otherwise, abort and output $\perp$.
  - Encryption Query $\mathcal{Q}_{\mathbf{Enc}}(W, \mathbf{Pub})$: When adversary $\mathcal{A}$ issues a keyword $W \in \mathcal{W}$ and the public part $\mathbf{Pub} \in \mathbf{PSet}$ to query the corresponding ciphertext, algorithm $\mathcal{B}$ performs the following steps:
    1) If $(W, *, *, *) \notin \mathbf{H_1List}$, query $\mathcal{Q}_{\mathbf{H}_1}(W)$;
    2) Retrieve $(W, X, x, Coin)$ and $(\mathbf{Pub}, u, Coin')$ via W and **Pub** from $\mathbf{H_1List}$ and **SList**, respectively;
    3) Seek $(W, \mathbf{Pub}, K_W)$ via W in **KList**;
    4) If not found, set $K_W = 1$ and insert $(W, \mathbf{Pub}, K_W)$ into **KList**
    5) Otherwise, set $K_W = K_W + 1$ and update **KList**;
    6) Perform the following steps:
       a) If $Coin = 1 \land Coin' = 1$, output ciphertext $C \overset{\$}{\leftarrow} \{0, 1\}^n$;
       b) If $Coin = 0 \land Coin' = 1$, output ciphertext $C = \mathcal{Q}_{\mathbf{H}_2}(\hat{e}(g^{b*x*K_W}, g^{a*u}))$;
       c) If $Coin = 1 \land Coin' = 0$, output ciphertext $C = \mathcal{Q}_{\mathbf{H}_2}(\hat{e}(g^{c*x*K_W}, g^{a*u}))$
       d) If $Coin = 0 \land Coin' = 0$, output ciphertext $C = \mathcal{Q}_{\mathbf{H}_2}(\hat{e}(g^{x*K_W}, g^{a*u}))$

    Note that when $Coin = 1 \land Coin' = 1$ holds, algorithm $\mathcal{B}$ does not respond with the correct ciphertext. However, this exception can be found by adversary $\mathcal{A}$ only when he can help algorithm $\mathcal{B}$ solve the CBDH problem. Hence, this exception is good for our proof. More details will be analyzed later. In addition, for the other three cases, algorithm $\mathcal{B}$ can always respond with the correct ciphertext.
- **Challenge** phase: Adversary $\mathcal{A}$ sends two challenge keyword and hidden relationship pairs $(W_0^*, \mathbf{Pub}_0^*)$ and $(W_1^*, \mathbf{Pub}_1^*)$ to algorithm $\mathcal{B}$, where $\mathcal{B}$ selects $d \overset{\$}{\leftarrow} \{0, 1\}$ and performs the following steps:
  1) Retrieve $(\mathbf{Pub}_0^*, u_0^*, PCoin_0^*)$ and $(\mathbf{Pub}_1^*, u_1^*, PCoin_1^*)$ via $\mathbf{Pub}_0^*$ and $\mathbf{Pub}_1^*$, respectively, from **SList**;

2) If $PCoin_0^* = 0 \vee PCoin_1^* = 0$, then abort and output $\perp$.
3) If $(W_0^*, *, *, *)$ or $(W_0^*, *, *, *) \notin$ **H₁List**, query $\mathcal{Q}_{\mathbf{H}_1}(W_0^*)$ or $\mathcal{Q}_{\mathbf{H}_1}(W_1^*)$;
4) Retrieve $(W_0^*, X_0^*, x_0^*, WCoin_0^*)$ and $(W_1^*, X_1^*, x_1^*, WCoin_1^*)$ via W_0^* and W_1^*, respectively, from **H₁List**;
5) If $WCoin_0^* = 0 \vee WCoin_1^* = 0$, then abort and output $\perp$.
6) Seek $(W_d^*, \mathbf{Pub}_d^*, K_{W_d^*})$ via W_d^* and $\mathbf{Pub}_d^*$ from **KList**;
7) If not found, set $K_{W_d^*} = 1$ and insert $(W_d^*, \mathbf{Pub}, K_{W_d^*})$ into **KList**;
8) Otherwise, update $K_{W_d^*} = K_{W_d^*} + 1$ in **KList**;
9) Randomly choose a challenge ciphertext $C_d^* \xleftarrow{\$} \{0, 1\}^n$ and send it to adversary $\mathcal{A}$;

- **Query 2** phase: This phase is the same as the **query 1** phase. Note that in the **query 1** phase and the **query 2** phase, adversary $\mathcal{A}$ cannot query the keyword trapdoors of both W_0^* and W_1^* and the corresponding private parts of both $\mathbf{Pub}_0^*$ and $\mathbf{Pub}_1^*$.
- **Guess** phase: Adversary $\mathcal{A}$ sends a guess d' to algorithm $\mathcal{B}$. Irrespective of whether the guess is correct, algorithm $\mathcal{B}$ selects a pair (Y, V) randomly from **H₂List** and solves the CBDH problem by outputting $Y^{1/(x_d^* * K_{W_d^*} * u_d^*)}$.

Next, we will compute the probability of algorithm $\mathcal{B}$ to solve the CBDH problem via leveraging the capability of adversary $\mathcal{A}$. The computation consists of three steps: the first step is to compute the probability without considering the event that algorithm $\mathcal{B}$ aborts in the above SS-CKRA game, the second step is to compute the probability of algorithm $\mathcal{B}$ to abort, and the final step is to combine the above two types of probabilities.

Suppose that algorithm $\mathcal{B}$ does not abort in the above SS-CKRA game. Under this assumption, algorithm $\mathcal{B}$ simulates a SS-CKRA game that is indistinguishable from a real SS-CKRA game in the view of adversary $\mathcal{A}$, except the incorrect simulations in both step 6)-a) of encryption query $\mathcal{Q}_{\mathbf{Enc}}$ and step 9) of the **challenge** phase. These two places cannot be correctly simulated since algorithm $\mathcal{B}$ cannot compute the values of the special form $\hat{e}(g, g)^{abc*z}$ by itself, where variable z is assigned according to the encryption query $\mathcal{Q}_{\mathbf{Enc}}$ and the challenge pairs of adversary $\mathcal{A}$. Although these two simulations are incorrect, they cannot be found by adversary $\mathcal{A}$, except that $\mathcal{A}$ queries $\mathcal{Q}_{\mathbf{H}_2}$ with that special issue. Moreover, if the exception never appears, adversary $\mathcal{A}$ has no advantage to win the SS-CKRA game since the challenge ciphertext is independent of the two challenge keyword and hidden relationship pairs.

Let *Query* denote the event that adversary $\mathcal{A}$ queries $\mathcal{Q}_{\mathbf{H}_2}$ with that special issue. According to the definition of the SS-CKRA security and the above analysis, we have

$$\begin{aligned} Adv_{\mathrm{SPCHS},\mathcal{A}}^{\mathrm{SS\text{-}CKRA}} &= Pr[Win] - \frac{1}{2} \\ &= Pr[Win|Query] * Pr[Query] \\ &\quad + Pr[Win|\overline{Query}] * Pr[\overline{Query}] - \frac{1}{2} \\ &= (Pr[Win|Query] - \frac{1}{2}) * Pr[Query] \end{aligned}$$

It implies that $Pr[Query] > Adv_{\mathrm{SPCHS},\mathcal{A}}^{\mathrm{SS\text{-}CKRA}}$.

In the **guess** phase, algorithm $\mathcal{B}$ randomly chooses a pair (Y, V) from **H₂List** and computes $Y^{1/(x_d^* * K_{W_d^*} * u_d^*)}$ as its solution to the CBDH problem. If $Y = \hat{e}(g, g)^{abc*x_d^* K_{W_d^*} u_d^*}$, then it is clear that algorithm $\mathcal{B}$ successfully solves the CBDH problem. Hence, we need to compute the probability of the event that $Y = \hat{e}(g, g)^{abc*x_d^* K_{W_d^*} u_d^*}$ holds. To complete this task, we compute the probability of the event that adversary $\mathcal{A}$ queries $\mathcal{Q}_{\mathbf{H}_2}$ with the issue $\hat{e}(g, g)^{abc*x_d^* K_{W_d^*} u_d^*}$ at first. Let $Query_d$ denote this event. Since this issue belongs to the special issues defined in the event *Query*, the maximum number of special issues is $q_e + 2$. Hence, we have

$$Pr[Query_d] > \frac{1}{q_e + 2} Pr[Query] > \frac{1}{q_e + 2} Adv_{\mathrm{SPCHS},\mathcal{A}}^{\mathrm{SS\text{-}CKRA}}.$$

In addition, since **H₂List** has at most $q_2 + q_e$ records, algorithm $\mathcal{B}$ has a probability of at least $\frac{1}{q_2+q_e}$ to choose a value Y having $Y = \hat{e}(g, g)^{abc*x_d^* K_{W_d^*} u_d^*}$. To summarize, under the assumption that algorithm $\mathcal{B}$ does not abort, it can solve the CBDH problem with probability

$$Adv_{\mathcal{B}}^{\mathrm{CBDH}} > \frac{1}{(q_2 + q_e) * (q_e + 2)} Adv_{\mathrm{SPCHS},\mathcal{A}}^{\mathrm{SS\text{-}CKRA}}.$$

Second, we compute the probability of the event that algorithm $\mathcal{B}$ does not abort. Let $\overline{Abort}$ be the event. According to the above SS-CKRA game, we have $Pr[\overline{Abort}] = (1 - \sigma)^{q_t + q_p} \sigma^4$. Let $\sigma = \frac{4}{q_t * q_p + 4}$. We have $Pr[\overline{Abort}] > (\frac{4}{e * q_t * q_p})^4$, where e is the base of the natural logarithm.

To summarize, if there is a PPT adversary $\mathcal{A}$ that has an advantage of $Adv_{\mathrm{SPCHS},\mathcal{A}}^{\mathrm{SS\text{-}CKRA}}$ to break our new SPCHS instance in the SS-CKRA game, algorithm $\mathcal{B}$ can solve the CBDH problem with a probability of more than

$$Adv_{\mathcal{B}}^{\mathrm{CBDH}} > \frac{256}{(e * q_t * q_p)^4 * (q_2 + q_e) * (q_e + 2)} Adv_{\mathrm{SPCHS},\mathcal{A}}^{\mathrm{SS\text{-}CKRA}},$$

where e is the base of the natural logarithm. ∎

IV. THE APPLICATION OF OUR NEW INSTANCE IN THE CLOUD-ASSISTED IoT MODEL

In this section, we will briefly introduce the application of our new instance in the cloud-assisted IoT model. A classic cloud-assisted IoT model consists of some IoT devices, the cloud and the owner of those devices. All IoT devices collect the real-world data and upload them to the cloud, and the owner can retrieve the intended data from the cloud. When applying our new instance in this model, the resulting system includes the following phases:

- **Deploying Devices.** In this phase, the owner runs algorithm **SystemSetup** to generate a pair of the master public and private keys $(\mathbf{PK}, \mathbf{SK})$, generates a pair of the traditional public and private keys $(\mathbf{PK}', \mathbf{SK}')$ (such

TABLE 2. Configuration of system parameters.

Hardware	Intel Xeon CPU E5-2420 v2 @ 2.20 GHz
OS	CentOS
Program Library	Pairing-Based Cryptography (PBC)
Mathematical Parameters	
Elliptic Curve	$y^2 = x^3 + x$
Base Field	2^{512}
Order	$2^{159} + 2^{127} + 1$
The default unit is decimal	

as the public and private keys of the RSA algorithm), stores (**PK**, **PK**′) in all IoT devices, and finally deploys these devices to the real world.

- **Uploading Data.** In this phase, all IoT devices collect data and upload them to the cloud. For example, an IoT device runs algorithm **StructureInitialization** to generate a pair of its public and private parts (**Pub**, **Pri**), and it uploads the public part **Pub** to the cloud. When the device has collected data F, it extracts some keywords from F, generates searchable ciphertexts using algorithm **StructuredEncryption** for those keywords, encrypts F by **PK**′, and uploads all ciphertexts to the cloud.
- **Retrieving Data.** In this phase, the owner generates a keyword search trapdoor using algorithm **Trapdoor** and sends it to the cloud. Then, the cloud runs algorithm **StructuredSearch** to find all matching keyword-searchable ciphertexts and returns the corresponding ciphertexts of data to the owner. Finally, the owner decrypts data F using **SK**′.

Since all IoT data are encrypted in a public-key setting, the cloud cannot learn their content. In addition, all extracted keywords are encrypted by our new instance. Hence, the corresponding keyword-searchable ciphertexts are semantically secure in the view of the cloud according to Theorem 2.

V. COMPARISONS AND EXPERIMENTS

In this section, we experimentally compare our new instance with XW'15 in terms of the time cost to generate keyword-searchable ciphertexts, the communication cost to transfer ciphertexts, and the time cost to search keywords. Table 2 shows the hardware and operating system for the experiment. We code our new instance and XW'15 using the PBC library [3] with the parameters of the elliptic curve in Table 2.

A. COMPARING SEARCH PERFORMANCE

This experiment contains 1 million ciphertexts in total. We respectively code our new instance and XW'15 to find the matching ciphertexts in an appointed time, and we run these two instances using a CPU with 6 cores. Fig. 4 shows our results. For a given time, Fig. 4 shows the number of the matching ciphertexts that can be found by our new instance and XW'15. For example, given 5 seconds, our new instance can find approximately 21500 matching ciphertexts, and XW'15 can find approximately 4300 matching

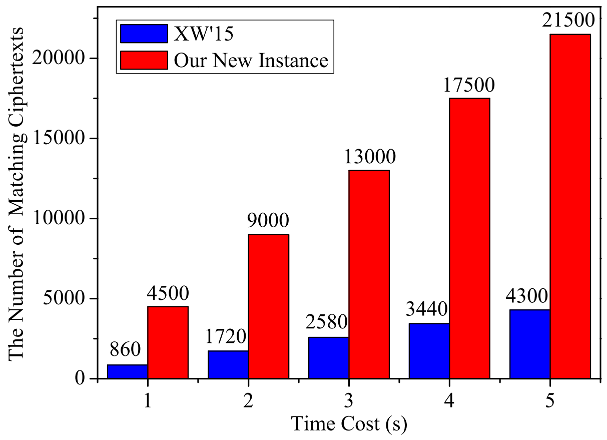

FIGURE 4. Comparing the Search Performance of Our New Instance and XW'15.

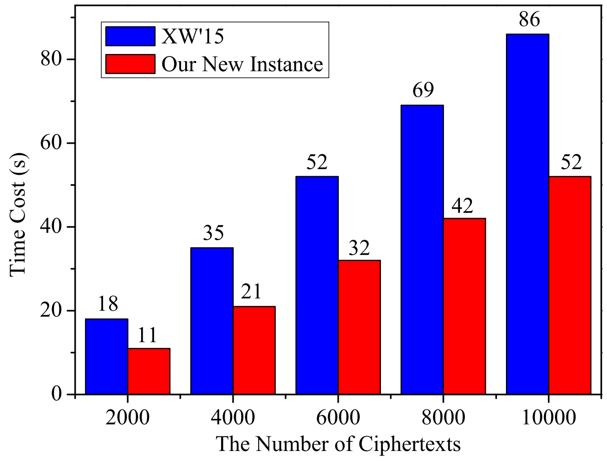

FIGURE 5. Comparing the Encryption Performance of Our New Instance and XW'15.

ciphertexts. Hence, our new instance has search performance that is approximately 5 times greater than that of XW'15. Moreover, this type of high performance also is preserved for different given times by our new instance.

B. COMPARING ENCRYPTION PERFORMANCE

In this experiment, we code our new instance and XW'15 to generate keyword-searchable ciphertexts, and we test the time cost to generate a given number of ciphertexts. Fig. 5 shows our results. For example, to generate 10000 keyword-searchable ciphertexts, XW'15 takes approximately 86 seconds, and our new instance takes approximately 52 seconds. Hence, to generate keyword-searchable ciphertexts, our new instance saves approximately 40% of the time cost compared with XW'15.

C. COMPARING COMMUNICATION COST

This experiment compares the communication cost of our new instance and XW'15 by statistically evaluating the byte size of the generated keyword-searchable ciphertexts. Fig. 6

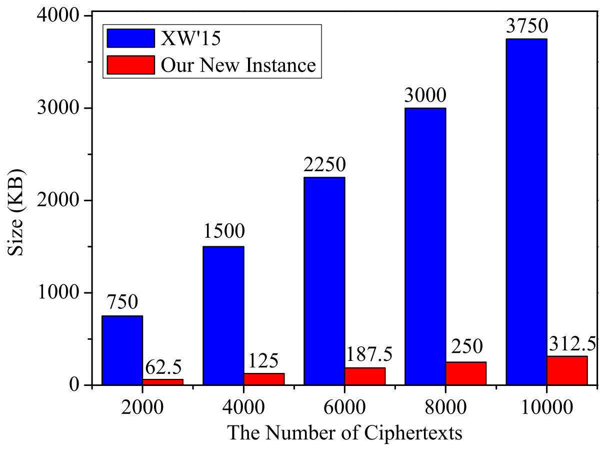

FIGURE 6. Comparing the Communication Cost of Our New Instance and XW'15.

shows our results. For example, it shows that when generating and transferring 10000 keyword-searchable ciphertexts, our new instance takes approximately 312.5 KB and XW'15 takes approximately 3750 KB. Hence, our new instance saves approximately 91% of the communication cost compared with XW'15.

To summarize, the above experimental comparisons clearly show that our new instance is considerably more efficient than XW'15 in terms of the time cost to search a keyword and generate ciphertexts and the communication cost to transfer the generated ciphertexts.

VI. OTHER RELATED WORKS

SPKE was first introduced by Boneh *et al.* [1]. Following this seminal work, Abdalla *et al.* [4] redefined the consistency of SPKE and proposed a general method to construct an instance with enhanced consistency.

Recently, most of the works on SPKE have been dedicated to realizing diverse searches, including continuous search [5]–[10], parallel search [11]–[13], subset search [13], timestamp search [4], [14], similarity search [15], delegation search [16], [17] and fuzzy search [18]. In addition, some works, like [19], dedicate to achieve the SPKE scheme resisting keyword guessing attack. However, all above works can not improve their search performance.

To address this problem, Bellare *et al.* proposed a deterministic encryption scheme [20]. In this scheme, the ciphertexts of the same keyword are also the same. Hence, it achieves logarithmic search complexity. However, it fails to maintain the semantic security in practice. Without losing the semantic security, Xu *et al.* proposed the first SPKE scheme with sub-linear search complexity. To the best of our knowledge, their scheme achieves the best search complexity while maintaining the semantic security.

VII. CONCLUSION

SPKE is a fundamental tool to achieve data security for the cloud-assisted IoT model. However, the existing works are limited in achieving the parallel search, the sub-linear search complexity or the semantic security. Hence, this paper proposes a new instance to achieve more practical performance while retaining the semantic security. When applying our new instance in the cloud-assisted IoT model, the resulting system reduces the time cost of the IoT devices to generate searchable ciphertexts, saves the communication cost of the IoT devices to transfer ciphertexts, and improves the search performance of the cloud to retrieve the intended data. In addition, this system maintains the security of the data even if the cloud is untrusted.

REFERENCES

[1] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 3027, C. Cachin and J. Camenisch, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 506–522.

[2] P. Xu, Q. Wu, W. Wang, W. Susilo, J. Domingo-Ferrer, and H. Jin, "Generating searchable public-key ciphertexts with hidden structures for fast keyword search," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 9, pp. 1993–2006, Sep. 2015.

[3] B. Lynn. *PBC Library*. Accessed: Jun. 2017. [Online]. Available: https://crypto.stanford.edu/pbc/

[4] M. Abdalla *et al.*, "Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions," in *Proc. Annu. Int. Cryptol. Conf.*, vol. 3621. 2005, pp. 205–222.

[5] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," in *Proc. Workshop Intell. Syst. Appl.*, vol. 3325. 2004, pp. 73–86.

[6] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 3089. Berlin, Germany: Springer, 2004, pp. 31–45.

[7] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. Int. Conf. Inf. Commun. Secur.*, vol. 3783. 2005, pp. 414–426.

[8] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Pairing-Based Cryptography—Pairing* (Lecture Notes in Computer Science), vol. 4575. Berlin, Germany: Springer, 2007, pp. 2–22.

[9] E.-K. Ryu and T. Takagi, "Efficient conjunctive keyword-searchable encryption," in *Proc. 21st Int. Conf. Adv. Inf. Netw. Appl. Workshops*, May 2007, pp. 409–414.

[10] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Proc. Int. Conf. ICCSA*, vol. 5072. Perugia, Italy, Jun./Jul. 2008, pp. 1249–1259.

[11] J. Bethencourt, T.-H. H. Chan, A. Perrig, E. Shi, and D. Song, "Anonymous multi-attribute encryption with range query and conditional decryption," School Comput. Sci., Carnegie Mellon Univ. Pittsburgh, Pittsburgh, PA, USA, Tech. Rep. CMU-CS-06-135, 2006.

[12] E. Shi, J. Bethencourt, T.-H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 350–364.

[13] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. 4th TCC*, vol. 4392. 2007, pp. 535–554.

[14] D. Davis, F. Monrose, and M. K. Reiter, "Time-scoped searching of encrypted audit logs," in *Proc. Int. Conf. Inf. Commun. Secur.*, vol. 3269. 2004, pp. 532–545.

[15] D. W. Cheung, N. Mamoulis, W. K. Wong, S. M. Yiu, and Y. Zhang, "Anonymous fuzzy identity-based encryption for similarity search," in *Proc. Int. Symp. Algorithms Comput.*, vol. 6506. 2010, pp. 61–72.

[16] Q. Tang and X. Chen, "Towards asymmetric searchable encryption with message recovery and flexible search authorization," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur.*, 2013, pp. 253–264.

[17] L. Ibraimi, S. Nikova, P. Hartel, and W. Jonker, "Public-key encryption with delegated search," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 6715. Heidelberg, Germany: Springer, 2011, pp. 532–549.

[18] 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. Comput.*, vol. 62, no. 11, pp. 2266–2277, Nov. 2013.

[19] L. Sun, C. Xu, M. Zhang, K. Chen, and H. Li, "Secure searchable public key encryption against insider keyword guessing attacks from indistinguishability obfuscation," *Sci. China Inf. Sci.*, vol. 61, p. 038106, Mar. 2018, doi: 10.1007/s11432-017-9124-0.

[20] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. Annu. Int. Cryptol. Conf.*, vol. 4622. 2007, pp. 535–552.

PENG XU (M'13) received the B.A. degree in computer science from the Wuhan University of Science and Technology, Wuhan, China, in 2003, and the master's and Ph.D. degrees in computer science from the Huazhong University of Science and Technology, Wuhan, in 2006 and 2010, respectively. Since 2010, he has been a Post-Doctor with the Huazhong University of Science and Technology. He was PI in four grants, respectively, from the National Natural Science Foundation of China, the China Postdoctoral Science Foundation, and the Shenzhen Fundamental Research Program. He was also a Key Member in several projects supported by the 973 Program. He has authored over 30 research papers. He is a member of ACM.

XIAOLAN TANG received the B.A. degree in software engineering from Hunan University, Hunan, China, in 2011. She is currently pursuing the master's degree in cyberspace security with the Huazhong University of Science and Technology. Her research interests include cryptography and cloud security.

WEI WANG (M'13) received the B.S. and Ph.D. degrees in electronic and communication engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2006 and 2011, respectively. She is currently a Researcher with the Cyber-Physical-Social Systems Laboratory, Huazhong University of Science and Technology. She has authored over ten papers in international journals and at conferences. Her research interests include cloud security, network coding, and multimedia transmission.

HAI JIN (SM'06) received the Ph.D. degree in computer engineering from HUST in 1994. He was with The University of Hong Kong from 1998 to 2000. He was a Visiting Scholar with the University of Southern California from 1999 to 2000. In 1996, he received a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz, Germany. He received the Excellent Youth Award from the National Science Foundation of China in 2001. He is the Chief Scientist of the National 973 Basic Research Program Project of Virtualization Technology of Computing System. He has co-authored 15 books and published over 400 research papers. He is a member of the ACM.

LAURENCE T. YANG (SM'15) received the B.E. degree in computer science and technology from Tsinghua University, China, and the Ph.D. degree in computer science from the University of Victoria, Canada. He is currently a Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology, China, and the Department of Computer Science, St. Francis Xavier University, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous computing, and big data. His research had been supported by the National Sciences and Engineering Research Council and the Canada Foundation for Innovation.