Private Web Search with Malicious Adversaries*

Yehuda Lindell† Erez Waisbard†

March 24, 2011

Abstract

Web search has become an integral part of our lives and we use it daily for business and pleasure. Unfortunately, however, we unwittingly reveal a huge amount of private information about ourselves when we search the web. A look at a user's search terms over a period of a few months paints a frighteningly clear and detailed picture about the user's life. In this paper, we build on previous work by Castellà-Roca et al. (Computer Communications 2009) and show how to achieve privacy in web searches efficiently and practically without resorting to full-blown anonymous routing. In contrast to previous work, our protocol is secure in the presence of malicious adversaries.

1 Introduction

It is well known that users' search terms to web search engines contain significant amounts of sensitive information and, as such, the aggregation and use of these terms constitutes a severe privacy breach. The only way that a user can protect him or herself from this breach today is to use an anonymous routing system like Tor [9]. However, this can sometimes be an "overkill" measure. This is especially the case since in order to achieve a high level of security, such systems cause a considerable slowdown.

Recently, an interesting model for solving this problem was suggested by [2]. Essentially, their proposal is for a group of users to first *shuffle* their search words amongst themselves. After the shuffle, each user has someone's search word (but doesn't know whose), and the parties then query the search engine with the word obtained. Finally, the parties all broadcast the result to all others. This model is especially attractive because it doesn't involve the overhead of installing a full-blown anonymous routing system, and can be provided as a simple web service.

In [2], the authors present a protocol for private web search in the above model that is secure in the presence of semi-honest adversaries. That is, users' privacy is maintained only if all parties follow the protocol specification exactly. We argue that this level of security is not sufficient, especially due to the fact that the protocol of [2] has the property that a *single* adversarial participant can easily learn the queries of all users, without any malicious behavior being detected. This means that an adversarial entity who is a participant in many searches can learn all of the users' queries without any threat of retribution.

Our results. In this paper we construct a protocol for private web search in the model of [2] that is secure in the presence of *malicious adversaries* that may arbitrarily deviate from the protocol specification in order to attack the system. Our main technical tool is a highly efficient cryptographic protocol for parties to mix their inputs [3] that guarantees privacy in the presence of malicious adversaries. Unlike the usual setting of mix-nets, here the parties themselves carry out the mix. The novelty of our approach is based on the observation that, unlike the setting of voting where mix-nets are usually applied, the guarantee of *correctness* is not necessary for private web search. That is, we allow a malicious participant to carry out a "denial of service" type attack, causing the search to fail. In return, we are able to omit the expensive zero-knowledge proofs of correctness in every stage of the mix.

*An extended abstract of this work appeared in *PETS* 2010 [13]. The main protocol there contained a serious error which is fixed in this draft.

†Dept. of Computer Science, Bar-Ilan University, Israel. Email: {lindell,waisbard}@cs.biu.ac.il. This research was generously supported by the European Research Council as part of the ERC projects "LAST" and "SFEROT".

We stress that simply removing the correctness proofs from a standard mix protocol yields a completely insecure protocol that provides no privacy. For example, we still have to deal with "replacement attacks" where the first party carrying out the mix replaces all of the encrypted search words with terms of its own, except for the *one* ciphertext belonging to the user under attack. In this case, the result of the mix completely reveals the search word of the targeted user (because all other search words belong to the attacker). Our solution to this problem (and others that arise; see Section 3) is based on the following novel idea: instead of inputting search words into the mix, each party inputs an encrypted version of its search word. Then, after all stages of the mix are concluded, each party checks that its encrypted value appears. If yes, it sends true to all parties, and if not it sends false. If all parties send true, they can then proceed to decrypt the search words because this ensures that no honest party's search word was replaced. However, this raises a new challenge regarding how to decrypt the encrypted search word. Namely, a naive solution to the problem fails. For example, if each party encrypted their search word using a one-time symmetric key, then sending this key for decryption reveals the identity of the party whose search word it is. We therefore use a "one-time" threshold encryption scheme based on ElGamal [10] and have the parties encrypt the search words with the combined key. The parties then send their key-part in the case that all parties sent true (a similar idea to this appears in [2] but for a different purpose). We call this a *private shuffle* in order to distinguish it from a standard mix-net. We provide a formal definition of security for a private shuffle and have a rigorous proof of security under this definition.

As we have mentioned, the private shuffle is the main technical tool used for obtaining private web search. However, as is often the case, the cryptographic protocol at its core does not suffice for obtaining a secure *overall solution*. In Section 5 we therefore discuss how a private shuffle primitive can be used to obtain private web search, and in particular how to bypass non-cryptographic attacks that can be fatal. One major issue that arises is how to choose the group of participants, and in particular, how to prevent the case that the adversary controls all but one participant (in which case the adversary will clearly learn the input of the sole honest party). This issue was not addressed in previous solutions.

Related work. A number of different anonymity-preserving techniques can be used in principal for private web search. For example, private information retrieval [4, 14] provides the appropriate guarantees. However, it is far too inefficient. A more natural candidate is a to use a mix-net [3]. However, as we have mentioned, considerable expense goes into proving correctness in these protocols. In addition, doing this efficiently and securely turns out to be quite a challenge; see for example [12, 8]. For further comparisons of existing techniques to the model that we adopt here, we refer the reader to [2] and the references within. We remark that our protocol is about twice as expensive as the protocol of [2], and thus the efficiency comparisons between their solution and other existing techniques can be extrapolated to our solution.

2 Definitions

In this section we present our definition of security for a private shuffle primitive. The shuffle functionality is simply the n-ary probabilistic function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, such that for every i, $y_i = x_{\pi(i)}$ where π is a random permutation over $[n]$. Intuitively, a shuffle is *private* if an adversary cannot link between the inputs of the protocol and the outputs of the protocol. Namely, the adversary should not be able to link y_j to an honest party P_i where $j = \pi(i)$. Denoting the number of corrupted parties by t, we have that a random guess regarding a "link" is correct with probability $\frac{1}{n-t}$. Thus, we formalize security by requiring that an adversary controlling t parties can output (i, j) where P_i is honest and $j = \pi(i)$ with probability that is at most negligibly greater than $\frac{1}{n-t}$.

The security experiment. We assume that the parties communicate over an open network with unauthenticated channels. We model this network by having all communication go through an adversary that can listen to all the communication, delete messages and inject messages of its choice. This is formally modeled by providing the adversary with stateful oracles that model the honest parties, as in [1]. The experiment modeling the success of the adversary appears in Figure 1.

FIGURE 1 (The Security Experiment $\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k)$)

1. Invoke the adversary $\mathcal{A}$ with input 1^k and with parameters t and n (k is the security parameter, t the number of corrupted parties, and n the overall number of parties).
2. Receive from $\mathcal{A}$ a set of t indices $I \subset [n]$ designating the corrupted parties (note that $|I| = t$), and a vector of $n - t$ *distinct* inputs $w_1, \ldots, w_{n-t}$ for the honest parties.
3. Choose a random permutation π over $\{1, \ldots, n - t\}$ and initialize the ith honest-party oracle with input $w_{\pi(i)}$.
4. Execute the shuffle protocol, where $\mathcal{A}$ interacts with the $n - t$ oracles (each oracle runs the specified shuffle protocol as an honest party responding to the messages it receives from $\mathcal{A}$).
5. When it concludes, the adversary outputs a pair (i, j) for any i, j of its choice.

We say that the adversary succeeds in the experiment, in which case the output of the experiment $\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k)$ equals 1, if and only if $\pi(i) = j$.

Defining security. We are now ready to define security. First, we require non-triviality, meaning that if all parties are honest, then the protocol output is a permuted vector of the inputs. Next, we require that an adversary controlling t out of the n parties can succeed in the experiment ExptShuffle with probability that is only negligibly greater than $\frac{1}{n-t}$ (where negl is a negligible function if for every polynomial p and all large enough k's it holds that $\mathsf{negl}(k) < 1/p(k)$):

Definition 2 *A protocol π is a* private shuffle *if it is non-trivial, and if for every probabilistic polynomial-time algorithm $\mathcal{A}$, every integer $n \in \mathbb{N}$ and every $0 < t < n$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\Pr\left[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1\right] \leq \frac{1}{n-t} + \mathsf{negl}(k).$$

3 Constructing a Private Shuffle

In order to motivate our construction, we begin by describing the protocol of [2] that is secure in the presence of semi-honest adversaries. We then describe the difficulties that arise when moving to the malicious model. The main tool that is used in [2] is called *ElGamal remasking*, which takes a ciphertext and rerandomizes it into a new ciphertext c' without requiring knowledge of the secret key.[1] See Protocol 3 for an outline of the construction of [2].

Attacks on private shuffle protocols. Although Protocol 3 was defined for the semi-honest model, it is instructive to see what attacks can be carried out by a malicious party:

Targeted public-key attack: A malicious P_n may compute its share of the public key after given all of the g^{x_i} values of the other parties. Specifically, P_n sets its share of the public-key to be $h = g^{x_n}/(\prod_{i=1}^{n-1} g^{x_i})$ for a random x_n. Observe that any encryption under $y = \prod_{i=1}^{n} g^{x_i}$ is actually an encryption under g^{x_n} alone because $h \cdot y = g^{x_n}$. Thus, P_n can decrypt the values of all parties and learn who sent what. In order to escape detection, in its remasking step, party P_n can re-encrypt all of the search terms (which it already learned) under a new $y' = g^{x'_n} \cdot \prod_{i=1}^{n} g^{x_i}$ for a value x'_n that it knows (if it doesn't do this, then it cannot complete the last step because it doesn't know its share of y). This enables it to complete the last stage correctly and thus the attack would go completely unnoticed.

Stage-skipping attack: A malicious party P_n may remask and permute the initial vector of ciphertexts sent by the parties instead of the vector that it received from P_{n-1}. In this case, when the vector is decrypted P_n will know exactly which party sent which message. Observe that this behavior would not be detected because the remask operation looks identical when applied once or n times.

[1] A rerandomization of a ciphertext is a procedure that takes an encryption of some message m and generates a new ciphertext that is also an encryption of m, where the randomness used to generate the two ciphertext is independent.

Protocol 3 (The protocol of [2] for semi-honest adversaries (overview))

1. Parties $P_1, \ldots, P_n$ generate a joint ElGamal public key $y = \prod_{i=1}^{n} g^{x_i}$, where x_i denotes the private key of each party.
2. Every party P_j encrypts its search word w_j using the joint public key, obtaining $c_j^0 = (u_j^0, v_j^0)$, and sends it to everyone.
3. For every $i = 1, \ldots, n$, party P_i does the following:
   (a) P_i *remasks* the ciphertexts $(c_1^{i-1}, \ldots, c_n^{i-1})$ it received from P_{i-1}.
   (b) P_i randomly *permutes* the remasked ciphertexts.
   (c) P_i sends the shuffled and remasked ciphertexts to P_{i+1}, except for party P_n who broadcasts the result to all the parties.
4. Given the shuffled and remasked ciphertexts $(c_1^n, \ldots, c_n^n)$, every party P_i decrypts a single ciphertext $c_i^n = (u_i^n, v_i^n)$. This is carried out as follows:
   (a) Every party P_j sends each P_i the share $(u_i^n)^{x_j}$ for every $i, j \in \{1, \ldots, n\}$, where x_j is P_j's private key.
   (b) Given the shares from all parties, every P_i computes $w_i = \frac{v_i^n}{\prod_{j=1}^{n} (u_i^n)^{x_j}}$.

Input-replacement attack: A malicious party P_1 can learn the input w_j of an honest party P_j by replacing all the ciphertexts in the input vector with individually remasked copies of the *initial* ciphertext (u_j^0, v_j^0). In this case, all of the parties receive w_j; in particular P_1 receives w_j and so knows the search term of P_j.

Input-tagging attack: A malicious party P_1 can change its own message to effectively "tag" a message of an honest party P_j. Specifically, P_1 can set c_1^1 to equal a remasked version of $(u_j^0, v_j^0 \cdot \delta)$ for some random δ and this would result in the keyword of P_1 being $w_1 = w_j \cdot \delta$. This enables P_1 to learn P_j's exact keyword by searching for a pair of terms w, w' in the final result for which $w' = \delta \cdot w$. We note that this attack can also be carried out on the protocol that appears in the extended abstract of this work [13].

Private shuffle for malicious adversaries. Our protocol for private shuffle that achieves security in the presence of malicious adversaries is based on the following ideas. First, we don't use a "remask" method as in [2] since this is inherently vulnerable to an input-tagging attack; rather we use an onion-layered encryption method with CCA2-secure encryption. This ensures non-malleability and so no party can make its ciphertext be related to another party's ciphertext. Next, in order to guarantee privacy, we need to ensure that at least one honest user "remasks and permutes" all of the ciphertext values. This involves ensuring that all parties take part in the shuffle and that the parties shuffle the actual input values (that is, we need to ensure that neither a stage-skipping nor input-replacement attack is carried out). The classic way of achieving this in the mix-net literature [3, 12, 8] is to have each party P_i prove (at each stage) that the values that it passed onto P_{i+1} are indeed a remasked and permuted version of what P_i received from P_{i-1}. However, this is a costly step that we want to avoid. We therefore provide an alternative solution that is based on a two-stage protocol with two phases of encryption of each input. First, each party encrypts its search term with threshold El Gamal, where the shares of all parties are needed for decryption. Next, the El Gamal ciphertext is encrypted in an onion form using CCA2-secure encryption. That is, the El Gamal ciphertext is re-encrypted n times under the encryption key of each party yielding $E_{pk_1}(E_{pk_2}(\cdots E_{pk_n}(c)\cdots))$, where c is the El-Gamal encrypted search term. Now, given all of these ciphertexts, the parties in turn randomly permute the ciphertexts and remove the encryption layer under their CCA2-secure key. Then, at the end of this stage there is a *verification step* in which all parties check that their input value is still in the shuffled array (under the inner encryption). Note that since the result is the series of El-Gamal encrypted search terms, these have high entropy and so cannot be guessed. If all parties acknowledge that their value is present then we are guaranteed that all parties participated in the shuffle and that no inputs were replaced.

We can therefore safely proceed to the second stage of the protocol where the inner encryption is privately removed, revealing the shuffled inputs. In addition to the above, we prevent the aforementioned targeted public-key attack by having each party prove that it knows its associated secret key.

We note that in order to prevent a powerful man-in-the-middle adversary from playing the role of all parties except for one, we assume the existence of a PKI for digital signatures; see Section 5 for a discussion of how to achieve this in practice. In addition, we assume that all parties hold a unique *session identifier sid* (e.g., this could be a timestamp), and a group $\mathbb{G}$ of order q with generator g, to be used for the El Gamal encryption. Let $\mathcal{E} = (G, E, D)$ denote a CCA2-secure public-key encryption scheme.

Protocol 4 (Private Shuffle with Malicious Adversaries)

Input: Each P_j has a search word w_j, and auxiliary input $(\mathbb{G}, g, q)$ as described.

Initialization Stage:

1. Each party P_j chooses a random $\alpha_j \in Z_q^*$ and computes $h_j = g^{\alpha_j}$, and chooses a pair of keys $(sk_j, pk_j) \leftarrow G(1^k)$ for the CCA2-secure encryption. P_j sends (h_j, pk_j) to all the other parties and proves knowledge of α_j using a *non-malleable non-interactive zero-knowledge proof of knowledge* as in [7]. P_j signs the message it sends together with the identifier *sid* using its certified private signing key (from the PKI).
2. Each party verifies the signatures on the messages and the proofs that it received and aborts unless all are correct.
3. Each party P_j encrypts its input w_j using El Gamal with the public key $h = \prod_{i=1}^{n} h_i = g^{\sum_{i=1}^{n} \alpha_i}$. That is, it chooses a random $\rho_j \in_R Z_q^*$ and computes an encryption $m_j = (g^{\rho_j}, h^{\rho_j} \cdot w_j)$.
4. Each party P_j computes $c_j = E_{pk_1}(E_{pk_2}(\cdots(E_{pk_n}(m_j))\cdots))$ and sends c_j to P_1.

The output of this phase is the list of the encrypted c_j's of all the parties, denoted $\mu_0 = \langle c_1^0, \ldots, c_n^0 \rangle$.

Shuffle stage: For $j = 1, \ldots, n$, P_j receives vector μ_{j-1} and computes a shuffled version μ_j as follows:

1. P_j checks that there are no duplications in μ_{j-1}. If there are, it aborts.
2. P_j decrypts every c_i^{j-1} in μ_{j-1} by computing $c_i^j = D_{sk_j}(c_i^{j-1})$
3. P_j randomly permutes the list of values c_i^j computed above; denote the result by μ_j.
4. P_j sends μ_j to P_{j+1}; the last party P_n sends μ_n to all parties.

Verification stage:

1. Every party P_j checks that its El-Gamal ciphertext m_j appears in the vector μ_n. If yes it sends $(sid, P_j, \mathsf{true})$, signed with its private signing key, to all the other users. Otherwise it sends (P_j, false).
2. If P_j sent false in the previous step, or did not receive a validly signed message $(sid, P_i, \mathsf{true})$ from all other parties P_i, then it aborts. Otherwise, it proceeds to the next step.

Reveal stage:

1. For every (u_i, v_i) in μ_n, party P_j sends its share of the decryption $s_i^j = u_i^{\alpha_j}$ to P_i.
2. After receiving all the shares s_j^i, every party P_j computes $w'_j = \frac{v_j}{\prod_{i=1}^{n} s_j^i} = \frac{v_j}{u_j^{\sum_{i=1}^{n} \alpha_i}}$, thereby decrypting the El Gamal ciphertext and recovering the search word w'_j (here j denotes the current index in μ_n and not the index of the party who had input w_j at the beginning of the protocol).

Remarks on the protocol:

1. For the sake of efficiency, the non-malleable non-interactive zero-knowledge proof of knowledge in the initialization stage can be implemented by applying the Fiat-Shamir heuristic [11] to Schnorr's protocol for discrete log [15]. In order to achieve independence (i.e., the ability to extract from the adversary while simulating proofs from the honest parties), we also include the *sid* and the party ID of the prover inside the hash for generating the "verifier query"; see Appendix A for more details. It is also possible

to use the methodology of [5] and an interactive zero-knowledge proof of knowledge, at the expense of $\log n$ rounds of communication, but then without relying on random oracles for efficiency.

2. Any public-key CCA2-secure encryption scheme can be used in our protocol; the Cramer-Shoup scheme [6] is a good choice in that it provides CCA2-security without random oracles.

3. Observe that each input w_j is encrypted $n+1$ times; first with ElGamal and then n times using a CCA2-secure encryption scheme. This can cause a problem since each encryption increases the size of the ciphertext, whereas the input to a given scheme is usually limited (e.g., with Cramer-Shoup, the plaintext is a single group element, whereas the ciphertext is a number of group elements). This is solved using standard techniques of hybrid encryption. Specifically, just define $\mathbb{E}_{pk}(m) = (E_{pk}(K), E'_K(m))$ where E is a CCA2-secure public-key encryption scheme and E' is a CCA2-secure symmetric encryption scheme. Observe that two layers of encryption are computed as

$$\mathbb{E}_{pk_1}(\mathbb{E}_{pk_2}(m)) = E_{pk_1}(K_1), E'_{K_1}(E_{pk_2}(K_2), E'_{K_2}(m)).$$

   We therefore have a CCA2-secure encryption scheme that can take inputs of any length.

4. In the first stage of the protocol every party participates in the shuffle. However, as we will see in the proof it suffices to ensure that *one* honest party participated. Thus, if we assume that at most t parties are malicious (for $t < n$), then we can run the shuffle stage for $j = 1$ to $t+1$ only, reducing the number of rounds from n to $t+1$.

Non-triviality. The non-triviality requirement of a private shuffle is that if all parties are honest then the output is a permutation of the input values $(w_1, \ldots, w_n)$. It is clear that after the shuffle stage, the result is a permuted vector containing the El-Gamal ciphertexts $m_1, \ldots, m_n$. The fact that the decryption in the last stage works follows from the fact that

$$\frac{v_j}{\prod_{i=1}^n s_j^i} = \frac{v_j}{u_j^{\sum_{i=1}^n \alpha_i}} = \frac{h^{\rho_j} \cdot w_j}{g^{\rho_j \cdot \sum_{i=1}^n \alpha_i}} = \frac{g^{\rho_j \cdot \sum_{i=1}^n \alpha_i} \cdot w_j}{g^{\rho_j \cdot \sum_{i=1}^n \alpha_i}} = w_j,$$

as required.

4 Security of the Shuffle Protocol

As we have discussed above, the security of the protocol is based on the fact that the ciphertexts are shuffled by everyone while removing the layers of the outer encryption. The fact that a CCA2-secure encryption scheme is used prevents an attacker not only from changing any of the messages of the honest parties during the *shuffle stage* in a way that would not be detected in the *verification stage*, but also from creating ciphertexts which are related to the plaintexts of the honest parties. Assuming that everyone sent true in the verification stage we know that indeed *all of the original* ciphertexts were shuffled and that *all of the parties* took part in the shuffle. We can therefore open the El-Gamal encryption decryption layer, and be confident that every party learns exactly one plaintext without knowing to which party it belongs. Of course, this decryption stage must not reveal anything about who encrypted which El-Gamal ciphertext. This is achieved by using the threshold decryption method described in the protocol. In this section we prove the following theorem, stating that an adversary can succeed in guessing an honest party's search word with probability at most negligibly greater than $2/(n-t)$. We actually conjecture that the true success probability is $1/(n-t)$, but currently have only proven the bound of $2/(n-t)$.

Theorem 5 *Assume that the decisional Diffie-Hellman (DDH) assumption holds in $(\mathbb{G}, g, q)$ and that $\mathcal{E} = (G, E, D)$ is a CCA2-secure encryption scheme. Then, for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every integer $n \in \mathbb{N}$ and every $0 < t < n$,*

$$\Pr\left[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1\right] \leq \frac{2}{n-t} + \mathsf{negl}(k).$$

Proof: The structure of our proof is as follows. We first prove that if the adversary tampers with any of the ciphertexts of the honest parties in any of the rounds of the shuffle, then it will be detected in the verification stage with overwhelming probability. This follows by a reduction to the security of the CCA2-secure encryption scheme (and the fact that the El-Gamal encryption have high entropy). Next, we prove that if the adversary is detected in the verification stage, then the adversary can succeed in the shuffle experiment with probability at most $\frac{1}{n-t} + \mathsf{negl}(k)$. Intuitively, this is due to the fact that the honest parties do not open the El-Gamal encryptions and so the original search terms are kept secret. Indeed, this is formally proven via a reduction to the security of El Gamal. Finally, we prove that if the adversary does not tamper with any of the ciphertexts during the rounds of the shuffle (and so the verification stage passes), then it cannot succeed in the shuffle experiment with probability greater than $\frac{1}{n-t} + \mathsf{negl}(k)$. Intuitively, this is due to the fact that the CCA2-secure encryption and the random permutations ensure that a shuffle carried out by an honest party breaks all connection between the original search terms and the parties who encrypted them. Once again, this is formally proven via a reduction to the CCA2-secure encryption scheme. Combining the above together, we have that the probability that the adversary succeeds is at most $2/(n-t) + \mathsf{negl}(k)$, as required.

Notation: Throughout the proof we use the subscript i to denote a message that *was originated* by party i regardless of its location in the ciphertext vector at any given round. In particular, m_i is the result of applying the inner ElGamal encryption to w_i and c_i is the result of applying all the layers of the outer encryption to m_i.

We now define two random variables $\mathsf{missing}$ and $\mathsf{AllVerify}$. Intuitively, $\mathsf{missing} = 1$ if there exists a round ℓ where at least one of the ciphertexts of the honest parties (appropriately decrypted to that round) does not appear in the vector μ_ℓ. The random variable $\mathsf{AllVerify} = 1$ if all the honest parties send true in the verification stage of the protocol. Formally:

Definition 6 *Let $\mathcal{C}_\ell$ be the set of ciphertexts in the vector $\mu_{\ell-1}$, sent from $P_{\ell-1}$ to P_ℓ in the protocol execution, where P_ℓ is honest. We define the random variable $\mathsf{missing}_\ell$ where $\mathsf{missing}_\ell = 1$ if and only if $\ell \in [n] \setminus I$ and*

$$\left\{D_{pk_{\ell-1}}(D_{pk_{\ell-2}}(\cdots(D_{pk_1}(c_i))\cdots))\right\}_{i\in[n]\setminus I} \not\subseteq \mathcal{C}_\ell$$

where $\{c_i\}_{i\in[n]\setminus I}$ is the set of ciphertexts sent by all honest parties to P_1 in the protocol execution. We define $\mathsf{missing}$ so that $\mathsf{missing} = 1$ if and only if there exists an ℓ such that $\mathsf{missing}_\ell = 1$. Finally, we define the random variable $\mathsf{AllVerify}$ such that $\mathsf{AllVerify} = 1$ if and only if all honest parties send true in the verification stage of the protocol.

We note that if an honest party P_ℓ does not receive vector $\mu_{\ell-1}$ at all in the execution (e.g., if it is bypassed by the adversary), then $\mathsf{missing}_\ell$ certainly holds. We are now ready to state our first claim. Intuitively, it states that the probability that an honest party's ciphertext was missing and yet all honest parties send true is negligible.

Claim 7 *If $\mathcal{E} = (G, E, D)$ is a CCA2-secure encryption scheme, then there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr[\mathsf{missing} = 1 \ \wedge \ \mathsf{AllVerify} = 1] \leq \mathsf{negl}(k).$$

Proof: We prove that if $\mathsf{missing} = 1$ and $\mathsf{AllVerify} = 1$, then the adversary must have guessed the value of the El-Gamal encryption m_i, where c_i is the "missing ciphertext"; i.e., $D_{pk_{\ell-1}}(D_{pk_{\ell-2}}(\cdots(D_{pk_1}(c_i))\cdots)) \notin \mathcal{C}_\ell$. Intuitively, this is the case since c_j is not decrypted by one of the honest parties. Thus, the security of $\mathcal{E}$ under that honest party's key implies that m_i remains secret. Now, all of the encryptions m_i are of high entropy (simply by the fact that $\mathbb{G}$ is a large group). Thus, the probability that an adversary succeeds in guessing m_i so that party P_i will send true in the verification stage is negligible. Let $\mathsf{missing}^i_\ell = 1$ if

$$D_{pk_{\ell-1}}(D_{pk_{\ell-2}}(\cdots(D_{pk_1}(c_i))\cdots)) \notin \mathcal{C}_\ell.$$

Let ℓ be the smallest round for which $\mathsf{missing}_\ell = 1$ and let i be the smallest index for which $\mathsf{missing}_\ell^i = 1$; we say that such an (ℓ, i) are minimal for $\mathsf{missing}_\ell^i = 1$ (note that the minimality of ℓ comes first, although all we need is to specific a specific ℓ, i). We now construct a CCA-2 adversary $\mathcal{A}_{\mathsf{cca}}$ that works as follows.[2] First, $\mathcal{A}_{\mathsf{cca}}$ guesses i and ℓ, hoping that they will be minimal for $\mathsf{missing}_\ell^i = 1$. Next, $\mathcal{A}_{\mathsf{cca}}$ sets $pk_\ell = pk$ where pk is the CCA2 public-key that it receives as part of the CCA2 encryption experiment. Then, it computes two independent El-Gamal encryptions m_i, m_i' of w_i and computes $c_i^{\ell+1} = E_{pk_{\ell+1}}(\cdots(E_{pk_n}(m_i))\cdots)$ and ${c'}_i^{\ell+1} = E_{pk_{\ell+1}}(\cdots(E_{pk_n}(m_i'))\cdots)$ as the pair of plaintexts for generating the challenge ciphertext in the CCA2 experiment. Observe that the challenge ciphertext that $\mathcal{A}_{\mathsf{cca}}$ obtains in the encryption experiment is the encryption c_i^ℓ of m_i (or m_i') under the keys $pk_n, \ldots, pk_\ell$. Thus, $\mathcal{A}_{\mathsf{cca}}$ can continue to encrypt the ciphertext with keys $pk_{\ell-1}, \ldots, pk_1$ and the result will be

$$c_i = E_{pk_1}(\cdots(E_{pk_n}(EG(w_i))$$

where $EG(w)$ denotes an El Gamal encryption of w_i. From here on, $\mathcal{A}_{\mathsf{cca}}$ perfectly simulates the shuffle experiment for adversary $\mathcal{A}$. This involves playing all honest parties, which is straightforward for every P_j with $j \neq \ell$ because it can choose all of the keys itself. The only problem is that $\mathcal{A}_{\mathsf{cca}}$ must also play the honest party P_ℓ, including running the ℓth shuffle which involves *decrypting* all of the ciphertexts in $\mu_{\ell-1}$ with sk_ℓ. However, $\mathcal{A}_{\mathsf{cca}}$ does not have sk_ℓ. Here, we use the fact that $\mathcal{E}$ is CCA2-secure and so in the CCA2 encryption experiment, $\mathcal{A}_{\mathsf{cca}}$ can use the decryption oracle to decrypt everything but its challenge $c = c_i^\ell$. This is the crucial point of the proof here; in the case that $\mathsf{missing} = 1$ and $\mathcal{A}_{\mathsf{cca}}$ guessed (i, ℓ) correctly, we have that $\mathsf{missing}_\ell^i = 1$ and so c_i^ℓ *does not appear* in the vector $\mu_{\ell-1}$. Thus, $\mathcal{A}_{\mathsf{cca}}$ can decrypt the entire vector using its decryption oracle and can generate a valid μ_ℓ, just like in the real shuffle experiment. Now, if $\mathsf{AllVerify} = 1$, then either the El Gamal encryption m_i or the El Gamal encryption m_i' must appear in the verification stage. However, these are of high entropy (they each contain a random element g^{ρ_i} from $\mathbb{G}$) and the view of the shuffle experiment only contains one of them, depending on if the challenge ciphertext is an encryption of $c_i^{\ell+1}$ (in which case only m_i appears) or ${c'}_i^{\ell+1}$ (in which case only m_i' appears). Thus, the encryption m_i or m_i' appearing in the verification stage must be from the challenge ciphertext, except with negligible probability. This implies that $\mathcal{A}_{\mathsf{cca}}$ is able to guess which plaintext was encrypted, in contradiction to the security of $\mathcal{E}$.

We now formally describe $\mathcal{A}_{\mathsf{cca}}$. Let t and n be parameters, let $\epsilon : \mathbb{N} \to [0,1]$ be a function, and let $\mathcal{A}$ be an adversary for the private shuffle experiment such that $\Pr[\mathsf{missing} = 1 \ \wedge \ \mathsf{AllVerify} = 1] = \epsilon(k)$ with t and n.

The adversary $\mathcal{A}_{\mathsf{cca}}$: *Upon input 1^k and public-key pk, and with oracle access to $D_{sk}(\cdot)$:*

1. *$\mathcal{A}_{\mathsf{cca}}$ invokes $\mathcal{A}$ upon input 1^k and with parameters t and n and obtains back the set I of indices of the corrupted parties, and the distinct words $w_1, \ldots, w_{n-t}$.*
2. *$\mathcal{A}_{\mathsf{cca}}$ chooses random $\ell, i \in_R [n] \setminus I$.*
3. *$\mathcal{A}_{\mathsf{cca}}$ interacts with $\mathcal{A}$, playing the honest parties in the shuffle protocol as follows:*
   (a) *$\mathcal{A}_{\mathsf{cca}}$ sets up the keys of the honest parties like in the protocol, except for P_ℓ for which it chooses a random α_ℓ like an honest party, but sets $pk_\ell = pk$, where pk is its input above.*
   (b) *$\mathcal{A}_{\mathsf{cca}}$ computes the ciphertexts c_j exactly like the honest parties in the protocol, except for c_i. This ciphertext is computed by computing two independent El-Gamal encryptions m_i, m_i' of w_i and computing $c_i^{\ell+1} = E_{pk_{\ell+1}}(\cdots(E_{pk_n}(m_i))\cdots)$ and ${c'}_i^{\ell+1} = E_{pk_{\ell+1}}(\cdots(E_{pk_n}(m_i'))\cdots)$. Then, $\mathcal{A}_{\mathsf{cca}}$ outputs $(c_i^{\ell+1}, {c'}_i^{\ell+1})$ and receives back the challenge ciphertext c. Finally, c_i is set to $E_{pk_1}(\cdots(E_{pk_{\ell-1}}(c))\cdots)$. In addition, in the ℓth round of the shuffle stage,*

[2] Recall that in the CCA2 encryption experiment, $\mathcal{A}_{\mathsf{cca}}$ receives a public key pk and oracle access to $D_{sk}(\cdot)$. Then, $\mathcal{A}_{\mathsf{cca}}$ outputs a pair of plaintexts m_0, m_1 and receives back a challenge ciphertext $c = E_{pk}(m_b)$ for a random bit $b \in_R \{0,1\}$. Finally, $\mathcal{A}_{\mathsf{cca}}$ continues to receive oracle access to $D_{sk}(\cdot)$ with the exception that it cannot query c itself, and outputs a guess b'.

(c) $\mathcal{A}_{\mathsf{cca}}$ *runs the shuffle stages of the protocol like the honest parties, with the following exceptions. First, if there exists an* (ℓ', i') *such* $\mathsf{missing}_\ell^i = 1$ *and* $\ell' < \ell$ *or* $\ell' = \ell$ *but* $i' < i$ *(where* ℓ *and* i *are as guessed above), then* $\mathcal{A}_{\mathsf{cca}}$ *aborts and outputs a random bit. Furthermore, if* $\mathsf{missing}_\ell^i = 0$*, then* $\mathcal{A}_{\mathsf{cca}}$ *aborts and outputs a random bit. Otherwise,* $\mathcal{A}_{\mathsf{cca}}$ *carries out the shuffles of all honest parties apart from* P_ℓ *exactly as in the protocol specification, and it carries out the shuffle of* P_ℓ *using* $D_{sk}(\cdot) = D_{sk_\ell}(\cdot)$*; it can do this since* $\mathsf{missing}_\ell^i = 1$ *and so the challenge ciphertext* c *does not appear in* $\mu_{\ell-1}$*.*

4. *In the verification stage* $\mathcal{A}_{\mathsf{cca}}$ *checks if either* m_i, m_i' *appear. If neither appear, then* $\mathcal{A}_{\mathsf{cca}}$ *outputs a random bit. Otherwise, if* m_i *appears it outputs 0 and if* m_i' *appears in outputs 1.*

We have already discussed the intuition behind this attack strategy by $\mathcal{A}_{\mathsf{cca}}$, and so we now proceed to prove that $\mathcal{A}_{\mathsf{cca}}$ outputs the correct b with probability $1/2 + \epsilon(k)/(n-t)^2$ which is non-negligible if $\epsilon(k)$ is non-negligible.

Define fail to be the event causing $\mathcal{A}_{\mathsf{cca}}$ to output a random bit in its attack. We have:

$$\Pr[\mathsf{Expt}^{\mathsf{cca}}_{\mathcal{A}_{\mathsf{cca}},\mathcal{E}}(k) = 1] = \Pr[\mathsf{Expt}^{\mathsf{cca}}_{\mathcal{A}_{\mathsf{cca}},\mathcal{E}}(k) = 1 \mid \neg\mathsf{fail}] \cdot \Pr[\neg\mathsf{fail}] + \Pr[\mathsf{Expt}^{\mathsf{cca}}_{\mathcal{A}_{\mathsf{cca}},\mathcal{E}}(k) = 1 \mid \mathsf{fail}] \cdot \Pr[\mathsf{fail}]. \quad (1)$$

Now, by the definition of fail, we have that $\Pr[\mathsf{Expt}^{\mathsf{cca}}_{\mathcal{A}_{\mathsf{cca}},\mathcal{E}}(k) = 1 \mid \mathsf{fail}] = 1/2$. In addition, if fail does not occur then one of m_i, m_i' appear in the last vector and in the verification stage. Now, if c_i^ℓ was encrypted in the encryption experiment, then information theoretically the value of m_i' does not appear anywhere in the shuffle experiment. Thus, the probability that m_i' appears in the verification stage is at most $1/q$ which is negligible (this holds because $\mathbb{G}$ is of order q and the first element of m_i' is a random element of $\mathbb{G}$). Thus, the probability that $\mathcal{A}_{\mathsf{cca}}$ outputs an incorrect bit, condition on fail not happening is at most negligible, and

$$\Pr[\mathsf{Expt}^{\mathsf{cca}}_{\mathcal{A}_{\mathsf{cca}},\mathcal{E}}(k) = 1 \mid \neg\mathsf{fail}] \geq 1 - \mathsf{negl}(k)$$

for some negligible function $\mathsf{negl}(\cdot)$. It remains to compute $\Pr[\mathsf{fail}]$ and $\Pr[\neg\mathsf{fail}]$, in order to evaluate Eq. (1).

We will compute $\Pr[\neg\mathsf{fail}]$:

$$\begin{aligned}\Pr[\neg\mathsf{fail}] = {} & \Pr[\neg\mathsf{fail} \mid \mathsf{missing} = \mathsf{AllVerify} = 1] \cdot \Pr[\mathsf{missing} = \mathsf{AllVerify} = 1] \\ & + \Pr[\neg\mathsf{fail} \mid \mathsf{missing} = 0 \vee \mathsf{AllVerify} = 0] \cdot \Pr[\mathsf{missing} = 0 \vee \mathsf{AllVerify} = 0]\end{aligned}$$

By the assumption regarding $\mathcal{A}$, we have that $\Pr[\mathsf{missing} = \mathsf{AllVerify} = 1] = \epsilon(k)$, and so it follows that $\Pr[\mathsf{missing} = 0 \vee \mathsf{AllVerify} = 0] = 1 - \epsilon(k)$. Next, if $\mathsf{missing} = \mathsf{AllVerify} = 1$, then $\mathcal{A}_{\mathsf{cca}}$ only outputs a random bit if it did not choose the minimal (ℓ, i). Since $\ell, i \in_R [n] \setminus I$, we have that $\mathcal{A}_{\mathsf{cca}}$ chooses the minimal (ℓ, i) with probability exactly $1/(n-t)^2$. Thus,

$$\Pr[\neg\mathsf{fail} \mid \mathsf{missing} = \mathsf{AllVerify} = 1] = \frac{1}{(n-t)^2}.$$

In contrast, when $\mathsf{missing} = 0$ or $\mathsf{AllVerify} = 0$, then $\mathcal{A}_{\mathsf{cca}}$ always outputs a random bit. Thus,

$$\Pr[\neg\mathsf{fail} \mid \mathsf{missing} = 0 \vee \mathsf{AllVerify} = 0] = 0.$$

Combining the above, we have that

$$\Pr[\neg\mathsf{fail}] = \frac{\epsilon(k)}{(n-t)^2} \quad \text{and} \quad \Pr[\mathsf{fail}] = 1 - \frac{\epsilon(k)}{(n-t)^2}.$$

Plugging this into Eq. (1), we conclude that

$$\begin{aligned}\Pr[\mathsf{Expt}^{\mathsf{cca}}_{\mathcal{A}_{\mathsf{cca}},\mathcal{E}}(k) = 1] &= (1 - \mathsf{negl}(k)) \cdot \frac{\epsilon(k)}{(n-t)^2} + \frac{1}{2} \cdot \left(1 - \frac{\epsilon(k)}{(n-t)^2}\right) \\ &= \frac{\epsilon(k)}{(n-t)^2} + \frac{1}{2} - \frac{\epsilon(k)}{2(n-t)^2} - \mathsf{negl}'(k) \\ &= \frac{1}{2} + \frac{\epsilon(k)}{2(n-t)^2} - \mathsf{negl}'(k).\end{aligned}$$

Thus, if $\epsilon(k)$ is non-negligible, then $\mathcal{A}_{\mathsf{cca}}$ succeeds in $\mathsf{Expt}^{\mathsf{cca}}_{\mathcal{A}_{\mathsf{cca}},\mathcal{E}}(k)$ with probability that is non-negligibly greater than 1/2. By the CCA2-security of $\mathcal{E}$ we conclude that $\epsilon(k)$, is negligible, thereby completing the proof. ■

We have proven that $\Pr[\mathsf{missing} = 1 \wedge \mathsf{AllVerify} = 1]$ is negligible. In order to see how the rest of the proof will proceed, we observe that:

$$\begin{aligned}
&\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1] \\
&= \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 1] + \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] \\
&= \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 1 \wedge \mathsf{AllVerify} = 1] \\
&\quad + \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 1 \wedge \mathsf{AllVerify} = 0] \\
&\quad + \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] \\
&\leq \Pr[\mathsf{missing} = 1 \wedge \mathsf{AllVerify} = 1] + \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{AllVerify} = 0] \\
&\quad + \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] \\
&\leq \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{AllVerify} = 0] + \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] + \mathsf{negl}(k)
\end{aligned}$$

where the last equation is by what we have already proven. Thus, it remains to prove that

$$\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{AllVerify} = 0] \leq \frac{1}{n-t} + \mathsf{negl}(k) \tag{2}$$

and

$$\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] \leq \frac{1}{n-t} + \mathsf{negl}(k) \tag{3}$$

in order to conclude that

$$\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1] \leq \frac{2}{n-t} + \mathsf{negl}(k),$$

as required.

Stated in words, Eq. (2) means that if the verification fails and thus the protocol aborts, the adversary cannot succeed more than by just making a random guess. Intuitively, this is the case since when the verification fails the honest parties do not remove the El Gamal encryption of the actual search words. Next, Eq. (3) states that if the ciphertexts of all the honest parties appear in every shuffle carried out by the honest parties, then once again the adversary cannot succeed more than by just making a random guess. Intuitively, this is because this guarantees that at least one honest party shuffled all of the honest parties' ciphertexts, and so the adversary cannot connect the output search words to the users.

We prove both of these equations by first defining a one round permutation experiment. The experiment is defined similarly for the case of CPA and CCA2 secure encryption; the CPA-case is used for proving Eq. (2) (which is based on the CPA security of El Gamal) and the CCA2-case is used for proving Eq. (3) (which is based on the CCA2-security of the encryption scheme). Note that in the case of CCA2-security, the adversary is also provided with access to a decryption oracle. We denote the attack type (CPA or CCA2) in the definition by T; formally $T \in \{CPA, CCA2\}$. We also denote the encryption scheme used in the experiment by $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Experiment Permutation $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},T}(n)$:

1. Run $\mathsf{Gen}(1^n)$ to get a key pair (sk, pk).
2. Invoke the adversary $\mathcal{A}$ on security parameter 1^n and public-key pk. If $T = CCA$, then $\mathcal{A}$ is also given oracle access to $\mathsf{Dec}_{sk}(\cdot)$.
3. $\mathcal{A}$ returns a list of *distinct* messages $w_1, \ldots, w_{n-t}$ of the same length.

4. Choose a random permutation π over $\{1,\ldots,n-t\}$ and compute $c_1,\ldots,c_{n-t}$ where for every i, $c_i = E_{pk}(m_{\pi(i)})$.

5. Give the adversary $\mathcal{A}$ the ciphertexts $c_1,\ldots,c_{n-t}$. If $T = CCA$, then $\mathcal{A}$ is also given oracle access to $\mathsf{Dec}_{sk}(\cdot)$, with the exception that it cannot query any of $c_1,\ldots,c_{n-t}$.

6. $\mathcal{A}$ outputs a pair (i,j)

7. The output of the experiment is 1 if and only if $j = \pi(i)$.

Claim 8 *If the encryption scheme $\mathcal{E}$ is T-secure for $T \in \{CPA, CCA2\}$, then for every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},T}(n) = 1\right] \leq \frac{1}{n-t} + \mathsf{negl}(k)$$

Proof Sketch: We construct an adversary for the regular CPA/CCA2 encryption experiment for multiple messages (i.e., the adversary outputs two vectors of messages and tries to guess which one was encrypted). This adversary takes the vector $\alpha_1,\ldots,\alpha_{n-t}$ of messages output by the adversary for $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},T}$ and outputs two random permutations of the messages. It then receives back an encryption of one of them. After completing the experiment it receives back $(i,\pi(i))$ and checks if this is correct for exactly one of the permutations. If yes, it guesses that this is the encrypted one. If not (either it is correct for both or incorrect for both), then it outputs a random guess. It can be shown that this strategy yields a non-negligible advantage in the CPA/CCA2 experiment if there is an adversary with a non-negligible advantage in the $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},T}$ experiment. ■

We now use Claim 8 in order to prove Eq. (2).

Claim 9 *Assume that the decisional Diffie-Hellman (DDH) assumption holds relative to $(\mathbb{G},q,g)$ as used in the protocol. Then, for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a negligible function negl such that*

$$\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \wedge \mathsf{AllVerify} = 0] \leq \frac{1}{n-t} + \mathsf{negl}(k)$$

Proof: Assume, by contradiction, that there exists an adversary $\mathcal{A}$ for which the event "$\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) \wedge \mathsf{AllVerify} = 0$" holds with probability that is non-negligibly greater than $1/(n-t)$. Let $\epsilon(k)$ be such that

$$\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \wedge \mathsf{AllVerify} = 0] = \frac{1}{n-t} + \epsilon(k)$$

We use $\mathcal{A}$ to construct an adversary $\mathcal{A}_{\text{CPA}}$ that breaks $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},\text{CPA}}$ with the encryption scheme being El Gamal (using $(\mathbb{G},q,g)$).

The adversary $\mathcal{A}_{\text{CPA}}$: Upon input security parameter 1^k and a public key $h = g^\alpha$ (where α is unknown):

1. $\mathcal{A}_{\text{CPA}}$ invokes $\mathcal{A}$ with k,t,n in $\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k)$ and receives back a set of t indices I and a vector $w_1,\ldots,w_{n-t}$ of distinct inputs for the honest parties.

2. $\mathcal{A}_{\text{CPA}}$ sets the El Gamal key h of the first honest party to be g^α and picks $n-t-1$ random El Gamal private keys $\alpha_j \in Z_q^*$ for all other honest parties. Let ℓ denote the index of the first honest party; i.e., $h_\ell = h = g^\alpha$. In addition, $\mathcal{A}_{\text{CPA}}$ chooses $n-t$ key pairs (sk_j, pk_j) for the CCA2-secure scheme in the protocol.

3. $\mathcal{A}_{\text{CPA}}$ gives $\mathcal{A}$ all of the public keys of the honest parties $\{h_j, pk_j\}_{j\notin I}$ along with the proof of knowledge (this proof is simulated for the first honest party since $\mathcal{A}_{\text{CPA}}$ doesn't know α).

4. $\mathcal{A}_{\text{CPA}}$ outputs $w_1, \ldots, w_{n-t}$ (to the experiment) and receives a vector of ciphertexts $\hat{m}_1, \ldots, \hat{m}_{n-t}$ encrypted under $h_\ell = h = g^\alpha$ and permuted.

5. $\mathcal{A}_{\text{CPA}}$ extracts all of the $\{\alpha_i\}_{i \in I}$ secret keys from adversary using the extractor of the proof of knowledge.

6. Given all of the El Gamal keys except for α, the adversary $\mathcal{A}_{\text{CPA}}$ transforms the El Gamal encryptions under pk to El Gamal encryptions under all keys $\sum_{i=1}^{n} \alpha_i$ as follows. For every $\hat{m}_j = (u_j, v_j) = (g^{r_j}, h^{r_j} \cdot w_{\pi(j)}) = (g^{r_j}, (g^{r_j})^{\alpha_\ell} \cdot w_j)$ it computes

$$m_j = \left(u_j, (u_j)^{\sum_{i \neq \ell} \alpha_i} \cdot v_j\right) = \left(g^{r_j}, (g^{r_j})^{\sum_{i \neq \ell} \alpha_i} \cdot (g^{r_j})^{\alpha_\ell} \cdot w_{\pi(j)}\right) = \left(g^{r_j}, \left(g^{\sum_{i=1}^{n} \alpha_i}\right)^{r_j} \cdot w_{\pi(j)}\right)$$

   which is therefore a valid encryption of $w_{\pi(j)}$ under all keys.

7. $\mathcal{A}_{\text{CPA}}$ encrypts each m_j in n layers using $\mathcal{E}$, as in the protocol.

8. $\mathcal{A}_{\text{CPA}}$ executes the private shuffle protocols, for every round between 1 and n, as follows:

   (a) If it is an honest user's round, it permutes and decrypts one layer of the outer encryption.

   (b) If it is an corrupted party's round, $\mathcal{A}_{\text{CPA}}$ gives the vector to $\mathcal{A}$ and receives back a new vector.

9. At the verification stage:

   (a) If $\mathsf{AllVerify} = 1$ (which $\mathcal{A}_{\text{CPA}}$ can check) then $\mathcal{A}_{\text{CPA}}$ just aborts.

   (b) Otherwise, $\mathcal{A}_{\text{CPA}}$ simulates all the honest parties aborting. Then, $\mathcal{A}_{\text{CPA}}$ outputs the pair (i, j) that is output by $\mathcal{A}$.

It is clear from the simulation that the view of $\mathcal{A}$ is computationally indistinguishable to that of a real protocol execution within $\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k)$ (the only difference is due to the simulated proof of knowledge). Furthermore, whenever the verification fails, $\mathcal{A}_{\text{CPA}}$ succeeds with the same probability as $\mathcal{A}$. We therefore have:

$$\begin{aligned} \Pr[\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A}_{\text{CPA}},\mathcal{E},\text{CPA}} = 1] &\geq \Pr[\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A}_{\text{CPA}},\mathcal{E},\text{CPA}} = 1 \ \wedge \ \mathsf{AllVerify} = 0] \\ &= \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \wedge \mathsf{AllVerify} = 0] \\ &= \frac{1}{n-t} + \epsilon(k). \end{aligned}$$

From the security of $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},\text{CPA}}$ we conclude that ϵ must be negligible. This completes the proof of the claim. ■

We complete the proof of security by proving Eq. (3).

Claim 10 *Assume that $\mathcal{E}$ is a CCA2 secure encryption scheme. Then, for every probabilistic polynomial-time adversary $\mathcal{A}'$ there exists a negligible function* negl *such that*

$$\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] \leq \frac{1}{n-t} + \mathsf{negl}(k)$$

Proof: Let $\mathcal{A}'$ be an adversary for $\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k)$ and let $\epsilon(k)$ be a function such that

$$\Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] = \frac{1}{n-t} + \epsilon(k).$$

We show that ϵ is a negligible function, by constructing an adversary $\mathcal{A}$ for $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},\text{CCA}}$ that succeeds also with probability $1/(n-t) + \epsilon(k)$.

The adversary $\mathcal{A}$: Upon input security parameter 1^k and a public key pk for the CCA2-secure encryption scheme, $\mathcal{A}$ (who is also given oracle access to $\mathsf{Dec}_{sk}(\cdot)$) works as follows:

1. $\mathcal{A}$ receives from $\mathcal{A}'$ a set of t indices $I \subset [n]$ to indicate which parties are corrupted, and a set of distinct inputs $w_1, \ldots, w_{n-t}$
2. $\mathcal{A}$ picks a random $\ell \notin I$; let $pk_\ell = pk$.
3. $\mathcal{A}$ picks $n-t$ random $\alpha_j \in Z_q^*$, and $n-t-1$ key pairs (sk_j, pk_j) for $\mathcal{E}$. These are the keys of the honest parties.
4. $\mathcal{A}$ gives $\mathcal{A}'$ the public keys of the honest parties $\{h_j = g^{\alpha_j}, pk_j\}_{j=1}^{n-t}$ along with a proof of knowledge of the value α_j for every j.
5. $\mathcal{A}$ receives t pairs of public keys (h_i, pk_i) for El Gamal and for $\mathcal{E}$, along with a proof of knowledge of the exponents α_i.
6. $\mathcal{A}$ encrypts $w_1, \ldots, w_{n-t}$ as follows:
   (a) $\mathcal{A}$ first encrypts each w_i using the threshold ElGamal
   (b) $\mathcal{A}$ encrypts $m_1, \ldots, m_{n-t}$ with $n-\ell$ layers of encryption (starting from pk_n and working backwords) using $\mathcal{E}$, and obtains $c_1^{\ell+1}, \ldots, c_{n-t}^{\ell+1}$
   (c) $\mathcal{A}$ gives $c_1^{\ell+1}, \ldots, c_{n-t}^{\ell+1}$ to $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},\mathrm{CCA}}$ and receives back $c^\ell_{\pi(1)}, \ldots, c^\ell_{\pi(n-t)}$
   (d) $\mathcal{A}$ continues to encrypt $c^\ell_{\pi(1)}, \ldots, c^\ell_{\pi(n-t)}$ using $\mathcal{E}$ for layers $\ell-1$ to 1 and obtains $c_1, \ldots, c_{n-t}$.
7. $\mathcal{A}$ executes the private shuffle protocols as follows:
   (a) For every round j between 1 and $\ell-1$:
      - If it is an honest user's round, $\mathcal{A}$ permutes and decrypts one layer of the outer encryption.
      - If it is an corrupted party's round, $\mathcal{A}$ gives the vector to $\mathcal{A}'$ and receives a new vector.
   (b) In the ℓth round, $\mathcal{A}$ first checks if $\mathsf{missing} = 1$. If yes, it just aborts. Otherwise, $\mathcal{A}$ "decrypts" the honest party's ciphertexts. We stress that it cannot actually decrypt them since it doesn't know sk_ℓ. However, since it knows the mapping of $c_1^{\ell+1}, \ldots, c_{n-t}^{\ell+1}$ to $c^\ell_{\pi(1)}, \ldots, c^\ell_{\pi(n-t)}$ (albeit after a random permutation), it can just use $c^\ell_{\pi(1)}, \ldots, c^\ell_{\pi(n-t)}$ as the decrypted values.[3] The remaining ciphertexts (generated by the adversary) are given to the decryption oracle (unless there are ciphertexts copied from the honest, in which case the honest party aborts in the protocol and thus so does $\mathcal{A}$, outputting whatever pair $\mathcal{A}'$ outputs). $\mathcal{A}$ then permutes the entire vector of decrypted ciphertexts. (Note that $c^\ell_{\pi(1)}, \ldots, c^\ell_{\pi(n-t)}$ are not in the same order as $c_1^{\ell+1}, \ldots, c_{n-t}^{\ell+1}$. However, since the honest party permutes the result anyway, this makes no difference.
   (c) For every round j between $\ell+1$ and n, $\mathcal{A}$ decrypts exactly as in rounds j between 1 and $\ell-1$.
8. $\mathcal{A}$ completes the verification stage exactly like the honest parties (it can do this since it knows all of the α_j values of the honest parties).
9. $\mathcal{A}'$ outputs the pair (i, j) that is output by $\mathcal{A}$.

[3]Here it is crucial that none of the ciphertexts is missing. Otherwise, $\mathcal{A}$ cannot know which of the ciphertexts should be included and which not. In order to see this, note that $\mathcal{A}$ knows all of $c_1^{\ell+1}, \ldots, c_{n-t}^{\ell+1}$ and $c^\ell_{\pi(1)}, \ldots, c^\ell_{\pi(n-t)}$, but does not know which ciphertext is mapped to which. Thus, if $c_1^{\ell+1}$ is missing, then $\mathcal{A}$ cannot know which of the c_j^ℓ values to remove. This is because $\mathcal{A}$ does not know the permutation π.

$\mathcal{A}$ does not know the order of the messages in the ℓth round, but it does not need to since it only needs to give a random permutation to the next party. Here $\mathcal{A}$ uses both its knowledge of how the ciphertexts of the honest parties looked prior to applying the ℓth encryption along with the answers it received from the decryption oracle for the ciphertexts of the remaining parties. After applying a random permutation to all we obtain a view that is identical to that of $\mathcal{A}'$ in a real execution in which the ℓth party really decrypts and permutes these messages. Thus, $\mathcal{A}$ succeeds in its guess with exactly the same probability as $\mathcal{A}'$. We conclude:

$$\begin{aligned}
\Pr\left[\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},\mathrm{CCA}} = 1\right] &\geq \Pr\left[\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},\mathrm{CCA}} = 1 \ \wedge \mathsf{missing} = 0\right] \\
&= \Pr[\mathsf{ExptShuffle}^{t,n}_{\mathcal{A},\pi}(k) = 1 \ \wedge \ \mathsf{missing} = 0] \\
&= \frac{1}{n-t} + \epsilon(k)
\end{aligned}$$

and so from the security of $\mathsf{Expt}^{\mathsf{permute}}_{\mathcal{A},\mathcal{E},\mathrm{CCA}}$, ϵ must be negligible. ■

This completes the proof of the theorem. ■

5 Private Web Search

In this section we show how to use a *private shuffle* in order to achieve private web search. As we will show below, a system for private web search needs to take into account additional considerations that are not covered by the notion of a private shuffle (or even a fully secure mix-net). In this section we address these considerations, describe the assumptions that we make, and present a general scheme that models real-world threats and is thus implementable in practice.

5.1 Background

As in [2], the basic idea of the scheme is to allow many users who wish to submit a web query to team up in a group, shuffle their queries in a private manner and then have each of them perform one of the queries without knowing who it belongs to. Upon receiving back the query results, each party just sends them to all others in the group so that the original party who sent the query can learn the result. This methodology prevents the search engine from linking between a user and its search query. Furthermore, the users in the group do not know on whose behalf they send a query; all they know is that it belongs to someone within the group. An important question in such a system is how to group users together. One possibility is to do this in a *peer-to-peer* way, so that whenever a user has a query it can notify the peer network in order to find out who else has a query at this time. The parties with queries can then join in an ad-hoc way in order to privately shuffle them before sending them to the search engine. (Note that parties who are currently idle can help by sending dummy queries, if they like.) This is a feasible model, but has significant implementation difficulties. The alternative suggested by [2], and one that we follow for the remainder of this section, is to use a *central server* whom anyone interested in searching can approach. The server then notifies the parties wishing to currently search of each others' identities so that they can form a group in order to carry out a private shuffle. This model is easily implemented by simply having the server be a website offering a "private search" utility.

As we mentioned in the introduction, the problem with the scheme suggested by [2] was that it assumed that all parties are semi-honest. In our view this is highly unrealistic, especially since a single corrupt party can completely break the privacy of the scheme and learn every party's search query. We now show how to achieve private web search in the presence of malicious adversaries. In order to do this, we use the *private shuffle* protocol presented in Section 3 that maintains privacy in the presence of malicious adversaries. We stress that private shuffle within itself does not suffice for obtaining private web search in practice for the following reasons:

1. A malicious central server can choose the group so that it controls all but one user. As we explain below, this completely bypasses the security guarantees of the shuffle.

2. The result of the web search queries must be sent to all parties because we don't know which user sent which query. This means that users learn the search results for all the members in their group, which is much more information than necessary (although the search engine must learn all queries, this is not the case for users).

Below, we will present a system for web search that uses the *private shuffle* protocol, while addressing the above concerns.

5.2 A Private Web Search System

Our solution is comprised of four phases that together enable private web search:

- **Phase 0:** Installation and initialization
- **Phase 1:** Ad-hoc group setup
- **Phase 2:** Private shuffle of the search queries
- **Phase 3:** Query submission and private response retrieval

We remark that an ad-hoc group can be used for many searches, and ideally would be used for a session of a reasonable amount of time. This enables us to reduce the overhead due to running phase 1.

5.2.1 Phase 0 – installation and initialization:

Our *private shuffle* protocol requires a PKI and communication with a central server. A natural realization of this would be as an Add-on to a web browser that would supply a functionality which is similar to the search window in the most common web browsers. This Add-on would contain the address of a central server (or a list of servers). Regarding the PKI, since most users do not have a certificate for digital signatures, we have to generate one. The most practical way to do this would be to use a one-time activation of the Add-on after installation, in which a key pair is generated and a digital certificate then downloaded from a CA. Recall that without a PKI, the efficient verification in our *private shuffle* protocol does not guarantee that it was the honest parties in the group that sent `true` in the verification of the shuffle stage. We stress that a different certificate can be installed on every machine using the Add-on.

5.2.2 Phase 1 – ad-hoc group setup:

As mentioned above, users group together with the help of a server $\mathcal{S}$ that aggregates the identities of users that wish to currently engage in private web search. Conceptually speaking, in terms of role and trust, the server should be no more than a bulletin board for anonymous users who wish to create an ad-hoc group. In [2], the server was assumed to be a trusted entity who does not collude with any of the users nor with the web search engines. However, the role of grouping users together carries with it a lot of power that can easily be abused. Specifically, consider a server that has $t \geq n$ machines at its disposal (or even a single machine that can pretend to be t different users), where n is the size of the group. Then, the server can always group some single honest user with $n-1$ of the t server-owned users. If an honest user runs a private shuffle in this way, then its privacy is completely lost because the server knows the search queries of all the users except for the honest one. Thus, at the end of the protocol when all queries are revealed, the server knows the exact query made by the honest user. We stress that this holds even if the mix carried out is *perfectly secure*.

In order to prevent the server from grouping the users as it wishes, we have all parties run a type of joint coin tossing protocol so that the t parties controlled by a malicious server are uniformly distributed within all the groups running the shuffle. Let N denote the overall number of parties in the system, let t denote the overall number of parties under the control of the malicious server, and let n be the size of each

group running the shuffle. Our coin-tossing protocol uses two random oracles H_1 and H_2. Each party P_i sends $H_1(IP_i, PK_i, r_i)$ to the server to be posted (where r_i is a long random string). Then, the groups are formed by applying H_2 to all the values $H_1(IP_1, PK_1, r_1), \ldots, H_1(IP_N, PK_N, r_N)$. Denote the output of H_2 by $o = (o_1, \ldots, o_N)$ where each o_i is of length $\log N$. Letting o_i be the temporary name of party P_i, we have that the output of H_2 induces an order on the parties by taking the lexicographic ordering of the temporary names. Using this order the users are grouped into groups of size n. Observe that the server can choose the r_j values in $H_1(IP_j, PK_j, r_j)$ after it received all of the honest parties' H_1 values (where P_j is a party under its control). Furthermore, it can do so many times in an attempt to obtain a "bad group" in which all but one party are under its control We therefore need to make sure that the probability that a group is "bad" is very small (e.g., 10^{-40}). This will ensure that the server, after seeing the inputs from the honest users, still cannot find input values that would create a "bad group" in sufficient time. The reason that we use the random oracle H_1 in the process of sending the inputs, instead of just having the parties send (IP_i, PK_i, r_i) is in order to protect the identities of the users. Specifically, the server $\mathcal{S}$ will send the relevant IP addresses only to the relevant group, and so only the server $\mathcal{S}$ providing the service knows the history of which party participated in each group. As we will see below, it is important to prevent this information from being leaked, especially to the web search engine. Otherwise, statistical attacks can be carried out; see below for more details. The group setup appears in Protocol 11.

Protocol 11 (Group setup protocol)

Let H_1 and H_2 be two random oracles where $H_1 : \{0,1\}^* \to \{0,1\}^k$ and $H_2 : \{0,1\}^* \to \{0,1\}^{N \cdot logN}$. Let n be the size of each group for the shuffle. We set the initial indexing of the parties according to the lexicographical order of their IP addresses.

1. Each P_i chooses a random r_i and sends $H_1(IP_i, PK_i, r_i)$ to the center.
2. After a short predefined time everyone queries the center for the list of parties who have registered.
3. Each party computes $o = H_2(H_1(IP_1, PK_1, r_1), \ldots, H_1(IP_N, PK_N, r_N))$ and divides the result o into chunks of size $log\ N$, denoted $o_1, \ldots, o_N$. Party P_i is associated with o_i and the list is sorted according to the o_i values.
4. Grouping is carried out by taking groups of n parties according to the sorting. That is, for $i = 1, \ldots, \lfloor N/n \rfloor$, the ith group G_i is set to be the parties associated with the values $(o_{n\cdot(i-1)+1}, \ldots, o_{n\cdot i})$.
5. The center sends the IP addresses of the group members to the members of each group (i.e. each member gets only the IP addresses of the members in its group).
6. Members of each group send each other their IP address, public key and randomness that were used when registering with the center.
7. Each group member computes $H_1(IP_j, PK_j, r_j)$ for every party P_j in its group and verifies that it matches what was recorded by the center during registration. In addition, it verifies that it received the IP address of all parties that are in its group, by the computation of H_2. If no, then it sends abort to all the parties in its group.

We now analyze the security of Protocol 11. Recall that in the random oracle model, the output of H_2 is uniformly distributed every time that it is applied to a new value. We begin by analyzing the probability that a bad grouping occurs for a *given* set of values $\{(IP_i, PK_i, r_i)\}_{i=1}^N$. (Below we will analyze what this means when the server is malicious.) We call a group "bad" if it consists of $n-1$ malicious parties together with a single honest party. Clearly, this is bad because the server $\mathcal{S}$ then learns the search query of that party. The cases that a group has only a few honest parties is also quite bad, but there is still ambiguity regarding each user's search term. Furthermore, in Section 5.3 we discuss how to further improve this.

Let bad_i denote the event that the ith group is bad as defined above. We begin by computing the probability that the first group is bad; i.e., that bad_1 occurs. Since the output of H_2 is uniformly distributed, we can compute this by counting the number of ways to choose $n-1$ parties out of t malicious ones times the number of ways to choose a single honest party, divided by the total number of ways to choose a group

of size n out of N parties. That is, we have:

$$\begin{aligned}\Pr[\mathsf{bad}_1] &= \frac{\binom{N-t}{1}\cdot\binom{t}{n-1}}{\binom{N}{n}} = \frac{(N-t)\cdot\frac{t!}{(t-n+1)!(n-1)!}}{\frac{N!}{(N-n)!n!}} \\ &= \frac{\prod_{i=1}^{n-2}(t-i)}{\prod_{j=1}^{n-1}(N-j)}\cdot(N-t)\cdot n \\ &= \prod_{i=1}^{n-2}\frac{(t-i)}{(N-i)}\cdot\frac{N-t}{N-n+1}\cdot n\end{aligned}$$

Noting again that H_2 is a random function, it follows that the above calculation is true for any fixed group. Thus, the above gives the probability of bad_i for every $i = 1, \ldots, \lfloor N/n \rfloor$. As we have mentioned, a grouping is bad if there exists a bad group. Thus, applying the union bound over all $\lfloor N/n \rfloor$ groups we have that:

$$\Pr[\exists\ i\ :\ \mathsf{bad}_i] \leq \sum_{i=1}^{N/n}\Pr[\mathsf{bad}_i] = \frac{N}{n}\Pr[\mathsf{bad}_1] = \prod_{i=1}^{n-2}\frac{(t-i)}{(N-i)}\cdot\frac{N-t}{N-n+1}\cdot N$$

Assuming now that $N >> t$, we have that $\Pr[\exists\ i\ \text{ s.t. } \mathsf{bad}_i]$ is approximately $(\frac{t}{N})^{n-2}\cdot N$. Concretely, consider the case of millions of users running this protocol, a malicious server $\mathcal{S}$ that controls a few thousand of them, and a group size of about 20. In this case, we have that the probability that there exists a bad group for a given set of H_1 values is smaller than $10^6 \cdot (\frac{10^3}{10^6})^{18}$, which is 10^{-48}.

We stress that the above analysis alone is not sufficient. This is due to the fact that, as we have mentioned, it is possible for a malicious server $\mathcal{S}$ to modify the H_1 values many times in the aim of obtaining a bad grouping. Specifically, once all honest parties have submitted their values, the server can repeatedly modify the r_j portion of party P_j's input to H_1, where P_j is a malicious user under its control. Since any change to any of the H_1 values results in a completely different ordering of the parties (because H_2 is a random function), we have that the probability of a bad grouping is T times the above, where T equals the number of hashes that the server can compute in the required time interval. With the above example parameters where the probability of a bad grouping is 10^{-48}, the probability that a malicious server achieves a bad grouping within seconds is very small.

5.2.3 Phase 2 – private shuffle of the search queries:

Once the users have been grouped together, they run the *private shuffle* protocol of Section 3. However, as we discussed earlier in Section 5.1 (item 2 at the end of the section), we would like to prevent the group members from learning all the search results. This seems problematic because the parties do not know whose query they have and they must therefore broadcast the result to everyone. We overcome this problem by instructing each party to first choose a random symmetric encryption key k_j and then input the pair $wk_j = (w_j, k_j)$ to the shuffle. As we will see next, k_j will be used to encrypt the search result.

5.2.4 Phase 3 – query submission and private response retrieval:

After the shuffle protocol is completed, each party holds a pair (w', k'). Each party then submits the search query w' to the search engine and receives back the result. The search result along with the original search term is then encrypted using the key k' with a symmetric encryption scheme (e.g., AES) and broadcast to all group members. Each party attempts to decrypt all search results; the one that decrypts correctly is its own result. In this way, each party only learns its own result and the result of one other random user. Thus, privacy of the queries is better preserved.

5.3 Additional Considerations

We now address some of the issues that concern deployment of our scheme in the real world and discuss the privacy that it provides.

5.3.1 Blending into a crowd:

The main idea of our scheme is blending into a crowd. The fact that millions of people from all over the world can participate in the protocol provides a strong sense of privacy, but consideration should be given to the way different populations are grouped together. If 20 people from all over the world are grouped together and all submit the query in their native language, then it is easy to learn the query of each party based on the geographic location of its IP address. When deploying such a system, consideration should be given to these issues and blending into a crowd should actually be blending into a crowd of people with similar characteristics.

5.3.2 The size of a group:

Our *private shuffle* protocol provides anonymity with respect to the size of the group; thus the bigger the group the more anonymity one enjoys. Since the size of the group affects both the number of modular operations each party needs to perform and the number of rounds in the *private shuffle* protocol, the size of the group is bounded by the computing power of the users' computers and the acceptable latency. Nevertheless, it is possible to hide in a larger group at the expense of more modular exponentiations but without increasing the number of rounds, as follows. As we have described in remark 3 after Protocol 4, if we can assume that the number of malicious parties within a group is some $t' < n$, then it suffices to run the shuffle stage for $t' + 1$ rounds. Performing a similar analysis to the one above, we have that the probability of having 19 malicious parties within a group of size 50 is actually very close to the probability of having 19 malicious parties within a group of size 20 (when the total number of parties is about a million and the total number of malicious parties is several thousand). Thus, if one can afford the additional number of modular exponentiations that comes with increasing the group size, we can enhance privacy significantly by increasing the size of the group, without paying much more in latency. Observe that in this calculation a group is "bad" if there are $t' + 1$ malicious parties. Thus, if a group is not bad, each honest party's search query is guaranteed to be hidden amongst $n - t'$ other search queries.

5.3.3 Lifetime of a group:

Our scheme creates ad-hoc groups that can be changed over time. In terms of efficiency, it is easy to see that remaining within a group for a while saves the cost of running the group selection process. However, users may submit a query to the search engine and logout. In this case the group size would shrink and if it is too small then privacy is compromised. This can be dealt with by starting with a larger group and regrouping once the group becomes too small.

5.3.4 Statistical analysis and changing groups:

In terms of privacy, it may seem that the more often people change groups, the more privacy they gain. However, this actually may not always be the case. Consider a central server that colludes with the web search engine. The server $\mathcal{S}$ and search engine can then run a statistical analysis to group together queries that are likely to belong to the same user (e.g., by grouping together very low-probability queries). Now, if these queries are carried out in different groups, then the server $\mathcal{S}$ can find the (most likely) unique IP address that appears in all of the different groups, and conclude that the queries originated from this address. Thus, changing groups can be problematic. (Of course, without such collusion, this problem does not arise.)

5.3.5 An additional privacy enhancement:

The system presented above has the property that each user's search query is revealed to one other *random* group member. However, in some cases a user may prefer to be able to say which user will submit and therefore learn their query (and which users will not learn their query). We can extend our system for private web search to allow this by adding one more layer of encryption to the messages, using the public key of the designated party. Specifically, if a party P_j wishes to have party P_i be the one who submits its

query, then it encrypts wk_j along with some redundancy (to verify the correctness when opening) using g^{α_i}. Then P_j executes the private shuffle protocol with the encrypted wk_j. After the messages are shuffled, each party sends the message it received to everyone else, and all parties decrypt the results. In this way, only the designated party P_i is the one that can learn wk_j and it will send the query.

Acknowledgements

We would like to thank Gilad Asharov and Meital Levy for many helpful discussions, and the anonymous referees for helpful comments. A special thanks to Ian Goldberg for pointing out a serious error in the protocol appearing in the extended abstract of this work [13].

References

[1] M. Bellare and P. Rogaway Entity Authentication and Key Distribution. In *CRYPTO93*, Springer-Verlag (LNCS 773), pages 232-249, 1994.

[2] J. Castellà-Roca, A. Viejo and J. Herrera-Joancomarti Preserving User's Privacy in Web Search Engines. In *Computer Communications*, 32(13-14):1541–1551, 2009.

[3] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[4] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private Information Retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[5] B. Chor and M. Rabin. Achieving Independence in Logarithmic Number of Rounds. In the 6*th PODC*, pages 260–268, 1987.

[6] R. Cramer and V. Shoup, A Practical Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attacks. In *CRYPTO98*, Springer (LNCS 1462), pages 13-25, 1998.

[7] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, A. Sahai. Robust Non-interactive Zero-Knowledge. In *CRYPTO 2001,* Springer-Verlag (LNCS 2139), pages 566–598, 2001.

[8] Y. Desmedt and K. Kurosawa. How to Break a Practical MIX and Design a New One. In *EUROCRYPT 2000*, Springer-Verlag (LNCS 1807), pages 557–572, 2000.

[9] R. Dingledine, N. Mathewson and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the* 13*th USENIX Security Symposium*, pages 303–320, 2004.

[10] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO'84*, Springer-Verlag (LNCS 196), 1984.

[11] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86*, Springer-Verlag (LNCS 263), pages 186–194, 1986.

[12] M. Jakobsson. A Practical MIX. In *EUROCRYPT'98*, Springer-Verlag (LNCS 1403), pages 448–461, 1998.

[13] Y. Lindell, E. Waisbard. Private Web Search with Malicious Adversaries. *Privacy Enhancing Technologies*, pages 220–235, 2010.

[14] R. Ostrovsky and W.E. Skeith. A Survey of Single-Database PIR: Techniques and Applications. In the 10*th PKC*, Springer-Verlag (LNCS 4450), pages 393–411, 2007.

[15] C.P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 239–252, 1989.

A Efficient NIZK Proof of Knowledge of Discrete Log

The users in Protocol 4 need to prove knowledge of α_i in an efficient manner. Furthermore, it is crucial that these proofs be "independent" (technically, this means that it is possible to simultaneously simulate the proofs of the honest parties and extract the witnesses α_i of the corrupt parties). This can be achieved using the method of [5], but this requires interaction. In the random oracle model, a very fast alternative is to use the Fiat-Shamir heuristic [11] applied to the discrete log protocol of Schnorr [15]. This is standard, but we repeat it below for the sake of completeness. Observe also that the user's ID is included in the hash for generating the "verifier query" in order to ensure independence and that no party copies another party's proof. Let H be a hash function (modeled as a random oracle). The proof system appears in Protocol 12.

Protocol 12 (NIZK Proof of Knowledge of the Discrete Log of $y = g^x$)

- **Prover's instructions:**
  1. Choose a random $r \in \mathbb{Z}_q^*$ and compute $t = g^r$
  2. Compute $c = H(ID \| t)$ (where $\|$ denotes concatenation and ID is the prover's unique identifier of known length)
  3. Compute $s = r + cx$
  4. Output (ID, g^x, t, c, s).
- **Verifier's instructions:** Upon receiving (ID, y, t, c, s), accept if and only if $c = H(ID \| t)$ and $g^s = t \cdot y^c$

The fact that Protocol 12 constitutes a zero-knowledge proof of knowledge (with independence) is well known.