International Journal of Information Security (2019) 18:533–548
https://doi.org/10.1007/s10207-018-00426-7

REGULAR CONTRIBUTION

Provably secure public-key encryption with conjunctive and subset keyword search

Oriol Farràs[1] · Jordi Ribes-González[1]

Published online: 4 January 2019

Abstract
Public-key encryption with keyword search (PEKS) schemes enable public key holders to encrypt documents, while the secret key holder is able to generate queries for the encrypted data. In this paper, we present two PEKS schemes with extended functionalities. The first proposed scheme supports conjunctive queries. That is, it enables searching for encrypted documents containing a chosen list of keywords. We prove the computational consistency of our scheme, and we prove security under the asymmetric DBDH assumption. We show that it improves previous related schemes in terms of efficiency and in terms of index and trapdoor size. The second proposed scheme supports subset queries and some more general predicates. We prove the computational consistency of our scheme, and we prove our scheme secure under the p-BDHI assumption. We show that it improves previous related schemes in terms of efficiency and expressiveness. Moreover, unlike previous related schemes, it admits an arbitrary keyword space.

Keywords Searchable encryption · Conjunctive keyword search · Subset keyword search · Public-key encryption with keyword search

1 Introduction

External storage servers allow users to retrieve outsourced data selectively. For instance, a database located in a cloud storage server can be queried for the segment of records satisfying a certain condition. However, since such servers are managed by untrusted third parties, users are usually reluctant to outsource their sensitive data in the clear.

Encrypting the data to be outsourced is a good approach to overcome this security concern. Nevertheless, traditional encryption schemes fail to provide selective retrieval in an efficient and secure way. Searchable encryption deals with this problem by allowing data owners to issue queries for encrypted outsourced data. Much like traditional encryption schemes, searchable encryption schemes generally come in two distinct types, serving different purposes: symmetric-key searchable encryption (SSE) [15] and public-key searchable encryption.

Public-key searchable encryption (also named *public-key encryption with keyword search*), or PEKS, was firstly proposed by Boneh et al. [9]. Since their pioneering work, there have appeared several PEKS schemes in the literature [3,4,11,14,21,23,31–33,40,42], improving the scheme in [9] in terms of efficiency, security or functionality.

The keyword search protocol considered in public-key searchable encryption involves the following entities:

- a set of *data suppliers*, which provides and encrypts the data to be outsourced,
- a storage *server* (e.g., an e-mail gateway or a database), which stores the outsourced data, and
- a *client* of the storage server, who retains the ability to generate queries for the encrypted data.

In the protocol, the client firstly sets up the scheme by generating some public parameters and shares them with the server and the data suppliers. Then, it generates a public and private key pair and shares the public key with the data suppliers. Afterward, the data suppliers may wish to share a collection of documents with the client, where each document is

✉ Jordi Ribes-González
jordi.ribes@urv.cat

Oriol Farràs
oriol.farras@urv.cat

[1] Universitat Rovira i Virgili, Av. Països Catalans 26, 43007 Tarragona, Catalonia, Spain

indexed by a set of keywords. To do so, they encrypt this collection and upload the resulting ciphertexts to the storage server. Also, the client wants to receive the stored documents indexed by all keywords in a chosen list from the storage server. To let the storage server know which encrypted document it should forward, the client generates a query and sends it to the storage server. The storage server is then able to use the received information to select the encrypted stored documents satisfying the conditions in the queries and to return those documents to the client. Note that this is done without direct interaction between the client and the data providers.

As for query expressiveness, the scheme in [9] achieves *single-keyword* queries, i.e., queries matched by documents indexed by a single chosen keyword. Single-keyword PEKS schemes enable data providers to generate an encryption of a keyword w by using the public key of the client and upload it to the storage server. We call this an *encrypted index*, and we denote it by $\mathbf{I}(w)$. The client, holding the secret key, can build a *trapdoor* $\mathbf{T}(w')$ corresponding to some keyword w'. By sending $\mathbf{T}(w')$ to the storage server, the client empowers the storage server to learn whether any encrypted index $\mathbf{I}(w)$ satisfies $w = w'$, but no other information about $\mathbf{I}(w)$ is revealed in this process.

One of the most common enhancements of PEKS is *conjunctive PEKS* [11,21,30,31], which consists in enabling *conjunctive field keyword* queries. Typically, in conjunctive PEKS, data providers encrypt a tuple (that is, an ordered set) of keywords $(w_1, \ldots, w_m)$ by using the public key of the client, generating an encrypted index $\mathbf{I}(w_1, \ldots, w_m)$. The client can produce a trapdoor associated with a tuple of keywords $(w'_1, \ldots, w'_l)$, along with a set of keyword fields (or positions) $\{j_1, \ldots, j_l\} \subseteq \{1, \ldots, m\}$. On receiving this trapdoor, the storage server can check if the predicate $(w_{j_1} = w'_1) \wedge \cdots \wedge (w_{j_l} = w'_l)$ holds by using the index $\mathbf{I}(w_1, \ldots, w_m)$ and the trapdoor. This usage of keyword fields is standard in the PEKS literature and in many tools and applications related to encrypted search, such as in relational DBMS (in the form of table fields), in the metadata of network packets and e-mails and in some of the proposed applications of PEKS [9,13,30,36,41].

Another enhancement of PEKS is *subset PEKS*, first defined in [11], which enables *subset* queries. In subset PEKS, data providers encrypt a tuple of keywords $(w_1, \ldots, w_m)$ by using the public key of the client, generating an encrypted index $\mathbf{I}(w_1, \ldots, w_m)$. The client can produce a trapdoor associated to m arbitrary sets of keywords $(A_1, \ldots, A_m)$. When receiving this trapdoor, the storage server can check if the conjunctive subset query predicate $(w_1 \in A_1) \wedge \cdots \wedge (w_m \in A_m)$ holds by using the index $\mathbf{I}(w_1, \ldots, w_m)$ and the trapdoor.

In most of the proposed applications for single-keyword PEKS, the exchanged ciphertexts take the form

$$\mathrm{Enc}_{\mathrm{pk}}(D) \| \mathbf{I}(w_1) \| \cdots \| \mathbf{I}(w_m),$$

where pk is the public key of the client, Enc is some public-key encryption scheme, document D is indexed by keywords $w_1, \ldots, w_m$, and $\mathbf{I}(w_1), \ldots, \mathbf{I}(w_m)$ are the corresponding encrypted indexes. Such ciphertexts are uploaded to the server by the data suppliers. The client can recover the documents that are indexed by the keyword w in position $i \in \{1, \ldots, m\}$ by sending the trapdoor $\mathbf{T}(w)$ and the position i to the server. The basic security property of PEKS schemes is that the server does not learn any information about the encrypted indexes unless it has the knowledge of a matching trapdoor.

Note that one could achieve conjunctive queries by using single-keyword PEKS schemes, simply by querying for particular keywords as stated above, and computing the intersection of the results locally or in the storage server. When doing so, the server learns which documents are indexed by each of the keywords. The benefits of using conjunctive PEKS against using single-keyword PEKS in this way are mainly in efficiency and security, since trapdoors are usually much shorter, the intersection predicate is embedded in the trapdoor and the intersection is computed at the storage server.

The earliest application scenario for PEKS, as suggested by Boneh et al. [9], is e-mail gateways. In this scenario, a user Alice wishes to read her e-mails, which are stored in an untrusted e-mail gateway in an encrypted form. When retrieving her e-mails, she may want the e-mail gateway to forward her just e-mails satisfying certain conditions, e.g., containing the keyword "urgent" or having a particular sender "Bob". To enable the e-mail gateway to do so, she sends the trapdoors corresponding to these keywords to the gateway, e.g., $\mathbf{T}$("tag:urgent") and $\mathbf{T}$("sender:Bob").

Now, suppose user Bob wishes to send Alice an e-mail D. He may encrypt D by using Alice's public key pk and attach the sender information and the "urgent" tag in the form of encrypted indexes of the form $\mathbf{I}$("sender:Bob") and $\mathbf{I}$("tag:urgent"). Thus, Alice's gateway would receive the message

$$\mathrm{Enc}_{\mathrm{pk}}(D) \| \mathbf{I}(\text{"sender:Bob"}) \| \mathbf{I}(\text{"tag:urgent"}).$$

By matching the attached index with the stored trapdoors, the e-mail gateway is able to know which e-mails should be forwarded to Alice by checking which trapdoors match which encrypted indexes. However, it learns nothing else about the e-mails in the process. This example illustrates an application covered by PEKS that seems, a priori, hard to cover by using exclusively symmetric-key mechanisms such as SSE.

Another natural application for PEKS schemes is related to secure audit logs, and it was devised by Waters et al. [41]. Audit logs are stored in an untrusted storage server in an encrypted form by using an Identity-Based Encryption (IBE) scheme (e.g., [8,10,24,25,39]), and PEKS encrypted indexes are attached to it. Attributes for IBE and keywords for PEKS are related to the audit record at hand, e.g., date and time. An investigator Bob may wish to be granted access to audit logs recorded, for instance, in a particular date and time. To do so, Bob asks the key escrow agent, say Alice, for the trapdoors and decryption keys corresponding to this particular date and time. If Alice authorized Bob to issue this particular search, she would serve Bob the requested IBE decryption keys and PEKS trapdoor, and Bob would then be able to retrieve the audit logs of interest from the untrusted storage server.

Other applications include secure cloud storage [13], decryption key delegation systems [30] and context-based forwarding [36]. Although SSE represents a much more efficient solution than PEKS for cloud storage in the symmetric setting, PEKS schemes can be useful in applications involving asymmetric architectures, such as when sending or sharing outsourced data.

The first symmetric-key searchable encryption scheme was proposed in 2000 by Song et al. [37]. In 2004, Boneh et al. presented the earliest PEKS scheme in [9]. This was immediately followed by the first conjunctive searchable encryption scheme in the symmetric setting by Golle et al. [20] and by an extension of PEKS to conjunctive PEKS by Park et al. [31]. Many authors then presented alternative PEKS schemes that allow decryption of indexes [11,30,32], multi-dimensional range queries [35], reduction of communication and storage costs [14,42], extension to multi-user systems [21] or improvements of security in various ways [3,13,17,28,33,40]. Among them, one of the most relevant is the work by Boneh and Waters [11], in which they define the general notion of a Hidden-Vector Encryption (HVE) scheme, providing an enhancement of expressiveness of the queries that allows for conjunctive and keyword search, and also decryption of indexes. We overlook the existing work on the symmetric-key setting of searchable encryption, since it lies outside the scope of this article.

Also, the relationship between PEKS and Anonymous Identity-Based Encryption, abbreviated AIBE, was first established in [9]. Most AIBE schemes (e.g., see [8,10,24,25,39]) can be easily translated to PEKS schemes and vice-versa via a generic blackbox transformation [1,9].

We propose two PEKS schemes. The first one achieves conjunctive field keyword search. Under the proposed security definition, it does not provide any security enhancement against using a single-keyword PEKS scheme to issue conjunctive field keyword queries. However, as we see in Sect. 7, it improves all previous related schemes in terms of efficiency in the most critical operations. Moreover, the trapdoors generated by using this first scheme consist of just one group element, and the index size improves all previous PEKS schemes.

The second proposed scheme enables a class of generalized subset queries, which includes subset queries as defined in [11]. The proposed security definition guarantees that nothing is leaked from encrypted indexes apart from the output of the search process. To the best of our knowledge, apart from the scheme in [11], no other subset PEKS schemes have been proposed in the literature. The proposed scheme improves [11] in terms of efficiency and expressiveness, and it does not assume that keywords are taken from a finite keyword space.

The security of the two proposed schemes relies on the intractability of the asymmetric DBDH problem and of the p-BDHI problem, respectively.

The remainder of this paper is structured as follows. In Sect. 2, we outline the preliminaries needed in this work. Our constructions for conjunctive and subset PEKS are described in Sects. 3 and 5. Sections 4 and 6 feature the consistency and security proofs for our conjunctive and subset PEKS schemes, respectively. In Sect. 7, we analyze the efficiency of our schemes. We conclude the article in Sect. 8 with some final remarks and future work directions.

2 Preliminaries

We start this section by giving some general notation and by stating the general model for the proposed PEKS schemes. We then give the consistency and security definitions, providing the hardness assumptions on which we base the security of our schemes. We finally give some implementation remarks.

2.1 Notation

We start by giving some standard notation and definitions used in searchable encryption. In this work, a *keyword* denotes a binary string $w \in \{0,1\}^*$. We define a *document* as a tuple of keywords $\mathbf{D} = (w_1, \ldots, w_m)$, and we say that keyword w_i is in *keyword field* (or *position*) i of $\mathbf{D}$. Note that, in this definition of document, we drop the data items (e.g., files, e-mails...) and consider only the indexing keywords. We make this choice since PEKS works exclusively over the indexing keywords, and data items may be protected by other cryptographic means (as explained in Sect. 1 and as studied in [17]).

If $\mathbf{D}_0, \mathbf{D}_1$ are two documents, we denote by $\mathbf{D}_0 \Delta \mathbf{D}_1$ the set of keywords appearing in either $\mathbf{D}_0$ or $\mathbf{D}_1$, but not in both at the same time. So if, for instance, $\mathbf{D}_0 =$ (“a”, “b”, “c”) and

$\mathbf{D}_1$ = ("a", "d", "e"), we would then have that $\mathbf{D}_0 \Delta \mathbf{D}_1$ = {"b", "c", "d", "e"}. We also name $\mathbf{D}_0 \Delta \mathbf{D}_1$ as the set of keywords *distinguishing* $\mathbf{D}_0$ and $\mathbf{D}_1$.

Given a positive integer m, we denote by $[m]$ the set $\{1, \ldots, m\}$. Given a function $f : \mathbb{N} \to \mathbb{R}$, we say that $f(\lambda)$ is *negligible in* λ if for every positive polynomial g there exists an integer λ_0 such that, for all $\lambda > \lambda_0$, $|f(\lambda)| < 1/g(\lambda)$. That is, if it decreases faster than any positive polynomial.

See [12] for an extensive and recent survey on the subject of searchable encryption.

2.2 Model for PEKS scheme

We now give the general model for the proposed public-key searchable encryption schemes. Although not stated, every algorithm apart from Setup takes the public parameters as input.

Definition 1 We define a PEKS scheme $\mathscr{S}$ as consisting of five polynomial-time algorithms:

$\mathscr{S}$.Setup(λ):	Probabilistic algorithm run by the client that, given a security parameter λ, returns the public parameters params of the scheme.
$\mathscr{S}$.KeyGen():	Probabilistic algorithm run by the client that derives a private key sk and a public key pk from the public parameters params.
$\mathscr{S}.\text{BuildIndex}_{\text{pk}}(\mathbf{D})$:	Probabilistic algorithm, to be run by data providers. It takes as input a document $\mathbf{D}$ and returns a corresponding encrypted index $\mathbf{I}$.
$\mathscr{S}.\text{Trapdoor}_{\text{sk}}(\mathbf{L}, J)$:	Algorithm run by the client that takes as input a tuple $\mathbf{L}$ of keywords and a set J of positions. It returns a corresponding trapdoor $\mathbf{T}$.
$\mathscr{S}.\text{Search}(\mathbf{I}, \mathbf{T})$:	Deterministic algorithm run by the server and taking as input an encrypted index $\mathbf{I}$ and a trapdoor $\mathbf{T}$. It returns either 1 or 0.

2.3 Consistency definition

The consistency property relates to the correctness of the scheme, in the sense that an encrypted index and a trapdoor should match in the search process exactly when the underlying document and query also match. If a document and a query match, then by construction of our schemes the corresponding encrypted index and trapdoor match in the search process. However, the converse does not necessarily hold. In this regard, the usage of hash functions in the proposed schemes induces the existence of false positives in the search process. Therefore, we must analyze the extent to which false positives can be produced, and we recur to a notion of consistency defined by Abdalla et al. [1].

The consistency notions defined by Abdalla et al. [1] are, in increasing strength order, *computational*, *statistical* and *perfect*. We prove consistency under the random oracle model and under an adaptation of the weakest definition of consistency in [1], namely computational consistency. Informally, their definition states that the advantage of any polynomial-time adversary in finding a matching encrypted index and trapdoor coming from a non-matching document and query is negligible in the security parameter, where the adversary has access to the public parameters and to the public key.

Let $\mathscr{S}$ be a PEKS scheme. Given a security parameter λ, we introduce a consistency game in the following three phases:

- *Setup.* The challenger runs $\mathscr{S}$.Setup on input λ and then hands over the public parameters to the adversary. It also runs $\mathscr{S}$.KeyGen, keeps the private key sk secret and hands over the public key pk to the adversary.
- *Guess.* The adversary outputs a document of the form $\mathbf{D} = (w_1, \ldots, w_m)$ and a tuple of keywords of the form $\mathbf{L} = (w'_1, \ldots, w'_l)$ together with a set of positions $J = \{j_1, \ldots, j_l\} \subseteq [m]$.
- *Output.* The challenger hands over to the adversary the trapdoor $\mathbf{T} = \mathscr{S}.\text{Trapdoor}_{\text{sk}}(\mathbf{L}, J)$, and then, the adversary computes $\mathbf{I} = \mathscr{S}.\text{BuildIndex}_{\text{pk}}(\mathbf{D})$. If it holds that $\mathscr{S}.\text{Search}(\mathbf{I}, \mathbf{T}) = 1$ and if there exists a $j_i \in J$ such that $w_{j_i} \neq w'_i$, then the adversary outputs a bit $b = 1$. Otherwise, it outputs $b = 0$.

Definition 2 (*Computational consistency of PEKS* [1]) A PEKS scheme $\mathscr{S}$ is *computationally consistent* if the advantage of every probabilistic polynomial-time (PPT) adversary $\mathscr{A}$ in the above game

$$\text{Adv}_{\mathscr{A}}(\lambda) = \Pr(b = 1)$$

is negligible in λ.

2.4 Security definition

In this section, we provide the hardness assumptions and the security definitions used in the security analysis of the proposed schemes. All proofs in this work are set in the random oracle model (see [7]).

2.4.1 Hardness assumptions

We now define symmetric and asymmetric bilinear groups. In this article, group operations are always written multiplicatively.

Definition 3 (*Bilinear Groups*) Let $(\mathbb{G}_1, \cdot), (\mathbb{G}_2, \cdot)$ be two cyclic groups of prime order q with generators g, h, respectively (usually denoted by $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$), and suppose that there exists a cyclic group $\mathbb{G}_T$ of order q and a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

We say that $\mathbb{G}_1$ is a *symmetric bilinear group* if there exists an efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. Under such an isomorphism, we denote $\mathbb{G}_1$ and $\mathbb{G}_2$ by $\mathbb{G}$.

Similarly, we say that $\mathbb{G}_1, \mathbb{G}_2$ are *asymmetric bilinear groups* if there exist no non-trivial efficiently computable homomorphisms from $\mathbb{G}_2$ to $\mathbb{G}_1$.

The bilinear groups $\mathbb{G}, \mathbb{G}_1, \mathbb{G}_2$ are taken to be subgroups of the group of points of an elliptic curve, and $\mathbb{G}_T$ is a subgroup of the multiplicative group of a finite field [29]. The definition of symmetric and asymmetric bilinear groups corresponds to Type 1 and Type 3 pairings in the paper by Galbraith et al. [19]. The term *pairing* refers to the non-degenerate bilinear map in the definition of bilinear group. We refer the reader to their article for properties of particular instantiations and to [5,34] for techniques to speed up pairing computation.

The first scheme we propose is proved secure under the asymmetric Decisional Bilinear Diffie–Hellman assumption (asymmetric DBDH). This assumption is proposed in the work [8] by Boneh and Boyen as a generalization of the DBDH assumption (see [22]) to the asymmetric setting. The DBDH assumption is easily seen to imply DDH in the target group $\mathbb{G}_T$.

Definition 4 (*Asymmetric DBDH Assumption*) Let $\mathbb{G}_1 = \langle g \rangle$, $\mathbb{G}_2 = \langle h \rangle$ be asymmetric bilinear groups deterministically generated according to a security parameter λ. We say the *asymmetric* DBDH *assumption* holds in $\mathbb{G}_1$ and $\mathbb{G}_2$ if for every PPT algorithm $\mathscr{B}$,

$$\mathrm{Adv}_{\mathscr{B}}(\lambda) = \Big|\Pr\left(\mathscr{B}(g, g^a, g^b, h, h^a, h^c, e(g,h)^{abc}) = 1\right) - \Pr\left(\mathscr{B}(g, g^a, g^b, h, h^a, h^c, e(g,h)^{r}) = 1\right)\Big|$$

is negligible in λ, where the probabilities are taken over a, b, c, r uniformly distributed in $\mathbb{F}_q$ and over the random bits of $\mathscr{B}$.

The second scheme we propose is proved secure under the Bilinear Diffie–Hellman Inversion Assumption ($p-$BDHI). This assumption is also proposed in [8]. According to [18], the best known algorithm breaking p-BDHI is to solve the Discrete Logarithm Problem (DLP) in $\mathbb{G}$.

Definition 5 (p-BDHI *Assumption*) Let $\mathbb{G} = \langle g \rangle$ denote a symmetric bilinear group deterministically generated according to a security parameter λ, and let p be a positive integer. We say the p-BDHI *assumption* holds in $\mathbb{G}$ if for every PPT algorithm $\mathscr{B}$,

$$\mathrm{Adv}_{\mathscr{B}}(\lambda) = \Pr\left(\mathscr{B}(g, g^a, g^{a^2}, \ldots, g^{a^p}) = e(g,g)^{1/a}\right)$$

is negligible in λ, where the probabilities are taken over uniformly distributed $a \in \mathbb{F}_q$ and over the random bits of $\mathscr{B}$.

In the proposed schemes and in the definitions above, bilinear groups are generated according to the security parameter λ. We choose bilinear groups to have exponential order in λ. See the security and consistency proofs for more details about this choice.

2.4.2 Security definition

We now introduce the security definition used in this article. We adapt the security definition introduced by Boneh et al. [9] to the conjunctive and subset case of Definition 1.

The used security definition is a semantic-security style definition that guarantees encrypted index indistinguishability in the face of an adversary with access to the public key and to trapdoors not containing any keyword distinguishing the challenge candidate documents. Therefore, in the security definition we propose, the adversary is not allowed to obtain trapdoors associated to any word that appears in one of the challenge candidate documents, but not in both.

Let $\mathscr{S}$ be a PEKS scheme. Given a security parameter λ, we introduce a security game in the following five phases:

- *Setup.* The challenger runs Setup on input λ and hands over the public parameters to the adversary. It also runs $\mathscr{S}$.KeyGen, keeps the private key sk secret and hands over the public key pk to the adversary.
- *Query Phase 1.* The adversary adaptively requests the challenger for q_T trapdoors of its own choice, where q_T is a polynomial value in the security parameter λ. We denote the set of all keywords queried in this phase by $\mathscr{W}_1$.
- *Challenge.* The adversary outputs two challenge candidate documents $\mathbf{D}_0, \mathbf{D}_1$ subject to the restriction that keywords appearing in $\mathbf{D}_0 \Delta \mathbf{D}_1$ have not been queried in Query Phase 1. That is, $(\mathbf{D}_0 \Delta \mathbf{D}_1) \cap \mathscr{W}_1 = \emptyset$. The challenger throws a fair coin $b \in \{0, 1\}$, and outputs the encrypted index $\mathscr{S}$.BuildIndex$(\mathbf{D}_b)$ corresponding to $\mathbf{D}_b$.
- *Query Phase 2.* The adversary proceeds just as in Query Phase 1, but it is not allowed to ask for trapdoors containing keywords in $\mathbf{D}_0 \Delta \mathbf{D}_1$. That is, if the set of all keywords queried in this phase is $\mathscr{W}_2$, we impose $(\mathbf{D}_0 \Delta \mathbf{D}_1) \cap \mathscr{W}_2 = \emptyset$.
- *Guess.* The adversary outputs a guess $b' \in \{0, 1\}$ for b.

Definition 6 (*Semantic security against adaptive chosen keyword attacks*) We say that a PEKS scheme $\mathcal{S}$ is semantically secure against adaptive chosen keyword attacks if the advantage of every PPT adversary $\mathcal{A}$ in distinguishing b in the above game

$$\begin{aligned}\mathrm{Adv}_{\mathcal{A}}(\lambda) &= |\Pr(b' = b) - 1/2| \\ &= |\Pr(\mathcal{A}(X) = b | X = b) \\ &\quad - \Pr(\mathcal{A}(X) = b | X = 1 - b)|\end{aligned}$$

is negligible in λ.

For conjunctive PEKS, the security definition we consider is slightly weaker than in other related works, in the following sense. Works such as [4,11,14,16,20,21] impose the natural restriction of serving the adversary just trapdoors coming from queries with equal search outcome over the two challenge candidate documents. In the case of conjunctive queries, the restriction we pose is stronger, since served trapdoors can not contain any keywords distinguishing the challenge candidate documents. This implies that trapdoors could leak which encrypted indexes contain some of the keywords in the underlying query, even if there is not a match.

In addition, the considered security definition does not provide trapdoor unlinkability or remove the need for a secure channel for trapdoors, as studied, for instance, in [3,13,32,40].

2.4.3 Implementation remarks

We refer the reader to [2] for remarks and references on the following statements and for a recent review on the state of the art of pairing computation.

The implementation of asymmetric bilinear groups for elliptic curve cryptography is often based on BN, BLS, KSS or MNT elliptic curves. In turn, symmetric bilinear groups are implemented in practice on supersingular elliptic curves.

Supersingular elliptic curves are well known to require large prime order groups for the DLP to be intractable (since they have a small MOV exponent), and this would enlarge the size of exchanged information in the proposed schemes. Moreover, recent results on the discrete logarithm problem [6] have rendered symmetric bilinear groups effectively obsolete for cryptographic purposes. Nevertheless, as in [26], we note that the second scheme we propose can be implemented in asymmetric bilinear groups as well, thus reducing the group order and increasing efficiency and security. In this context, symmetric bilinear groups are used just to facilitate the construction of the formal security proof.

We should note that asymmetric bilinear groups guarantee that we can securely and efficiently hash onto $\mathbb{G}_1$. In particular, it is possible to efficiently and uniformly sample from $\mathbb{G}_1$ without computing multiples of the generator g. The fact that we prove security under the random oracle model forces the use of such hash functions in the proposed schemes. See [38] for an explicit solution on secure hashing for BN curves.

3 Conjunctive PEKS scheme

The proposed scheme can be seen as an analog to Boneh et al.'s scheme [9] by replacing the symmetric computational-type hardness assumption with an asymmetric decisional-type one. This replacement by a stronger assumption allows one to take advantage of the bilinearity of pairings and build a conjunctive PEKS scheme with small trapdoors and indexes and efficient search process.

Following [14,20,21,30,31], we assume that the documents to be encrypted satisfy that

1. two different keyword fields never hold the same keyword, and
2. every keyword field is defined.

As noted in the literature [31], this can be effectively achieved by appending a keyword field identifier to every keyword. For instance, when encrypting a document of the form $(w_1, \ldots, w_n)$, one can assume that $w_i = i \| w_i'$ for some keyword w_i' (which could be NULL or $\perp$) and for all $i \in [m]$. We implicitly assume keywords in documents and trapdoors to be of this form.

Although not stated, every algorithm apart from Setup takes the public parameters as input.

Definition 7 We define a public-key encryption with conjunctive keyword search scheme $\mathcal{S}_1$ by means of the following five polynomial-time algorithms:

$\mathcal{S}_1$.Setup(λ): Given a security parameter $\lambda \in \mathbb{Z}$, fix two asymmetric bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order $q \geq 2^\lambda$ and denote the corresponding pairing by $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let g, h be random generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively. Let $H : \{0,1\}^* \to \mathbb{G}_1$ be a collision-free hash function. Define $m \in \mathbb{Z}$ as the fixed number of keywords in every document, which we assume constant in λ and satisfying $m \leq (1 + \log q)/2$. Output the public parameters $\mathrm{params} = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g, h, H, m\}$.

$\mathcal{S}_1$.KeyGen(): Choose $\beta \in \mathbb{F}_q$ uniformly at random. Output the private key β and the public key $\alpha = h^\beta$.

$\mathcal{S}_1$.BuildIndex$_\alpha(\mathbf{D})$: Denote by $\mathbf{D} = (w_1, \ldots, w_m)$ the input document consisting of a tuple of m keywords

$w_i \in \{0,1\}^*$. Then, uniformly generate a random nonce $r \in \mathbb{F}_q$ and set

$$I_0 = h^r$$
$$I_i = e\left(H(w_i), \alpha^r\right) \quad \text{for } i \in [m].$$

Output the index $\mathbf{I} = (I_0, I_1, \ldots, I_m)$.

$\mathcal{S}_1$.Trapdoor$_\beta(\mathbf{L}, J)$: Denote by $\mathbf{L} = (w_1, \ldots, w_l)$ the input tuple of keywords (where $l \leq m$ and $w_i \in \{0,1\}^*$) and by $J = \{j_1, \ldots, j_l\} \subseteq [m]$ the input set of positions. Set

$$T_0 = \left(\prod_{i=1}^{l} H(w_i)\right)^{\beta}.$$

Output the trapdoor $\mathbf{T}$, consisting of T_0 along with the fields J to be queried.

$\mathcal{S}_1$.Search$(\mathbf{I}, \mathbf{T})$: Given the index $\mathbf{I} = (I_0, I_1, \ldots, I_m)$ and the trapdoor $\mathbf{T} = (T_0, J)$, where $J = \{j_1, \ldots, j_l\}$, output 1 if

$$e(T_0, I_0) = \prod_{i=1}^{l} I_{j_i}.$$

Otherwise output 0.

We next give the consistency and security theorems for our scheme. The proofs are deferred to Sects. 4.1 and 4.2, respectively.

Theorem 1 *The proposed conjunctive PEKS scheme $\mathcal{S}_1$ is computationally consistent under the random oracle model.*

Theorem 2 *Assume the* DBDH *assumption holds. Then, the proposed conjunctive PEKS scheme $\mathcal{S}_1$ is semantically secure against adaptive chosen keyword attacks under the random oracle model.*

4 Consistency and security proofs for the conjunctive PEKS scheme $\mathcal{S}_1$

In this section, we present the consistency and the security proofs of our conjunctive PEKS scheme $\mathcal{S}_1$.

4.1 Consistency proof for the conjunctive PEKS scheme $\mathcal{S}_1$

We dedicate this section to the proof of Theorem 1. By proceeding in a similar way than in the proof by Abdalla et al. [1], we prove consistency of the scheme $\mathcal{S}_1$ in the random oracle model.

Let $\mathcal{A}$ be a PPT adversary in the consistency game defined in Sect. 2.3, having access to the public parameters, to the public key pk and to the hash oracle H modeled as a random oracle. Let WSet be the set of keywords queried to the hash oracle H throughout the game, whose size q_H is polynomial in λ. Let $\mathbf{D} = (w_1, \ldots, w_m)$, $\mathbf{L} = (w'_1, \ldots, w'_l)$ and $J = \{j_1, \ldots, j_l\} \subseteq [m]$ denote the guess of $\mathcal{A}$ in the *Guess* phase, where keywords are taken from WSet, and let $\tilde{J}$ be the set of positions $j_i \in J$ such that $w_{j_i} \neq w'_i$. Without loss of generality, we rule out adversaries choosing $\tilde{J} = \emptyset$ in the *Guess* phase. Let $r \in \mathbb{F}_q$ denote the random nonce generated by $\mathcal{A}$ in the encrypted index generation of the *Output* phase.

Denote $X = e\left(\prod_{i \in J} H(w_i), h^{\beta r}\right)$ and also denote $X' = e\left(\prod_{i=1}^{l} H(w'_i), h^{\beta r}\right)$. Now note that the output of $\mathcal{A}$ in the consistency game is 1 if and only if $X = X'$. We proceed to bound the probability of this event, which is $\mathrm{Adv}_{\mathcal{A}}(\lambda)$ by definition.

Let E be the event that there exist $\mathbf{D} = (w_1, \ldots, w_m)$, $\mathbf{L} = (w'_1, \ldots, w'_l)$ and $J = \{j_1, \ldots, j_l\} \subseteq [m]$, among all possible guesses taking words in WSet, in such a way that the equality $\prod_{i=1}^{l} H(w_{j_i}) = \prod_{i=1}^{l} H(w'_i)$ is satisfied. If $r\beta = 0$, then $\mathcal{A}$ always outputs 1. Otherwise, notice that $X = X'$ happens only when E happens. Therefore,

$$\mathrm{Adv}_{\mathcal{A}}(\lambda) \leq \frac{(q-1)^2}{q^2} \Pr(E) + \frac{2}{q}$$

Since $q \geq 2^\lambda$, it suffices to see that $\Pr(E)$ is negligible in λ.

Since H is modeled as a random oracle and since inversion permutes group elements, by using Lemma 1 we see that $\Pr(E) \leq q_H^{2m} \frac{m 2^{2m}}{q}$. This bound is negligible in λ, since $q \geq 2^\lambda$ and m, q_H are assumed to be constant and polynomial in λ, respectively.

As a consequence of this result, we conclude the proof of Theorem 1. We next state the lemma used above.

Lemma 1 *Let $\mathbb{G}$ be a finite group of order q and neutral element* 1. *Let m, n be positive integers with $m \leq (1 + \log q)/2$ and set $X_1, \ldots, X_n$ independent and identically distributed uniform random variables with support $\mathbb{G}$.*

Let $A_{n,2m}$ denote the event that there exists a $S \subseteq [n]$ with $|S| \leq 2m$ such that $\prod_{i \in S} X_i = 1$. Then, we have

$$\Pr\left(A_{n,2m}\right) \leq n^{2m} \frac{m 2^{2m}}{q}.$$

Proof To make the notation simpler, denote $A_{t,t}$ by A_t and set $t = 2m$. Notice that $A_{n,t}$ happens for $X_1, \ldots, X_n$ exactly when it happens for some subset of $X_1, \ldots, X_n$ with $\min(n, t)$ terms. Therefore, by the union bound

$$\Pr(A_{n,t}) \leq \binom{n}{t} \Pr(A_t) \leq n^t \Pr(A_t).$$

We now lower bound the probability of the complementary event A_t^c.

We first prove $\Pr(A_t^c) \geq \prod_{i=0}^{t-1} \frac{q-2^i}{q}$ by induction on t over positive integers. For $t = 1$ we have $\Pr(A_1^c) = \frac{q-1}{q}$. For $t > 1$ note that, for A_t^c to happen with $X_1, \ldots, X_t$, the event A_{t-1} must happen with $X_1, \ldots, X_{t-1}$ and X_t can not take as a value any of the inverses of the subproducts of $X_1, \ldots, X_{t-1}$. Therefore, there are at least $q - 2^{t-1}$ possible values for X_t such that A_t happens and we get that $\Pr(A_t^c) \geq \frac{q-2^{t-1}}{q} \Pr(A_{t-1}^c)$ as claimed.

Now we have

$$\Pr(A_t) \leq 1 - \prod_{i=0}^{t-1} \frac{q-2^i}{q} \leq 1 - \left(1 - \frac{2^{t-1}}{q}\right)^t.$$

Since $t \leq 1 + \log q$, we can bound this last expression by using the binomial inequality, obtaining $\Pr(A_t) \leq \frac{t2^{t-1}}{q}$, and the result is proved. □

4.2 Security proof for the conjunctive PEKS scheme $\mathscr{S}_1$

We dedicate this section to the proof of Theorem 2. As in [11], security is here proved in the random oracle model by means of a sequence of hybrid games.

Given two documents $\mathbf{D}_0 = (w_{0,1}, \ldots, w_{0,m})$ and $\mathbf{D}_1 = (w_{1,1}, \ldots, w_{1,m})$, let $\Delta \subseteq [m]$ denote the set of positions corresponding to keywords in $\mathbf{D}_0 \Delta \mathbf{D}_1$. For $j \in [m]$ let Δ_j denote the first $\min(j, |\Delta|)$ elements of Δ.

Let G_0 be the security game defined in Sect. 2.4.2. Given $j \in [m]$, we define a hybrid game G_j, differing from G_0 only in that the keywords in positions in Δ_j of the challenge index are chosen uniformly at random by the challenger.

Specifically, we introduce the security game G_j for $j \in [m]$, consisting of the following five phases:

- *Setup.* The challenger runs Setup and hands over the public parameters to the adversary. It also runs KeyGen, keeps the private key sk secret and hands over the public key pk to the adversary.
- *Query Phase 1.* The adversary adaptively asks the challenger for q_T trapdoors of its own choice, where q_T is a polynomial value in the security parameter λ. We denote the set of all keywords queried in this phase by $\mathscr{W}_1$.
- *Challenge.* The adversary outputs two challenge candidate documents $\mathbf{D}_0$, $\mathbf{D}_1$, subject to the restriction that keywords appearing in $\mathbf{D}_0 \Delta \mathbf{D}_1$ have not been queried in Query Phase 1. That is, $(\mathbf{D}_0 \Delta \mathbf{D}_1) \cap \mathscr{W}_1 = \emptyset$. The challenger throws a fair coin $b \in \{0, 1\}$ and computes the index $\mathbf{I} = (I_0, I_1, \ldots, I_m)$ corresponding to $\mathbf{D}_b$. Then, for every $i \in \Delta_j$, the challenger replaces I_i with uniformly sampled random elements from $\mathbb{G}_1$ and hands over this modified index to the adversary as the challenge.
- *Query Phase 2.* The adversary proceeds just as in Query Phase 1, but it is not allowed to ask for trapdoors containing keywords in $\mathbf{D}_0 \Delta \mathbf{D}_1$. That is, if the set of all keywords queried in this phase is $\mathscr{W}_2$, we impose $(\mathbf{D}_0 \Delta \mathbf{D}_1) \cap \mathscr{W}_2 = \emptyset$.
- *Guess.* The adversary outputs a guess $b' \in \{0, 1\}$ for b.

Let $\mathrm{Adv}_{\mathscr{A}, G_j}(\lambda)$ denote the advantage of the PPT adversary $\mathscr{A}$ in guessing b in the game G_j. It is clear that $\mathrm{Adv}_{\mathscr{A}, G_m}(\lambda)$ is negligible in λ for every PPT adversary $\mathscr{A}$ because in G_m the two challenge candidate documents share the same information with the challenge index.

Note that G_0 is identical to the security game defined in Sect. 2.4.2. We prove through Proposition 1 that the proposed conjunctive PEKS scheme $\mathscr{S}_1$ is semantically secure against adaptive chosen keyword attacks provided the DBDH assumption holds.

Proposition 1 *Assume that the* DBDH *assumption holds. For any $j \in \{0, \ldots, m-1\}$ and for any PPT adversary $\mathscr{A}$, the advantages of $\mathscr{A}$ in the games G_j and G_{j+1}, when using the scheme $\mathscr{S}_1$, are negligibly close in λ. That is,*

$$|\mathrm{Adv}_{\mathscr{A}, G_j}(\lambda) - \mathrm{Adv}_{\mathscr{A}, G_{j+1}}(\lambda)|$$

is negligible in λ.

Proof Let $\mathscr{A}$ be a PPT adversary. For every $j \in \{0, \ldots, m-1\}$, we build a PPT DBDH distinguisher $\mathscr{B}_j$ taking a DBDH challenge tuple $(g, g^a, g^b, h, h^a, h^c, v)$ as input and interacting with $\mathscr{A}$ as the challenger in the security game of the scheme.

The distinguisher $\mathscr{B}_j$ is built in such a way that, for tuples with $v = e(g, g)^{abc}$, $\mathscr{A}$ is playing the game G_j, and for tuples with v random $\mathscr{A}$ is playing the game G_{j+1}. The output of the DBDH distinguisher $\mathscr{B}_j$ depends on the output of $\mathscr{A}$.

- *Setup.* The challenger $\mathscr{B}_j$ runs $\mathscr{S}_1$.Setup(λ) to obtain params $= \{\mathbb{G}_1, \mathbb{G}_2, q, e, g, h, H, m\}$ the public parameters of the scheme, where H is the hash oracle described below. $\mathscr{B}_j$ hands over the public parameters to $\mathscr{A}$.
- *Keygen.* The challenger $\mathscr{B}_j$ hands over the public key h^a to $\mathscr{A}$.
- *Hash Oracle.* The hash oracle H is operated by $\mathscr{B}_j$, and it maintains a list of tuples of the form $\langle w, s, c \rangle$ with $w \in \{0, 1\}^*$, $s \in \mathbb{F}_q$ and $c \in \{0, 1\}$. The list is initially empty. On input a keyword $w \in \{0, 1\}^*$, the oracle H operates as follows:
  1. If there is an item in the list whose first element is keyword w, denote it by $\langle w, s, c \rangle$. Then:

(a) If $c = 0$, the oracle returns g^s.
(b) If $c = 1$, the oracle returns $\left(g^b\right)^s$.

2. If there is no item in the list whose first element is keyword w, then the oracle flips a coin $c \in \{0, 1\}$ with $\Pr(c = 1) = 1/(2q_T m + 1)$, samples $s \in \mathbb{F}_q$ uniformly at random and inserts $\langle w, s, c\rangle$ into the list. Then, it proceeds to give an output as in the previous point.

– *Query Phase 1.* When $\mathscr{A}$ requests a trapdoor for keywords $\mathbf{L} = (w_1, \ldots, w_l)$ in the set of keyword fields $J = \{j_1, \ldots, j_l\}$, the algorithm $\mathscr{B}_j$ first calls the oracle on input each keyword w_i and retrieves the associated oracle list tuples $\langle w_i, s_i, c_i\rangle$. Then, if some coin flip $c_i = 1$, $\mathscr{B}_j$ halts. Otherwise, $\mathscr{B}_j$ hands over to $\mathscr{A}$ the trapdoor $\mathbf{T}$ consisting of $T_0 = \prod_{i=1}^{l} (g^a)^{s_i}$ and J.
– *Challenge.* In this phase, the adversary $\mathscr{A}$ outputs two documents $\mathbf{D}_0 = (w_{0,1}, \ldots, w_{0,m})$, $\mathbf{D}_1 = (w_{1,1}, \ldots, w_{1,m})$ with the restrictions stated in the security game defined in Sect. 2.4.2 and above, and $\mathscr{B}_j$ throws a fair coin $b \in \{0, 1\}$.
Then, $\mathscr{B}_j$ calls the hash oracle on every keyword $w_{b,i}$ to fill the H-list with tuples $\langle w_{b,i}, s_{b,i}, c_{b,i}\rangle$. The algorithm $\mathscr{B}_j$ halts if:
  – For some $i \in [m]\backslash\Delta_{j+1}$ we have $c_{b,i} = 1$, or
  – $c_{b,t} = 0$, where $\{t\} = \Delta_{j+1}\backslash\Delta_j$.

  Then $\mathscr{B}_j$ samples a value $r \in \mathbb{F}_q$ uniformly at random and computes the challenge $\mathbf{I} = (I_0, I_1, \ldots, I_m)$ in the following way

$$I_0 = h^r,$$
$$I_i = \begin{cases} \text{unif. sampled from } \mathbb{G}_T & \text{if } i \in \Delta_j \\ v^{r s_{b,i}} & \text{if } i \in \Delta_{j+1}\backslash\Delta_j \neq \emptyset \\ e\left((g^a)^r, (h^c)^{s_{b,i}}\right) & \text{if } i \in [m]\backslash\Delta_{j+1} \neq \emptyset \end{cases}$$

  and hands over $\mathbf{I}$ to $\mathscr{A}$.
– *Query Phase 2.* $\mathscr{B}_j$ proceeds as in Query Phase 1.
– *Guess.* The adversary $\mathscr{A}$ outputs a guess $b' \in \{0, 1\}$ for b. If $b = b'$, $\mathscr{B}_j$ outputs 1, and if $b \neq b'$, $\mathscr{B}_j$ outputs 0.

Since the DBDH assumption holds, $\mathrm{Adv}_{\mathscr{B}_j}(\lambda)$ must be negligible in λ. But

$$\begin{aligned} \mathrm{Adv}_{\mathscr{B}_j}(\lambda) &= |\Pr(\mathscr{B}_j(X) = 1 | X = 1) \\ &\quad - \Pr(\mathscr{B}_j(X) = 1 | X = 0)| \\ &= \Pr(\mathscr{B}_j \text{ does not halt}) \\ &\quad \cdot |\mathrm{Adv}_{\mathscr{A},G_j}(\lambda) - \mathrm{Adv}_{\mathscr{A},G_{j+1}}(\lambda)|. \end{aligned}$$

By Lemma 2, $\Pr(\mathscr{B}_j \text{ does not halt})$ is non-negligible in λ, and the result is proved. □

As a consequence of this result, we conclude the proof of Theorem 2. We next state and prove the lemma referenced in the proof of Proposition 1, which is an adaptation of a result in [9].

Lemma 2 ([9]) *The probability that algorithm $\mathscr{B}_j$ does not halt is non-negligible in the security parameter λ.*

Proof We split the calculations between the query phases and the challenge phase.

In each of the query phases, we allow $\mathscr{A}$ to ask for a polynomial amount q_T (in λ) of trapdoor queries. This amounts to throwing at most $2mq_T$ coins c with $\Pr(c = 1) = 1/(2q_T m + 1)$. Since $\mathscr{B}_j$ does not halt exactly when each and every one of these throws outcome is 0, we have

$$\Pr(\mathscr{B}_j \text{ does not halt in query phases}) \geq \left(1 - \frac{1}{2mq_T + 1}\right)^{2mq_T} \geq 1/e,$$

which is non-negligible in λ.

For the challenge phase, $\mathscr{B}$ does not halt exactly when the coin throw corresponding to the keyword in position $\Delta_{j+1}\backslash\Delta_j$ (if nonempty) of the chosen challenge document is 1 and the coin throws corresponding to the keywords in positions in $[m]\backslash\Delta_{j+1}$ of the chosen challenge document are all 0. Since, if $\mathbf{D}_0 \neq \mathbf{D}_1$ then $|[m]\backslash\Delta_{j+1}| \leq m - 1$, we have:

$$\Pr(\mathscr{B}_j \text{ does not halt in the challenge phase}) \geq \left(1 - \frac{1}{2mq_T + 1}\right)^{m-1} \frac{1}{2mq_T + 1} \geq \frac{1}{e}\frac{1}{2mq_T + 1},$$

which is non-negligible in λ since m is constant in λ and q_T is polynomial in λ, and we get the stated lemma. □

5 Subset PEKS scheme

The second PEKS scheme we propose enables a class of subset queries. This class includes subset queries as defined in [11].

Subset queries, as understood by [11], are specified by an ordered tuple of m sets of keywords $(A_1, \ldots, A_m)$. Then, a document $\mathbf{D} = (w_1, \ldots, w_m)$ satisfies such a query if and only if the predicate $(w_1 \in A_1) \wedge \cdots \wedge (w_m \in A_m)$ holds. The scheme we propose considers subsets in a partition of $\mathbf{D}$ instead of keywords w_i in this last predicate.

More concretely, in the setup algorithm we fix a partition $J_1, \ldots, J_m$ of $[m]$. Given a document $\mathbf{D} = (w_1, \ldots, w_m)$, write $B_i = \{w_j\}_{j \in J_i}$ for every $i \in [m]$. Given a query $\mathbf{L} = (w'_1, \ldots, w'_l)$, $J = \{j_1, \ldots, j_l\}$, where J is written in increasing order, consider $A_i = \{w'_k\}_{j_k \in J_i}$ for every $i \in [m]$. Then, the document $\mathbf{D}$ satisfies the query $\mathbf{L}$, J if and only if

the predicate $(B_1 \subseteq A_1) \wedge \cdots \wedge (B_m \subseteq A_m)$ holds. Note that we also admit empty keyword fields in documents, which are denoted by keywords $\perp$.

For the sake of clarity, before formally stating the proposed construction, we give a brief example illustrating the internal workings of the scheme.

The Setup algorithm of the scheme fixes a tuple of possibly repeated field identifiers $(f_1, \ldots, f_m)$. We take $m = 8$ and $(f_1, \ldots, f_8) = (1, 1, 1, 2, 2, 2, 3, 3)$ as an example.

When encrypting documents in BuildIndex, the documents $\mathbf{D} = (w_1, \ldots, w_m)$ can be thought of as a collection of sets of keywords, where keywords in positions having the same field identifier belong to the same set. Also, the keyword $\perp$ is allowed, and it stands for a null entry. For instance, following the example above, the document $\mathbf{D} = (w_1, w_2, w_3, w_4, w_5, \perp, w_7, \perp)$ can be thought of as the following collection of sets $(\{w_1, w_2, w_3\}, \{w_4, w_5\}, \{w_7\})$.

When generating trapdoors, we input a query consisting of a tuple of keywords $\mathbf{L} = (w_1, \ldots, w_l)$ and a set of positions $J = \{j_1, \ldots, j_l\}$ written in increasing order. As above, keywords in positions having the same field identifier are thought to belong to the same set. Thus, in the example above, the query for words $\mathbf{L} = (w'_1, w'_2, w'_3, w'_4, w'_5, w'_6, w'_7)$ at positions $J = \{1, 2, 3, 4, 5, 6, 8\}$ can be thought of as the collection of sets $(\{w'_1, w'_2, w'_3\}, \{w'_4, w'_5, w'_6\}, \{w_7\})$.

Now, a document matches a query in the Search algorithm exactly when the sets of keywords defined by the document are contained in the sets of keywords defined by the query, in a sequential way. That is, following the example above, a match happens exactly when

$$(\{w_1, w_2, w_3\} \subseteq \{w'_1, w'_2, w'_3\}) \wedge (\{w_4, w_5\} \subseteq \{w'_4, w'_5, w'_6\}) \wedge (\{w_7\} \subseteq \{w'_7\}).$$

We now describe the proposed subset PEKS scheme. Although not stated, every algorithm apart from Setup takes the public parameters as input.

Definition 8 We define a public-key encryption with subset keyword search scheme $\mathscr{S}_2$ by means of the following five polynomial-time algorithms:

$\mathscr{S}_2$.Setup(λ): Given a security parameter $\lambda \in \mathbb{Z}$, fix a symmetric bilinear group $\mathbb{G}$ of prime order $q \geq 2^\lambda$ and denote the corresponding pairing by $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let g be a random generator of $\mathbb{G}$. Let $H : \{0,1\}^* \to \mathbb{G}$ and $H_1 : \mathbb{G}_T \to \{0,1\}^*$ be collision-free hash functions. Set $m \in \mathbb{Z}$ the maximum number of keywords in every document, which we assume constant in λ and satisfying $m \leq (1 + \log q)/2$. Define a tuple $(f_1, \ldots, f_m)$ of possibly repeated field identifiers describing which field does each word in the documents belong to, where each $f_i \in [m]$.
Output params $= \{\mathbb{G}, q, e, g, H, H_1, m, (f_1, \ldots, f_m)\}$ the public parameters of the scheme.

$\mathscr{S}_2$.KeyGen(): Choose $a \in \mathbb{F}_q$ uniformly at random.
Output the private key $\beta = (\beta_i)_{i=1}^m$ and the public key $\alpha = (\alpha_i)_{i=1}^m$, where $\beta_i = a^{-i}$ and $\alpha_i = g^{a^i}$.

$\mathscr{S}_2$.BuildIndex$_\alpha(\mathbf{D})$: Given as input $\mathbf{D} = (w_1, \ldots, w_m)$ the document consisting of a tuple of m keywords w_i in the domain $\{0,1\}^* \cup \{\perp\}$, generate $r_1, \ldots, r_m \in \mathbb{F}_q$ uniform random nonces in such a way that $f_i = f_j$ implies $r_i = r_j$.
Set

$$I_0 = H_1\left(e\left(\prod_{i \in [m]: w_i \neq \perp} H(w_i)^{r_i}, g\right)\right)$$
$$I_i = \alpha_i^{r_i} \quad \text{for } i \in [m].$$

Output the index $\mathbf{I} = (I_0, I_1, \ldots, I_m)$.

$\mathscr{S}_2$.Trapdoor$_\beta(\mathbf{L}, J)$: Given $\mathbf{L} = (w_1, \ldots, w_l)$ the input tuple of keywords with $l \leq m$, and the set of keyword fields $J = \{j_1, \ldots, j_l\} \subseteq [m]$ written in increasing order, set

$$T_i = H(w_i)^{\beta_{j_i}} \quad \text{for } i \in \{1, \ldots, l\}.$$

Output the trapdoor $\mathbf{T}$, consisting of $T_1, \ldots, T_l$ along with the fields J to be queried.

$\mathscr{S}_2$.Search$(\mathbf{I}, \mathbf{T})$: Denote the index $\mathbf{I}$ by $\mathbf{I} = (I_0, I_1, \ldots, I_m)$ and the trapdoor $\mathbf{T}$ by $\mathbf{T} = (T_0, \{j_1, \ldots, j_l\})$. For every $t \in [m]$, let J_t denote the set of elements $i \in [l]$ such that $f_{j_i} = t$. For every $i \in [l]$, compute $v_i = e\left(T_i, I_{j_i}\right)$.
Output 1 if there exists subsets $J'_t \subseteq J_t$ for $t \in [m]$ such that

$$I_0 = H_1\left(\prod_{t=1}^{m} \prod_{i \in J'_t} v_i\right).$$

Otherwise output 0.

In the following example, we describe an application of our subset PEKS scheme. We follow the e-mail gateway scenario mentioned in Sect. 1 and proposed by Boneh et al. [9].

Example 1 Consider a user Alice that reads her e-mail on various devices (such as laptop, smartphone and desktop). Suppose that each message is tagged with a sequence of at most four keywords to aid classification. Further suppose that the first tag defines the priority of the message (e.g., "urgent" or "low_priority") and that the last three describe its category

(e.g., "social", "advertising" or "work"), so e-mails have the structure

message‖priority_tag‖cat_tag_1‖cat_tag_2‖cat_tag_3.

Alice receives her e-mail through a gateway, who distributes all messages to her devices according to the attached tags. Due to privacy reasons, Alice does not wish her e-mail gateway to be able to read her e-mail messages nor to have any knowledge of the attached tags. However, she still wants her e-mail gateway to classify and distribute messages to her devices correctly. Hence, she sets up a public-key encryption scheme (Gen, Enc, Dec) and disseminates her public-key material, so that senders can send her messages in an encrypted form.

In this context, our subset PEKS scheme can be used to encrypt the keyword tags.

To set up our subset PEKS scheme in the described setting, Alice would first execute $\mathscr{S}_2$.Setup(λ) and set $m = 4$ and $(f_1, \ldots, f_4) = (1, 2, 2, 2)$. Then, she would generate the public key α and the private key β by calling $\mathscr{S}_2$.Keygen() and disseminate the public key.

Suppose that a user Bob wants to send Alice an e-mail message M with low priority to schedule a work meeting. He thus chooses the priority tag "low_priority" and the category tags "work" and "meeting". He can then generate the index

$$\mathbf{I} = \mathscr{S}_2.\text{BuildIndex}_\alpha((\text{low\_priority}, \text{work}, \text{meeting}, \perp))$$

and send $\text{Enc}(M)\|\mathbf{I}$ to her e-mail gateway.

Now, suppose that Alice wants to restrict the e-mails she receives on her smartphone to personal e-mails, urgent work e-mails and work e-mails for scheduling meetings. To do so, she can send the following trapdoors to her e-mail gateway

$$\mathbf{T} = \mathscr{S}_2.\text{Trapdoor}_\beta((\text{personal}), \{2\})$$
$$\mathbf{T}' = \mathscr{S}_2.\text{Trapdoor}_\beta((\text{urgent}, \text{work}), \{1, 2\})$$
$$\mathbf{T}'' = \mathscr{S}_2.\text{Trapdoor}_\beta((\text{work}, \text{meeting}), \{2, 3\}).$$

When receiving an encrypted e-mail of the form $\text{Enc}(M)\|\mathbf{I}$, Alice's e-mail gateway is trusted to forward it to her smartphone exactly when any of the evaluations $\mathscr{S}_2$.Search($\mathbf{I}, \mathbf{T}$), $\mathscr{S}_2$.Search($\mathbf{I}, \mathbf{T}'$) and $\mathscr{S}_2$.Search($\mathbf{I}, \mathbf{T}''$) return 1.

We next give the consistency and security theorems for our scheme. The proofs are deferred to Sects. 6.1 and 6.2, respectively.

Theorem 3 *The proposed subset PEKS scheme $\mathscr{S}_2$ is computationally consistent under the random oracle model.*

Theorem 4 *Assume that the $(m + 1)$ − BDHI assumption holds. Then, the proposed subset PEKS scheme $\mathscr{S}_2$ is semantically secure against adaptive chosen keyword attacks under the random oracle model.*

6 Consistency and security proofs for the subset PEKS scheme $\mathscr{S}_2$

In this section, we give the consistency and security proofs for the subset PEKS scheme $\mathscr{S}_2$.

6.1 Consistency proof for the subset PEKS scheme $\mathscr{S}_2$

We dedicate this section to the proof of Theorem 3.

We now prove consistency in the random oracle model of the scheme $\mathscr{S}_2$ in a similar fashion than in the proof by Abdalla et al. [1].

Let $\mathscr{A}$ be a PPT adversary in the consistency game defined in Sect. 2.3, which has access to the public parameters, to the public key pk and to the hash oracles H, H_1 modeled as random oracles. Let WSet, TSet be the sets of polynomial (in λ) size q_H, q_{H_1} which consist of keywords queried to the hash oracles H, H_1 throughout the game, respectively. Write $(f_1, \ldots, f_m)$ the tuple of field identifiers in the public parameters. Let $\mathbf{D} = (w_1, \ldots, w_m)$ and $\mathbf{L} = (w'_1, \ldots, w'_l)$ and $J = \{j_1, \ldots, j_l\} \subseteq [m]$ (written in increasing order) denote the guess of $\mathscr{A}$ in the *Guess* phase.

For $j \in [m]$, let $\mathbf{D}_{f_j}$ denote the set of keywords in $\mathbf{D}$ at positions having field identifier f_j. Let $\tilde{J}$ be the set of positions $j_i \in J$ such that $w'_i \nsubseteq \mathbf{D}_{f_{j_i}}$. Without loss of generality, we rule out adversaries choosing $\tilde{J} = \emptyset$ in the *Guess* phase. Let $r_1, \ldots, r_m \in \mathbb{F}_q$ denote the random nonces generated by $\mathscr{A}$ in the encrypted index generation of the *Output* phase.

Denote $X = e\left(\prod_{i\in[m]:w_i\neq\perp} H(w_i)^{r_i}, g\right)$ and, given a subset $J' \subset J$, denote $X' = e\left(\prod_{j_i\in J'} H(w'_i)^{r_{j_i}}, g\right)$. Now note that the output of $\mathscr{A}$ in the consistency game is 1 if and only if $X = X'$ or $H_1(X) = H_1(X')$ for some $J' \subseteq J$ with $J' \cap \tilde{J} \neq \emptyset$.

Let E denote the event that there exist $\mathbf{D} = (w_1, \ldots, w_m)$, $\mathbf{L} = (w'_1, \ldots, w'_l)$ and $J = \{j_1, \ldots, j_l\} \subseteq [m]$, among all possible guesses taking words in WSet in such a way that we have $\prod_{i\in[m]:w_i\neq\perp} H(w_i)^{r_i} = \prod_{i\in[l]} H(w'_i)^{r_{j_i}}$. Likewise, let E_1 be the event that there exist $T, T' \in$ TSet in such a way that $H_1(T) = H_1(T')$.

If all $r_1, \ldots, r_m \neq 0$, then $X = X'$ has nonzero probability of happening only when E happens (note that by ranging over all possible J we remove the need to include the J' above in the argument). Likewise $H_1(X) = H_1(X')$ has nonzero probability of happening only when E_1 happens. Therefore,

$$\text{Adv}_{\mathscr{A}}(\lambda) \leq \left(1 - \frac{m}{q}\right)(\Pr(E) + \Pr(E_1)) + \frac{m}{q}.$$

Since $q \geq 2^\lambda$ and m is constant in λ, it suffices to prove that $\Pr(E)$ and $\Pr(E_1)$ are negligible in λ.

By computing the probability of the complementary and using the binomial inequality, we see that $\Pr(E_1) \leq q_{H_1}^2/q$. Now, since H is modeled as a random oracle and since inversion permutes group elements, by using Lemma 1 we see that $\Pr(E) \leq q_H^{2m} \frac{m 2^{2m}}{q}$. The obtained bounds are indeed negligible in λ, since $q \geq 2^\lambda$, and m, q_H are assumed to be constant and polynomial in λ, respectively.

As a consequence of this result, we conclude the proof of Theorem 3.

6.2 Security proof for the subset PEKS scheme $\mathscr{S}_2$

We dedicate this section to the proof of Theorem 4.

We prove security in the random oracle model by following a similar technique than [9].

Suppose that there exists a PPT adversary $\mathscr{A}$ breaking the security game defined in Sect. 2.4.2 with advantage not negligible in λ. We then build a successful PPT $(m+1)-$BDHI distinguisher $\mathscr{B}$ taking an $(m+1)-$BDHI challenge tuple $(g, g^a, \ldots, g^{a^{m+1}})$ as input. By interacting with $\mathscr{A}$ as the challenger in the security game defined in Sect. 2.4.2, $\mathscr{B}$ computes $e(g,g)^{1/a}$ with non-negligible advantage in λ.

- *Setup.* The challenger $\mathscr{B}$ runs Setup(λ) to generate the public parameters of the scheme

  $$\text{params} = \{\mathbb{G}, q, e, g, H, H_1, m, (f_1, \ldots, f_m)\},$$

  where H, H_1 are handles to the hash oracles described below. $\mathscr{B}$ hands over the public parameters to $\mathscr{A}$.
- *Keygen.* The challenger $\mathscr{B}$ hands over the public key $(g^a, \ldots, g^{a^m})$ to $\mathscr{A}$.
- *Hash Oracle H.* The oracle is operated by $\mathscr{B}$, which maintains a list of tuples of the form $\langle w, s, c\rangle$ with $w \in \{0,1\}^*$, $s \in \mathbb{F}_q$ and $c \in \{0,1\}$. The list is initially empty. On input a keyword $w \in \{0,1\}^*$, the oracle H operates as follows:
  1. If there is an item in the list whose first element is keyword w, denote it by $\langle w, s, c\rangle$. Then:
     (a) If $c = 0$, the oracle returns g^s.
     (b) If $c = 1$, the oracle returns $\left(g^{a^{m+1}}\right)^s$.
  2. If there is no item in the list whose first element is keyword w, then the oracle flips a coin $c \in \{0,1\}$ with $\Pr(c=1) = 1/(2q_T m + 1)$, samples $s \in \mathbb{F}_q$ uniformly at random and inserts $\langle w, s, c\rangle$ into the list. Then, it proceeds to give an output as in the previous point.
- *Hash Oracle H_1.* The oracle is operated by $\mathscr{B}$, which maintains a list of tuples of the form $\langle t, V\rangle$ with $t \in \mathbb{G}_T$ and $V \in \{0,1\}^*$. The list is initially empty. On input an element $t \in \mathbb{G}_T$, the oracle H_1 operates as follows:
  1. If there is an item in the list whose first element is t, denote it by $\langle t, V\rangle$. The oracle returns V.
  2. If there is no item in the list whose first element is t, then the oracle samples $V \in \mathbb{G}_T$ uniformly at random and inserts $\langle t, V\rangle$ into the list. Then, it proceeds to give an output as in the previous point.
- *Query Phase 1.* When $\mathscr{A}$ requests a trapdoor for keywords $\mathbf{L} = (w_1, \ldots, w_l)$ in positions $J = \{j_1, \ldots, j_l\}$ written in increasing order, the algorithm $\mathscr{B}$ first calls the H oracle on input each keyword w_i and retrieves the associated oracle list tuples $\langle w_i, s_i, c_i\rangle$. Then, if some coin flip $c_i = 1$, $\mathscr{B}$ halts. Otherwise, $\mathscr{B}$ hands over to $\mathscr{A}$ the trapdoor $\mathbf{T}$ consisting of $T_i = (g^{a^{m-j_i+1}})^{s_i}$ for $i \in \{1, \ldots, l\}$, and J.
- *Challenge.* The adversary outputs two documents $\mathbf{D}_0 = (w_{0,1}, \ldots, w_{0,m})$, $\mathbf{D}_1 = (w_{1,1}, \ldots, w_{1,m})$ with the restrictions stated in the security game defined in Sect. 2.4.2, and $\mathscr{B}$ throws a fair coin $b \in \{0,1\}$.
  Then, $\mathscr{B}$ calls the hash oracle on every keyword $w_{b,i}$ to fill the H-list with tuples $\langle w_{b,i}, s_{b,i}, c_{b,i}\rangle$. The algorithm $\mathscr{B}$ halts if some $c_{b,i} = 1$.
  Then $\mathscr{B}$ uniformly chooses $J \in \mathbb{G}_T$ and random nonces $r_1, \ldots, r_m \in \mathbb{F}_q$ in such a way that if $f_i = f_j$ then $r_i = r_j$, and it computes the challenge $\mathbf{I} = (I_0, I_1, \ldots, I_m)$ in the following way

  $$I_0 = J, \quad I_i = \left(g^{a^{i-1}}\right)^{r_i}$$

  and hands over $\mathbf{I}$ to $\mathscr{A}$. In addition, $\mathscr{B}$ halts if $\sum_i s_{b,i} r_i \equiv 0 \pmod q$, and if not, it stores $C = (\sum_i s_{b,i} r_i)^{-1} \pmod q$.
- *Query Phase 2.* $\mathscr{B}$ proceeds as in Query Phase 1.
- *Guess.* The adversary $\mathscr{A}$ outputs a guess $b' \in \{0,1\}$ for b. Then, $\mathscr{B}$ picks a random element $\langle t, V\rangle$ from the list in H_1, and returns t^C.

Note that the challenge is well-formed and that it implicitly imposes the equality $J = H_1\left(e\left(g^{\sum_i s_{b,i} r_i}, g\right)^{1/a}\right)$. If $\mathscr{B}$ does not halt, then it perfectly simulates a real attack game up until the moment when $\mathscr{A}$ issues an H_1 oracle query for $t_0 = e\left(g^{\sum_i s_{0,i} r_i}, g\right)^{1/a}$ or $t_1 = e\left(g^{\sum_i s_{1,i} r_i}, g\right)^{1/a}$.

Let $\mathscr{E}$ denote the event that $\mathscr{A}$ issues a query for t_0 or t_1 in a real attack game. We now lower bound the probability of $\mathscr{E}$. Under the random oracle model, if $\mathscr{E}$ does not happen, then $\mathscr{B}$ does not reveal any information about b. Therefore,

$$\begin{aligned}\Pr(b' = b) &= \Pr(\mathscr{E})\Pr(b' = b|\mathscr{E}) + \frac{1}{2}\Pr(\neg\mathscr{E}) \\ &= \frac{1}{2} + \Pr(\mathscr{E})\left(\Pr(b' = b|\mathscr{E}) - \frac{1}{2}\right)\end{aligned}$$

so we can express the advantage of $\mathscr{A}$ by

$$\begin{aligned}\mathrm{Adv}_{\mathscr{A}}(\lambda) &= \left|\Pr(b' = b) - \frac{1}{2}\right| \\ &= \Pr(\mathscr{E}) \cdot \left|\Pr(b' = b|\mathscr{E}) - \frac{1}{2}\right| \leq \frac{1}{2}\Pr(\mathscr{E})\end{aligned}$$

and $\Pr(\mathscr{E}) \geq 2\mathrm{Adv}_{\mathscr{A}}(\lambda)$.

Now, suppose that

1. $\mathscr{B}$ does not abort,
2. $\mathscr{A}$ eventually issues an H_1 oracle query for either t_0 or t_1, i.e., $\mathscr{E}$ happens, and
3. $\mathscr{B}$ chooses b such that $\mathscr{A}$ queries the H_1 oracle for t_b (this is well defined, since $\mathscr{A}$ does not receive any information about b until $\mathscr{E}$ happens).

Then, if $\mathscr{B}$ calls the hash oracle H_1 on input t_b in the Guess phase, it successfully computes $e(g, g)^{1/a}$. Since $\mathscr{B}$ uniformly samples an element from all inputs processed by the hash oracle H_1 to generate its output in the *Guess* phase, the probability of $\mathscr{B}$ breaking the $(m+1)-$BDHI assumption in the above situation is at least $1/q_{H_1}$, where q_{H_1} is the polynomial amount of queries issued to the H_1 oracle. This implies

$$\begin{aligned}\mathrm{Adv}_{\mathscr{B}}(\lambda) &= \Pr\left(\mathscr{B}\left(g, \ldots, g^{a^{m+1}}\right) = e(g, g)^{1/a}\right) \\ &\geq \frac{1}{q_{H_1}}\Pr(\mathscr{B} \text{ does not abort})\Pr(\mathscr{E})\frac{1}{2} \\ &\geq \frac{1}{q_{H_1}}\Pr(\mathscr{B} \text{ does not abort})\mathrm{Adv}_{\mathscr{A}}(\lambda).\end{aligned}$$

By following the same argument as in Lemma 2, we see that $\Pr(\mathscr{B}$ does not abort$)$ is non-negligible in λ. Since $\mathscr{A}$ breaks the security game defined in Sect. 2.4.2 with non-negligible advantage and q_{H_1} is polynomial in λ, we conclude that $\mathrm{Adv}_{\mathscr{B}}(\lambda)$ is non-negligible as well.

As a consequence of this result, we conclude the proof of Theorem 4.

7 Efficiency analysis

We next lay out the efficiency measures of the proposed schemes and of other similar searchable encryption schemes. We state the size and the time needed to generate an encrypted index and a trapdoor and also the time taken to perform a search operation. We omit multiplication time, hash evaluation time, key setup time, field identifiers size and key storage size in the efficiency analysis. Notice that the search time refers to performing a search operation for a single encrypted index. Note that this is the case of application examples stated in Sect. 1. Thus, the search time over a set of encrypted indexes scales linearly in the number of encrypted indexes.

To analyze performance, we implement our schemes $\mathscr{S}_1$ and $\mathscr{S}_2$ and the schemes in [31] by using the PBC library [27], and we provide the estimated running times for each algorithm. All simulations ran on an Intel® $Core^{\text{TM}}$ i7-4510U CPU at 2.00 GHz and 8 GB memory under Ubuntu 16.04.1 LTS.

For the sake of comparison, we use symmetric bilinear groups (*type A* pairings in the PBC library documentation) in all implementations. Also, as suggested in the PBC library documentation, we fix a 512-bit base field order and a 160-bit group order to instantiate the bilinear group. In our implementations, we did not use preprocessing or any of the functions that the PBC library provides in order to speed up computations.

7.1 Conjunctive PEKS scheme

Since we restrict the analysis to conjunctive PEKS schemes, the schemes [3,16,32,33,35] lie out of the scope of the analysis. However, we include the single-keyword PEKS scheme [9] by Boneh et al. in our analysis for the sake of comparison, extending it to the conjunctive case by considering the concatenation of indexes and trapdoors and the sequential evaluation of the Search algorithm. Note that, for the single-keyword case $m = l = 1$, our scheme and [9] have similar efficiency marks.

We also restrict the analysis to the public-key setting, so we leave out schemes such as [4,13,14]. Other schemes such as [20,42] are omitted due to security considerations.

The size and time efficiency measures can be found in Tables 1 and 2, respectively.

Note that the proposed scheme achieves the lowest efficiency marks for index size, trapdoor size, trapdoor generation time and search time. This is not so for the index generation time, which is constrained by the computing power of the senders. However, in several applications, the search time and the index and trapdoor size measures are far

Table 1 Size efficiency comparison of conjunctive PEKS schemes

Scheme	Index size	Trapdoor size
[9] BDOP	$mE + mF$	lE
[31] PKL I	$2E + mF$	E
[31] PKL II	$2mE$	$2E$
[30] PCL	$(2m + 1)E$	$2E + F$
[11] BW	$(2m + 2)E$	$(2l + 1)E$
[21] HL	$(m + 2)E$	$3E$
$\mathscr{S}_1$	$E + mF$	E

m: number of keywords in the index. l: number of keywords in the trapdoor. E: size of elliptic curve point. F: size of finite field element

Table 2 Time efficiency comparison of conjunctive PEKS schemes

Scheme	Index time	Trapdoor time	Search time
[9] BDOP	$me + 2mG$	lG	le
[31] PKL I	$me + (m + 2)G$	G	$e + G$
[31] PKL II	$(4m + 1)G + X$	$2G$	$2e + G$
[30] PCL	$(4m + 1)G + 2X$	$2G$	$2e$
[11] BW	$(6m + 2)G$	$(5l + 1)G$	$(2l + 1)e$
[21] HL	$(2m + 2)G$	$3G$	$3e$
$\mathscr{S}_1$	$me + 2G_2$	G_1	e

m: number of keywords in the index. l: number of keywords in the trapdoor. e: pairing evaluation time. G: exponentiation time in symmetric bilinear group $\mathbb{G}$. G_1: exp. time in asymmetric bilinear group $\mathbb{G}_1$. G_2: exp. time in asymmetric bilinear group $\mathbb{G}_2$. X: exponentiation time in finite field

Table 3 Performance analysis of conjunctive PEKS schemes

Scheme	Index time (ms)	Trapdoor time (ms)	Search time (ms)
[9] BDOP	58.7	36.5	7.33
[31] PKL I	53.1	29.7	2.73
[31] PKL II	59.8	3.08	3.26
$\mathscr{S}_1$	35.9	26.5	0.94

more critical, since they are constrained by the throughput of the network and by the computing power of the storage server, and search operations may be executed more than once per encrypted index.

In Table 3, we give the estimated running times in milliseconds for Table 2 by arbitrarily fixing $m = 8$ and $l = 8$ and using 48-bit keywords. For implementation reasons, we do not give the performance analysis for the schemes in [21,30], mainly because they require the evaluation of multiple independent hash functions. We also omit the analysis of the conjunctive PEKS scheme in [11] for efficiency reasons.

We observe that $\mathscr{S}_1$ achieves the best index computation and search time in the studied case. Also, the second scheme in [31] gives the best trapdoor computation time. This is achieved in [31] by removing the need for an *admissible encoding scheme*, thus replacing products in the bilinear group by sums in the underlying finite field.

7.2 Subset PEKS scheme

We now give the efficiency measures for the proposed subset PEKS scheme and the related subset PEKS scheme by Boneh and Waters [11]. For the sake of comparison, we assume that queries are always the ones supported by [11]. See the beginning of Sect. 5 for more details.

One of the main differences between both schemes is that in the proposed scheme the keyword space is an arbitrary exponential-sized keyword space $\{0, 1\}^*$, while in [11], keywords are taken from a finite polynomial-sized keyword space. We denote by n the size of this keyword space in the efficiency analysis. Another difference is that, in the scheme we propose, the number of keywords in queries is limited at $\mathscr{S}_2$.Setup. We denote by L the maximum number of keywords in a trapdoor. The size and time efficiency measures can be found in Tables 4 and 5, respectively.

Table 4 Size efficiency comparison between subset PEKS schemes

Scheme	Index size	Trapdoor size
[11] BW	$(2nm + 2)E$	$(2nm + 1)E$
$\mathscr{S}_2$	$LE + H$	lE

n: size of the keyword space. m: number of keywords in the index. l: number of keywords in the trapdoor. L: maximum number of keywords in a trapdoor. E: size of elliptic curve point. F: size of finite field element. H: size of strings in $\mathrm{Im}(H_1) \subseteq \{0, 1\}^*$

Table 5 Time efficiency comparison between subset PEKS schemes

Scheme	Index time	Trapdoor time	Search time
[11] BW	$(6nm + 2)G$	$(5nm + 1)G$	$(2nm + 1)e$
$\mathscr{S}_2$	$e + 2LG$	lG	le

n: size of the keyword space. m: number of keywords in the index. l: number of keywords in the trapdoor. L: maximum number of keywords in a trapdoor. e: pairing evaluation time. G: exponentiation time in symmetric bilinear group $\mathbb{G}$. X: exponentiation time in finite field

We now give the estimated running times in milliseconds for the $\mathscr{S}_2$ scheme. Since the performance of the subset PEKS scheme in [11] depends strongly on the size of the keyword space, it is difficult to choose parameters allowing a sensible comparison. Therefore, we omit the performance analysis of [11].

We arbitrarily fix $m = 8$, $l = 8$ and use 48-bit keywords. We also instantiate the index and the trapdoor so that the search operation takes the longest possible. In the proposed scheme, the computation of an index and a trapdoor takes an estimated time of 39.5ms and 36.6ms, respectively, and the search time is approximately 7.57ms.

8 Conclusion and future work

Public-Key Encryption with Keyword Search (PEKS) schemes enable public key holders to encrypt documents, while the secret key holder is able to generate queries for the encrypted data. In this article, we have presented two PEKS schemes enabling conjunctive and subset queries. We have proposed a security notion for PEKS, and we have proved the proposed schemes secure under the asymmetric DBDH problem and the p-BDHI problem, respectively. We have also proved the computational consistence of our constructions.

The main strength of our schemes lies in their efficiency since, as shown in the provided efficiency analysis, they improve all previous related schemes in some of the most critical operations.

The proposed schemes could possibly admit various extensions. For example, we believe it is possible to extend our subset PEKS scheme to allow decryption of encrypted indexes by embedding messages in the target group, as done in works such as [11]. Such an extension would allow the retrieval of messages in the search process.

In [1], Abdalla et al. prove computational consistency for the PEKS scheme [9] by Boneh et al., and give a modified scheme achieving the stronger notion of statistical consistency. Since the conjunctive PEKS scheme we propose here can be seen as a natural extension to the scheme in [9] to the conjunctive case, it would be interesting to find similar modifications that improve consistency.

It would also be interesting to maintain a good efficiency and security trade-off while improving the security notion, for example, by providing tight security proofs in the standard model, or by removing the need for a secure channel for trapdoors.

Funding This study was funded by the European Commission (H2020-ICT-2014-1-644024 "CLARUS"), by the Government of Spain (TIN2014-57364-C2-1-R and TIN2016-80250-R "Sec-MCloud") and by the Government of Catalonia (Grant 2014 SGR 537).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical standard This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions. In: CRYPTO'05, pp. 205–222 (2005)
2. Aranha, D.F., Barreto, P.S.L.M., Longa, P., Ricardini, J.E.: The realm of the pairings. In: SAC'13, v.8282, pp. 3–25 (2014)
3. Baek, J., Safavi-Naini, R., Susilo, W.: Public key encryption with keyword search revisited. In: ICCSA'08, vol. 5072, pp. 1249–1259 (2008)
4. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: ICICS'05, vol. 3783, pp. 414–426 (2005)
5. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: CRYPTO'02, Springer, pp. 354–368 (2002)
6. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In: Advances in Cryptology—EUROCRYPT 2014, LNCS 8441. Springer, pp. 1–16 (2014)
7. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS'93. ACM, New York, NY, USA, pp. 62–73 (1993)
8. Boneh, D., Boyen, X.: Efficient selective identity-based encryption without random oracles. J. Cryptol. **24**(4), 659–693 (2011)
9. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In EUROCRYPT'04, LNCS, vol. 3027. Springer, pp. 506–522 (2004)
10. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. SIAM J. Comput. **32**(3), 586–615 (2003)
11. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: TCC'07. Springer, pp. 535–554 (2007)
12. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. In: ACM Computing Surveys, vol. 47(2), pp. 18:1–18:51 (2014)
13. Byun, J.W., Lee, D.H.: On a security model of conjunctive keyword search over encrypted relational database. J. Syst. Softw. **84**(8), 1364–1372 (2011)
14. Byun, J.W., Lee, D.H., Lim, J.: Efficient conjunctive keyword search on encrypted data storage system. In: EuroPKI '06, vol. 4043. Springer, pp. 184–196 (2006)
15. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
16. Chen, Z., Wu, C., Wang, D., Li, S.: Conjunctive keywords searchable encryption with efficient pairing, constant ciphertext and short trapdoor. In: PAISI'12. Springer, pp. 176–189 (2012)
17. Chen, Y., Zhang, J., Lin, D., Zhang, Z.: Generic constructions of integrated PKE and PEKS. Des. Codes Cryptogr. **78**(2), 493–526 (2016)
18. D.MAYA.6: final report on main computational assumptions in cryptography. ECRYPT II Project co-funded by the European Commission within the 7th Framework Programme, ICT-2007-216676. http://cordis.europa.eu/docs/projects/cnect/6/216676/080/deliverables/001-DMAYA6.pdf
19. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Appl. Math. **156**(16), 3113–3121 (2008)
20. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: ACNS 2004, vol. 3089. Springer, pp. 31–45 (2004)
21. Hwang, Y.H., Lee, P.J.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Pairing'07. Springer, pp. 2–22 (2007)
22. Joux, A.: A one round protocol for tripartite Diffie–Hellman. In: ANTS-IV. Springer, pp. 385–394 (2000)
23. 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)
24. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. J. Cryptol. **26**(2), 191–224 (2013)
25. Kiltz, E., Galindo, D.: Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. Theor. Comput. Sci. **410**(47–49), 5093–5111 (2009)
26. Libert, B., Quisquater, J.-J.: On constructing certificateless cryptosystems from identity based encryption. In: PKC 2006, vol. 3958, pp. 474–490 (2006)
27. Lynn, B.: PBC library. The pairing-based cryptography library. http://crypto.stanford.edu/pbc/. Accessed 24 Sept 2016
28. Liu, C., Zhu, L., Wang, M., Tan, Y.A.: Search pattern leakage in searchable encryption: attacks and new construction. Inf. Sci. Int. J. **265**, 176–188 (2014)

29. Miller, V.S.: Short programs for functions on curves. Unpublished, 1986. https://crypto.stanford.edu/miller/miller.pdf. l. Algebra Eng., Commun. Comput. 17, 5, 379–392 (2006)
30. Park, D.J., Cha, J., Lee, P.J.: Searchable keyword-based encryption. In: IACR Cryptology ePrint Archive, 367 (2005). https://eprint.iacr.org/2005/367.pdf
31. Park, D.J., Kim, K., Lee, P.J.: Public key encryption with conjunctive field keyword search. In: WISA'04, pp. 73–86 (2004)
32. Rhee H.S., Park J.H., Susilo W., Lee D.H.: Improved searchable public key encryption with designated tester. In ASIACCS'09. ACM, pp. 376–379 (2009)
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. Scott, M., Barreto, P.S.L.M.: Compressed pairings. CRYPTO 2004, LNCS 3152. Springer, pp. 140–156 (2004)
35. Shi, E., Bethencourt, J., Hubert Chan, T.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy. IEEE Computer Society, Washington, DC, USA, pp. 350–364 (2007)
36. Shikfa, A., Önen, M., Molva, R.: Privacy and confidentiality in context-based and epidemic forwarding. Comput. Commun. **33**(13), 1493–1504 (2010)
37. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: SP'00, IEEE Computer Society, p. 44 (2000)
38. Tibouchi, M.: A note on hashing to BN curves. In: SCIS. IEICE (2012)
39. Waters, B.: Efficient identity-based encryption without random oracles. In: EUROCRYPT 2005, vol. 3494. Springer, pp. 114–127 (2005)
40. Wang, T., Au, M.H., Wu, W.: An efficient secure channel free searchable encryption scheme with multiple keywords. In: NSS 2016: Network and System Security, pp. 251–265 (2016)
41. Waters, B., Balfanz, D., Durfee, G., Smetters, D.: Building an encrypted and searchable audit log. In: NDSS (2004)
42. Zhang, B., Zhang, F.: An efficient public key encryption with conjunctive-subset keywords search. J. Netw. Comput. Appl. **34**(1), 262–267 (2011)