Public-Key Encryption with Delegated Search

Luan Ibraimi, Svetla Nikova, Pieter Hartel, and Willem Jonker

Faculty of EEMCS, University of Twente, The Netherlands

Abstract. In a public key setting, Alice encrypts an email with the public key of Bob, so that only Bob will be able to learn the contents of the email. Consider a scenario where the computer of Alice is infected and unbeknown to Alice it also embeds a malware into the message. Bob's company, Carol, cannot scan his email for malicious content as it is encrypted so the burden is on Bob to do the scan. This is not efficient. We construct a mechanism that enables Bob to provide trapdoors to Carol such that Carol, given an encrypted data and a malware signature, is able to check whether the encrypted data contains the malware signature, without decrypting it. We refer to this mechanism as *public-key encryption with delegated search* ($\mathcal{PKEDS}$).

We formalize $\mathcal{PKEDS}$ and give a construction based on ElGamal public-key encryption ($\mathcal{PKE}$). The proposed scheme has ciphertexts which are both searchable and decryptable. This property of the scheme is crucial since an entity can search the entire content of the message, in contrast to existing searchable public-key encryption schemes where the search is done only in the metadata part. We prove in the standard model that the scheme is *ciphertext indistinguishable* and *trapdoor indistinguishable* under the Symmetric External Diffie-Hellman (SXDH) assumption. We prove also the *ciphertext one-wayness* of the scheme under the modified Computational Diffie-Hellman (mCDH) assumption. We show that our $\mathcal{PKEDS}$ scheme can be used in different applications such as detecting encrypted malware and forwarding encrypted email.

1 Introduction

Consider an organization, Carol, whose employees use public-key encryption to communicate with other users from outside the organization. All organizational incoming encrypted emails are stored in a server which is managed by Carol. Bob (an employee) can download his encrypted email from the server and decrypt it locally using his private key. Encryption prevents an attacker from learning confidential information, but it opens another problem: the server cannot search the ciphertext for malware. While encryption helps Bob to protect his sensitive data, the hardness of processing the ciphertext without decrypting it, helps the attacker to hide malicious content from the server. Suppose Alice, who resides outside the organization, encrypts a message with the public key of Bob, so that only Bob will be able to learn the contents of the message. Unbeknown to Alice, her computer is infected and embeds malware into the message. Since the malware is encrypted, the server is unable to scan the ciphertext for malicious

J. Lopez and G. Tsudik (Eds.): ACNS 2011, LNCS 6715, pp. 532–549, 2011.

code. A naive solution to detect encrypted malware is for Bob to send his private key to the server. Once the server gets the key, it decrypts the ciphertext and then scans the plain data for a malicious content. However this solution is too risky since the server accesses the plain data and a compromise of the server compromises all Bob's data. Another solution would be to force Bob to scan his email for malicious content. However this approach is not efficient.

In this paper we focus on finding mechanisms which allow the server to search the ciphertext, without decrypting it. Searching encrypted data [BDCOP04] is an attractive technique that might address the aforementioned problem. It allows the server to search encrypted data without learning information about the plain data or the search query. Boneh et al. [BDCOP04] were the first to propose *public-key encryption with keyword search* (a.k.a $\mathcal{PEKS}$ or searchable encryption). It works as follows. Alice creates a ciphertext c_w which encrypts the keyword w and Bob creates a trapdoor t_w for a keyword w. The trapdoor t_w is sent to the server, which on receipt of the searchable ciphertext c_w and the trapdoor t_w, runs the Test function which returns *true* only if both the searchable ciphertext c_w and the trapdoor t_w are associated with the same keyword, otherwise it outputs *false*. $\mathcal{PEKS}$, is only used to encrypt keywords (meta-data) describing the document, while to encrypt the entire document Alice must use a traditional public-key encryption $\mathcal{PKE}$ scheme, where the ciphertext is decryptable but not searchable. This approach is not suitable for some applications, such as detecting encrypted malware, for the following reasons: a) the server can search only inside the $\mathcal{PEKS}$ ciphertext, the other part of the ciphertext created by the $\mathcal{PKE}$ scheme is not searchable, and b) Bob has to stay online - the malware signature database maintained by the server might get updated frequently, therefore Bob has to create trapdoors and send them to the server.

1.1 Our Contribution

In this paper we construct a public-key encryption scheme with delegated search ($\mathcal{PKEDS}$) which has the following properties:

1. Each part of the encrypted data is both searchable and decryptable, unlike in $\mathcal{PEKS}$ where only the metadata part of the ciphertext is searchable. Hence, our scheme can be used alone, without employing an additional $\mathcal{PKE}$ scheme, to provide end-to-end security.
2. Once delegated by Bob, the server is allowed to create any trapdoor without contacting Bob, thus, once the delegation is done, Bob can go offline. We construct a mechanism that enables Bob to provide the server with a master trapdoor t_* such that, given the encrypted data and a word w picked by the server, the server can test whether the word w is in the encrypted data, without decrypting it.
3. The server can answer queries made by Bob. In the proposed scheme, Bob can provide the server with a trapdoor t_w associated with a specific word w such that the server can test whether the word w occurs in the encrypted data, without allowing the server to learn the word w.

We provide a security proof in the standard model and show that the scheme is *ciphertext indistinguishable* and *trapdoor indistinguishable* under the Symmetric External Diffie-Hellman (SXDH) assumption. Note that in our scheme it is *inherently* impossible to achieve these properties against the server (i.e. we can achieve these properties against any adversary excluding the server). The first limitation comes as a result of allowing the server to hold the master trapdoor t_*, from which the server can create any trapdoor associated to any word and break ciphertext indistinguishability security. The second limitation comes from the nature of public-key encryption where an entity (i.e. the server) which holds a trapdoor t_w associated with a specific word w and knows the public key of the receiver can create a valid ciphertext and break trapdoor indistinguishability. This is also observed by Shen, Shi and Waters [SSW09] and to date the property of trapdoor indistinguishability is only achieved in the symmetric key setting where only the secret key holder can create a valid ciphertext. The best that we can achieve against the server is *ciphertext one-wayness* and we show that our scheme is secure in this respect under the modified Computation Diffie-Hellman (mCDH) assumption in the standard model. The construction of the scheme is based on ElGamal private-key encryption ($\mathcal{PKE}$) [ElG85]. Indeed, our scheme can be viewed as an extension of ElGamal, with additional features: it allows the receiver to create trapdoors and the server to search the encrypted data. The proposed construction uses Type-3 pairings [GPS08] which are employed by the server to search the ciphertext and by the receiver to run the TrapGen function in order to generate the trapdoor. The use of Type-3 pairing is crucial, both for running testing functions and for preventing an adversary (other than the server) to break the security of the scheme.

1.2 Related Work

In the public key setting, Boneh et al. [BDCOP04] introduced the first *private-key encryption with keyword search* ($\mathcal{PEKS}$) in which everyone can create a searchable ciphertext but only the owner of a private key can create a trapdoor. The proposed $\mathcal{PEKS}$ scheme [BDCOP04] is based on anonymous identity-based encryption (IBE) as introduced in [BF01]. Abdalla et al. [ABC$^+$05] fix a consistency flaw from [BDCOP04] and provide a transform of an anonymous IBE scheme from Boyen and Waters [BW06] to construct a $\mathcal{PEKS}$ scheme in the standard model. In addition, they show how to extend $\mathcal{PEKS}$ scheme to design a private-key encryption with temporary keyword search.

There are number of improvements to the initial concept of $\mathcal{PEKS}$ in which the search is only done by comparing the keyword of the ciphertext with the keyword of the trapdoor. Boneh and Waters [BW07] propose a scheme which supports conjunctive, subset, and range queries over the keywords. Hwang and Lee [HL07] propose a $\mathcal{PEKS}$ scheme which works in the multiuser setting, where the keyword is encrypted under many public keys for many receivers. Fuhr and Paillier [FP07] propose a decryptable $\mathcal{PEKS}$ scheme with a security proof in the heuristic random oracle model, and Hofeinz and Weinreb [HW08] propose a decryptable $\mathcal{PEKS}$ scheme with a security proof in the standard model.

A similar concept to decryptable $\mathcal{PEKS}$ is the *hybrid model* [BSNS06, ZI07] which integrates $\mathcal{PKE}$ and $\mathcal{PEKS}$ into a single scheme by allowing both schemes to share the same key pair (pk, sk). The difference between the hybrid model and the original $\mathcal{PEKS}$ scheme, is that the first integrates both $\mathcal{PEKS}$ and $\mathcal{PKE}$ schemes into a single scheme, while the later assumes that in addition to $\mathcal{PEKS}$ scheme there is an another separate $\mathcal{PKE}$ scheme. While the hybrid model ties $\mathcal{PEKS}$ and $\mathcal{PKE}$, it does not guarantee any relation between messages encrypted under $\mathcal{PEKS}$ and messages encrypted under $\mathcal{PKE}$ scheme. Particularly, an attacker can always encrypt one message using $\mathcal{PEKS}$ scheme and encrypt a different message using $\mathcal{PKE}$ scheme, in this way causing the server to send emails in which the receivers are not interested. The proposed $\mathcal{PKEDS}$ scheme guarantees this relation since it allows the receiver to decrypt the searchable ciphertext and check whether the keywords indeed describe the original message.

Organization of the Paper. In Section 2 we give a brief description of bilinear groups and complexity assumptions. In Section 3 we define the algorithms of the $\mathcal{PKEDS}$ scheme and formalize the security requirements. In Section 4 we present our $\mathcal{PKEDS}$ construction. In Section 5 we prove its security and in Section 6 we discuss its applications. The last section concludes the paper.

2 Bilinear Groups and Complexity Assumptions

Our construction uses prime order bilinear groups. Let $\mathbb{G}$, Γ and $\mathbb{G}_T$ be groups of prime order p, and let g and γ be generator of $\mathbb{G}$ and Γ, respectively. A pairing (or bilinear map) $\hat{e} : \mathbb{G} \times \Gamma \rightarrow \mathbb{G}_T$ has the following properties [BF01]:

1. Bilinearity: for all $u \in \mathbb{G}$, $v \in \Gamma$ and $a, b \in \mathbb{Z}_p^*$, we have $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.
2. Non-degeneracy: $\hat{e}(g, \gamma) \neq 1$.
3. The function $\hat{e}$ can be efficiently computed.

Pairings can be categorized into three types [GPS08]: a) Type-1: is known as symmetric pairing and $\mathbb{G} = \Gamma$, b) Type-2: is known as asymmetric pairing and $\mathbb{G} \neq \Gamma$, but there is an efficiently computable isomorphism $\psi : \mathbb{G} \rightarrow \Gamma$, and c) Type-3: is same as Type-2 except that there is no known efficiently computable isomorphism $\psi : \mathbb{G} \rightarrow \Gamma$.

The ciphertext and trapdoor indistinguishability of the proposed scheme are based on the Symmetric External Diffie-Hellman (SXDH).

Definition 1. ***Symmetric External Diffie-Hellman*** (SXDH) ***Assumption****: In Type-3 pairings the Decision Diffie-Hellman Problem (DDH) is intractable both in $\mathbb{G}$ and Γ, i.e. given a tuple $(g, g^a, g^b, g^c) \in \mathbb{G}$ or $(\gamma, \gamma^a, \gamma^b, \gamma^c) \in \Gamma$ with $a, b \in \mathbb{Z}_p$, decide whether $c = ab$ or $c \in_R \mathbb{Z}_p$.*

To prove the one-wayness of the scheme, we use a slightly stronger variant of the CDH assumption which we call it the *modified* CDH (mCDH).

Definition 2. ***Modified Computational Diffie-Hellman*** (mCDH) ***Assumption:*** *Given tuples $(g, g^a,\ g^b) \in \mathbb{G}$ and $(\gamma, \gamma^b) \in \Gamma$ with $a, b \in \mathbb{Z}_p$, it is hard to compute g^{ab}.*

Note that the mCDH assumption is implied by the BDH-3 [CM09] assumption. Therefore, an algorithm that breaks the mCDH assumption can be converted to an algorithm that breaks the BDH-3 assumption.

3 Description and Security Model of $\mathcal{PKEDS}$ Scheme

Definition 3. *A private-key encryption scheme with delegated search ($\mathcal{PKEDS}$) consists of the following nine algorithms (Setup, KeyGen$_S$, KeyGen$_R$, Encrypt, Delegate, TrapGen, Test$_1$, Test$_2$, Decrypt):*

- $\mathsf{Setup}(\lambda)$: *The setup algorithm is run by a system administrator and takes as input a security parameter λ and outputs public parameters $\mathcal{PP}$.*
- $\mathsf{KeyGen}_S(\mathcal{PP})$: *This key generation algorithm is run by the server and takes as input public parameters $\mathcal{PP}$ and outputs the server's public/private key pair (pk_S, sk_S).*
- $\mathsf{KeyGen}_R(\mathcal{PP})$: *This key generation algorithm is run by Bob (the receiver) and takes as input public parameters $\mathcal{PP}$ and outputs Bob's public/private key pair (pk_R, sk_R).*
- $\mathsf{Encrypt}(pk_R, w)$: *The encryption algorithm is run by Alice (the message sender) and takes as input Bob's public key pk_R and a word w, and outputs a ciphertext c_w.*
- $\mathsf{Delegate}(sk_R, pk_S)$: *The delegate algorithm is run by Bob and takes as input Bob's private key sk_R, the server's public key pk_S, and outputs the master trapdoor t_*.*
- $\mathsf{TrapGen}(sk_R, pk_S, w)$: *The trapdoor generation algorithm is run by Bob and takes as input Bob's private key sk_R, the server's public key pk_S and a word w, and outputs the trapdoor t_w.*
- $\mathsf{Test}_1(c_w, t_*, t_w, sk_S)$: *The first testing algorithm is run by the server and takes as input a ciphertext c_w, a master trapdoor t_*, a trapdoor t_w associated with the word w, and the server's private key sk_S, and outputs* true *if the ciphertext and the trapdoor are associated with the same word, otherwise outputs $\perp$.*
- $\mathsf{Test}_2(c_w, t_*, w, sk_S)$: *The second testing algorithm is run by the server and takes as input a ciphertext c_w, a master trapdoor t_*, a word w, and the server's private key sk_S, and outputs* true *if the ciphertext contains the word w, otherwise outputs $\perp$.*
- $\mathsf{Decrypt}(c_w, sk_R)$: *The decryption algorithm is run by Bob and takes as input a ciphertext c_w and Bob's private key sk_R, and outputs the word w or $\perp$ if c_w is invalid.*

Correctness. We say that $\mathcal{PKEDS}$ is *correct* if for all security parameters $\lambda \in \mathbb{N}$, for all server public/private key pairs produced by KeyGen_S, for all receiver public/private key pairs produced by KeyGen_R, for all ciphertexts c_w produced by Encrypt, for all master trapdoors t_* produced by Delegate and for all trapdoors t_w produced by TrapGen , we should have:

$$\Pr\left[\begin{array}{c}\mathcal{PP} \leftarrow \mathsf{Setup}(\lambda), (pk_S, sk_S) \leftarrow \mathsf{KeyGen}_S(\mathcal{PP}), (pk_R, sk_R) \leftarrow \mathsf{KeyGen}_R(\mathcal{PP}), \\ c_w \leftarrow \mathsf{Encrypt}(pk_R, w), t_* \leftarrow \mathsf{Delegate}(sk_R, pk_S), t_w \leftarrow \mathsf{TrapGen}(sk_R, pk_S, w) : \\ w \leftarrow \mathsf{Decrypt}(c_w, sk_R) \wedge \mathit{true} \leftarrow \mathsf{Test}_1(c_w, t_*, t_w, sk_S) \\ \wedge\ \mathit{true} \leftarrow \mathsf{Test}_2(c_w, t_*, w, sk_S)\end{array}\right] = 1$$

Ciphertext Indistinguishability. In the following we describe the basic security property for a $\mathcal{PKEDS}$ scheme which is ciphertext indistinguishability. This property guarantees that it is infeasible for an adversary (other than the server) to learn any information about any word from the ciphertext. The following definition formally captures this property.

Definition 4. ***(CI-ATK)*** *Let $\mathcal{PKEDS}$ be a private-key encryption with delegated search scheme and let $\mathcal{A}$ be a polynomial-time (PPT) adversary. Let*

$$\mathsf{ADV}^{\mathcal{CI}-ATK}_{\mathcal{A},\mathcal{PKEDS}}(\lambda) \overset{def}{=} \Pr\left[\begin{array}{c}\mathcal{PP} \leftarrow \mathsf{Setup}(\lambda), (pk_S, sk_S) \leftarrow \mathsf{KeyGen}_S(\mathcal{PP}), \\ (w_0, w_1, R^*) \leftarrow \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}(pk_S, \mathcal{PP}), v \leftarrow \{0,1\}, \\ c_{w_v} \leftarrow \mathsf{Encrypt}(pk_{R^*}, w_v), \\ v' \leftarrow \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}(c_{w_v}, w_0, w_1, pk_{R^*}, pk_S) : v' = v\end{array}\right] - \frac{1}{2}$$

where $w_0 \neq w_1 \wedge |w_0| = |w_1|$, the key-generation oracle $\mathcal{O}_1(R)$ is defined as $\mathsf{KeyGen}_R(\mathcal{PP})$ and returns (pk_R, sk_R) , the registration oracle $\mathcal{O}_2(R)$ is defined as $\mathsf{Delegate}(sk_R, pk_S)$ and returns t_*, the trapdoor generation oracle $\mathcal{O}_3(R, w)$ is defined as $\mathsf{TrapGen}(sk_R, w)$ and returns t_w. We restrict $\mathcal{A}$ such that if $\mathcal{A}$ queries $\mathcal{O}_1$ for the receiver key pair R^*, then $\mathcal{O}_1$ returns only the public key pk_{R^*} of the receiver R^*. We say that $\mathcal{PKEDS}$ is secure from ciphertext indistinguishable attacks if $\mathsf{ADV}^{\mathcal{CI}-ATK}_{\mathcal{A},\mathcal{PKEDS}}$ is negligible for any $\mathcal{A}$.

The scheme does not achieve $\mathcal{CI}$ against the server. Given the challenge ciphertext c_{w_b}, the master trapdoor t_* and words (w_0, w_1), the server runs $\mathsf{Test}_2(c_{w_b}, t_*, w_0, sk_S)$ to check whether the trapdoor and the ciphertext are associated with the same word. If the output of Test_2 is *true* then the server learns that $b = 0$, otherwise it learns that $b = 1$. As $\mathcal{CI}$ against the server is inherently not possible (remember that our focus is to allow the server to search the ciphertext); the best we can achieve against the server is ciphertext one-wayness.

Note. The security model that we consider in this paper is *weaker* than the original security model considered in $\mathcal{PEKS}$ [BDCOP04]. Our model gives more power to the server since the scheme allows the server to generate any trapdoor. This is risky for low entropy messages since the server by trial and error can find out the message. Nevertheless, for high entropy messages this attack is hard. For instance, if the server scans a ciphertext which contains user fingerprints, then its hard for the server to guess the fingerprint and run the trial and error attack. Indeed, our scheme is suitable for situations when the server is managed by an organization which wants to protect their employees from potential malicious senders while avoiding the need of giving the private key to the server.

Trapdoor Indistinguishability. The *trapdoor indistinguishability* property guarantees that it is infeasible for an adversary (except the server) to learn any information about any word from the trapdoor. Baek et al. [BSNS08] observe that $\mathcal{PEKS}$ scheme presented by Boneh et al. [BDCOP04] assumes a secure channel between the server and the receiver. If there is no secure channel, then everyone can break the *trapdoor indistinguishability* property since everyone can play the role of the server. To remove the secure channel between the receiver and the server, Baek et al. [BSNS08] propose a scheme where the sender encrypts the $\mathcal{PEKS}$ ciphertext with the public key of the server, in such a way that only the server who knows the private key can reveal the $\mathcal{PEKS}$ ciphertext. In this paper we take a different approach to achieve *trapdoor indistinguishability* against outside adversaries. The Baek et al. [BSNS08] solution is not suitable in our setting since we allow the receiver to decrypt the $\mathcal{PKEDS}$ ciphertext, otherwise if we encrypt the $\mathcal{PKEDS}$ ciphertext with the server's public key, then the receiver cannot decrypt the ciphertext without getting help from the server. Instead, the role of the server is to search the encrypted data and not to help the receiver to decrypt the ciphertext. To achieve *trapdoor indistinguishability*, we need the secure channel established between the receiver and the server, as assumed in [BDCOP04]. This implies that, instead of encrypting the communication between the sender and the receiver, we encrypt the communication between the receiver and the server. Namely, before sending the trapdoor to the server, the receiver encrypts the trapdoor under the server's public key. Since only the server has the private key, only the server can reveal the trapdoor and search the encrypted data. We capture the property of *trapdoor indistinguishability* through the following definition.

Definition 5. ***(TI-ATK)*** *Let $\mathcal{PKEDS}$ be a private-key encryption with delegated search scheme and let $\mathcal{A}$ be a polynomial-time (PPT) adversary. Let*

$$\mathsf{ADV}^{\mathcal{TI}-ATK}_{\mathcal{A},\mathcal{PKEDS}}(\lambda) \overset{def}{=} \Pr\left[\begin{array}{c} \mathcal{PP} \leftarrow \mathsf{Setup}(\lambda), (pk_S, sk_S) \leftarrow \mathsf{KeyGen}_S(\mathcal{PP}), \\ (w_0, w_1, R^*) \leftarrow \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}(pk_S, \mathcal{PP}), v \leftarrow \{0,1\}, \\ t_{w_v} \leftarrow \mathsf{TrapGen}(pk_{R^*}, w_v), \\ v' \leftarrow \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}(t_{w_v}, w_0, w_1, pk_{R^*}, pk_S) : v' = v \end{array}\right] - \frac{1}{2}$$

where $w_0 \neq w_1 \wedge |w_0| = |w_1|$ and oracles $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3$ are defined in the same way as in $\mathcal{CI} - ATK$, but the difference is that the adversary is not limited in his queries. We say that $\mathcal{PKEDS}$ is secure from trapdoor indistinguishable attacks if $\mathsf{ADV}^{\mathcal{TI}-ATK}_{\mathcal{A},\mathcal{PKEDS}}$ is negligible for any $\mathcal{A}$.

Under this definition we achieve $\mathcal{TI}$ only for adversaries other than the server. Informally speaking, we achieve this property by not allowing an adversary to search the encrypted data. In particular, we cannot achieve $\mathcal{TI}$ from the server who runs the Test_1 function since the server can guess the word in the following way: the server sends to the challenger two words $(w_0,\ w_1)$ and the challenger replies to the server by sending t_{w_v} for a random bit $v \in \{0,1\}$. Next, the server chooses a random bit $v' \in \{0,1\}$ and runs $c_{v'} \leftarrow \mathsf{Encrypt}(pk_R, w_{v'})$. Finally, the server run $\mathsf{Test}_1(c_{v'}, t_*, t_{w_v}, sk_S)$ and outputs $v' = v$ if the output of Test_1 is *true.*

Ciphertext One-Wayness. The property of *ciphertext one-wayness* guarantees that it is hard for an adversary to invert the ciphertext and to learn the word even if the adversary holds the server's private key, the master trapdoor and the trapdoor associated with that word, but the adversary does not hold the receiver's private key. The following definition formally captures this attack.

Definition 6. ***(COW-ATK)*** *Let $\mathcal{PKEDS}$ be a private-key encryption with delegated search scheme and let $\mathcal{A}$ be a polynomial-time (PPT) adversary. Let*

$$\mathsf{ADV}^{\mathcal{COW}-ATK}_{\mathcal{A},\mathcal{PKEDS}}(\lambda) \overset{def}{=} \Pr\left[\begin{array}{c} \mathcal{PP} \leftarrow \mathsf{Setup}(\lambda), (pk_S, sk_S) \leftarrow \mathsf{KeyGen}_S(\mathcal{PP}), \\ R^* \leftarrow \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}(pk_S, sk_S, \mathcal{PP}), w^* \leftarrow \mathcal{M}(k), \\ c_{w^*} \leftarrow \mathsf{Encrypt}(pk_{R^*}, w^*), t_{w^*} \leftarrow \mathsf{Encrypt}(sk_{R^*}, pk_S, w^*), \\ w' \leftarrow \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3}(c_{w^*}, t_{w^*}, pk_{R^*}, sk_S, pk_S) : w' = w^* \end{array}\right]$$

where oracles $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3$ are defined in the same way as in $\mathcal{CI} - ATK$ with the restriction that $\mathcal{A}$ is not allowed to learn sk_{R^*} from the oracle $\mathcal{O}_1$. We say that $\mathcal{PKEDS}$ is secure from ciphertext one-way attacks if $\mathsf{ADV}^{\mathcal{COW}-ATK}_{\mathcal{A},\mathcal{PKEDS}}$ is negligible for any $\mathcal{A}$.

4 A Construction of $\mathcal{PKEDS}$ Scheme

We are now ready to present our construction. When explaining the scheme we consider the single-user setting. The scheme consists of nine algorithms (Setup, KeyGen_S, KeyGen_R, $\mathsf{Encrypt}$, $\mathsf{Delegate}$, $\mathsf{TrapGen}$, Test_1, Test_2, $\mathsf{Decrypt}$) defined as follows:

Setup: On input of the security parameter λ the algorithm outputs public parameters ($\mathcal{PP}$) which contain the description of groups $< \mathbb{G}, \mathit{\Gamma} >$ of order p, the bilinear map $\hat{e} : \mathbb{G} \times \mathit{\Gamma} \rightarrow \mathbb{G}_T$, generators g and γ of groups $\mathbb{G}$ and $\mathit{\Gamma}$, respectively.

$\mathsf{KeyGen}_S(\mathcal{PP})$: On input of public parameters $\mathcal{PP}$ the algorithm picks a random $x \in \mathbb{Z}_p$ and outputs the server's key pair:

$$(sk_S, pk_S) = (x, p_s = \gamma^x)$$

$\mathsf{KeyGen}_R(\mathcal{PP})$: On input of public parameters $\mathcal{PP}$ the algorithm picks a random $\alpha, y \in \mathbb{Z}_p$ and outputs the receiver's key pair:

$$(sk_R, pk_R) = ((y, \gamma^\alpha), p_r = g^y)$$

$\mathsf{Encrypt}(pk_R, w)$: On input of the receiver's public key and a word $w \in \mathbb{G}$ the algorithm picks a random $k \in \mathbb{Z}_p$ and outputs the ElGamal ciphertext:

$$c_w = (c_1, c_2) = \left(w \cdot p_r^k, g^k\right)$$

$\mathsf{Delegate}(pk_S, sk_R)$: The algorithm creates a master trapdoor to allow the server to search the encrypted data for any word of his choice. The algorithm picks at random $r_1, r_2 \in \mathbb{Z}_p$ and outputs the master trapdoor:

$$t_* = (t_1, t_2, t_3, t_4) = (\gamma^\alpha \cdot p_s^{r_1}, \gamma^{r_1}, \gamma^{y\alpha} \cdot p_s^{r_2}, \gamma^{r_2})$$

$\mathsf{TrapGen}(sk_R, pk_S, w)$**:** The algorithm creates a trapdoor to allow the server to search for a specific message w. The algorithm picks a random $\delta \in \mathbb{Z}_p$ and outputs the trapdoor:

$$t_w = (t_5, t_6) = \big(\hat{e}(w, \gamma^{\alpha}) \cdot \hat{e}(p_r, p_s^{\delta}), \gamma^{\delta}\big)$$

$\mathsf{Test}_1(c_w, t_*, t_w, sk_S)$: The algorithm tests whether the ciphertext contains the same message as the trapdoor. The algorithm parses c_w as (c_1, c_2), t_* as (t_1, t_2, t_3, t_4), t_w as (t_5, t_6) and defines:

$$t_7 = \frac{t_1}{t_2^x} \quad , \quad t_8 = \frac{t_3}{t_4^x} \quad , \quad \tilde{a} = \frac{\hat{e}(p_r, t_6^x) \cdot \hat{e}(c_1, t_7)}{t_5} \quad , \quad \tilde{b} = \hat{e}(c_2, t_8).$$

Finally, the algorithm checks whether $\tilde{a} \overset{?}{=} \tilde{b}$. If this equation holds, the algorithm outputs *true* indicating that the ciphertext contains the same message as the trapdoor, otherwise it outputs *false*.

$\mathsf{Test}_2(c_w, t_*, w, sk_S)$: The algorithm tests whether the ciphertext contains the word w. The algorithm parse c_w as (c_1, c_2), t_* as (t_1, t_2, t_3, t_4) and defines:

$$t_7 = \frac{t_1}{t_2^x} \quad , \quad t_8 = \frac{t_3}{t_4^x} \quad , \quad \tilde{c} = \hat{e}(c_1, t_7) \quad , \quad \tilde{d} = \hat{e}(c_2, t_8)$$

Finally, the algorithm checks whether $\frac{\tilde{c}}{\tilde{d}} \overset{?}{=} \hat{e}(w, t_7)$. If this equation holds, the algorithm outputs *true* indicating that the ciphertext contains the word w, otherwise it outputs *false*.

$\mathsf{Decrypt}(sk_R, c_w)$**:** On input of the ciphertext and the private key the algorithm outputs:

$$w = \frac{c_1}{c_2^y}.$$

4.1 Efficiency

In Table 1 we count the number of calculations in KeyGen_S, KeyGen_R, $\mathsf{Encrypt}$, $\mathsf{Delegate}$, $\mathsf{TrapGen}$, Test_1, Test_2 and $\mathsf{Decrypt}$. KeyGen_S requires one exponentiation in Γ and KeyGen_R requires one exponentiation in $\mathbb{G}$ and one exponentiation in Γ. $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ are same as in ElGamal. $\mathsf{Encrypt}$ requires two exponentiations in $\mathbb{G}$ which are independent of the message and can be computed ahead of time and $\mathsf{Decrypt}$ requires only one exponentiation in $\mathbb{G}$. $\mathsf{Delegate}$ requires five exponentiations in Γ. $\mathsf{TrapGen}$ requires one exponentiation in Γ, one exponentiation in $\mathbb{G}_T$ and two pairing operations. Test_1 requires two exponentiations in Γ, one exponentiation in $\mathbb{G}_T$ and three pairing operations. Test_2 requires two exponentiations in Γ and three pairing operations.

5 Security

We now show that the scheme satisfies the correctness property, and is ciphertext indistinguishable, trapdoor indistinguishable and that the scheme offers ciphertext one-wayness.

Table 1. Efficiency of $\mathcal{PKEDS}$

	Exp.($\mathbb{G}$)	Exp.(Γ)	Exp.($\mathbb{G}_T$)	Pairing
KeyGen$_S$	//	1	//	//
KeyGen$_R$	1	1	//	//
Encrypt	2	//	//	//
Delegate	//	5	//	//
TrapGen	//	1	1	2
Test$_1$	//	2	1	3
Test$_2$	//	2	//	3
Decrypt	1	//	//	//

Correctness. Firstly, we show that when a ciphertext is created as a result of running Encrypt on input of the word w and the receiver's public key pk_R, then the same word w is revealed when running Decrypt on input of the ciphertext and the receiver's private key sk_R. This proof is ElGamal encryption/decyption and proceeds as follows:

$$\frac{c_1}{c_2^y} = \frac{w \cdot p_r^k}{g^{rs}} = \frac{w \cdot g^{rs}}{g^{rs}} = w$$

Next, we show the correctness for Test$_1$ algorithm. We observe that:

$$t_7 = \frac{\gamma^{\alpha} \cdot p_s^{r_1}}{\gamma^{xr_1}} = \gamma^{\alpha} \qquad t_8 = \frac{\gamma^{y\alpha} \cdot p_s^{r_2}}{\gamma^{xr_2}} = \gamma^{y\alpha}$$

$$\tilde{a} = \frac{\hat{e}(p_r, \gamma^{x\delta}) \cdot \hat{e}(w \cdot p_r^k, \gamma^{\alpha})}{\hat{e}(w, \gamma^{\alpha}) \cdot \hat{e}(p_r, p_s^{\delta})} = \hat{e}(p_r^k, \gamma^{\alpha}) \qquad \tilde{b} = \hat{e}(g^k, \gamma^{y\alpha}) = \hat{e}(p_r^k, \gamma^{\alpha})$$

Thus $\tilde{a} = \tilde{b}$ and the output is *true* indicating that the word associated with the ciphertext and the word associated with the trapdoor are the same. Finally, we show the correctness for the Test$_2$ algorithm. We observe that:

$$t_7 = \frac{\gamma^{\alpha} \cdot p_s^{r_1}}{\gamma^{xr_1}} = \gamma^{\alpha} \qquad t_8 = \frac{\gamma^{y\alpha} \cdot p_s^{r_2}}{\gamma^{xr_2}} = \gamma^{y\alpha} \qquad \tilde{c} = \hat{e}(w \cdot p_r^k, \gamma^{\alpha})$$

$$\tilde{d} = \hat{e}(g^k, \gamma^{y\alpha}) \qquad \frac{\tilde{c}}{\tilde{d}} = \hat{e}(w, \gamma^{\alpha})$$

Thus $\frac{\tilde{c}}{\tilde{d}} = \hat{e}(w, t_7)$ and the output is *true* indicating that the ciphertext is an encryption of the word w.

Ciphertext Indistinguishability. When proving this property we will closely follow the security proof of [BGMM05]. In the following we show that our construction is CI-ATK secure as long as the SXDH assumption holds.

Theorem 1. *Suppose that there exists an adversary $\mathcal{A}$ that can break the CI-ATK of the $\mathcal{PKEDS}$ scheme with advantage ε. Then we can construct a reduction $\mathcal{B}$ that breaks the* SXDH *assumption with advantage $(1 - \frac{q}{n})\frac{1}{n}\frac{\varepsilon}{2}$ where q is the number of queries asked by $\mathcal{A}$, and n is the number of receivers in the system.*

Proof. The challenger selects a bilinear map $\hat{e} : \mathbb{G} \times \Gamma \rightarrow \mathbb{G}_T$, and generators g and γ of groups $\mathbb{G}$ and Γ, respectively. Then, it picks at random $a, b \in \mathbb{Z}_p$, computes $T_0 = g^{ab}$ and picks at random $T_1 \in_R \mathbb{G}$. It flips a fair coin $\mu \in_R \{0, 1\}$ and gives the SXDH tuple $(g, g^a, g^b, T_\mu) \in \mathbb{G}$ to the reduction $\mathcal{B}$. The goal of $\mathcal{B}$ is to solve the SXDH assumption and acts as $\mathcal{A}$'s challenger as follows:

1. Setup : $\mathcal{B}$ generates the server's key pair $(sk_S, pk_S) = (x, p_s = \gamma^x)$, where $x \in_R \mathbb{Z}_p$ is chosen in the same way as in the scheme. $\mathcal{B}$ publishes $\mathcal{PP}$ and the server's key pair. The distribution of the $\mathcal{PP}$ and the server's key pair is identical to the $\mathcal{PP}$ and the server's key pair of the scheme since g and γ are random generators, and x is a random exponent, all chosen in the same way as in the scheme.
2. KeyGen$_R$ *to* $\mathcal{O}_1$: $\mathcal{B}$ answers the receiver's key generation queries by computing $(sk_R, pk_R) = ((y, \gamma^\alpha), p_r = g^y)$ where $\alpha, y \in_R \mathbb{Z}_p$ are chosen in the same way as in the scheme (each user has different α and y value). If the query is for R^*, $\mathcal{B}$ sets the public key equal to $p_r = g^a$ (this parameter is taken from the SXDH instance). Note that $\mathcal{B}$ does not know the private key of R^* ($\mathcal{B}$ does not know a). The distribution of the receiver's key pair is identical to the distribution of the receiver's key pair of the scheme since g is a random generator, α, a and y are random exponents, all chosen in the same way as in the scheme.
3. Delegate Query *to* $\mathcal{O}_2$: $\mathcal{A}$ requests a master trapdoor for the receiver R. If R is equal to R^*, $\mathcal{B}$ aborts the simulation and returns a guess μ'. Otherwise, if R is not equal to R^*, $\mathcal{B}$ computes the random elements $r_1, r_2 \in \mathbb{Z}_p$ and outputs the master trapdoor $t_* = (t_1, t_2, t_3, t_4) = (\gamma^\alpha \cdot p_s^{r_1}, \gamma^{r_1}, \gamma^{y\alpha} \cdot p_s^{r_2}, \gamma^{r_2})$. When $\mathcal{B}$ does not abort, the distribution of the master trapdoor is identical to the distribution of the master trapdoor in the scheme since r_1 and r_2 are random elements from $\mathbb{Z}_p$, same as in the real scheme.
4. TrapGen Query *to* $\mathcal{O}_3$: $\mathcal{A}$ requests a trapdoor for the pair (R, w) [1]. If R is equal to R^*, $\mathcal{B}$ aborts the simulation and returns a guess μ'. Otherwise, if R is not equal to R^*, $\mathcal{B}$ picks random $\delta \in \mathbb{Z}_p$ and outputs the trapdoor $t_w = (t_5, t_6) = \big(\hat{e}(w, \gamma^\alpha) \cdot \hat{e}(p_r, p_s^\delta), \gamma^\delta\big)$.

   When $\mathcal{B}$ does not abort, the distribution of the trapdoor is identical to the distribution of trapdoor in the scheme since δ is chosen at random from $\mathbb{Z}_p$, same as in the real scheme.
5. Challenge : $\mathcal{A}$ requests a ciphertext for one of the two words w_0 and w_1 generated under the public key of the receiver R. If R is equal to R^*, $\mathcal{B}$ flips a fair binary coin $v \in \{0, 1\}$ and outputs the ciphertext $\hat{c}_{w_v} = (c_1, c_2) = (w_v \cdot T_\mu, g^b)$, where g^b and T_μ are parameters taken from SXDH instance. The distribution of the ciphertext is identical to the distribution of the ciphertext in the scheme only if $T_\mu = g^{ab}$. Otherwise, if $T_\mu \in_R \mathbb{G}$, the ciphertext is a random element from $\mathbb{G}$. If R is not equal to R^*, $\mathcal{B}$ aborts the simulation and returns a guess μ'.
6. Guess : At the end of the game, without loss of generality, we assume that $\mathcal{A}$ has ciphertexts for all keywords generated by each user, and has requested

[1] A trapdoor associated with the word w generated by user (receiver) R.

trapdoor queries to oracles $\mathcal{O}_2$ and $\mathcal{O}_3$ generated from all but one user. Therefore, we assume that at the end of the game, in a non-aborted simulation case, $\mathcal{A}$ should have ciphertexts generated by all users for each keyword, and have at least one challenge ciphertext, denoted as $\hat{c}_v$, which is either a valid or invalid ciphertext generated by R^* for which $\mathcal{A}$ does not have the corresponding trapdoor. Lastly, $\mathcal{A}$ outputs a guess v'. If the guess is correct $v' = v$, then $\mathcal{B}$ sets $\mu' = 0$ indicating that $T_0 = g^{ab}$, otherwise $\mathcal{B}$ sets $\mu' = 1$ indicating that $T_1 \in_R \mathbb{G}$.

Suppose $\mathcal{B}$ does not abort (noted as $\overline{abort}$) during the simulation. If $\mu = 0$ then the ciphertext $\hat{c}_v$ is a valid ciphertext generated by user R^* and $\mathcal{A}$ sees an encryption of w_v. In this case we have: $\Pr\left[v' = v | \overline{abort} \wedge \mu = 0\right] = \frac{1}{2} + \varepsilon$. If $\mu = 1$ then the ciphertext $\hat{c}_v$ is random ciphertext for $\mathcal{A}$ (i.e. $\mathcal{A}$ gains no information about w_v). Hence we have: $\Pr\left[v' \neq v | \overline{abort} \wedge \mu = 1\right] = \frac{1}{2}$. Note that the advantage of $\mathcal{B}$ is same as the advantage of $\mathcal{A}$. For the first case when the guess of $\mathcal{A}$ is correct $v' = v$, $\mathcal{B}$ will output $\mu' = 0$ and we have $\Pr\left[\mu' = \mu | \overline{abort} \wedge \mu = 0\right] = \frac{1}{2} + \varepsilon$. For the second case when the guess is not correct $v' \neq v$, $\mathcal{B}$ will output $\mu' = 1$ and we have $\Pr\left[\mu' = \mu | \overline{abort} \wedge \mu = 1\right] = \frac{1}{2}$.

Now assume that $\mathcal{B}$ aborts (noted as $abort$) the simulation when running either TrapGen Query or Challenge phase. In this case $\mathcal{B}$ outputs its guess μ' which is independent of the guess given by $\mathcal{A}$ in Guess phase. Therefore the advantage of $\mathcal{B}$ in the abort case is: $\Pr\left[\mu' = \mu | abort\right] = \frac{1}{2}$. Putting all together we define the overall advantage of the reduction $\mathcal{B}$:

$$\Pr[abort]\Pr\left[\mu' = \mu | abort\right] + \Pr[\overline{abort}](\Pr[\mu = 0]\Pr[\mu' = \mu | \overline{abort} \wedge \mu = 0] +$$
$$\Pr[\mu = 1]\Pr[\mu' = \mu | \overline{abort} \wedge \mu = 1]) - \frac{1}{2} = \frac{\Pr[\overline{abort}]\varepsilon}{2}.$$

Now we have to define exactly the value of $\Pr[\overline{abort}]$ and give the exact overall advantage of $\mathcal{B}$. Let assume that $\mathcal{A}$ makes at most q queries during TrapGen Query phase and there are n users in the system. Since there is only one user for whom $\mathcal{B}$ cannot answer in TrapGen Query phase, the probability that a query causes $\mathcal{B}$ to abort is at most $\frac{1}{n}$. Since $\mathcal{A}$ can make q queries the overall probability that $\mathcal{A}$ aborts during TrapGen Query phase is $\frac{q}{n}$. Thus the probability that $\mathcal{B}$ does not abort in the TrapGen Query is $1 - \frac{q}{n}$. The probability that $\mathcal{B}$ will not abort in the Challenge phase is at least $\frac{1}{n}$. We now conclude that the reduction $\mathcal{B}$ solves the SXDH assumption with advantage at least $(1 - \frac{q}{n})\frac{1}{n}\frac{\varepsilon}{2}$, as required. □

Trapdoor Indistinguishability. We prove that our scheme is TI-ATK secure as long as the SXDH assumption is intractable. Unlike the ciphertext indistinguishability where the reduction had an SXDH instance from the group $\mathbb{G}$, when proving this property the reduction has an SXDH instance from the group Γ.

Theorem 2. *Suppose that there exists an adversary $\mathcal{A}$ that can break the trapdoor indistinguishability of the $\mathcal{PKEDS}$ scheme with advantage ε. Then we can construct a reduction $\mathcal{B}$ that solves the* SXDH *assumption with advantage $\frac{\varepsilon}{2}$.*

Proof. The challenger selects a bilinear map $\hat{e} : \mathbb{G} \times \Gamma \rightarrow \mathbb{G}_T$, and generators g and γ of groups $\mathbb{G}$ and Γ, respectively. Next, the challenger defines $T_0 = \gamma^{ab}$ for a random $a, b \in_R \mathbb{Z}_p$ and picks at random $T_1 \in_R \mathit{\Gamma}$. After flipping a fair coin $\mu \in_R \{0,1\}$, the challenger gives the SXDH tuple $(\gamma, \gamma^a, \gamma^b, T_\mu) \in \mathit{\Gamma}$ to $\mathcal{B}$. The reduction $\mathcal{B}$ solves the SXDH assumption by running $\mathcal{A}$ as a subroutine:

1. Setup: $\mathcal{B}$ sets the server's public key $pk_S = (p_s = \gamma^a)$, where γ^a is taken from the SXDH instance, and implicitly sets the server's secret-key $sk_S = a$. The reduction publishes the $\mathcal{PP}$ and the server's public keys which distribution is identical to the $\mathcal{PP}$ and the server's public keys of the scheme since g and γ are random generators, a is random exponent, all chosen in the same way as in the scheme.
2. KeyGen_R *to* $\mathcal{O}_1$: $\mathcal{B}$ answers receiver's key generation queries by computing $(sk_R, pk_R) = ((y, \gamma^\alpha), p_r = g^y)$ where $\alpha, y \in_R \mathbb{Z}_p$ are chosen in the same way as in the scheme (each user has different α and y value). The distribution of receiver's key pair is identical to the distribution of receiver's key pair of the scheme since g is a random generator, α and y are random exponents, all chosen in the same way as in the scheme.
3. $\mathsf{Delegate\ Query}$ *to* $\mathcal{O}_2$: $\mathcal{A}$ requests a master trapdoor for the receiver R. $\mathcal{B}$ compute random elements $r_1, r_2 \in \mathbb{Z}_p$ and outputs the master trapdoor $t_* = (t_1, t_2, t_3, t_4) = (\gamma^\alpha \cdot p_s^{r_1}, \gamma^{r_1}, \gamma^{y\alpha} \cdot p_s^{r_2}, \gamma^{r_2})$.
   The distribution of the master trapdoor is identical to the distribution of the master trapdoor in the scheme since r_1 and r_2 are random elements from $\mathbb{Z}_p$, same as in the real scheme.
4. $\mathsf{TrapGen\ Query}$ *to* $\mathcal{O}_3$: $\mathcal{A}$ requests a trapdoor for the pair (R, w). $\mathcal{B}$ picks random $\delta \in \mathbb{Z}_p$ and outputs the trapdoor t_w associated with the keyword w, $t_w = (t_5, t_6) = \big(\hat{e}(w, \gamma^\alpha) \cdot \hat{e}(p_r, p_s^\delta), \gamma^\delta\big)$. The distribution of the trapdoor is identical to the distribution of the trapdoor in the scheme since δ is randomly chosen from $\mathbb{Z}_p$, same as in the real scheme.
5. $\mathsf{Challenge}$: $\mathcal{A}$ sends two words w_0 and w_1 to $\mathcal{B}$ and asks for a trapdoor generated by user R^*. $\mathcal{B}$ flips a fair coin $v \in_R \{0,1\}$, picks at random $\alpha \in \mathbb{Z}_p$ and implicitly sets $\delta = b$ (where b is an exponent from the SXDH instance) and returns the trapdoor to $\mathcal{A}$: $t_{w_v} = (t_5, t_6) = \big(\hat{e}(w_v, \gamma^\alpha) \cdot \hat{e}(p_r, T_\mu), \gamma^b\big)$, where $(p_r = g^y,\ y)$ is R^*'s public/private key pair.
6. Guess : $\mathcal{A}$ outputs a guess v'.

If $\mu = 0$ and $T_\mu = \gamma^{ab}$, then the generated challenged trapdoor t_{w_v} is a valid trapdoor generated by user R^* and the view of $\mathcal{A}$ is distributed as if it had received the trapdoor from the real scheme. In this case we have: $\Pr[v' = v | \mu = 0] = \frac{1}{2} + \varepsilon$. If $\mu = 1$ and $T_\mu \in_R \mathit{\Gamma}$, then the generated trapdoor t_{w_v} is an invalid trapdoor. In this case we have: $\Pr[v' \neq v | \mu = 1] = \frac{1}{2}$. Putting all together we define the overall advantage of $\mathcal{B}$:

$$(\Pr[\mu = 0] \Pr[\mu' = \mu | \mu = 0] + \Pr[\mu = 1] \Pr[\mu' = \mu | \mu = 1]) - \frac{1}{2} = \frac{\varepsilon}{2}.$$

Ciphertext One-Wayness. In this section we show that our construction is COW-ATK secure in the standard model.

Theorem 3. *The $\mathcal{PKEDS}$ scheme with the message space in $\mathbb{G}$ is $\mathcal{COW}-\mathcal{ATK}$ secure in the standard model assuming* mCDH *is intractable.*

Proof. The challenger selects a bilinear map $\hat{e} : \mathbb{G} \times \Gamma \rightarrow \mathbb{G}_T$, and generators g and γ of groups $\mathbb{G}$ and Γ, respectively. Then, it picks at random $a, b \in \mathbb{Z}_p$, and gives mCDH tuples $(g, g^a, g^b) \in \mathbb{G}$ and $(\gamma, \gamma^b) \in \Gamma$ to the reduction $\mathcal{B}$. The goal of $\mathcal{B}$ is to solve the mCDH assumption and acts as $\mathcal{A}$'s challenger as follows:

1. Setup : $\mathcal{B}$ generates the server's key pair $(sk_S, pk_S) = (x, p_s = \gamma^x)$, where $x \in_R \mathbb{Z}_p$ is chosen in the same way as in the scheme. $\mathcal{B}$ publishes $\mathcal{PP}$ and the server's key pair. The distribution of the $\mathcal{PP}$ and the server's key pair is identical to the $\mathcal{PP}$ and the server's key pair of the scheme since g and γ are random generators, and x is a random exponent, all chosen in the same way as in the scheme.
2. KeyGen$_R$ *to* $\mathcal{O}_1$: $\mathcal{B}$ answers receiver's key generation queries by computing $(sk_R, pk_R) = ((y, \gamma^\alpha), p_r = g^y)$ where $\alpha, y \in_R \mathbb{Z}_p$ are chosen in the same way as in the scheme (each user has different α and y value). If the query is for R^*, $\mathcal{B}$ sets the public key equal to $p_r = g^a$ (this parameter is taken from the mCDH instance). Note that $\mathcal{B}$ does not know the private key of R^* (the reduction does not know a). The distribution of receiver's key pair is identical to the distribution of receiver's key pair of the scheme since g is a random generator, α, a and y are random exponents, all chosen in the same way as in the scheme.
3. Delegate Query *to* $\mathcal{O}_2$: $\mathcal{A}$ requests a master trapdoor for the receiver R. $\mathcal{B}$ computes random elements $r_1, r_2 \in \mathbb{Z}_p$ and outputs the master trapdoor $t_* = (t_1, t_2, t_3, t_4) = (\gamma^\alpha \cdot p_s^{r_1}, \gamma^{r_1}, \gamma^{y\alpha} \cdot p_s^{r_2}, \gamma^{r_2})$. If $R = R^*$ then $y = a$. The distribution of the master trapdoor is identical to the distribution of the master trapdoor in the scheme since r_1 and r_2 are random elements from $\mathbb{Z}_p$, same as in the real scheme.
4. TrapGen Query *to* $\mathcal{O}_3$: $\mathcal{A}$ requests a trapdoor for the pair (R, w). $\mathcal{B}$ picks random $\delta \in \mathbb{Z}_p$ and outputs the trapdoor $t_w = (t_5, t_6) = (\hat{e}(w, \gamma^\alpha) \cdot \hat{e}(p_r, p_s^\delta), \gamma^\delta)$. The distribution of the trapdoor is identical to the distribution of trapdoor in the scheme since δ is chosen at random from $\mathbb{Z}_p$, same as in the real scheme.
5. Challenge : The reduction picks at random $c' \in \mathbb{Z}_p$, computes $\frac{g^{c'}}{g^a} = g^{\bar{c}}$ (thus, $c' = a + \bar{c}$), implicitly sets $w^* = g^{b\bar{c}}$ and outputs the challenge ciphertext $\hat{c}_{w^*} = (c_1, c_2) = (g^{bc'}, g^b)$ and the challenge trapdoor $t_{w^*} = (t_5, t_6) = (\hat{e}(g^{\bar{c}}, \gamma^{b\alpha}) \cdot \hat{e}(p_r, p_s^\delta), \gamma^\delta)$. The challenge ciphertext $\hat{c}_{w^*} = (c_1, c_2) = (g^{bc'}, g^b) = (g^{b\bar{c}} \cdot g^{ab}, g^b)$ is a valid encryption of the message $w^* = g^{b\bar{c}}$ under the public key of R^* and has the same distribution as the ciphertext in the real scheme. The same holds for the trapdoor t_{w^*}.
6. Output : At the end of the game, $\mathcal{A}$ outputs the message w'.

$\mathcal{B}$ checks whether $\hat{e}(w', \gamma) \stackrel{?}{=} \hat{e}(g^a, \gamma^b)$. If so, then $\mathcal{A}$ has decrypted the challenge ciphertext and $\mathcal{B}$ uses the output of $\mathcal{A}$ to solve the mCDH assumption: $g^{ab} = \frac{c_1}{w'}$, which will reach a contradiction and proof the theorem. □

6 Applications

In this section we illustrate two applications of the $\mathcal{PKEDS}$ scheme: to detect encrypted malwares and to forward encrypted emails.

Detecting Encrypted Malware. A polymorphic virus uses encryption to modify its form as it spreads in such a way that different infected files have different byte-strings (each file is encrypted with a different key) [MC00]. The polymorphic virus instance is divided into three parts: the decryption algorithm, the decryption key and the encrypted virus. The decryption algorithm uses the decryption key in order to decrypt and run the virus. The fact that polymorphic viruses store the decryption key within each virus instance, makes them detectable by a virus scanner. As pointed out in the introduction, in this paper we consider attackers who use encryption to hide malware even in a more powerful way. Namely, in our attack scenario an attacker does not include the decryption key within the encrypted data, indeed an attacker does not know the decryption key which belongs to the receiver. Hence, the virus instance that we consider contains only the encrypted malware and the decryption algorithm.

In the proposed $\mathcal{PKEDS}$, the server can use the master trapdoor, a ciphertext and a malware signature, to check whether the ciphertext contains the malware signature, without decrypting it. This kind of detection is known as signature-based detection and is performed by most existing antivirus software packages, which maintain a database with known malware signatures and check whether the scanned data has the same signature as one of the signatures stored in the database. If the signatures match, then a malware is found and the antivirus takes further steps to quarantine, repair or delete the data. In our context we assume that the server has a database with known malware signatures and for each signature, using the master trapdoor t_*, it creates a trapdoor and checks whether the trapdoor and the scanned ciphertext have the same signature. If so, then a malware is detected and the server takes further steps to quarantine or delete the ciphertext, otherwise the ciphertext is clean and is forwarded to the receiver. The crucial property of the scheme is that ciphertexts are both searchable and decryptable, thus the server can search every part of the ciphertext for a possible malware. Note that allowing the server to search the encrypted data does not mean that the server can perform any kind of intrusion detection. For instance, 0-day attacks cannot be detected through signature-based detection.

Roschke et al. [RICM10] propose a technique which detects malicious content in the encrypted data. The solution presented in [RICM10] is based on the IBE [BF01] scheme and suffers from the key escrow property where the compromise of the master secret key compromises the whole system. The conceptual difference between our approach and the approach presented in [RICM10], is that the technique in [RICM10] uses the master secret key of the IBE to decrypt the ciphertext and then uses the virus scanner to scan the plaintext, while in our approach scanning can be done in the encrypted data, without having to decrypt it.

Forwarding Encrypted Emails. The original motivation for a $\mathcal{PEKS}$ scheme is to allow an email server to categorize user encrypted emails based on keywords contained in the message text. Using this property, Bob can create trapdoors t_{work} and t_{family}, and instruct the server to forward his encrypted emails tagged with the word "work" to his secretary and encrypted emails tagged with the word "family" to one of his family members. Waters et al. [WBDS04] showed that $\mathcal{PEKS}$ schemes can also be used to build an searchable audit log which is encrypted. The $\mathcal{PKEDS}$ scheme adds the decryption property to the $\mathcal{PEKS}$ scheme and as such can be used in every application that $\mathcal{PEKS}$ can be used. The $\mathcal{PKEDS}$ scheme has the following additional advantages compared to $\mathcal{PEKS}$:

- $\mathcal{PKEDS}$ can be used alone, without employing an additional $\mathcal{PKE}$ encryption scheme, to send encrypted messages in an open environment. This property inherently brings an additional advantage - each word of the message becomes searchable by the server. For instance, to encrypt a message m with consists from words $w_1, ..., w_k$, the sender generates the ciphertext:

  $$(\mathsf{Encrypt}_{\mathcal{PKEDS}}(w_1)||...||\mathsf{Encrypt}_{\mathcal{PKEDS}}(w_k))$$

  Where $\mathsf{Encrypt}_{\mathcal{PKEDS}}$ is the encryption algorithm for the $\mathcal{PKEDS}$ scheme. It is clear that to reveal the message m, the receiver has to decrypt each searchable ciphertext separately. From the computational point of view, using $\mathcal{PKEDS}$ alone might be expensive since it requires a number of $\mathcal{PKEDS}$ ciphertexts linear in the number of words in the document. Note that in $\mathcal{PEKS}$ the server can search only for keywords and this might be a problem for scenarios when the original message might contain some words that appear in the receiver's query but do not appear in the keyword list, and as a result the server will not forward these documents to the receiver.
- $\mathcal{PKEDS}$ can be used in the same way as $\mathcal{PEKS}$ is used, namely, use $\mathcal{PKEDS}$ to encrypt only keywords in addition to a non-searchable $\mathcal{PKE}$ scheme which encrypts the original message. For instance, to encrypt a message m with keywords $w_1, ..., w_k$, the sender generates the ciphertext:

  $$(\mathsf{Encrypt}_{\mathcal{PKE}}(m)||\mathsf{Encrypt}_{\mathcal{PKEDS}}(w_1)||...||\mathsf{Encrypt}_{\mathcal{PKEDS}}(w_k))$$

  Where $\mathsf{Encrypt}_{\mathcal{PKE}}$ is a regular encryption function for the $\mathcal{PKE}$ scheme and $\mathsf{Encrypt}_{\mathcal{PKEDS}}$ is the encryption function for the $\mathcal{PKEDS}$ scheme. Unlike $\mathcal{PEKS}$ which does not guarantee any relation between keywords (encrypted under $\mathcal{PEKS}$) and the original message (encrypted under $\mathcal{PKE}$), $\mathcal{PKEDS}$ guarantees this *relation* since it allows the receiver to decrypt the searchable ciphertext and check whether the keywords indeed describe the original message. Another benefit is that the receiver can categorize her messages according to keywords, unlike in $\mathcal{PEKS}$ where the receiver cannot categorize her messages since the searchable ciphertext is not decryptable and consequently the receiver, without decrypting the ciphertext, does not know which keywords describe the message. From the computational point of view, using $\mathcal{PKEDS}$ in addition to another $\mathcal{PKE}$ scheme would require a number of $\mathcal{PKEDS}$ ciphertexts linear in the number of *keywords* in the message, same as in $\mathcal{PEKS}$.

7 Conclusion

In this work we have presented a private-key encryption with delegated search ($\mathcal{PKEDS}$) with a security proof in the standard model. In the proposed scheme the private key holder creates a master trapdoor t_* and delegates to another entity (i.e. the server) the ability to search ciphertexts intended for the receiver without decrypting it. The main property of the scheme is that ciphertexts are both searchable and decryptable, thus the scheme can be used to search not only for keywords describing the document, but search also for words inside the document. The proposed scheme also allows the receiver to provide the server with a special key (a.k.a. trapdoor t_w) associated with a specific word w, such that it enables the server to test whether the word w is in the ciphertext. As an application, we show how $\mathcal{PKEDS}$ can be used for detecting encrypted malware and for forwarding encrypted email.

Acknowledgments

We thank Qiang Tang and the anonymous reviewers for their valuable suggestions and comments.

References

[ABC$^+$05] 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: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 205–222. Springer, Heidelberg (2005)

[BDCOP04] 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)

[BF01] Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)

[BGMM05] Ballard, L., Green, M., Medeiros, D.B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. Technical report, Cryptology ePrint Archive, Report 2005/417 (2005), `http://eprint.iacr.org/2005/417`

[BSNS06] Baek, J., Safavi-Naini, R., Susilo, W.: On the integration of public key data encryption and public key encryption with keyword search. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 217–232. Springer, Heidelberg (2006)

[BSNS08] 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, Part I. LNCS, vol. 5072, pp. 1249–1259. Springer, Heidelberg (2008)

[BW06] Boyen, X., Waters, B.: Anonymous hierarchical identity-based encryption (without random oracles). In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 290–307. Springer, Heidelberg (2006)

[BW07] Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)

[CM09] Chatterjee, S., Menezes, A.: On Cryptographic Protocols Employing Asymmetric Pairings – The Role of Ψ Revisited. Technical report, Cryptology ePrint Archive, Report 2009/480 (2009), http://eprint.iacr.org/2009/480

[ElG85] El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)

[FP07] Fuhr, T., Paillier, P.: Decryptable searchable encryption. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 228–236. Springer, Heidelberg (2007)

[GPS08] Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics 156(16), 3113–3121 (2008)

[HL07] Hwang, Y., Lee, P.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 2–22. Springer, Heidelberg (2007)

[HW08] Hofheinz, D., Weinreb, E.: Searchable encryption with decryption in the standard model. Technical report, Cryptology ePrint Archive, Report 2008/423 (2008), http://eprint.iacr.org/2008/423

[MC00] Morar, J.F., Chess, D.M.: Can Cryptography Prevent Computer Viruses? In: VIRUS, vol. 127 (2000)

[RICM10] Roschke, S., Ibraimi, L., Cheng, F., Meinel, C.: Secure Communication Using Identity Based Encryption. In: De Decker, B., Schaumüller-Bichl, I. (eds.) CMS 2010. LNCS, vol. 6109, pp. 256–267. Springer, Heidelberg (2010)

[SSW09] Shen, E., Shi, E., Waters, B.: Predicate privacy in encryption systems. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 457–473. Springer, Heidelberg (2009)

[WBDS04] Waters, B.R., Balfanz, D., Durfee, G., Smetters, D.K.: Building an encrypted and searchable audit log. In: Proceedings of ISOC Network and Distributed System Security Symposium (NDSS 2004), Citeseer (2004)

[ZI07] Zhang, R., Imai, H.: Generic combination of public key encryption with keyword search and public key encryption. In: Bao, F., Ling, S., Okamoto, T., Wang, H., Xing, C. (eds.) CANS 2007. LNCS, vol. 4856, pp. 159–174. Springer, Heidelberg (2007)