Efficient Public Key Encryption with Keyword Search Schemes from Pairings

Chunxiang Gu, Yuefei Zhu, Yajuan Zhang

Network Engineering Department of Information Engineering University,

P.O. Box 1001-770, Zhengzhou, 450002, P.R.China

E-mail: gcxiang5209@Yahoo.com.cn

Abstract

Public key encryption with keyword search (PEKS) enables user Alice to send a secret key T_W to a server that will enable the server to locate all encrypted messages containing the keyword W, but learn nothing else. In this paper, we propose a new PKES scheme based on pairings. There is no pairing operation involved in the encryption procedure. Then, we provide further discussion on removing secure channel from PKES, and present an efficient *secure channel free PKES* scheme. Our two new schemes can be proved secure in the random oracle model, under the appropriate computational assumptions.

Keywords: Public key encryption with keyword search, pairings, provable secure.

I. Introduction

In 2004, Boneh et.al [1] proposed the concept of *public key encryption with keyword search* (PEKS) scheme to enable one to search encrypted keywords without compromising the security of the original data. Suppose Bob wants to send Alice a message M with keywords $W_1, W_2, ..., W_n$. Let pk_A be Alice's public key. Bob encrypts M using a standard public key encryption $E(.)$. He then appends to the resulting ciphertext a list of PEKS ciphertext of each keyword. That is $E(M, pk_A)||PKES(W_1, pk_A)||...||PKES(W_n, pk_A)$. This kind of encrypted messages may be stored in a server. Alice can give the server a certain trapdoor T_W through a secure channel that enables the server to test whether one of the keywords associated with the message is equal to the word W of Alice's choice. Given $PKES(W', pk_A)$ and T_W, the server can test whether $W = W'$. If $W \neq W'$ the server learns nothing more about W'.

Such PEKS scheme can be widely used in many practical applications. For instance, Boneh et.al [1] explain that PEKS provides a mechanism that allows user Alice to have his email server extract encrypted emails that contain a particular keyword by providing a trapdoor corresponding to the keyword, while the email server and other parties excluding Alice do not learn anything else about the email. Shortly after Boneh et al.'s work, Waters et al. [2] showed that the PEKS scheme can be applied to build encrypted and searchable audit logs.

The scheme of Boneh et.al [1] needs secure channel to transmit trapdoors to the server. However, building a secure channel is usually expensive. Very recently, Baek et al. [3] discussed ”removing secure channel”, and provided a notion of *secure channel free public key encryption with keyword search* (SCF-PEKS) scheme.

In this paper, we propose a new PKES scheme based on pairings. Its encryption procedure needs no pairing operation. So our scheme is more efficient than that of Boneh et.al’s. Then, we provide further discussion on the notion and security model for SCF-PEKS scheme, and present an efficient SCF-PEKS scheme. The new schemes can be proved secure in the random oracle model, under the appropriate computational assumptions.

The rest of this paper is organized as follows: In Section 2, we recall some preliminary works. In Section 3, we present a new PKES scheme with efficiency discussion and security proof. In Section 4, we provide further discussion on the formal model for SCF-PEKS schemes, and present an new efficient SCF-PEKS scheme with provable security. Finally, we end the paper with a brief conclusion.

II. Preliminaries

A. Public Key Encryption with Keyword Search

Definition 1. A public key encryption with Keyword Search (PEKS) scheme consists of four polynomial-time algorithms:

- ***KeyGen*:** Take as input a security parameter λ, generate a public/private key pair (pk, sk).
- ***Trapdoor*:** Take as input the receiver’s private key sk and a word W, produce a trapdoor T_W.
- ***PKES*:** Take as input the receiver’s a public key pk and a word W, produce a searchable encryption of W.
- ***Test*:** Take as input the receiver’s public key pk, a searchable encryption $C = PEKS(pk, W')$, and a trapdoor $T_W = Trapdoor(sk, W)$, output 1 (”yes”) if $W = W'$ and 0 (”no”) otherwise.

Consistency requires that for any keyword W, $(pk, sk) = KeyGen(1^\lambda)$, $T_W = Trapdoor(sk, W)$, we have $Test(pk, PEKS(pk, W), T_W) = 1$.

In [1], Boneh et.al defined a security notion for PEKS schemes– ”indistinguishability of PEKS against chosen keyword attack” (IND-CKA).

IND-CKA game:

- **KeyGen**: The challenger runs the $KeyGen(\lambda)$ algorithm to generate (pk, sk). It gives pk to the attacker.
- **Phase 1**: The attacker can adaptively ask the challenger for the trapdoor T_W for any keyword $W \in \{0,1\}^*$ of his choice.
- **Challenge**: At some point, the attacker $\mathcal{A}$ sends the challenger two words W_0, W_1 on which it wishes to be challenged. The only restriction is that the attacker did not previously ask for the trapdoors T_{W_0} or T_{W_1}. The challenger picks a random $b \in \{0,1\}$ and gives the attacker $C = PEKS(pk, W_b)$ as the challenge PEKS ciphertext.

- **Phase 2**: The attacker can continue to ask for trapdoors T_W for any keyword W of his choice as long as $W \neq W_0, W_1$.
- **Guess**: Eventually, the attacker $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

Such an adversary $\mathcal{A}$ is called an IND-CKA adversary. $\mathcal{A}$'s advantage in attacking the scheme $\mathcal{E}$ is defined as the following function of the security parameter λ:

$$Adv_{\mathcal{E},\mathcal{A}}(\lambda) = |Pr[b = b'] - 1/2|.$$

The probability is over the random bits used by the challenger and the adversary.

Definition 2. A PKES scheme $\mathcal{E}$ is IND-CKA secure if for any polynomially time adversary $\mathcal{A}$, $Adv_{\mathcal{E},\mathcal{A}}(\lambda)$ is negligible.

B. Bilinear Pairings

Let $(G_1, +)$ and $(G_2, \cdot)$ be two cyclic groups of prime order q. $\hat{e} : G_1 \times G_1 \rightarrow G_2$ be a map which satisfies the following properties.

1) Bilinear: $\forall P, Q \in G_1, \forall \alpha, \beta \in Z_q, \hat{e}(\alpha P, \beta Q) = \hat{e}(P, Q)^{\alpha\beta}$;
2) Non-degenerate: If P is a generator of G_1, then $\hat{e}(P, P)$ is a generator of G_2;
3) Computable: There is an efficient algorithm to compute $\hat{e}(P, Q)$ for any $P, Q \in G_1$.

Such an bilinear map is called an *admissible bilinear pairing* [4]. The Weil pairings and the Tate pairings of elliptic curves can be used to construct efficient admissible bilinear pairings.

We review two complexity problems related to bilinear pairings: the Bilinear Diffie-Hellman (BDH) problem [4] and the Bilinear Diffie-Hellman Inverse (BDHI) problem [5], [6]. Let P be a generator of G_1, and $a, b, c \in Z_q^*$.

- **BDH problem**: given $P, aP, bP, cP \in G_1$, output $\hat{e}(P, P)^{abc}$. An algorithm $\mathcal{A}$ solves BDH problem with the probability ε if
$$Pr[\mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}] \geq \varepsilon,$$
where the probability is over the random choice of generator $P \in G_1^*$, the random choice of $a, b, c \in Z_q^*$ and random coins consumed by $\mathcal{A}$.
- k**-BDHI problem**: given $(P, aP, a^2P, ...a^kP \in (G_1^*)^{k+1}$, output $\hat{e}(P, P)^{a^{-1}}$. An algorithm $\mathcal{A}$ solves k-BDHI problem with the probability ε if
$$Pr[\mathcal{A}(P, aP, a^2P, ...a^kP) = \hat{e}(P, P)^{a^{-1}}] \geq \varepsilon,$$
where the probability is over the random choice of generator $P \in G_1^*$, the random choice of $a \in Z_q^*$ and random coins consumed by $\mathcal{A}$.

We assume through this paper that BDH problem and k-BDHI problem are intractable, which means that there is no polynomial time algorithm to solve BDH problem or k-BDHI problem with non-negligible probability.

III. A New PKES Scheme from Pairings

A. The Scheme

Let $(G_1,+)$ and $(G_2,\cdot)$ be two cyclic groups of prime order q, $\hat{e}: G_1 \times G_1 \to G_2$ be an admissible bilinear pairing, $H_1 : \{0,1\}^* \to Z_q^*$ and $H_2 : G_2 \to \{0,1\}^{\log q}$ be two hash functions. P is a generator of G_1, $\mu = \hat{e}(P,P)$. The scheme is described as following:

- ***KeyGen***: Pick a random $x \in Z_q^*$, compute $X = xP$, and output $pk = X$, and $sk = x$.
- ***Trapdoor***: Take as input secret key x and keyword W, and output $T_W = (H_1(W)+x)^{-1}P$.
- ***PEKS***: Take as input public key X and a keyword W, select randomly $r \in Z_q^*$, compute $U = rH_1(W)P + rX$, $c = H_2(\mu^r)$ and output (U,c).
- ***Test***: For input public key X, searchable encryption cipher-text (U,c) and trapdoor T_W, test if $H_2(\hat{e}(T_W,U)) = c$. If so, output 1; otherwise, output 0.

B. Consistency and Efficiency

Consistency of the scheme is easily proved as follows:

$$\begin{aligned}
H_2(\hat{e}(T_W,U)) &= H_2(\hat{e}((H_1(W)+x)^{-1}P, rH_1(W)P + rX)) \\
&= H_2(\hat{e}((H_1(W)+x)^{-1}P, r(H_1(W)+x)P)) \\
&= H_2(\hat{e}(P,P)^r) = c.
\end{aligned}$$

Denote by M an ordinary scalar multiplication in $(G_1,+)$, by E an Exp. operation in $(G_2,.)$, and by $\hat{e}$ a computation of the pairing. The hash function $H_1 : \{0,1\}^* \to G_1^*$ used by the scheme in [1] usually requires a "Maptopoint operation" [4] to map a keyword to an element in G_1. As discussed in [4], Maptopoint operation (denoted by P) is so inefficient that we can't neglect it. Do not take other operations into account. We compare our scheme to the scheme in [1] in the following table.

schemes	KeyGen	Trapdoor	PKES	Test
scheme in [1]	$1M$	$1M + 1P$	$2M + 1P + 1\hat{e}$	$1\hat{e}$
proposed	$1M$	$1M$	$2M + 1E$	$1\hat{e}$

Note: The hash function used in our scheme which maps a keyword to an element in Z_q^* is so efficient that we usually can neglect it.

The construction seems to be more efficient in performance. Although fruitful achievements [7], [8] have been made in enhancing the computation of pairings, the computation of pairings is still time consuming. Our new scheme requires no pairing operation in PKES procedure.

Some general performance enhancements can also be applied to our scheme. For pre-selected $P \in G_1$ and $\mu \in G_2$, there are efficient algorithms [9] to compute $rH_1(ID_X)P$ and μ^r for a random $r \in Z_q^*$ by pre-computing and storing.

C. Security Proof

Lemma 1. *Let $\mathcal{F}_0$ be an IND-CKA adversary that has advantage $\varepsilon(\lambda)$ within a time bound $T(\lambda)$. Suppose $\mathcal{F}_0$ makes at most $q_T > 0$* ***Trapdoor*** *queries, $q_1 > 0$ hash function queries to H_1 and $q_2 > 0$ hash function queries to H_2. Let $n = max\{q_1, 2q_T\}$. Then there is an algorithm $\mathcal{F}_1$ that solves the n-BDHI problem with advantage at least $\varepsilon(\lambda)/(nq_2)$ with a running time $\mathcal{O}(T(\lambda))$.*

Proof: $\mathcal{F}_1$ is given input parameters of pairing $(q, G_1, G_2, \hat{e})$ and a random instance $(P, aP, a^2P, ..., a^nP)$ of the n-BDHI problem, where P is random in G_1^* and a is a random in Z_q^*. $\mathcal{F}_1$ simulates the challenger and interacts with $\mathcal{F}_0$ as follows:

- **KeyGen**: 1. Randomly choose different $h_0, h_1, ...h_{n-1} \in Z_q^*$, and compute $f(x) = \prod_{i=1}^{n-1}(x + h_i) = \sum_{i=0}^{n-1} c_i x^i$.
  2. Compute $Q = \sum_{i=0}^{n-1} c_i a^i P = f(a)P$, $aQ = \sum_{i=0}^{n-1} c_i a^{i+1} P$, and $Q' = \sum_{i=1}^{n-1} c_i a^{i-1} P$. In the (unlikely) situation where $Q = 1_{G_1}$, there exist an $h_i = -a$, hence, $\mathcal{F}_1$ can solve the n-BDHI problem directly and abort.
  3.Compute $f_i(x) = f(x)/(x + h_i) = \sum_{j=0}^{n-2} d_j x^j$. Obviously, $(a + h_i)^{-1}Q = (a + h_i)^{-1}f(a)P = f_i(a)P = \sum_{j=0}^{n-2} d_j a^j P$ for $1 \le i \le n$.
  4. Randomly choose an index t with $1 \le t \le n$, set $v = 0$, and start by giving $\mathcal{F}_0$ the public key $Y = aQ - h_0Q$.
- **Phase 1:** H_1**-queries**. $\mathcal{F}_1$ maintains a H_1_list, initially empty. For a query W, if W already appears on the H_1_list in a tuple (W, g, D), $\mathcal{F}_1$ responds with g. Otherwise, sets $v = v + +$, $W_v = W$, if $v = t$, $\mathcal{F}_1$ sets $g_v = h_0$, $D_v = \perp$; otherwise, $\mathcal{F}_1$ selects a random $n \ge \iota > 0$ which has not been chosen and sets $g_v = h_\iota + h_0$, $D_v = (a + h_\iota)^{-1}Q$. In both case, adds the tuple (W_v, g_v, D_v) to H_1_list and responds with g_v.
- **Phase 1:** H_2**-queries**. $\mathcal{F}_1$ maintains a H_2_list, initially empty. For a query e_i, $\mathcal{F}_1$ checks if e_i appears on the H_2_list in a tuple (e_i, u_i). If not, $\mathcal{F}_1$ picks a random $u_i \in \{0,1\}^{\log q}$, and adds the tuple $(e_i, u_i)$to the H_2_list. $\mathcal{F}_1$ returns u_i to $\mathcal{F}_0$.
- **Phase 1:** ***Trapdoor*** **queries**: For input W_i, without any loss of generality, we can assume that W_i has already been asked to oracle H_1. $\mathcal{F}_1$ searches in H_1_list for (W_i, g_i, D_i). If $D_i = \perp$ then $\mathcal{F}_1$ aborts. Otherwise, $\mathcal{F}_1$ responds with D_i.
- **Challenge**: Once $\mathcal{F}_0$ decides that Phase 1 is over it outputs two keywords W'_0, W'_1 on which it wishes to be challenged. $\mathcal{F}_1$ responds as follows:
  1. $\mathcal{F}_1$ runs the above algorithm for responding to H_1-queries twice to obtain (W'_0, g'_0, D'_0) and (W'_1, g'_1, D'_1). If both $D'_0 \ne \perp$ and $D'_1 \ne \perp$ then $\mathcal{F}_1$ aborts. Otherwise, $\mathcal{F}_1$ responds with the challenge ciphertext (bQ, ξ) for random selected $b \in Z_q^*$ and $\xi \in \{0,1\}^{\log q}$. (Observe that if (bQ, ξ) is a cipher-text corresponding to W'_ι with $\iota \in \{0,1\}$ satisfying $D'_\iota = \perp$, by definition, the decryption of C is $\xi = H_2(\hat{e}(T_{W'_\iota}, bQ)) = H_2(\hat{e}(a^{-1}Q, bQ)) = H_2(\hat{e}(Q,Q)^{a^{-1}b})$.)
- **Phase 2:** H_1**-queries,** H_2**-queries,** ***Trapdoor*** **queries**. $\mathcal{F}_1$ responds to these queries in the same way it does in Phase 1 with the only restriction that $W_i \ne W'_0, W'_1$ for *Trapdoor* queries.

- **Guess**: Eventually $\mathcal{F}_0$ produces its guess $\iota' \in \{0,1\}$ for ι.

$\mathcal{F}_1$ keeps interacting with $\mathcal{F}_0$ until $\mathcal{F}_0$ halts or aborts. If $\mathcal{F}_0$ produces a guess ι', $\mathcal{F}_1$ picks a random tuple (e_i, u_i) from the H_2_list. $\mathcal{F}_1$ computes $\alpha = e_i^{b^{-1}}$, $\beta = \hat{e}(Q', Q + c_0P)$ and outputs $(\alpha/\beta)^{c_0^{-2}}$ as the solution to the given instance of q_1-BIDH problem. (Note that if $\alpha = \hat{e}(Q,Q)^{a^{-1}}$, then $(\alpha/\beta)^{c_0^{-2}} = \hat{e}(P,P)^{a^{-1}}$.)

This completes the description of $\mathcal{F}_1$.

Suppose that in a real attack game $\mathcal{F}_0$ is given the public key $(Q, Y = aQ - h_0Q)$ and $\mathcal{F}_0$ asks to be challenged on words W_0' and W_1'. In response, $\mathcal{F}_0$ is given a challenge (bQ, ξ). Then, just as discussed in [1], in the real attack game $\mathcal{F}_0$ issues an H_2 query for either $H_2(\hat{e}(T_{W_0'}, bQ))$ or $H_2(\hat{e}(T_{W_1'}, bQ))$ with probability at least $2\varepsilon(\lambda)$.

Now, assuming $\mathcal{F}_1$ does not abort, we know that $\mathcal{F}_1$ simulates a real attack game perfectly up to the moment when $\mathcal{F}_0$ issues a query for either $H_2(\hat{e}(T_{W_0'}, bQ))$ or $H_2(\hat{e}(T_{W_1'}, bQ))$. Therefore, the value $H_2(\hat{e}(T_{W_\iota'}, bQ)) = H_2(\hat{e}(Q,Q)^{a^{-1}b})$ will appear in the H_2-list with probability at least $\varepsilon(\lambda)$. $\mathcal{F}_1$ will choose the correct pair with probability at least $1/q_2$.

During the simulation, $\mathcal{F}_1$ does not abort in phases 1 or 2 because of $\mathcal{F}_0$'s ***Trapdoor*** queries is $1 - q_T/n$. The probability that $\mathcal{F}_1$ does not abort during the challenge step is $2/n$. Because $n \geq 2q_T$, we know that the probability that $\mathcal{F}_1$ does not abort during the simulation is $(1 - q_T/n)2/n \geq 1/n$.

Therefore, $\mathcal{F}_1$'s success probability overall is at least $\varepsilon(\lambda)/(nq_2)$.

IV. PKES Schemes Without Secure Channel

PKES schemes need secure (encrypted and authenticated) channels between users and servers. However, building a secure channel is usually expensive. In [3], Baek et.al suggested a formal model for *secure channel free public key encryption with keyword search* (SCF-PEKS) scheme, which defines SCF-PEKS scheme with six algorithms. In this section, we provide further discussion on the formal model for SCF-PEKS schemes, and present an new efficient SCF-PEKS scheme with provable security.

A. New Formal Model for SCF-PEKS Schemes

A SCF-PEKS scheme enables the sender to use the server's public key as well as the receiver's public key to generate PEKS ciphertexts. The receiver then can send a trapdoor to retrieve data associated with the encrypted keyword via a public channel.

Definition 3. A SCF-PEKS scheme consists of four polynomial-time algorithms:

- ***KeyGen***: Take as input a security parameter λ, generate a public/private key pairs (pk, sk). This algorithm is used to generate key pairs for users (including the receiver and the server).
- ***Trapdoor***: Take as input the receiver's private key sk_r and a word W, produce a trapdoor T_W.
- ***PKES***: Take as input the receiver's public key pk_r, the server's public key pk_s and a word W, produce a searchable encryption of W.

- ***Test***: Take as input the server's secret key sk_s and the receiver's public key pk_r, a searchable encryption $S = PEKS(pk_r, pk_s, W')$, and a trapdoor $T_W = Trapdoor(sk_r, W)$, output 1 ("yes") if $W = W'$ and 0 ("no") otherwise.

Consistency requires that for any keyword W, receiver's key pair $(pk_r, sk_r) = KeyGen(1^\lambda)$, server's key pair $(pk_s, sk_s) = KeyGen(1^\lambda)$, $T_W = Trapdoor(sk_r, W)$, we have $Test(sk_s, pk_r, PEKS(pk_r, pk_s, W), T_W) = 1$.

As to security, informally, we can say a SCF-PEKS scheme is secure if it can catch the following goals:

- The attacker without the trapdoors for given keywords cannot tell the PEKS ciphertext is produced from which keyword, even he knows the server's secret key. We call this security property "*indistinguishability against chosen keyword attack with server's secret key*" (IND-CKA-SSK).
- The attacker without the server's private key cannot make any decisions about the PEKS ciphertexts even though the attacker gets all the trapdoors for the keywords that it holds. We call this security property "*indistinguishability against chosen keyword attack with all trapdoors*" (IND-CKA-AT).

Formally, we define the following two security notions.

IND-CKA-SSK game:

- **KeyGen**:The challenger runs the $KeyGen(\lambda)$ algorithm twice to generate the server's key pair (pk_s, sk_s) and the receiver's key pair (pk_r, sk_r). It gives pk_s, pk_r, sk_s to the attacker.
- **Phase 1**, **Challenge**, **Phase 2**, **Guess**: The attacker $\mathcal{A}$ does these steps almost the same as that in IND-CKA game, except that the challenge ciphertext is $C = PEKS(pk_r, pk_s, W_b)$, where $b \in_R \{0, 1\}$, W_0, W_1 are the two words to be challenged.

The adversary $\mathcal{A}$ is called an IND-CKA-SSK adversary. $\mathcal{A}$'s advantage is defined as:

$$Adv_{\mathcal{E},\mathcal{A}}^{IND-CKA-SSK}(\lambda) = |Pr[b = b'] - 1/2|.$$

The probability is over the random bits used by the challenger and the adversary.

Definition 4. A SCF-PKES scheme $\mathcal{E}$ is IND-CKA-SSK secure if for any polynomially time adversary $\mathcal{A}$, $Adv_{\mathcal{E},\mathcal{A}}^{IND-CKA-SSK}(\lambda)$ is negligible.

IND-CKA-AT game:

- **KeyGen**: The challenger runs the $KeyGen(\lambda)$ algorithm twice to generate the server's key pair (pk_s, sk_s) and the receiver's key pair (pk_r, sk_r). It gives pk_s, pk_r to the attacker.
- **Phase 1**: The attacker can adaptively ask the challenger for the trapdoor T_W for any keyword $W \in \{0, 1\}^*$ of his choice.
- **Challenge**: At some point, the attacker $\mathcal{A}$ sends the challenger two words W_0, W_1 on which it wishes to be challenged. The challenger picks a random $b \in \{0, 1\}$ and gives the attacker $C = PEKS(pk_r, pk_s, W_b)$ as the challenge PEKS.

- **Phase 2**: The attacker can continue to ask for trapdoors T_W for any keyword W of his choice.
- **Guess**: Eventually, the attacker $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

The adversary $\mathcal{A}$ is called an IND-CKA-AT adversary. $\mathcal{A}$'s advantage is defined as:

$$Adv_{\mathcal{E},\mathcal{A}}^{IND-CKA-AT}(\lambda) = |Pr[b = b'] - 1/2|.$$

The probability is over the random bits used by the challenger and the adversary.

Definition 5. A SCF-PKES scheme $\mathcal{E}$ is IND-CKA-AT secure if for any polynomially time adversary $\mathcal{A}$, $Adv_{\mathcal{E},\mathcal{A}}^{IND-CKA-AT}(\lambda)$ is negligible.

B. A New SCF-PEKS Scheme from pairings

Let $(G_1, +)$ and $(G_2, \cdot)$ be two cyclic groups of prime order q, $\hat{e} : G_1 \times G_1 \to G_2$ be an admissible bilinear pairing, $H_1 : \{0,1\}^* \to Z_q^*$ and $H_2 : G_2 \to \{0,1\}^{\log q}$ be two hash functions. P is a generator of G_1, $\mu = \hat{e}(P, P)$. The scheme is described as following:

- ***KeyGen***: Pick a random $x \in Z_q^*$, compute $X = xP$, and output $pk = X$, and $sk = x$.
- ***Trapdoor***: Take as input secret key x and keyword W, output $T_W = (H_1(W) + x)^{-1}P$.
- ***PEKS***: Take as input a receiver's public key X, a server's public key Y and a keyword W, select randomly $r_1, r_2 \in Z_q^*$, compute $U = r_1 H_1(W)P + r_1 X$, $V = r_2 P$, $c = H_2(\hat{e}(r_1 P + r_2 U, Y))$ and output (U, V, c).
- ***Test***: Take as input the receiver's public key X, the server's private key $y \in Z_q^*$, a searchable encryption cipher-text (U, V, c) and trapdoor T_W, test if $H_2(\hat{e}(yU, T_W + V)) = c$. If so, output "yes"; otherwise, output "no".

C. Consistency and Efficiency

Consistency of the scheme is easily proved as follows:

$$\begin{aligned} H_2(\hat{e}(yU, T_W + V)) &= H_2(\hat{e}(U, (H_1(W) + x)^{-1}P + r_2 P)^y) \\ &= H_2(\hat{e}(r_1(H_1(W) + x)P, (H_1(W) + x)^{-1}P)^y \cdot \hat{e}(U, r_2 P)^y) \\ &= H_2(\hat{e}(r_1 P, yP) \cdot \hat{e}(r_2 U, yP)) \\ &= H_2(\hat{e}(r_1 P + r_2 U, Y)) = c. \end{aligned}$$

Denote by M an ordinary scalar multiplication in $(G_1, +)$, by E an Exp. operation in $(G_2, .)$, by $\hat{e}$ a computation of the pairing and by P a Maptopoint operation [4]. Do not take other operations into account. We compare our scheme to the scheme in [3] in the following table.

schemes	KeyGen	Trapdoor	PKES	Test
scheme in [3]	$1M$	$1M + 1P$	$1M + 1P + 1E + 2\hat{e}$	$2M + 1\hat{e}$
proposed	$1M$	$1M$	$5M + 1\hat{e}$	$1M + 1\hat{e}$

D. Security Proof

Lemma 2. *Let $\mathcal{F}_0$ be an IND-CKA-SSK adversary that has advantage $\varepsilon(\lambda)$ within a time bound $T(\lambda)$. Suppose $\mathcal{F}_0$ makes at most $q_T > 0$ **Trapdoor** queries, $q_1 > 0$ hash function queries to H_1 and $q_2 > 0$ hash function queries to H_2. Let $n = max\{q_1, 2q_T\}$. Then there is an algorithm $\mathcal{F}_1$ that solves the n-BDHI problem with advantage at least $\varepsilon(\lambda)/(nq_2)$ with a running time $\mathcal{O}(T(\lambda))$.*

Proof: $\mathcal{F}_1$ is given input parameters of pairing $(q, G_1, G_2, \hat{e})$ and an a random instance $(P, aP, a^2P, ..., a^nP)$ of the n-BDHI problem, where P is random in G_1^* and a is a random in Z_q^*. $\mathcal{F}_1$ simulates the challenger and interacts with $\mathcal{F}_0$ as follows:

- **KeyGen**: Randomly choose different $h_0, h_1, ...h_{n-1} \in Z_q^*$, and and compute $f(x)$, Q, aQ, Q', $(a + h_i)^{-1}Q$ for $1 \le i \le n$ the same as that in the proof of Lemma 1. In the (unlikely) situation where $Q = 1_{G_1}$, there exist an $h_i = -a$, hence, $\mathcal{F}_1$ can solve the q_1-BDHI problem directly and abort.
  2. Randomly choose an index t with $1 \le t \le n$, sets $v = 0$. Select a random $y \in Z_q^*$ and start by giving $\mathcal{F}_0$ the reciver's public key $X = aQ - h_0Q$ and the server's key pair (y, yQ).
- **Phase 1: H_1-queries, H_2-queries, *Trapdoor* queries**. $\mathcal{F}_1$ responds these queries the same way as that in the proof of Lemma 1.
- **Challenge**: Once $\mathcal{F}_0$ decides that Phase 1 is over it outputs two keywords W_0', W_1' on which it wishes to be challenged. $\mathcal{F}_1$ responds as follows:
  1. $\mathcal{F}_1$ runs the above algorithm for responding to H_1-queries twice to obtain (W_0', g_0', D_0') and (W_1', g_1', D_1'). If both $D_0' \neq \perp$ and $D_1' \neq \perp$ then $\mathcal{F}_1$ aborts. Otherwise, $\mathcal{F}_1$ responds with the challenge ciphertext $(\gamma_1 Q, \gamma_2 Q, \xi)$ for random selected $\gamma_1, \gamma_2 \in Z_q^*$ and $\xi \in \{0,1\}^{\log q}$. (Observe that if $(\gamma_1 Q, \gamma_2 Q, \xi)$ is a cipher-text corresponding to W_ι' with $\iota \in \{0, 1\}$ satisfying $D_\iota' = \perp$, by definition, the decryption of C is $\xi = H_2(\hat{e}(\gamma_1 Q, T_{W_\iota'} + \gamma_2 Q)^y) = H_2(\hat{e}(\gamma_1 Q, a^{-1}Q + \gamma_2 Q)^y) = H_2(\hat{e}(Q,Q)^{\gamma_1(a^{-1}+\gamma_2)y})$.)
- **Phase 2: H_1-queries, H_2-queries, *Trapdoor* queries**. $\mathcal{F}_1$ responds to these queries in the same way it does in Phase 1 with the only restriction that $W_i \neq W_0', W_1'$ for *Trapdoor* queries.
- **Guess**:Eventually $\mathcal{F}_0$ produces its guess $\iota' \in \{0, 1\}$ for ι.

$\mathcal{F}_1$ keeps interacting with $\mathcal{F}_0$ until $\mathcal{F}_0$ halts or aborts. If $\mathcal{F}_0$ produces a guess ι', $\mathcal{F}_1$ picks a random tuple (e_i, h_i) from the H_2_list and computes $\delta = \hat{e}(Q, \gamma_2 Q)$, $\alpha = e_i^{(\gamma_1 y)^{-1}}/\delta$, $\beta = \hat{e}(Q', Q + c_0 P)$ and outputs $(\alpha/\beta)^{c_0^{-2}}$ as the solution to the given instance of q_1-BIDH problem. (Note that if $e_i = \hat{e}(Q,Q)^{\gamma_1(a^{-1}+\gamma_2)y}$, then $\alpha = \hat{e}(Q,Q)^{a^{-1}}$, hence, $(\alpha/\beta)^{c_0^{-2}} = \hat{e}(P,P)^{a^{-1}}$.)

This completes the description of $\mathcal{F}_1$. Just as discussed in the proof of Lemma 1, $\mathcal{F}_1$'s success probability overall is at least $\varepsilon(\lambda)/(nq_2)$.

Lemma 3. *Let $\mathcal{F}_0$ be an IND-CKA-AT adversary that has advantage $\varepsilon(\lambda)$ within a time bound $T(\lambda)$. Suppose $\mathcal{F}_0$ makes at most $q_T > 0$ **Trapdoor** queries, $q_1 > 0$ hash function queries to H_1 and $q_2 > 0$ hash function queries to H_2. Then there is an algorithm $\mathcal{F}_1$ that solves the BDH problem with advantage at least $2\varepsilon(\lambda)/q_2$ with a running* *time* $\mathcal{O}(T(\lambda))$.

Proof: $\mathcal{F}_1$ is given input parameters of pairing $(q, G_1, G_2, \hat{e})$ and a random instance (P, aP, bP, cP) of the BDH problem, where P is random in G_1^* and a, b, c are random elements in Z_q^*. $\mathcal{F}_1$ simulates the challenger and interacts with $\mathcal{F}_0$ as follows:

- **KeyGen**: Select randomly $x \in Z_q^*$ and start by giving $\mathcal{F}_0$ the reciver's public key $X = xP$ and the server's public key aP.
- **Phase 1:** H_1**-queries**. $\mathcal{F}_1$ maintains a H_1_list, initially empty. For a query W_i, if W_i already appears on the H_1_list in a tuple (W_i, g_i), $\mathcal{F}_1$ responds with g_i. Otherwise, $\mathcal{F}_1$ selects a random $g_i \in Z_q^*$, adds the tuple $(W_i, g_i)$to the H_1_list and responds with g_i.
- **Phase 1:** $\mathbf{H_2}$**-queries**. $\mathcal{F}_1$ maintains a H_2_list, initially empty. For a query e_i, $\mathcal{F}_1$ checks if e_i appears on the H_2_list in a tuple (e_i, h_i). If not, $\mathcal{F}_1$ picks a random $h_i \in \{0,1\}^{\log q}$, and adds the tuple $(e_i, h_i)$to the H_2_list. $\mathcal{F}_1$ returns h_i to $\mathcal{F}_0$.
- **Phase 1:** ***Trapdoor*** **queries**: For input W_i, without any loss of generality, we can assume that W_i has already been asked to oracle H_1. $\mathcal{F}_1$ searches in H_1_list for (W_i, g_i) and $\mathcal{F}_1$ responds with $D_i = (g_i + x)^{-1}P$.
- **Challenge**: Once $\mathcal{F}_0$ decides that Phase 1 is over it outputs two keywords W_0', W_1' on which it wishes to be challenged. $\mathcal{F}_1$ runs the above algorithm for responding to H_1-queries twice to obtain (W_0', g_0') and (W_1', g_1'). Selects $\iota \in \{0,1\}$ and responds with the challenge ciphertext $(g_\iota' bP + xbP, cP, \xi)$ for random selected $\xi \in \{0,1\}^{\log q}$. (Observe that if $(g_\iota' bP + xbP, cP, \xi)$ is a cipher-text corresponding to W_ι', by definition, the test procedure of C is to test $\xi = H_2(\hat{e}(g_\iota' bP + xbP, (g_\iota' + x)^{-1}P + cP)^a)$.)
- **Phase 2:** H_1**-queries,** H_2**-queries,** ***Trapdoor*** **queries**. $\mathcal{F}_1$ responds to these queries in the same way it does in Phase 1.
- **Guess**:Eventually $\mathcal{F}_0$ produces its guess $\iota' \in \{0,1\}$ for ι.

$\mathcal{F}_1$ keeps interacting with $\mathcal{F}_0$ until $\mathcal{F}_0$ halts or aborts. If $\mathcal{F}_0$ produces a guess ι', $\mathcal{F}_1$ picks a random tuple (e_i, u_i) from the H_2_list. $\mathcal{F}_1$ computes and outputs $\alpha = (e_i/\hat{e}(aP, bP))^{(g_\iota'+x)^{-1}}$ as the solution to the given instance of BDH problem. (Note that if $e_i = \hat{e}(g_\iota' bP + xbP, (g_\iota' + x)^{-1}P + cP)^a$, then $e_i = \hat{e}(bP, P)^a \hat{e}(P, P)^{bca(g_\iota'+x)}$, hence $\alpha = (e_i/\hat{e}(aP, bP))^{(g_\iota'+x)^{-1}} = \hat{e}(P, P)^{abc}$.)

This completes the description of $\mathcal{F}_1$.

We know that in the real attack game $\mathcal{F}_0$ issues an H_2 query for $H_2(\hat{e}(g_0' bP + xbP, (g_0' + x)^{-1}P + cP)^a)$ $H_2(\hat{e}(g_1' bP + xbP, (g_1' + x)^{-1}P + cP)^a)$ with probability at least $2\varepsilon(\lambda)$. $\mathcal{F}_1$simulates a real attack game perfectly up to the moment when $\mathcal{F}_0$ issues a query for $H_2(\hat{e}(g_\iota' bP + xbP, (g_\iota' + x)^{-1}P + cP)^a)$ with $\iota \in \{0,1\}$. Therefore, the value $\hat{e}(g_\iota' bP + xbP, (g_\iota' + x)^{-1}P + cP)^a$ will appear in the H_2-list with probability at least $2\varepsilon(\lambda)$. $\mathcal{F}_1$ will choose the correct pair with probability at least $1/q_2$. Therefore, $\mathcal{F}_1$'s success probability overall is at least $2\varepsilon(\lambda)/q_2$.

V. Conclusion

In this paper, first, we propose a new PKES scheme based on pairings and prove its security in the random oracle model. The new scheme is more efficient than that of Boneh et.al's. Then, we provide further discussion on the notion of SCF-PEKS scheme, give a formal security model and present an efficient SCF-PEKS scheme. The new scheme can also be proved to be secure in the random oracle model.

References

[1] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public Key Encryption with Keyword Search, In Eurocrypt 2004, LNCS 3027, pages 506-522, Springer-Verlag, 2004.

[2] B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an Encrypted and Searchable Audit Log, In Network and Distributed System Security Symposium (NDSS 2004), 2004.

[3] J. Baek, R. Safiavi-Naini, W. Susilo. Public Key Encryption with Keyword Search Revisited. Available on Cryptology ePrint Archive, Report 2005/119.

[4] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, Advances in Cryptology- CRYPTO 2001, LNCS 2139, pp. 213-229. Springer-Verlag, 2001.

[5] D. Boneh, X. Boyen. Efficient Selective ID Secure Identity Based Encryption without Random Oracles. Advances In Cryptology-Eurocrypt 2004, LNCS 3027, pp. 223-238, Springer-Verlag, 2004.

[6] F. Zhang, R. Safavi-Naini, and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In PKC'2004, LNCS 2947, pages 277-290. Springer-Verlag, 2004.

[7] P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. Advances in Cryptology-Crypto'2002, LNCS 2442, pp. 354-368. Springer-Verlag, 2002.

[8] I. Duursma and H. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p + x + d$. Advances in Cryptology-Asiacrypt'2003, LNCS 2894, pp. 111-123. Springer-Verlag, 2003.

[9] Y. Sakai, K. Sakurai. Efficient Scalar Multiplications on Elliptic Curves without Repeated Doublings and Their Practical Performance. ACISP 2000, LNCS 1841, pp. 59-73. Springer-Verlag 2000.

[10] R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054.

[11] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In SIAM Journal on Computing, 2000. Early version in proceedings of STOC'91.