Private Web Search with Constant Round Efficiency

Bolam Kang, Sung Cheol Goh and Myungsun Kim

Department of Information Security
The University of Suwon
teapack@suwon.ac.kr, goh5703@hanmail.net, msunkim@suwon.ac.kr

Abstract. Web search is increasingly becoming an essential activity as it is frequently the most effective and convenient way of finding information. However, it can be a threat for the privacy of users because their queries may reveal their sensitive information. Private web search (PWS) solutions allow users to find information in the Internet while preserving their privacy. In particular, cryptography-based PWS (CB-PWS) systems provide strong privacy guarantees.
This paper introduces a constant-round CB-PWS protocol which remains computationally efficient, compared to known CB-PWS systems. Our construction is comparable to similar solutions regarding users' privacy.

1 Introduction

Private web search (PWS) is the problem of preventing web search service providers (e.g., Google and Yahoo) from building user profiles while still making users enjoy the search functionality during web search. User profiling is usually defined as the process of implicitly learning a user profile from search-engine queries submitted by the user. Then, performing user profiling, web search service providers use a user profile to classify a given user into predefined user segments (e.g., by demographics or tastes) or to capture the online behavior of the user including the users private interests and preferences. This raises *privacy concerns* because sensitive information such as a user's name and location can be inferred from search-engine queries. Aside from the query terms, other information such as the source IP address and timestamp may reveal sensitive things about the user.

Different approaches (e.g., [10, 21, 8, 14, 20, 13]) have been proposed to tackle this problem. In these systems, the main measure of efficiency is the *round complexity*, and it is important to construct constant-round PWS systems while guaranteeing privacy. In some cases, PWS schemes without strong privacy guarantees may suffice, and we know how to construct such protocols, e.g., using a proxy. However, in this work, we focus on cryptography-based PWS (CB-PWS) systems, where strong privacy is another important design goal.

To our knowledge, known CB-PWS constructions require $O(n)$ rounds where n is the number of users [5, 14, 20], or significantly restrict the length of messages to be encrypted and hence do not lead to practical solutions to the problem [13]. In addition, the latter requires web search engines to implement and run the protocols. Search engines however do not have any incentives to implement costly protocols they cannot profit from. Unfortunately, there are no known constructions of constant-round CB-PWS while being realistic in practice.

We briefly survey what's known in this regard for private web search. We then summarize our contributions and provide a high-level overview of our construction.

Proxy-based PWS. The first approach is the use of an anonymous proxy (e.g., [1, 2, 21]). Users can expect that anonymizers prohibit the creation of user profiles through query unlinkability. There are several options from simple mechanisms achieving just a low level of anonymity in web searches to more reliable, but complicate systems based on onion routing [19] such as the Tor network [7]. However, the effectiveness of simple solutions is clearly limited. Also, as pointed out by [5] Tor cannot be installed and configured with relative ease. Further, it is well known that the HTTP requests over Tor get significantly slow [21]. For example, it takes 10 seconds on average to submit a query to Google even in a setting of using paths of length 2 (the default length is 3).

Obfuscation-based PWS. Another approach to provide privacy during web search is based on a query obfuscation technique (e.g., [10, 8, 18, 3]). Roughly speaking, a class of solutions using query obfuscation is to blend the real queries into a stream of fake queries so that web search engines cannot create a correct profile. From the privacy point of view, these obfuscation-based solutions have a critical drawback: automated queries have different features from the actual queries entered by a user, such as randomness. The authors in [17] demonstrated a concrete classifier that can distinguish real queries from fake queries generated by TrackMeNot [3], only with a mean of misclassification around 0.02%.

Cryptography-based PWS. The last class of solutions is to use cryptographic algorithms such as public-key encryption and shuffle. One of the main advantages of CB-PWS over other approaches is that they provide strong privacy guarantees. Besides, they are not affected by the misclassification issue, and are generally faster than anonymizer-based solutions. To our knowledge, known solutions are [5, 14, 20, 13].

Their basic idea is that joining a small-sized group, each user encrypts his search query and sends it to other members; then according to a predefined order, one user hands over a shuffled list of encrypted queries to its neighbor and finally the last user broadcasts its shuffled version. After group decryption, each user gets a set of queries but he can not know who submitted which query. As a result, web search engines cannot build user profiles.

1.1 Our Contributions

Our main contribution is the first *practical* protocol that only has $O(n)$ modular exponentiations and a *constant number of rounds* in the user side, where n is the number of users (i.e., a group size). According to our analysis of existing CB-PWS schemes (i.e., [5, 14, 20]), existing solutions require $O(n)$ computation complexity and $O(n)$ rounds in the same conditions.

A user has words or sentences for a query, and then is about to submit it to a specific search engine. Using our protocol, the user first disperses his query term into n pieces using Shamir's secret sharing scheme. This can be very efficiently done in a finite field (see Section 4.1). Then encrypting each share into n ciphertexts under a public-key cryptosystem, which requires at most $O(n)$ modular exponentiations, the user sends to the encrypted shares to corresponding users. Finally all users send to a group manager a list of re-masked and shuffled ciphertexts. (In CB-PWS solutions, a small group of users is created and maintained by a specific entity called a group manager. We will explain the entity later.)

Our key technical contribution is the introduction of distributing a query term instead of distributed private-key generation, among n users. Roughly speaking, known CB-PWS schemes demand each user to compute only a singe ciphertext of the query term, but a collection of all n ciphertexts should be shuffled in a *relay manner* among all users. This step is essential to provide unlinkability between query terms and users, but leads to $O(n)$ round complexity. Our scheme does not require users to participate in the sequential shuffles.

1.2 Our Setting

We are working in a setting consisting of three semi-honest entities as follows:

– *Users*. They are the individuals who submit query terms to the search engine, and wish to protect the search engine to build their profiles. We use u to denote a user.
– *The group manager*. The role of the group manager, denoted by G, is to group users so that they execute our protocol that was introduced as above. We assume that the group manager has fairly powerful computing resources and storage capacity compared to users.
– *The search engine*. The web search service provider, denoted by W, is the entity that provides a list of best-matching web pages, usually along with a short summary and/or sometimes parts of the document. A typical example is Google. Note that the search engine has no incentives to protect users' privacy.

We consider that an adversary is not allowed to break current computationally secure encryption schemes. We assume that there are at least two honest users. However, different from [5], we allow collusions between two entities of the protocol.

1.3 A High-level Overview of Our Solution

In what follows we provide a high level description of our construction and the techniques used therein. To obtain our construction we build on the notion of secret sharing. The basic idea of Shamir's (t, n)-secret sharing is that a user can use a polynomial $f(X)$ of degree $t-1$ to split a secret q into n shares, $(\mathsf{v}_1, \dots, \mathsf{v}_n)$. Then, any collection of shares $\geq t$ from the distributed n shares allows to recover the secret using the Lagrange interpolation formula.

The starting point of the design of our solution is Shamir's secret sharing scheme: To submit a query term q, one chooses a random polynomial $f(X)$ of degree $n-1$ whose constant term is q, and evaluates $\mathsf{v}_i = f(u_i)$ at each user's label u_i. Then he computes encryptions $\bar{v}_i = \mathsf{E}_{pk}(\mathsf{v}_i)$ for all $1 \leq i \leq n$. Here $\mathsf{E}_{pk}(\cdot)$ is a public-key encryption algorithm and v_i is assumed to be in the message space of the encryption algorithm. Next, he sends each encrypted share to a corresponding user.

After all users obtain a list of encrypted shares, they perform re-masking and permuting the list and send it to a group manager. At first glance, our scheme may seem to have no differences from existing CB-PWS schemes. However, we emphasize that in the above step of our scheme, shuffling ciphertexts does not demand any interaction between neighbors. Completing the first step in existing solutions, every user holds the *same list of ciphertexts*. Hence, every user should join in the sequential shuffles to achieve unlinkability. We consider this as a very legitimate reason to incur a high round complexity $O(n)$. In contrast, our solution makes all users to have *different lists of ciphertexts* so that users do not need to perform shuffles sequentially.

Decrypting all received lists of ciphertexts, the group manager uses Lagrange interpolation to recover users' query terms. It then submits the recovered queries to the search engine and broadcasts the search results to the group users.

Outline of the paper: This work is organized as follows. Section 2 introduces cryptographic building blocks: secret sharing and public-key encryption. Section 3 provides a detailed desciption of our construction. In Section 4, we continue to provide its performance and security analysis.

2 Background

In this section, we review the concepts and notation of cryptographic building blocks. We begin with a review of secret sharing techniques, and continue to recalling public-key cryptography and its security definitions.

Notation. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \dots, n\}$. If A is a probabilistic polynomial-time (PPT) machine, we use $a \leftarrow A$ to denote making A produce an output according to its internal randomness. In particular, if U is a set, then $r \overset{\$}{\leftarrow} U$ is used to denote sampling from the uniform distribution on U.

We denote by λ a security parameter. A function $g : \mathbb{N} \to \mathbb{R}$ is called *negligible* if for every positive polynomial $\mu(\cdot)$ there is an integer N such that $g(n) < 1/\mu(n)$ for all $n > N$.

2.1 Secret Sharing

A secret sharing scheme is a method of distributing a secret, usually a key, among a group of users, requiring a cooperative effort to determine the key, so the plaintext can subsequently be decrypted. The ultimate goal of the scheme is to divide the secret being hidden into n shares, but any subset of t shares can be used together to solve for the value of the secret. Additionally, any subset of $t-1$ shares will prevent the secret from being reconstructed. This is defined as a (t,n)-threshold scheme, meaning that the secret is dispersed into n overall pieces, with any t pieces being able to recreate the original secret.

The original secret sharing scheme was proposed by Adi Shamir in 1979 [22], while George Blakley also designed a similar scheme on his own around the same time [4]. In this work, we use Shamir's secret sharing scheme instead of Blakley's scheme because the former is relatively efficient compared to the latter. Shamir's scheme is based on polynomial interpolation, and takes t points on the Cartesian plane, and with those t points, a unique polynomial $f(X)$ is guaranteed to exist such that $f(X) = y$ for each of the points given. Regardless, this polynomial $f(X)$ is of degree $t-1$, and the coefficient for the 0^{th} degree is equal to a given secret q. Overall, the full equation for $f(X)$ is given as such, with $\mathsf{q} = a_0$:

$$f(X) = a_0 + a_1 X + \cdots + a_{t-1} X^{t-1}.$$

For a concrete instantiation of Shamir's scheme, however we need to to determine an appropriate field $\mathcal{M}$ for the subsequent modular arithmetic. Because of this, an appropriate prime value p is needed, so that the needed arithmetic can be performed in $\mathcal{M} = \mathbb{Z}_p$, for instance. Following that, n unique elements are chosen, either systematically, or randomly, within the field. These values are labelled u_1 through u_n, and are made public. Assign one of these values to each of the participants that are to be given a share. The next step is to then choose $t-1$ random values in the field. These values are secret to all participants, and are labelled r_1 through r_{t-1}. Once these values are all determined, give each participant a share v_i based on their corresponding u_i value as such:

$$\mathsf{v}_i = \mathsf{q} + \sum_{j=1}^{t-1} r_j (u_i)^j \pmod{p}.$$

Reconstructing the secret can be done through the Lagrange interpolation that proves the solution to the equation is unique.

We use K to denote a list of public values chosen for a specific Shamir's secret sharing instantiation, i.e., $K = (t, n, \mathcal{M}_K; u_1, u_2, \ldots, u_n)$.

2.2 Public-key Encryption

A public-key encryption scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ consists of the following algorithms:

- KG is a randomized algorithm that takes a security parameter λ as input, and outputs a secret key sk and a public key pk; pk defines a plaintext space $\mathcal{M}_{pk}$ and a ciphertext space $\mathcal{C}_{pk}$.
- E is a randomized algorithm that takes pk and a plaintext $m \in \mathcal{M}_{pk}$ as input, and outputs a ciphertext $c \in \mathcal{C}_{pk}$. Especially, this process is usually randomized, using randomization value $r \in \mathcal{R}_{pk}$
$$c = \mathsf{E}_{pk}(m; r)$$
- D takes sk and $c \in \mathcal{C}_{pk}$ as input, and outputs the plaintext m.

Homomorphic public-key encryption schemes exhibit a particularly interesting algebraic property: when two ciphertexts are combined in a specific, the resulting ciphertext encodes the combination of the underlying plaintexts under a specific group operation, usually multiplication or addition. More formally, we say that a public-key cryptosystem $\mathcal{E} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ is *homomorphic* for binary relations $(\oplus, \otimes)$ if for all $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$,

- Given message domain $\mathcal{M}_{pk}$, $(\mathcal{M}_{pk}, \oplus)$ forms a group.
- Given ciphertext range $\mathcal{C}_{pk}$, $(\mathcal{C}_{pk}, \otimes)$ forms a group.
- For all $c_1, c_2 \in \mathcal{C}_{pk}$, $\mathsf{D}_{sk}(c_1 \otimes c_2) = \mathsf{D}_{sk}(c_1) \oplus \mathsf{D}_{sk}(c_2)$.

As a consequence, a cryptosystem's homomorphic property allows it to perform *reencryption*: given a ciphertext c, anyone can create a different ciphertext $\tilde{c}$ that encodes the same plaintext as c. Thus, given a homomorphic encryption scheme $\mathcal{E}$, we can define the reencryption algorithm as follows:

$$\mathsf{RE}_{pk}(c;r) = c \otimes \mathsf{E}_{pk}(m_0;r)$$

where m_0 is an identity message such that $\forall m \in \mathcal{M}_{pk}, m \oplus m_0 = m$. Further, $\mathsf{D}_{sk}(c) = m$, then $\mathsf{D}_{sk}(\mathsf{RE}_{pk}(c)) = m$, too.

Security Notions for Encryption Schemes. Given such a cryptosystem, one can consider different security definitions. Because the malleability of ciphertexts in homomorphic cryptosystems limits the security of such schemes, we only discuss security of homomorphic cryptosystems.

Semantic security was first defined in 1982 by Goldwasser and Micali [11]. Intuitively, a cryptosystem is said to be *semantically secure* if, given a ciphertext c, an adversary cannot determine any property of the underlying plaintext m. In other words, an adversary cannot extract any semantic information of plaintext m from an encryption of m. Formally, a public-key cryptosystem $\mathcal{E} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ is said to be semantically secure if there exists a negligible function $\mu(\cdot)$ such that, for all polynomial time algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\Pr\left[\begin{array}{l|l} (pk, sk) \xleftarrow{\$} \mathsf{KG}(1^\lambda); & \\ (m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_1(pk); & b = b' \\ c_b \leftarrow \mathsf{E}_{pk}(m_b); & \\ b' \leftarrow \mathcal{A}_2(m_0, m_1, c_b, \mathsf{state}) & \end{array}\right] < \frac{1}{2} + \mu(\lambda).$$

Instantiations. We know of a number of efficient schemes that are semantically secure. El Gamal [9] is the prime example of a semantically secure encryption scheme and Paillier [16] is another good example with semantic security.

2.3 Compatibility of Secret-sharing and Public-key Encryption

From now on, we will simply assume that we have some secret sharing scheme with key K and a cryptosystem with key pk. The keys may or may not overlap so that elements from one key are also included in another key. For instance, we could imagine the cryptosystem were an ElGamal scheme working on a group $\mathbb{G}$ of order q from primes $p, q|p-1$ and a generator g and the secret sharing scheme used the same message space as the encryption scheme, i.e., $\mathcal{M}_K = \mathcal{M}_{pk}$.

We require that the keys be selected so that the message space to be embedded into the group $\mathbb{G}$ allows field operations. In such a case, we are able to carry out polynomial arithmetic in the message space. Sometimes we then say that two schemes are compatible.

Readers refer to [15, 12] for further details of these cryptographic primitives.

3 Our Construction for PWS

We now proceed to describe our new technique for private web search. Our scheme enjoys computational efficiency similar to existing CB-PWS schemes but not require rounds in proportional to the number of users.

3.1 The Proposed Scheme

Our scheme is logically divided into the following phases:

- *Setup*. The main goal of the Setup phase is to create a group of users who would like to make searches to the search engine. Additionally, all system parameters for secret sharing and encryption will be published to all users in the group.

– *Mixing query*. Completing the Mixing Query phase, all users hold a re-encrypted and permuted version of distributed query terms.
– *Submitting query*. The group manager recovers a set of queries, but it cannot know who submitted which query. It then submits to the search engine. On receiving a set of query results from the search engine, it broadcasts the result.

Let n be a size of group. For convenience, we simply assume that the group manager knows the n and all users know where the group manager is and how to contact the group manager. Further, all messages are assumed to be automatically encoded into the working group of a public-key cryptosystem. Thus we do not use an extra notation for denoting that a plaintext was embedded into an element in the working group.

Setup. Let $\mathcal{E} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ be a semantically secure public-key cryptosystem with homomorphic property, and let K be a public key specifying a list of parameters for Shamir's secret sharing scheme. It also specifies how to efficiently perform polynomial evaluation and interpolation (see Section 2.1). The Setup phase is again divided into the following activities:

1. When the group manager G receives n requests for private query, it creates a group $\{u_1, u_2, \ldots, u_n\}$.
2. The group manager chooses a pair of keys (pk, sk) by invoking $(pk, sk) \overset{\$}{\leftarrow} \mathsf{KG}(1^\lambda)$ and publishes all system parameters (pk, K) for the protocol. As mentioned before, $\mathcal{M}_K = \mathcal{M}_{pk}$.

Mixing Query. After obtaining system parameters from G as a response for its query request, each user $u_{i\in[n]}$ does the followings:

1. On receiving the parameters, u_i chooses $n-1$ random coefficients $r_{i,j} \in \mathcal{M}_K$ and determines
$$R_i(X) = r_{i,n-1}X^{n-1} + \cdots + r_{i,1}X + \mathsf{q}_i$$
where $\mathsf{q}_i \in \mathcal{M}_K$ is u_i's query term.
2. Each user computes shares $\mathsf{v}_{i,j}$ of his query q_i for every $j \in [n]$ by
$$R_i(j) = \sum_{k=1}^{n-1} r_{i,k}j^k + \mathsf{q}_i.$$
We define $\mathsf{v}_{i,j} := R_i(j)$ for all $i, j \in [n]$.
3. For each $j \in [n]$, u_i computes $\bar{\mathsf{v}}_{i,j} = \mathsf{E}_{pk}(\mathsf{v}_{i,j})$, and sends $\bar{\mathsf{v}}_{i,j}$ to $u_{j\neq i}$.
4. After building a list of encrypted shares, $\bar{V}_i = (\bar{\mathsf{v}}_{1,i}, \ldots, \bar{\mathsf{v}}_{n,i})$, each user generates a new version of the list, $\tilde{V}_i = (\tilde{\mathsf{v}}_{1,i}, \ldots, \tilde{\mathsf{v}}_{n,i})$ where $\tilde{\mathsf{v}}_{\ell,i} = \mathsf{RE}_{pk}\left(\bar{\mathsf{v}}_{\pi_i(j),i}\right)$ for all $\ell, j \in [n]$ and for a random permutation π_i over $[n]$.
5. Each user sends $\tilde{V}_i$ to the group manager.

Submitting Query. The group manager performs the following steps:

1. The group manager constructs the following $n \times n$ matrix $\mathfrak{M}$ by decrypting all of ciphertexts in the n vectors:
$$\mathfrak{M} = \begin{bmatrix} \tilde{V}_1 \\ \tilde{V}_2 \\ \vdots \\ \tilde{V}_n \end{bmatrix} = \begin{bmatrix} \mathsf{v}_{\pi_1(1),1} & \mathsf{v}_{\pi_1(2),1} & \cdots & \mathsf{v}_{\pi_1(n),1} \\ \mathsf{v}_{\pi_2(1),2} & \mathsf{v}_{\pi_2(2),2} & \cdots & \mathsf{v}_{\pi_2(n),2} \\ \vdots & \vdots & \vdots & \vdots \\ \mathsf{v}_{\pi_n(1),n} & \mathsf{v}_{\pi_n(2),n} & \cdots & \mathsf{v}_{\pi_n(n),n} \end{bmatrix}$$

2. Since G has no order information for each row, it sequentially recovers each query term by applying the Lagrange interpolation formula to each element in $\mathfrak{M}$. During recovering, the group manager just ignores all meaningless query terms.
3. The group manager submits a set of query terms to the search engine W. The group manager broadcasts the output from W.

4 Analysis

This section analyzes the performance of our construction regarding the efficiency requirements. Next, we analyze the security of our protocol by examining all behaviors of the protocol.

4.1 Performance Analysis

Our protocol is compared to other CB-PWS solutions with respect to three efficiency measures– computation, communication and round. For this purpose, we first analyze the performance of our proposal. Then schemes proposed by [5] and [14] are compared to our proposal respectively. Since the scheme by [20] is identical to [5] except for replacing shuffle by permutation network, we decided to omit the scheme by [20]. Furthermore, we cannot give a fair comparison between the scheme proposed by [13] and our scheme because our scheme allows n times longer than the maximum length of query terms allowed in [13].

Parameter selection. Before giving the comparisons, we fist need to determine the system parameters: the group size (n) and the key size.

We believe that the time users must wait in order to form the group determines the group size n. Fast creating the group makes it possible to reduce the query delay. At the same time, the bigger size of the group, the more privacy the protocol achieves. Therefore, one way that the members can obtain strong privacy is that a user joins a different small-sized group every time the user submits a query. According to [5], $n = 3$ is the most realistic group size, in practice. The authors in [5] stated that with overwhelming probability a group of $n = 3$ users can be created in a hundredth of a second.

Regarding the key length, we take a 1024-bit key length like other solutions. Thus, a key of 1024-bit length can encrypt up to 128 bytes at a time and so a public-key cryptosystem that uses a 1024-bit key length can deal with queries approximately 64 characters.

Computation complexity. Now we analyze the computation cost for running the protocol. In general, since it is widely accepted that modular exponentiations dominates the total computation cost of a system, we also focus on the number of modular exponentiations that every user must perform during executing each protocol. For a fair comparison we assume that our construction also employes an ElGamal encryption scheme.

For this purpose, we denote by $\mathsf{ME}(\ell)$ a modular exponentiation modulo an ℓ-bit integer value. The scheme in [14] extensively uses a double encryption by combining ElGamal encryption and Cramer & Shoup's cryptosystem [6]. Thus, some modular exponentiations in [14] should be carried out modulus a 2048-bit integer rather than a 1024-bit integer like [5] and our scheme. Our experimental implementation without any optimization shows that a 1024-bit modular exponentiation is approximately 10 times faster than a 2048-bit modular exponentiation. More specifically, a 1024-bit modular exponentiation takes 18 msec on average, while a 2048-bit modular exponentiation takes 191 msec on average.

We give the comparison of schemes in Figure 1, regarding the computation complexity. We remark that evaluation of a polynomial in a finite field of degree less than n at n points can be performed using at most $O(\mathsf{M}(n) \log n)$ field operations where $\mathsf{M}(n)$ is the number of bit operations to multiply two n-bit integers. Similarly, Lagrange interpolation can be computed with the asymptotically same computation.

We see that our scheme obtains the lowest computation cost. Of course, our scheme and [5] do not provide any mechanism to protect the honest users against the malicious adversary.

However, even if we use zero-knowledge proofs to achieve active security, our scheme still has $O(n)$ computation complexity such as [14].

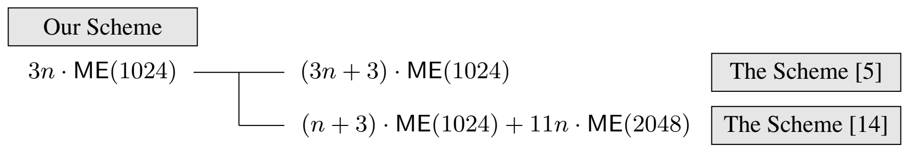
Our Scheme
$3n \cdot \mathsf{ME}(1024)$
$(3n + 3) \cdot \mathsf{ME}(1024)$
The Scheme [5]
$(n + 3) \cdot \mathsf{ME}(1024) + 11n \cdot \mathsf{ME}(2048)$
The Scheme [14]

Fig. 1. Computation Complexity

Communication & Round complexity. First of all, it is clear that our protocol incurs just a constant number of rounds. Next, we compare the communication complexity by counting the number of messages that every user should send in each step of the protocols. Table 1 summarizes the comparison results.

Our proposal consists of five rounds: (1) obtaining the system parameters, (2) applying Shamir's secret sharing and encryption to the query term and sending the resulting list, (3) mixing the resulting set and sending it to the group manager, (4) recovering a set of query terms and submitting to the search engine, and finally (5) broadcasting the search results to the users. We see that all schemes have $O(n)$ communication complexity. However, our scheme only requires 5 rounds, while other proposals have $O(n)$ round complexity.

Table 1. Communication & Round Complexity

	Communication Complexity	Round Complexity
The Scheme [5]	$3n - 2$	$n + 6$
The Scheme [14]	$4n - 2$	$n + 6$
Our Scheme	$4n$	5

Remark 1 *There is an interesting difference between our scheme and other two schemes. Our scheme demands the group manager to perform heavy computations, while the group manager in other schemes only plays a role of maintaining a group of users. However, in practice, since n is small (e.g., $n = 3$ or 4), we think that these computations may affect the performance but not considerable.*

4.2 Security Analysis

Our construction achieves the following privacy requirements of the users when they submit query terms to a search engine:

- *Unlinkability among users*. The users must not be able to link a certain query to a specific user who has entered the query.
- *Unlinkability between the group manager and the users*. The group manager must not be able to link a certain query with the user who have entered it.
- *Unlinkability between the search engine and the users*. The web search engine must not be able to build a plausible profile of a specific user.

Unlinkability among users. Let $J \subset [n]$ be a set of semi-honest users such that $|J| = \eta < n$. For notional convenience, suppose that $\hat{u} \in J$ is of index $\alpha \in [n]$, but it follows all steps of the protocol. At the end of the Mixing Query step, $\hat{u}$ receives a list of encrypted shares from other users, $(\bar{\mathsf{v}}_{1,\alpha}, \bar{\mathsf{v}}_{2,\alpha}, \ldots, \bar{\mathsf{v}}_{n,\alpha})$. If $\hat{u}$ would be able to decrypt these ciphertexts, it would be able to fill in some parts of the matrix $\mathfrak{M}$. Nevertheless, as long as $\mathfrak{M}$ is not completed, it cannot know the queries of honest users. For example, let u_1, u_2 be only the honest users. Denoting by $\boxtimes$ a decrypted share, $\hat{u}$ would be able to obtain at most the following matrix:

$$\begin{bmatrix} ? \ \boxtimes \ \boxtimes \cdots \boxtimes \\ \boxtimes \ ? \ \boxtimes \cdots \boxtimes \\ \boxtimes \ \boxtimes \ \boxtimes \cdots \boxtimes \\ \vdots \\ \boxtimes \ \boxtimes \ \boxtimes \cdots \boxtimes \end{bmatrix}$$

Hence, $\hat{u}$ is unable to obtain q_1 and q_2, and thus our solution preserves users' privacy.

Unlinkability between the group manager and the users. Our protocol allows that two entities (i.e., between $\hat{u}$ and G) collude. At the mid of the Submitting Query phase, a compromised group manager and semi-honest users have the valid matrix, but they still do not know the secret permutations of honest users. Therefore, even though the group manager would be compromised, the attacker has that a random guess regarding a "link" is correct with probability that is only negligibly greater than $\frac{1}{n-\eta}$.

Unlinkability between the search engine and the users. Our protocol allows the search engine to participate in the execution of the protocol, only at the end of the last phase. For similar reasons above, it cannot link a certain query to an honest user, and so it cannot build profiles for honest users.

5 Concluding Remarks and Further Research

Web searches have been shown to be often sensitive. Any information leaked from search histories could endanger user privacy. Search histories may contain health-related data and possibly other personal information, including, but not restricted to: political or religious views, sexual orientation, etc. For example, Google provides signed-in users with personalized search results based on the history of their searches and navigation. Furthermore, users typing search queries in the Web interface are prompted with suggestions resulting from their history. To this end, Google tracks all Web searches performed by a signed-in user, as well as the target web pages clicked from the search result page.

In this work, we presented a constant-round CB-PWS protocol for protecting users' privacy. Our solution can be easily deployed to the current systems because it does not require any changes in the service provider side. However, still there are some remaining work for further research as follows:

- We will try to provide a more rigorous security proof using the standard techniques like simulation or game-playing proof. For this purpose, we first need to define the notion of privacy in this setting.
- We should improve the performance of the group manager side, especially in the case that the size of group is huge.

References

1. Anonymizer. `http://www.anonymizer.com`, 2014.
2. Scroogle. `http://scroogle.org`, 2014.
3. TracMeNot. `http://mrl.nyu.edu/dhowe/trackmenot`, 2014.
4. G. Blakley. Safeguarding cryptographic keys. *American Federation of Information Processing Societies Proceedings*, 48:313–317, 1979.
5. J. Castellà-Roca, A. Viejo, and J. Herrera-Joancomartí. Preserving user's privacy in web search engines. *Computer Communications*, 32(13-14):1541–1551, 2009.

6. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Advances in Cryptology-Crypto*, LNCS 1462, pages 13–25, 1998.
7. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In M. Blaze, editor, *USENIX Security Symposium*, pages 303–320, 2004.
8. J. Domingo-Ferrer, A. Solanas, and J. Castellà-Roca. $h(k)$-private information retrieval from privacy-uncooperative queryable databases. *Online Information Review*, 33(4):720–744, 2009.
9. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology-Crypto*, LNCS 196, pages 10–18, 1984.
10. Y. Elovici, B. Shapira, and A. Meshiach. Cluster-analysis attack against a private web solution (PRAW). *Online Information Review*, 30(6):624–643, 2006.
11. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 1984.
12. J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC Press, 2007.
13. M. Kim and J. Kim. Privacy-preserving web search. In *ICUFN*, pages 480–481, 2012.
14. Y. Lindell and E. Waisbard. Private web search with malicious adversaries. In M. Atallah and N. Hopper, editors, *Privacy Enhancing Technologies*, LNCS 6205, pages 220–235, 2010.
15. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press, 1996.
16. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1592, pages 223–238, 1999.
17. S. T. Peddinti and N. Saxena. On the privacy of web search based on query obfuscation: A case study of TrackMeNot. In M. Atallah and N. Hopper, editors, *Privacy Enhancing Technologies*, LNCS 6205, pages 19–37, 2010.
18. D. Rebollo-Monedero and J. Forné. Optimized query forgery for private information retrieval. *IEEE Transactions on Information Theory*, 56(9):4631–4642, 2010.
19. M. Reed, P. Syverson, and D. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.
20. C. Romero-Tris, J. Castellà-Roca, and A. Viejo. Multi-party private web search with untrusted partners. In M. Rajarajan, F. Piper, H. Wang, and G. Kesidis, editors, *SecureComm*, pages 261–280, 2011.
21. F. Saint-Jean, A. Johnson, D. Boneh, and J. Feigenbaum. Private web search. In P. Ning and T. Yu, editors, *WPES*, pages 84–90, 2007.
22. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.