DEKS
A Secure Cloud-Based Searchable Service Can Make Attackers Pay

Zheng, Yubo; Xu, Peng; Wang, Wei; Chen, Tianyang; Susilo, Willy; Liang, Kaitai; Jin, Hai

DOI
10.1007/978-3-031-17146-8_5

Publication date
2022

Document Version
Final published version

Published in
Computer Security – ESORICS 2022 - 27th European Symposium on Research in Computer Security, Proceedings

DEKS: A Secure Cloud-Based Searchable Service Can Make Attackers Pay

Yubo Zheng[1], Peng Xu[1](✉), Wei Wang[2], Tianyang Chen[1], Willy Susilo[3], Kaitai Liang[4], and Hai Jin[1]

[1] National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China
{zhengyubo,xupeng,chentianyang,hjin}@mail.hust.edu.cn

[2] Cyber-Physical-Social Systems Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
viviawangwei@hust.edu.cn

[3] Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, Australia
wsusilo@uow.edu.au

[4] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands
Kaitai.Liang@tudelft.nl

Abstract. Many practical secure systems have been designed to prevent real-world attacks via maximizing the attacking cost so as to reduce attack intentions. Inspired by this philosophy, we propose a new concept named delay encryption with keyword search (DEKS) to resist the notorious keyword guessing attack (KGA), in the context of secure cloud-based searchable services. Avoiding the use of complex (and unreasonable) assumptions, as compared to existing works, DEKS optionally leverages a *catalyst* that enables one (e.g., a valid data user) to easily execute encryption; without the *catalyst*, any unauthenticated system insiders and outsiders take severe time consumption on encryption. By this, DEKS can overwhelm a KGA attacker in the encryption stage before it obtains any advantage. We leverage the repeated squaring function, which is the core building block of our design, to construct the first DEKS instance. The experimental results show that DEKS is practical in thwarting KGA for both small and large-scale datasets. For example, in the Wikipedia, a KGA attacker averagely takes 7.23 years to break DEKS when the delay parameter $T = 2^{24}$. The parameter T can be flexibly adjusted based on practical needs, and theoretically, its upper bound is infinite.

Keywords: Delay encryption with keyword search · Keyword guessing attack · Security · Privacy

1 Introduction

To date, an increasing number of individuals and companies choose to outsource their personal and business data to remote cloud. Combining with encryption technique, cloud-based searchable service, like CipherCloud [2], bitglass [1], and MVISION Cloud [3], provide privacy-preserving data query and retrieval for cloud users without violating data security. The secure service enables a data searcher to put, say a keyword, into a search query, and after that, the cloud server performs searching and returns matching files without knowing the information of the keyword and the files. This searchability can be captured by the use of *searchable symmetric-key encryption* (SSE) [36], in a single-sender scenario but with complex key distribution for multiple senders, or *searchable public-key encryption* (SPE) [10] (also well known as *public-key encryption with keyword search*, PEKS), in a multi-sender scenario.

In PEKS, each sender generates keyword-searchable ciphertexts with a target receiver's public key. The key is public so that any sender can generate a PEKS ciphertext without pre-communication with the receiver. The receiver can keep off-line, after publishing his public key, till he wants to retrieve the ciphertexts of an expected keyword. Upon getting a keyword search delegation of the receiver, the cloud server finds all matching ciphertexts and then returns them. However, many PEKS schemes suffer from *keyword guessing attack* (KGA).

KGA seriously threatens the security of PEKS. This is due to the intrinsic design of PEKS [27]. Boneh *et al.* [11] concluded that it is a challenge to resist KGA unless the target keyword is sufficiently unpredictable. Figure 1 shows two types of KGA categorized as their interactive pattern, namely, off-line KGA [13] and on-line KGA [41]. Specifically, an off-line KGA attacker guesses the keyword in a unique search request without interaction with the cloud server, for example, the malicious server itself and other attackers launch KGA locally. An attacker can exhaust all possible keywords, generate the corresponding PEKS ciphertexts, and guess the target keyword by testing if a given ciphertext matches the trapdoor. As for on-line KGA, attackers, like malicious senders, need public injecting guessing ciphertexts into the cloud server. They can upload all possible PEKS ciphertexts to the server, eavesdrop the search results to determine which ciphertext corresponds to which trapdoors, and reveal the target keyword.

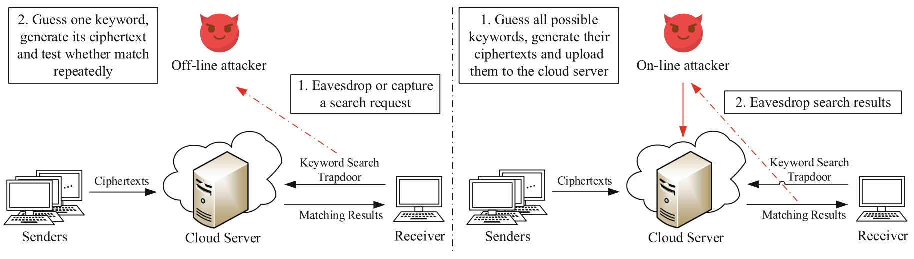
2. Guess one keyword, generate its ciphertext and test whether match repeatedly
Off-line attacker
1. Eavesdrop or capture a search request
Ciphertexts
Keyword Search Trapdoor
Matching Results
Senders
Cloud Server
Receiver
1. Guess all possible keywords, generate their ciphertexts and upload them to the cloud server
On-line attacker
2. Eavesdrop search results
Ciphertexts
Keyword Search Trapdoor
Matching Results
Senders
Cloud Server
Receiver

Fig. 1. Off-line and on-line KGA overview.

1.1 Motivation

Many research works have contributed to resisting KGA. We analyze and categorize them into the following types based on the core techniques they use. Note we here briefly introduce the types and will present a detailed review later.

– Type-I: The schemes, in [21,22,33,34], suppose that server is trusted. The receiver can designate that only trusted server can perform keyword search. Thus, KGA can be easily excluded.
– Type-II: The schemes, in [15,39,42], require that both senders and receiver know the distribution of all possible keywords, and the distribution is static. This facilitates the receiver to constrain the malicious server's search capability only to fuzzy keyword search. Thus, the server cannot guess the accurate target keyword.
– Type-III: The schemes, in [17,38], make good use of two non-colluding servers. The receiver can split a complete search procedure into two phases and delegate the phases to the two non-colluding servers, respectively. Neither of the servers can guess the target keyword independently.
– Type-IV: Those works, in [25,29–32,37], assume that the receiver can always identify if senders are trusted. The receiver may only search over these trusted senders' ciphertexts. Thus, on-line KGA may be easily prevented.
– Type-V: The solutions, in [16,43], requires a trusted server interactively to assist senders in generating ciphertexts (but does not launch on-line KGA). No sender can generate ciphertexts without the server. Thus, the server can limit the capability of malicious senders to launch on-line KGA.

We find that all the aforementioned solutions are valid only if some additional and strong assumptions (or unreasonable constraints) hold. These assumptions are hard to be captured in real-world scenarios, and they even deviate from the original PEKS. Specifically, the assumptions of the Type-I and Type-V solutions are too ideal in practice. The assumption of the Type-IV solutions limits the application of secure searchability, to some degree. The Type-II and Type-III solutions also restrict the scalability of PEKS. Note the comprehensive analyses will be given in Sect. 2. Without using the above assumptions, we make efforts to investigate a brand new design idea for KGA-resistant design.

1.2 A High-Level Overview of Our Idea

Proof of Honesty (PoH) is the main component we use to design a KGA-resistant scheme without any strong assumptions. Inspired by blockchain consensus mechanism, e.g., proof of work (PoW) [26] mechanism, we propose the notion PoH. Each participant pays his computations to prove his honesty. Specifically, the participants build reliability with only computational techniques and then build a secure environment together. PoH is a self-constructing honesty-aware mechanism. It provides two or more parties an approach to build up essential trust in fewer conversations, even non-interactively. Based on PoH, we successfully let attackers pay severe time consumption mandatorily in encryption. It results that a KGA attacker may lose intention or must pay a huge cost for attack. A considerable amount of cost overweighs the attacker's benefits such that KGA may be meaningless. Our approach does not yield complex interactions and expensive computational costs; meanwhile, it does not introduce any extra assumptions.

1.3 Our Contributions

This work aims at constructing a practical KGA-resistant scheme for cloud-based searchable service via the use of you-attack-then-you-pay philosophy. Our contributions are described as follows.

We investigate the existing KGA-resistant works and categorize them into five types according to their technical features. After comprehensively analyzing, we notice that we can design a KGA-resistant scheme in the PoH approach, to open a new vision in this research line. To this end, we propose a new scheme named *delay encryption with keyword search* (DEKS). DEKS allows the receiver to define the time cost of generating a keyword-searchable ciphertext when initializing his public key. With the receiver's public key, no sender can generate a ciphertext with a time cost less than the receiver's requirement, even when the sender is malicious. In other words, DEKS can delay the generation of ciphertext as the request of the receiver. Note such a delay process is non-interactive. Moreover, DEKS allows the receiver to initialize a *catalyst*, which enables the receiver to generate a keyword search trapdoor efficiently (without any delay). In application, the receiver optionally can give the *catalyst* to honest senders (e.g., top contacts of the email system who usually are the receiver's close friends or business partners) and speed up the generation of a ciphertext. In terms of security, the sender cannot leverage the *catalyst* to forge any legal search trapdoor.

We apply repeated squaring and bilinear mapping to realize a DEKS instance. In DEKS, we have four phases, namely, **Setup**, **DEKS**, **Trapdoor**, and **Test**. In **Setup**, the receiver generates his public-and-private keys and a *catalyst*. The public key contains a delay parameter T, which means the mandatory number of executing squaring modulo operations to generate a ciphertext in **DEKS**. However, the honest senders (like top contacts) can fast generate ciphertexts when getting the *catalyst* from the receiver. In **Trapdoor**, the receiver efficiently generates a trapdoor for an expected keyword with both the *catalyst* and the private key. We prove that if the generation of a ciphertext without the *catalyst* does not satisfy the above mandatory time limitation, the trapdoor is ineffective in testing the ciphertext even if they have the same keyword in the **Test** phase. We also prove that DEKS is semantically secure.

Finally, we evaluate DEKS through comprehensive experiments. We take the *Enron* mail data and *Wikipedia* article data to examine the practical security of DEKS for resisting KGA, respectively. DEKS is valid for both small and large-scale datasets. As the delay parameter T increases, an attacker's average consumed time to complete a successful KGA climbs exponentially. For example, for the Wikipedia dataset, when $T = 2^{24}$, it takes 7.23 years for an attacker to complete a successful KGA. And, the average time cost to break a traditional PEKS is less than 51 h. As for the small-scale dataset Enron, DEKS can also give a better performance on resisting KGA by increasing the value of T. At last, we consider the case where attackers are equipped with parallel computing ability. For mitigating the effect of parallel computing, the receiver can slightly increase the value of T. T belongs to a large set and does have an infinite upper bound. It, thus, can be a factor to properly limit the KGA ability.

Discussions. To the best of our knowledge, the repeated squaring function is a unique technique to construct DEKS, which is also implied in [12]. From the technical viewpoint, our DEKS is different from the scheme in [12]. Furthermore, we cannot construct DEKS from their design because they delay the generation of a private key instead of ciphertext. Besides, the following delay techniques cannot be used to construct DEKS either. The password-based key-derivation function is used to mitigate dictionary attacks on password [40]. It delays deriving keys so that the workload of key search attacks significantly increases. But it cannot be used to delay encryption. Both PoW [26] and proof of sequential work (PoSW) [19] can mandatorily assign the time cost of finding a valid proof of a work. However, they require many valid proofs for a given task, and the receiver cannot securely delegate a semi-trust server to verify different proofs. This approach may not be feasible for us. The verifiable delay function (VDF) [9], which is not based on the repeated squaring function, is also ineffective in constructing DEKS. Such a kind of component cannot achieves either the non-interactive delay and verification functions or the *catalyst*-optional delay function, which is similar to the non-interactive VDF in [23], failing to achieve the optional *short-cut* to speed up finding a delay proof. We note that the *short-cut* (or the *catalyst*) is a must for guaranteeing a DEKS is friendly to the receiver and honest senders.

2 KGA Revisited

Table 1 shows the comparisons among the aforementioned five types of existing schemes and our proposed DEKS. We here separately consider insiders and outsiders in off-line and on-line KGA. Outside attackers are eavesdroppers and malicious senders. Semi-trusted servers perform as insider attackers. Our comprehensive analyzations as follows.

Type-I. The Type-I solutions suppose a trusted server performs all keyword search tasks while any other entity cannot. Baek *et al.* [6] first proposed this idea and designed a secure channel-free PEKS, but their work cannot resist KGA. Rhee *et al.* [34] applied this idea to construct the first PEKS instance resisting outside off-line KGA. Some following works, like [21,22,33], also applied this

Table 1. Comparisons among existing and our works.

Type	Off-line KGA-resistant		On-line KGA-resistant		The particular required Assumption	Other disadvantages
	Outside	Inside	Outside	Inside		
Type-I	√	Omitted	√	Omitted	A trusted server to implement keyword searches	Make PEKS useless in practice
Type-II	√	√	√	√	The distribution of keywords is known	Additional cost to filter out the redundant ciphertexts
Type-III	√	√	×	×	Two non-colluding servers to implement keyword searches cooperatively	The communication cost between two servers
Type-IV	√	√	√	√	The receiver can always identify if senders are trusted	The size of a keyword trapdoor is linear with the count of senders
Type-V	√	Omitted	√	Omitted	A trusted server to assist the generation of ciphertexts and trapdoors	The assistant server is needed and could be a performance bottleneck
DEKS	√	√	√	√	No above assumptions	No above disadvantages

idea to resist outside off-line KGA. Generally, the Type-I solutions can be easily extended to resist outside on-line KGA by requesting the trusted server returns the encrypted search results to the receiver, as in [18]. However, supposing a trusted server is too ideal to realize in real-world scenarios. This assumption leads the Type-I solutions to omit inside off-line and on-line KGAs. Moreover, this assumption also makes PEKS meaningless in practice. A trusted server is allowed to search keywords directly over plaintexts while maintaining the confidentiality of keywords by applying a traditional secure communication technique.

Type-II. The Type-II solutions suppose that the distribution of keywords is known and public and allows the server only to implement fuzzy keyword search. Both the server and the malicious sender cannot know the accurate keyword that the receiver requests. Thus, the Type-II solutions can resist off-line and on-line KGAs. Xu *et al.* [39] proposed the first Type-II solution. Some following works utilize the same idea and achieve new properties, such as the works in [15,42]. However, it is difficult to know the distribution of keywords. In practice, different applications have different distributions of keywords. Moreover, the distribution of keywords can be dynamic due to the character of data. In addition, the Type-II solutions return many redundant ciphertexts to the receiver, and the receiver must pay an additional time cost to filter out these ciphertexts.

Type-III. The Type-III solutions suppose two non-colluding servers and split a search task into two parts, and these servers each implement a part. This idea first appeared in work [38]. However, if a single server is malicious, it can still launch the inside off-line KGA. Chen *et al.* overcame this limitation and designed a new Type-III solution [17]. However, it is not easy to guarantee that these two servers do not collude in practice. Moreover, the additional interaction between these two servers increases the communication cost and decreases the search performance.

Type-IV. The Type-IV solutions suppose the receiver can always identify if senders are trusted, and the receiver generates a trapdoor for each trusted sender to realize a keyword search over these senders' ciphertexts. Huang and Li [25] proposed the first and specific Type-IV solution to resist off-line and on-line KGAs. Some subsequent works applied the same idea to resist KGA, such as works in [29–32,37]. However, the Type-IV solutions limit the application of PEKS to some individual scenarios, such as the receiver knows all trusted senders in advance. Indeed, in such a scenario, SSE is more convenient than PEKS for achieving keyword searches over ciphertexts. In addition, the receiver has to generate many trapdoors for searching a keyword, and the count of trapdoors increases linearly with the count of trusted senders.

Type-V. The Type-V solutions suppose a trusted server to assist PEKS in generating its ciphertexts and trapdoors. Chen *et al.* [16] first proposed the Type-V solution. It designed a particular trusted keyword server to compute the hash values of keywords when senders encrypt these keywords or the receiver generates these keywords' trapdoors. Since this trusted server does not help the untrusted server and malicious senders generate the ciphertext, the Type-V solutions can resist the outside off-line and on-line KGAs. In addition, to resist the on-line KGA, the trusted server intentionally limits the speed to generate ciphertexts. Later, Zhang *et al.* [43] used multiple servers to avoid the single-point-of-failure problem. However, the Type-V solutions omit the case that the assistant server could be an inside attacker. Besides, the assistant server could be a performance bottleneck if many senders connect with the server simultaneously.

DEKS. As the first of its type, it avoids the unreasonable assumptions used in the previous types, and it can resist all kinds of KGAs. DEKS allows the receiver to non-interactively constrain the minimum time cost of generating a keyword-searchable ciphertext. Neither the malicious sender nor the untrusted server can break this constraint. Thus, launching a successful KGA will take a considerable time cost such that the cost could be beyond the benefits obtained by a successful attack. Compared with existing works, DEKS provides the best performance in all metrics, except encryption. But DEKS has a negligible impact on encryption throughput, and honest senders who hold a *catalyst* could generate ciphertexts efficiently. DEKS also does not affect testing throughput.

In summary, all previous KGA-resistant works rely on some particular assumptions. These assumptions are either difficult to realize or restrict the application of PEKS to some individual scenarios. In contrast, DEKS adopts the novel idea of delaying the encryption performance to resist KGA without any particular assumption. Any sender can independently achieve the encryption, but no one can avoid the mandatory encryption delay without a *catalyst*, even if the sender is malicious.

3 System Definition and Model

3.1 System Overview

A DEKS system contains three participants: receiver, sender, and server, and they work as follows, shown in Fig. 2.

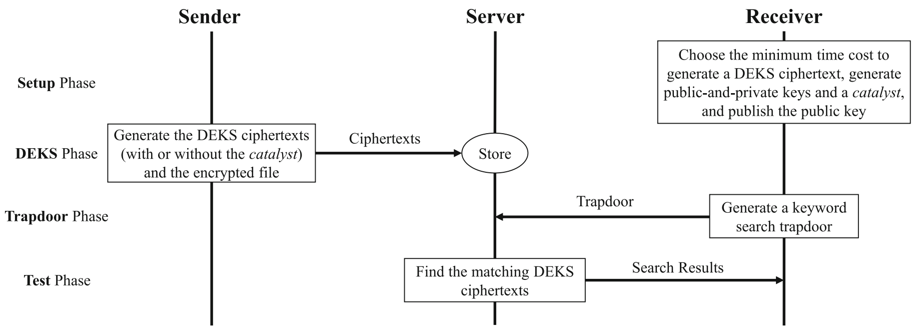
Sender
Server
Receiver
Setup Phase
DEKS Phase
Trapdoor Phase
Test Phase
Choose the minimum time cost to generate a DEKS ciphertext, generate public-and-private keys and a catalyst, and publish the public key
Generate the DEKS ciphertexts (with or without the catalyst) and the encrypted file
Ciphertexts
Store
Trapdoor
Generate a keyword search trapdoor
Find the matching DEKS ciphertexts
Search Results

Fig. 2. The workflow of DEKS.

- *Receiver*. In **Setup**, a receiver chooses the minimum time cost to generate a DEKS ciphertext, generates his public-and-private keys and a *catalyst*, and then publishes his public key. In **Trapdoor**, the receiver chooses an expected keyword and delegates the corresponding keyword trapdoor to the server. The receiver optionally could give the *catalyst* to the honest senders (such as top contacts in the email system) for accelerating encryption.
- *Sender*. In **DEKS**, each sender abstracts keywords from the to-upload file, then gets the corresponding ciphertexts by encrypting these keywords with a receiver's public key, and finally uploads these ciphertexts and the encrypted file to a semi-trusted server. With the *catalyst*, the honest senders can encrypt keywords fast. Other senders without the *catalyst* can only encrypt with a mandatory and slow speed.
- *Server*. In **DEKS**, the server stores the uploaded ciphertexts from senders. In **Test**, once getting a keyword search delegation from a receiver, the server finds and returns all matching ciphertexts to the receiver. Note that the server is honest in implementing a keyword search; however, meanwhile, the server is curious about the keywords that the receiver desires to search.

3.2 Definition of DEKS

The main difference between PEKS and DEKS appears in the definitions of the encryption algorithm and correctness or consistency. The encryption algorithm of DEKS has an optional input parameter. When the optional input parameter is a *catalyst*, the encryption algorithm performs fast; otherwise, the performance is relatively slow. In addition to the search correctness, we define the correctness for delaying the generation of a DEKS ciphertext. The proposed correctness guarantees that everyone must take a fixed minimum time to generate a ciphertext without a *catalyst* - even for the malicious parties.

Definition 1 (DEKS). *A DEKS scheme $\mathcal{DEKS}$ contains four algorithms,* **Setup***,* **DEKS***,* **Trapdoor***, and* **Test***, as shown below.*

- $(PK, SK, \pi) \leftarrow$ **Setup**(k, t)*: Take as inputs a security parameter k and a minimum time constraint t and probabilistically output a pair of public-and-private keys (PK, SK) and a* catalyst *π.*
- $C_w \leftarrow$ **DEKS**(PK, w, opt)*:Take as inputs the public key PK, a keyword w and an optional input parameter opt, probabilistically output a DEKS ciphertext C_w of keyword w with a high speed if the optional input parameter $opt = \pi$; Otherwise, slowly output a probabilistical DEKS ciphertext C_w of keyword w.*
- $T_{w'} \leftarrow$ **Trapdoor**(SK, π, w')*: Take as inputs the private key SK, the* catalyst *π, and a keyword w', output a keyword search trapdoor $T_{w'}$ of keyword w'.*
- *"0" or "1"*$\leftarrow$ **Test**$(T_{w'}, C_w)$*: Take as inputs a keyword search trapdoor $T_{w'}$ and a DEKS ciphertext C_w, output "1" if trapdoor $T_{w'}$ and ciphertext C_w contain the same keyword (namely, the representation $w = w'$ holds); otherwise, output "0".*

In addition, $\mathcal{DEKS}$ must be correct or consistent in the following senses:

- *Search Correctness: For any two keywords w and w', given the corresponding ciphertext $C_w \leftarrow$ **DEKS**(PK, w, opt) and the corresponding trapdoor $T_{w'} \leftarrow$ **Trapdoor**(SK, π, w'), scheme $\mathcal{DEKS}$ always has that algorithm **Test**$(T_{w'}, C_w)$ outputs "1" if $w = w'$ holds and "0" otherwise, except with a negligible probability.*
- *Delay Correctness: For any keyword w, no one can generate a valid DEKS ciphertext with a time cost less than parameter t if it is the first time to encrypt the keyword w without the* catalyst *π. A valid DEKS ciphertext means that the ciphertext can satisfy the above search correctness.*

Remark. In practice, suppose a receiver determines that a sender is honest. The former optionally can help the latter to accelerate ciphertext generation by giving the *catalyst* π securely, so that the time cost may decrease significantly. Beyond that, the sender cannot use the *catalyst* π to harm the scheme, like recovering a valid keyword search trapdoor.

3.3 SS-CKA Security

Semantic security is recognized as a strong enough guarantee in the public-key setting, especially with simple assumptions. The semantic security of DEKS is the same as that of PEKS. It is also defined as the *semantic security under chosen keyword attacks* (SS-CKA). The SS-CKA security is distinct from the security under KGA. The former focuses on if a ciphertext leaks its keyword under the assumption that the keyword's trapdoor is unknown; while the latter captures if a trapdoor leaks its keyword, which implies that the KGA adversary knows the target trapdoor. We briefly review SS-CKA security below. More details can be found in reference [10].

Definition 2 (SS-CKA). *A DEKS scheme $\mathcal{DEKS}$ is SS-CKA secure if any probabilistically polynomial time (PPT) adversary $\mathcal{A}$ wins the following SS-CKA game with only a negligible advantage:*

- ***Setup**: A challenger sets the public-and-private keys and the* catalyst *of $\mathcal{DEKS}$ and publishes the public key (and the* catalyst*) to adversary $\mathcal{A}$.*
- ***Query Phase 1**: Adversary $\mathcal{A}$ adaptively requests the expected keywords' trapdoors.*
- ***Challenge**: Adversary $\mathcal{A}$ chooses two challenge keywords and issues them to the challenger. The challenger randomly picks one of those keywords to generate the challenge ciphertext.*
- ***Query Phase 2**: Same as **Query Phase 1**. Noting that adversary $\mathcal{A}$ cannot request the trapdoors of the challenge keywords in **Query Phase 1 and 2**.*
- ***Guess**: Adversary $\mathcal{A}$ guesses which challenge keyword is contained in the challenge ciphertext. Adversary $\mathcal{A}$ wins this game if the guess is correct.*

4 A Concrete Construction for DEKS

4.1 Mathematical Tools

Bilinear Mapping Function [14]**.** Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative groups with the same prime order q. Let g_1 and g_2 denote generators of group $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The bilinear mapping function is defined as an efficient function $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the property that equation $\hat{e}(g_1^u, g_2^v) = \hat{e}(g_1, g_2)^{uv}$ holds for $\forall u, v \in \mathbb{Z}_q^*$.
Repeated Squaring Function [35]**.** Let P and Q be two special primes having the same binary size. We say that primes P and Q are special if equations $P = 2P_1 + 1$, $P_1 = 2P_2 + 1$, $Q = 2Q_1 + 1$, and $Q_1 = 2Q_2 + 1$ hold and P_1, P_2, Q_1, and Q_2 are also primes [8]. Without loss of generality, suppose P_2 is less than Q_2. Let $N = P \cdot Q$ and $\varphi(\cdot)$ denote Euler's totient function. Given $\forall \varpi \in \mathbb{Z}_N^*$ and $T < P_2$, a repeated squaring function is defined to compute $\varpi^{2^T} \mod N$.

With Euler number $\varphi(N)$, it is very efficient to compute $\varpi^{2^T} \mod N$ by computing $2^T \mod \varphi(N)$ first and then computing $\varpi^{2^T \mod \varphi(N)} \mod N$. In contrast, without Euler number $\varphi(N)$, we usually have to repeatedly compute $\varpi = \varpi^2 \mod N$ T times.

4.2 The Construction

We apply the repeated squaring function (RSF) to delay the time cost of ciphertext generation. Instead of directly generating a searchable ciphertext for a keyword, we take the keyword as input to implement the RSF and then compute the ciphertext from the resulting output. We state that no practical method can accelerate the implementation of the RSF without a *catalyst*. We also use a similar idea to efficiently generate a keyword search trapdoor (with the *catalyst*). Due to the delay capability, anyone who does not know the *catalyst* must take a mandatory time to generate a ciphertext. And this capability does not affect the search performance. The details of our instance are described below.

- **Setup**$(k, t, \mathcal{W})$: Take as inputs the security parameter k, the minimum time constraint t, and the keyword space $\mathcal{W}$ and implement the steps below:
  1. Generate parameters $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_2, \hat{e})$ of the bilinear mapping function according to the security parameter k;
  2. Pick parameters (P, Q, N, T) of the repeated squaring function, such that for any randomly chosen $\varpi \in \mathbb{Z}_N^*$, the time cost to compute $\varpi^{2^T} \mod N$ is more than the minimum time constraint t without $\varphi(N)$;
  3. Set $\pi = 2^T \mod \varphi(N)$, randomly pick $\sigma \in \mathbb{Z}_q^*$, and set $\eta = g_2^\sigma$;
  4. Choose three cryptographic hash functions $H_1 : \mathcal{W} \rightarrow \mathbb{Z}_N^*$, $H_2 : \mathbb{Z}_N^* \rightarrow \mathbb{G}_1$, and $H_3 : \mathbb{G}_T \rightarrow \{0,1\}^k$;
  5. Output public key $PK = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_2, \eta, \hat{e}, \mathcal{W}, N, T, H_1, H_2, H_3)$, private key $SK = \sigma$, and *catalyst* π.
- **DEKS**(PK, w, opt): Take as inputs public key PK, a keyword $w \in \mathcal{W}$, and an optional parameter $opt \in \{Null, \pi\}$ and execute the steps below:

1. Take the hash value $H_1(w)$ as input to compute its repeated squaring, where directly compute $\Delta = H_1(w)^{\pi} \mod N$ if *opt* $= \pi$, and otherwise, get the same result slowly by computing $\Delta = H_1(w)^{2^T} \mod N$;
2. Randomly pick $r \in \mathbb{Z}_q^*$ and compute $C_1 = g_2^r$ and $C_2 = H_3(\hat{e}(H_2(\Delta)^r, \eta))$;
3. Output the DEKS ciphertext $C_w = (C_1, C_2)$ of keyword w.

– **Trapdoor**(SK, π, w'): Take as inputs private key SK, *catalyst* π, and a keyword $w' \in \mathcal{W}$, compute the repeated squaring $\Delta' = H_1(w')^{\pi} \mod N$ of the hash value $H_1(w')$, and finally output the keyword search trapdoor $T_{w'} = H_2(\Delta')^{\sigma}$.
– **Test**$(T_{w'}, C_w)$: Take as inputs a keyword search trapdoor $T_{w'}$ and a DEKS ciphertext $C_w = (C_1, C_2)$, output "1" if $H_3(\hat{e}(T_{w'}, C_1)) = C_2$, and otherwise, output "0".

4.3 Correctness and Security Proof

Theorem 1 and 2 separately guarantee the search correctness and delay correctness of our DEKS instance. Due to the limit of space, we provide the proofs of these two theorems in the full version of this paper.

Theorem 1. *Suppose that hash functions H_1, H_2, and H_3 are random oracles. DEKS can maintain search correctness, except for a negligible probability.*

Theorem 2. *Given two special primes P and Q having $P \neq Q$, $N = P \cdot Q$, and a positive integer $T < P_2$; suppose that N is an n-bit composite number. The time cost to generate a DEKS ciphertext is at least $t = \mathcal{S}_n \cdot T$ without* catalyst *π, where $\pi = 2^T \mod \varphi(N)$, $\varphi(N) = (P-1)(Q-1)$, and $\mathcal{S}_n$ is the time cost to compute squaring modulo N.*

The SS-CKA security relies on the *computational bilinear Diffie-Hellman* (CBDH) assumption. According to the security parameter k, given parameters $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \hat{e})$ of the bilinear mapping function and parameters (g_2^a, g_2^b, g_1^c), the CBDH problem is to compute the value of $\hat{e}(g_1, g_2)^{abc}$, where (a, b, c) are randomly sampled from $\mathbb{Z}_q^*$. Let $\mathrm{Adv}_{\mathcal{B}}^{\mathrm{CBDH}}(k)$ denote the advantage of solving the CBDH problem by algorithm $\mathcal{B}$. The CBDH assumption holds if $\mathrm{Adv}_{\mathcal{B}}^{\mathrm{CBDH}}(k)$ is negligible.

In the proof, we show that a specially constructed algorithm can solve the CBDH problem if a PPT adversary can break the SS-CKA security of our DEKS instance. Formally, we have Theorem 3 below. Since the CBDH assumption holds in practice, Theorem 3 guarantees that no one can break the SS-CKA security. Due to the limit of space, we provide the proof in the full version of this paper.

Theorem 3. *Model hash functions H_1, H_2, and H_3 as three random oracles $\mathcal{Q}_{H_1}(\cdot)$, $\mathcal{Q}_{H_2}(\cdot)$, and $\mathcal{Q}_{H_3}(\cdot)$, respectively. Suppose a PPT adversary $\mathcal{A}$ wins with advantage $\mathrm{Adv}_{\mathcal{DEKS},\mathcal{A}}^{SS\text{-}CKA}$ in the SS-CKA game of our DEKS instance, in which $\mathcal{A}$ makes at most q_1 queries to $\mathcal{Q}_{H_1}(\cdot)$, at most q_2 queries to $\mathcal{Q}_{H_2}(\cdot)$, at most q_3 queries to $\mathcal{Q}_{H_3}(\cdot)$, and at most q_t queries to $\mathcal{Q}_{trapdoor}(\cdot)$. Then, the probability of a PPT algorithm $\mathcal{B}$ to solve the CBDH problem in parameters $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \hat{e}, g_2^a, g_2^b, g_1^c)$ is $\mathrm{Adv}_{\mathcal{B}}^{CBDH}(k) \geq \frac{2}{e^2 q_t^2 q_3} \mathrm{Adv}_{\mathcal{DEKS},\mathcal{A}}^{SS\text{-}CKA}$, where e denotes the base of the natural logarithm.*

5 Evaluation

We evaluate practical schemes in terms of computation and communication. And we experimentally compare DEKS with the two most-efficient schemes. We also leverage the *Enron* mail data [20] and *Wikipedia* article dataset [24] to evaluate the security of our DEKS under KGA.

5.1 Complexity Analysis

In terms of complexity, we compare DEKS with six classic PEKS instances. These schemes include the first PEKS (BC'04 [10]) and different types of KGA-resistant PEKS (RS'09 [34], XJ'13 [39], CM'16 [17], HL'17 [25], and CMY'16 [16]). For each of them, we analyze the efficiency in three algorithms: ciphertext generation, trapdoor computation, and test if a ciphertext matches. And we also analyze the size for the main public and private parameters. Moreover, we evaluate the number of expensive cryptographic operations implemented in those algorithms. The expensive operations mainly include the computations of exponentiation, multiplication, division, and bilinear mapping in different algebraic groups. To make the comparison simple and clear, Table 2 defines the symbols which we use later.

Compared with RS'09, XJ'13, CM'16, HL'17, and CMY'16, DEKS is much efficient in trapdoor generation and test stage. But DEKS is more complicated in ciphertext generation than others. We choose to use this tricky and subtle way to resist KGA. Further, both DEKS and BC'04 are most efficient in terms of communication complexity. If honest senders are given the *catalyst*, the complexity of encryption reduces significantly which could be quite close to the performance of BC'04. Besides, DEKS still performs well in terms of parameter size.

In summary, BC'04 has great performance in terms of the complexity and the interaction pattern but cannot thwart KGA. Under some particular assumptions, others may resist KGA while sacrificing some complexity and communication price. DEKS maintains analogous complexity and the same interaction pattern as BC'04, while merely sacrificing the efficiency of encryption.

Table 2. Theoretical comparison among DEKS and existing works in efficiency.

Instance	Computation complexity			Communication Cost			Parameter Size		Type
	Ciphertext	Trapdoor	Testing	Ciphertext	Trapdoor	Testing	Public	Private	
BC'04 [10]	$2 \times E_1 + B$	E_1	B	$\mathbb{G} + k$	$\mathbb{G}$	$\varOmega$	$2 \times \mathbb{G}$	$\mathbb{Z}_q^*$	Original
RS'09 [34]	$2 \times E_1 + B$	$3 \times E_1 + M_1$	$2 \times E_1 + D + B$	$\mathbb{G} + k$	$2 \times \mathbb{G}$	$\varOmega$	$3 \times \mathbb{G}$	$2 \times \mathbb{Z}_q^*$	Type-I
XJ'13 [39]	$4 \times E_1 + 2 \times B$	$2 \times E_1$	$2 \times B$	$2 \times \mathbb{G} + 2 \times k$	$\mathbb{G}$	$2 \times \varOmega$ or $3 \times \varOmega$	$2 \times \mathbb{G}$	$\mathbb{Z}_q^*$	Type-II
CM'16 [17]	$4 \times E_1 + 2 \times M_1$	$4 \times E_1 + D + M_1$	$7 \times E_1 + D + 5 \times M_1$	$6 \times \mathbb{G}$	$3 \times \mathbb{G}$	$\varOmega$	$2 \times \mathbb{G}$	$4 \times \mathbb{Z}_q^*$	Type-III
HL'17 [25]	$3 \times E_1 + M_1$	$\theta \times (E_1 + B)$	$M_2 + B$	$2 \times \mathbb{G}$	$\theta \times \mathbb{G}_T$	$\varOmega$	$2 \times \mathbb{G}$	$2 \times \mathbb{Z}_q^*$	Type-IV
CMY'16 [16]	$2 \times E_1 + 2 \times E_2 + 2 \times M_3 + B$	$E_1 + 2 \times E_2 + 2 \times M_3$	B	$2 \times \mathbb{Z}_N^* + \mathbb{G} + k$	$2 \times \mathbb{Z}_N^* + \mathbb{G}$	$\varOmega$	$2 \times \mathbb{G} + 2 \times \mathbb{Z}_N^*$	$\mathbb{Z}_q^* + \mathbb{Z}_N^*$	Type-V
DEKS	$2 \times E_1 + T \times M_3 + B$ or $2 \times E_1 + E_2 + B$	$E_1 + E_2$	B	$\mathbb{G} + k$	$\mathbb{G}$	$\varOmega$	$2 \times \mathbb{G} + \mathbb{Z}_N^*$	$\mathbb{Z}_q^* + \mathbb{Z}_N^*$	New
Notations*	E_1 and E_2 separately denote the exponentiation operation in groups $\mathbb{G}$ and $\mathbb{Z}_N^*$; θ denotes the number of possible senders; $\varOmega$ denotes the size of matching ciphertexts; T denotes the delay parameter in our DEKS instance; k denotes the security parameter; B denotes the bilinear mapping operation; D denotes the division operation in group $\mathbb{G}$; M_1, M_2, and M_3 separately denote the multiplication operation in groups $\mathbb{G}$, $\mathbb{G}_T$, and $\mathbb{Z}_N^*$.								

*Without loss of generality, we use $\mathbb{G}$ denote $\mathbb{G}_1$ or $\mathbb{G}_2$.

5.2 Experimental Analysis

We design two kinds of experiments to evaluate DEKS's performance, and further compare it with two efficient works, namely, BC'04 and CMY'16. The first experiment codes the above three instances. It compares their time costs in the aforementioned three algorithms. The second experiment tests the capability of DEKS in resisting KGA. It separately compares the time costs in launching KGA on DEKS and BC'04 in order to show that DEKS can make attackers cost massively. Further, we evaluate the average time cost on one guessing attempt for attackers equipped with parallel computing supports. The results show that our DEKS still makes attackers suffer well. Our experimental source codes are available on https://github.com/HustSecurityLab/DEKS_Exp, and the interested readers can use the codes to reproduce the results.

Table 3. Experimental environment.

Hardware & operation system	4 × Intel Xeon CPU E5-2630 v3 @ 2.40 GHz and 48 GB RAM; Ubuntu 20.04LTS
Compiler & interpreter	GCC v9.3.0 and Python v3.6
Program library	GMP v6.1.2, PBC v0.5.14, and NLTK v3.5
Dataset	Enron dataset [20] and Wikipedia dataset [24]
Elliptic curve $y^2 = x^3 + b$ with Embedding Degree 12 (unit: decimal)	
Base field	16283262548997601220198008118239886027035269286659395419233331082106632227801
Group order	16283262548997601220198008118239886026907663399064043451383740756301306087801
b	7322757890651446833342173470888950103129198494728585151431956701557392549679
Repeated Squaring Function $y = x^{2^T} \mod N$ with $N = P \cdot Q$; both P, Q are special primes (unit: hexadecimal)	
P	DD848D47E193DCF0F57DD9256ABF10B5869C2D5D600C21A4D36C29659C062542B5CDCB6CF1002D7177D720472078AFC0 193BAD7E0FCE7C07CABC83526F71CF2881993188748C07C52CF73D1A09BF38F22163909A7EBAEEC9A9D9019F6CE919AE F18BCD995F80E7823370D500B53DC85D169F4FBA383C9A2E7DA2393A11A9B171C86957B82E8115F9FB19670466155E50 E41ADF91FB392EBC53614A475F58F9959972E56346993923991BD15110D2393513243DFEB2C28FCDFA067535E7A8A4DF
Q	F9CE5FD04C169FC42F3C24C9E149EDCA7513A02648628C9AB80A9E9CE6F1FCD7EF4EA0FBC5AD4BE3E2B199A99969B749 01B46BAF632A3B653A2E0FDC37D9D44646247C104EAB0A38027725886DCCAC682A3E71A84F57E5CE3FAF8C6DD7DEA272 07AD6B3FBDDD51A4898884FB9C4853826C2836987179D4359122308CC6D44987562800D136BFB01CB3611E66B0F862EF A0E3769BE3795A9A75CA36A69E60851111849F8F0B8D46C5ACE50FCA7157B48B991C5AE30BC7B4198C464302C477CD0F

- **Experimental Environment.** Table 3 shows the experimental platform, including the hardware and software, the elliptic curve for realizing the bilinear mapping function, and the parameters of the repeated squaring function. We use the GNU Multiple Precision Arithmetic (GMP) library and the Pairing-Based Cryptography (PBC) library to implement BC'04, CMY'16, and DEKS. We produce the instances in the same bilinear-mapping-friendly elliptic curve [7] that offers efficient exponentiation operations. The elliptic curve and the composite number N both provide the same security level as AES-128. To test the performance, as in [28], we apply the Enron and Wikipedia and extract their keywords to generate a small-scale dataset and a large-scale dataset separately.
- **Performance Comparison.** This part shows the experimental results of the three instances BC'04, CMY'16, and DEKS. For Algorithm **Test**, **Trapdoor**, and **DEKS**, we choose the delay parameter T with values $2^0, 2^{12}$, and 2^{24}

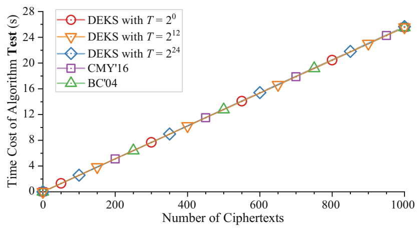

Fig. 3. Time cost: algorithm **Test**.

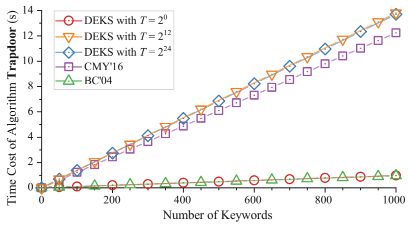

Fig. 4. Time cost: algorithm **Trapdoor**.

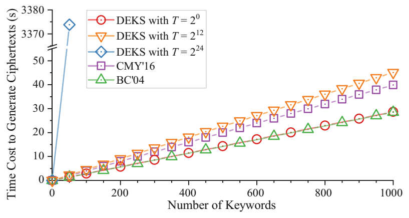

Fig. 5. Time cost: generate ciphertexts.

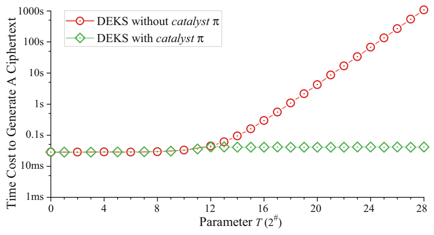

Fig. 6. DEKS time cost: generate a ciphertext with an increase in parameter T.

to clearly show the relations between the value of T and algorithms' performance. Meanwhile, we set a test size of 1,000 to clearly show the average time cost in each figure. Moreover, we present the average encryption time of DEKS with different delay parameters. Without lose generality, we use $T = 2^i (i \in [0, 28])$ to show the experimental results clearly. We let the delay parameter $T < 2^{28}$ since it is sufficient for us to practically resist KGA (which can be seen later).

Time Cost of Algorithm **Test**. Table 2 concludes that instances BC'04, CMY'16, and DEKS have the same complexity in testing a ciphertext. The experimental results, as shown in Fig. 3, also confirm this conclusion. Given different numbers of ciphertexts, we test the total time cost to find the matching ciphertexts and compute the average time to test a ciphertext. In summary, the above three instances have the same average time cost, which is approximately 25.55 milliseconds, in testing a ciphertext.

Time Cost of Algorithm **Trapdoor**. For each instance, we test the total time costs to generate the corresponding trapdoors for different numbers of keywords and compute the average cost. The numerical results, which is described in Fig. 4, imply that DEKS takes more time than BC'04 in trapdoor generation, and the time cost of DEKS is slightly higher than that of CMY'16. For example, the average time cost of BC'04, CMY'16, and DEKS are 0.96 milliseconds, 12.24

milliseconds, and 0.98 to 13.71 milliseconds (for $T \in [2^0, 2^{24}]$), respectively. And the average time cost of DEKS is constant for $T \geq 2^{12}$.

Time Cost to Generate A Ciphertext. For each instance, we test the total time costs to generate ciphertexts for different numbers of keywords and compute the average time cost. On the one hand, we test algorithm **DEKS** with the optional input parameter $opt = Null$. As Fig. 5 shows, when $T = 2^0$, DEKS and BC'04 take a similar cost, the average values are 28.63 ms and 28.47 ms, respectively. When the delay parameter $T = 2^{12}$, the average of DEKS and CMY'16 change to 45.11 ms and 39.87 ms, respectively. Note for the case where $T = 2^{24}$, we only present the result of 50 keywords in Fig. 5. We note that is sufficient to present a clear comparison. Setting T from 2^0 to 2^{28}, we can see that DEKS's average cost also increases significantly, as shown in Fig. 6. For example, when $T = 2^{20}$, the average time cost is 4.25 s. On the other hand, we test algorithm **DEKS** with the optional input parameter $opt = \pi$. The average cost is now quite stable even when T is increased to 2^{12}, approximately 41.27 ms. When $T \geq 2^{12}$, further increasing its value, we say, will not affect the encryption time. Note the generation of *catalyst* is very efficient, in particular, it only takes nearly 55 µs when $T \leq 2^{28}$.

- **Testing DEKS's Capability Against KGA**. We take the *Enron* mail data and *Wikipedia* article dataset as examples and test the average time cost on launching a successful KGA on DEKS and BC'04. First, we remove the wiki syntax from the entire *Wikipedia* article dataset using Wikipedia Extractor [5], then extract keywords using the PorterStemmer tool provided by the NLTK library [4] and remove stop words. Thus, we extract 6,756,439 different keywords from the *Wikipedia*. Similarly, we extract 400,087 keywords from the *Enron*.

Suppose a KGA attacker desires to know the keyword underlying a given keyword search trapdoor. The attacker picks a keyword from the keyword space, generates the keyword's searchable ciphertext, and tests if the searchable ciphertext matches the given trapdoor. If the test outputs a yes, the attacker successfully guesses the keyword and stops the attack. Otherwise, it chooses a new keyword and repeats the above steps.

Note BC'04 always provides fixed time cost w.r.t. ciphertext generation and keyword testing; and given a fixed T, DEKS also maintains fixed cost. Thus, it is feasible for us to compute the time cost to launch a successful KGA for a given trapdoor. For example, let $\mathcal{T}^{\text{Gen}}$ and $\mathcal{T}^{\text{Test}}$ be the time costs to generate and test a ciphertext, respectively. Suppose that the KGA attacker can get the keyword in a target trapdoor after guessing *num* keywords. Then, the time cost of the KGA attacker is equal to $num \cdot (\mathcal{T}^{\text{Gen}} + \mathcal{T}^{\text{Test}})$.

According to the above methods, we compute the time costs of launching a successful KGA, for each keyword. Then, we compute these costs' arithmetic mean, which is equal to the sum of them divided by the total count of distinct keywords. Table 4 and 5 present the results on launching KGA on DEKS (with some different delay parameters) and BC'04, with Wikipedia and Enron datasets.

Table 4. Time cost: launch KGA on DEKS and BC'04 with Wikipedia.

Instance	Arithmetic mean of time cost									
	T	Time	T	Time	T	Time	T	Time	T	Time
DEKS	2^{10}	54.76 h	2^{11}	58.90 h	2^{12}	65.99 h	2^{13}	3.39 days	2^{14}	4.69 days
	2^{15}	7.27 days	2^{16}	12.51 days	2^{17}	22.74 days	2^{18}	43.42 days	2^{19}	84.85 days
	2^{20}	167.18 days	2^{21}	340.10 days	2^{22}	1.81 yrs	2^{23}	3.63 yrs	2^{24}	7.23 yrs
BC'04	50.69 h									

Table 5. Time cost: launch KGA on DEKS and BC'04 with Enron.

Instance	Arithmetic mean of time cost									
	T	Time	T	Time	T	Time	T	Time	T	Time
DEKS	2^{10}	194.57 min	2^{11}	209.28 min	2^{12}	234.48 min	2^{13}	4.82 h	2^{14}	6.67 h
	2^{15}	10.33 h	2^{16}	17.78 h	2^{17}	32.32 h	2^{18}	61.70 h	2^{19}	5.02 days
	2^{20}	9.89 days	2^{21}	20.14 days	2^{22}	39.22 days	2^{23}	78.44 days	2^{24}	156.29 days
	2^{25}	312.84 days	2^{26}	1.71 yrs	2^{27}	3.44 yrs	2^{28}	6.86 yrs		
BC'04	180.09 min									

We find that the attack works quite well on BC'04 for both small and large-scale datasets. But, with the increase of T, DEKS can make attackers' time cost jump exponentially. For instance, for the Wikipedia, when $T = 2^{10}$, we make attackers take 54.76 h; when $T = 2^{17}$, the cost jumps to 22.74 days; and further $T = 2^{24}$, it exponentially increases to 7.23 years. As for the small-scale dataset, Enron, the time cost of the attacker also reaches 6.86 years, along with the increase of T.

In practice, a KGA attacker may guess and test possible keywords in parallel so as to enhance attack performance. To simulate this setting, we experimentally test the average time cost of one guessing attempt (namely, generating and testing a DEKS ciphertext), with different delay parameters and various degrees of parallelism. Figure 7 presents the main results. From the results, it can be seen that even a tiny increase of T may effectively mitigate the influence of parallel attacks. For instance, given $T = 2^{19}$, the attacker with 16 cores CPU can reduce the time cost of one attempt from 2.17 s to 0.17 s. If we put T to 2^{23}, the cost bounces back to 2.61 s. We state that DEKS resists KGA by adaptively increasing the time cost of generating a ciphertext. And this increase is unavoidable for any PPT adversary. According to the specific application, we can choose an

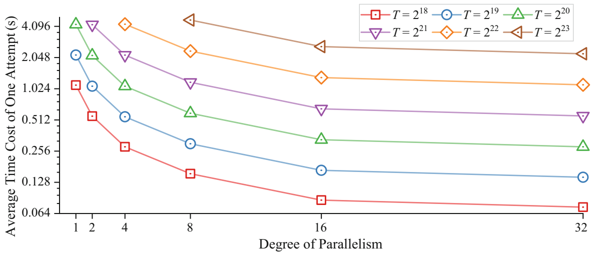

Fig. 7. DEKS average time cost: guess and test possible keywords in parallel.

appropriate value of T to increase the attack difficulty, so that the intaken cost of attack is far beyond the benefit attackers achieve.

6 Conclusion

To resist KGA on secure cloud-based searchable service, we propose a new scheme DEKS, which allows receivers to constrain the minimum time cost of a keyword-searchable ciphertext generation, in a non-interactive way. No sender who generates the ciphertext can break the time constraint without a *catalyst*. We apply both the RSF and bilinear mapping function to construct the first DEKS instance and prove its unique encryption delay capability. Compared with existing works, DEKS provides good performance and simple interaction pattern as the original PEKS, except that the time cost of ciphertext generation may go beyond the minimum constraint. Our experimental results show that our design can practically resist KGA in small and large-scale datasets.

Acknowledgements. We would like to thank our shepherd Anrin Chakraborti and the anonymous reviewers for their insightful comments and valuable suggestions. This work was partly supported by the National Key Research and Development Program of China under Grant No. 2021YFB3101304, the Wuhan Applied Foundational Frontier Project under Grant No. 2020010601012188, the National Natural Science Foundation of China under Grant No. 61872412, the Guangdong Provincial Key Research and Development Plan Project under Grant No. 2019B010139001, and the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 952697 (ASSURED) and No. 101021727 (IRIS).

References

1. Bitglass. https://www.bitglass.com/cloud-encryption
2. CipherCloud. https://www.ciphercloud.com/encryption-and-tokenization/
3. MVISION cloud. https://www.mcafee.com/enterprise/en-us/products/mvision-cloud/salesforce.html
4. Natural Language Toolkit (2020). http://www.nltk.org/
5. Attardi, G.: WikiExtractor (2015). https://github.com/attardi/wikiextractor
6. Baek, J., Safavi-Naini, R., Susilo, W.: Public key encryption with keyword search revisited. In: Gervasi, O., Murgante, B., Laganà, A., Taniar, D., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2008. LNCS, vol. 5072, pp. 1249–1259. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69839-5_96
7. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006). https://doi.org/10.1007/11693383_22
8. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. **15**(2), 364–383 (1986)
9. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 757–788. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_25

10. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_30
11. Boneh, D., Raghunathan, A., Segev, G.: Function-private identity-based encryption: hiding the function in functional encryption. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 461–478. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_26
12. Burdges, J., De Feo, L.: Delay encryption. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 302–326. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_11
13. Byun, J.W., Rhee, H.S., Park, H.-A., Lee, D.H.: Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In: Jonker, W., Petković, M. (eds.) SDM 2006. LNCS, vol. 4165, pp. 75–83. Springer, Heidelberg (2006). https://doi.org/10.1007/11844662_6
14. Chatterjee, S., Menezes, A.: On cryptographic protocols employing asymmetric pairings - the role of Ψ revisited. Discret. Appl. Math. **159**(13), 1311–1322 (2011)
15. Chen, H., Cao, Z., Dong, X., Shen, J.: SDKSE-KGA: a secure dynamic keyword searchable encryption scheme against keyword guessing attacks. In: Meng, W., Cofta, P., Jensen, C.D., Grandison, T. (eds.) IFIPTM 2019. IAICT, vol. 563, pp. 162–177. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33716-2_13
16. Chen, R., et al.: Server-aided public key encryption with keyword search. IEEE Trans. Inf. Forensics Secur. **11**(12), 2833–2842 (2016)
17. Chen, R., Mu, Y., Yang, G., Guo, F., Wang, X.: Dual-server public-key encryption with keyword search for secure cloud storage. IEEE Trans. Inf. Forensics Secur. **11**(4), 789–798 (2016)
18. Chen, Y.: SPEKS: secure server-designation public key encryption with keyword search against keyword guessing attacks. Comput. J. **58**(4), 922–933 (2015)
19. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 451–467. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_15
20. Cohen, W.W.: Enron Email Dataset (2015). https://www.cs.cmu.edu/./enron/
21. Emura, K., Ito, K., Ohigashi, T.: Secure-channel free searchable encryption with multiple keywords: a generic construction, an instantiation, and its implementation. J. Comput. Syst. Sci. **114**, 107–125 (2020)
22. Fang, L., Susilo, W., Ge, C., Wang, J.: Public key encryption with keyword search secure against keyword guessing attacks without random oracle. Inf. Sci. **238**, 221–241 (2013)
23. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 248–277. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_10
24. Foundation, W.: Wikimedia downloads (2020). https://dumps.wikimedia.org/enwiki/20201120/enwiki-20201120-pages-articles.xml.bz2
25. Huang, Q., Li, H.: An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. Inf. Sci. **403**, 1–14 (2017)
26. Jakobsson, M., Juels, A.: Proofs of work and bread pudding protocols (extended abstract). In: Preneel, B. (ed.) Secure Information Networks. ITIFIP, vol. 23, pp. 258–272. Springer, Boston, MA (1999). https://doi.org/10.1007/978-0-387-35568-9_18

27. Jeong, I.R., Kwon, J.O., Hong, D., Lee, D.H.: Constructing PEKS schemes secure against keyword guessing attacks is possible? Comput. Commun. **32**(2), 394–396 (2009)
28. Kim, K.S., Kim, M., Lee, D., Park, J.H., Kim, W.: Forward secure dynamic searchable symmetric encryption with efficient updates. In: CCS 2017, pp. 1449–1463 (2017)
29. Lu, Y., Li, J.: Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices. IEEE Trans. Mob. Comput. (2021). https://doi.org/10.1109/TMC.2021.3077508
30. Lu, Y., Li, J., Zhang, Y.: Secure channel free certificate-based searchable encryption withstanding outside and inside keyword guessing attacks. IEEE Trans. Serv. Comput. **14**(6), 2041–2054 (2021)
31. Miao, Y., Tong, Q., Deng, R.H., Choo, K.K.R., Liu, X., Li, H.: Verifiable searchable encryption framework against insider keyword-guessing attack in cloud storage. IEEE Trans. Cloud Comput. **10**(1), 835–848 (2022)
32. Qin, B., Chen, Y., Huang, Q., Liu, X., Zheng, D.: Public-key authenticated encryption with keyword search revisited: security model and constructions. Inf. Sci. **516**, 515–528 (2020)
33. Rhee, H.S., Park, J.H., Susilo, W., Lee, D.H.: Trapdoor security in a searchable public-key encryption scheme with a designated tester. J. Syst. Softw. **83**(5), 763–771 (2010)
34. Rhee, H.S., Susilo, W., Kim, H.: Secure searchable public key encryption scheme against keyword guessing attacks. IEICE Electron. Exp. **6**(5), 237–243 (2009)
35. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical report, MIT/LCS/TR-684 (1996). https://people.csail.mit.edu/rivest/pubs/RSW96.pdf
36. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: S&P 2000, pp. 44–55 (2000)
37. Sun, L., Xu, C., Zhang, M., Chen, K., Li, H.: Secure searchable public key encryption against insider keyword guessing attacks from indistinguishability obfuscation. Sci. China Inf. Sci. **61**(3), 1–3 (2017). https://doi.org/10.1007/s11432-017-9124-0
38. Wang, C., Tu, T.: Keyword search encryption scheme resistant against keyword-guessing attack by the untrusted server. J. Shanghai Jiaotong Univ. (Sci.) **19**(4), 440–442 (2014). https://doi.org/10.1007/s12204-014-1522-6
39. Xu, P., Jin, H., Wu, Q., Wang, W.: Public-key encryption with fuzzy keyword search: a provably secure scheme under keyword guessing attack. IEEE Trans. Comput. **62**(11), 2266–2277 (2013)
40. Yao, F.F., Yin, Y.L.: Design and analysis of password-based key derivation functions. IEEE Trans. Inf. Theory **51**(9), 3292–3297 (2005)
41. Yau, W., Phan, R.C., Heng, S., Goi, B.: Keyword guessing attacks on secure searchable public key encryption schemes with a designated tester. Int. J. Comput. Math. **90**(12), 2581–2587 (2013)
42. Yousefipoor, V., Ameri, M.H., Mohajeri, J., Eghlidos, T.: A secure attribute based keyword search scheme against keyword guessing attack. In: IST 2016, pp. 124–128 (2016)
43. Zhang, Y., Xu, C., Ni, J., Li, H., Shen, X.S.: Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage. IEEE Trans. Cloud Comput. **9**(4), 1335–1348 (2021)