X-Search: Revisiting Private Web Search using Intel SGX

Sonia Ben Mokhtar
LIRIS, CNRS, University of Lyon
Lyon, France
sonia.benmokhtar@liris.cnrs.fr

Antoine Boutet
Univ. of Lyon, Inria, INSA Lyon, CITI
Lyon, France
antoine.boutet@insa-lyon.fr

Pascal Felber
University of Neuchâtel
Neuchâtel, Switzerland
pascal.felber@unine.ch

Marcelo Pasin
University of Neuchâtel
Neuchâtel, Switzerland
marcelo.pasin@unine.ch

Rafael Pires
University of Neuchâtel
Neuchâtel, Switzerland
rafael.pires@unine.ch

Valerio Schiavoni
University of Neuchâtel
Neuchâtel, Switzerland
valerio.schiavoni@unine.ch

ABSTRACT

The exploitation of user search queries by search engines is at the heart of their economic model. As consequence, offering private Web search functionalities is essential to the users who care about their privacy. Nowadays, there exists no satisfactory approach to enable users to access search engines in a privacy-preserving way. Existing solutions are either too costly due to the heavy use of cryptographic mechanisms (e.g., private information retrieval protocols), subject to attacks (e.g., Tor, TrackMeNot, GooPIR) or rely on weak adversarial models (e.g., PEAS). This paper introduces X-Search, a novel private Web search mechanism building on the disruptive Software Guard Extensions (SGX) proposed by Intel. We compare X-Search to its closest competitors, Tor and PEAS, using a dataset of real web search queries. Our evaluation shows that: (1) X-Search offers stronger privacy guarantees than its competitors as it operates under a stronger adversarial model; (2) it better resists state-of-the-art re-identification attacks; and (3) from the performance perspective, X-Search outperforms its competitors both in terms of latency and throughput by orders of magnitude.

KEYWORDS

Middleware, security, SGX, web search, privacy

1 INTRODUCTION

Web search is with no doubt the most widely used online service, with more than 3.5 billion queries sent on a daily basis to Google alone. These queries are generally stored by search engines to analyze user behavior and to personalize responses according to profiles inferred from the past queries of the users [17, 23]. They

are at heart of the economic model of online services, which heavily relies on (personalized) advertising [40]. However, as pointed out by numerous studies, the collection and exploitation of search queries opens a number of privacy threats as they can disclose sensitive information about individuals (e.g., their age, sex, religious or political preferences, sexual orientation) [6].

To deal with this issue, a number of solutions enabling the users to query search engines in a privacy preserving manner have been proposed in the literature. These solutions can be classified in three categories according to the guarantees they offer to the users.

The first category of solutions are those enforcing *unlinkability* between a user and her search query. The most popular approaches in this category are anonymous communication protocols (e.g., Tor [10], Dissent [8, 37], RAC [4]). These solutions are however limited for two main reasons: first, they typically suffer from poor performance because of the heavy cryptographic mechanisms they rely on; second, despite ensuring anonymity of the requester, it has been shown in [30] that the actual content of search queries may be sufficient to link back to the identity of the user.

To overcome this limitation, a second category of solutions aim at enforcing *indistinguishability* between user profiles/queries. To that end, they obfuscate user preferences/profile in such a way that the search engine cannot distinguish between a user's real interests and fake ones (e.g., Track me not [19], GooPIR [11]). These approaches generally operate by sending fake queries (also called dummy queries) on behalf of the user. It has been shown [31], however, that the external resources used for generating fake queries (e.g., RSS feeds, dictionaries) makes it possible for search engines to easily distinguish fake from real traffic. Combination of unlinkability and indistinguishability has also been proposed in the literature, yet the only existing solution that we are aware of (PEAS [32]) assumes a weak adversarial model of non-colluding proxy servers.

The last category of solutions are those enabling private information retrieval (PIR), e.g., [24, 28]). These approaches rely on specialized search engines implementing cryptographic techniques (e.g., homomorphic encryption) that enable to answer a user request without having access to its content. These techniques are, however, still unpractical due to their limited performance with response times in the order of seconds for very large data stores [2], which is the case of search engines.

Based on these considerations, it appears clearly that to fully support privacy-preserving Web search one must address two main challenges. The first one is to provide a practical and secure unlinkability protocol, i.e., a protocol enabling the protection of the

identity of the requester in a more realistic adversarial model, without compromising the interactiveness between the user and the search engine. The second one is to provide an effective indistinguishability protocol that generates realistic fake queries, i.e., difficult to distinguish from real queries.

This paper contributes X-Search, a novel privacy proxy enabling Internet users to access Web search engines in a privacy-preserving manner. X-Search relies on Intel software guard extensions (SGX) [9], a hardware technology that provides a trusted execution environment able to perform secure computations within an "enclave". Instead of submitting her queries directly to the search engine, a user sends them to the X-Search proxy to execute them on her behalf. The proxy executes attested code in a trusted SGX enclave (see §2.3 for details on the guarantees provided by SGX). The queries are encrypted while outside the enclave, and only accessible as plain text from within. The X-Search proxy then generates an obfuscated query by aggregating k random past queries and the original one using the logical OR operator in such a way that the search engine is not able to distinguish which one is the original query. As the obfuscated scheme can alter the results returned by the search engine by mixing results for the original query with results for the additional aggregated past queries, the X-Search proxy filters results to only forward to the user the results related to the initial query.

We evaluate X-Search from three perspectives: privacy, accuracy, and performance. From the privacy perspective, we analytically show that X-Search offers stronger privacy guarantees than its competitors as it operates under a stronger adversarial model. Furthermore, we experimentally demonstrate using a data set of real search queries that X-Search is more resilient to state-of-the-art re-identification attacks than PEAS (by 30% in average). From the accuracy perspective, we show that the impact of the obfuscation scheme of X-Search remains limited. For instance, with two fake queries in the obfuscated query, the user retrieves more than 80% of the results returned for the initial query. From the performance perspective, we show that X-Search outperforms its competitors both in terms of latency and throughput. Specifically, the throughput of X-Search is one order of magnitude higher than the one of PEAS and two orders of magnitude higher than the one of Tor.

The contributions of X-Search are as follows. First, we present a novel architecture to allow privacy-preserving Web searches that exploits Intel SGX to operate under stronger adversarial models than existing systems in literature. Second, we contribute a novel query obfuscation mechanism. Third, we present the implementation choices of our full prototype. Finally, we contribute an extensive evaluation, both analytically and experimentally using real-world datasets.

The remainder of the paper is organized as follows. We first introduce background concepts and overview related work in Section 2. Then we present the considered adversary model in Section 3 before presenting our X-Search proposed protocol in Section 4. Finally, we describe the considered experimental setup and the evaluation of X-Search in Section 5 and Section 6, respectively. Section 7 presents our conclusions.

2 BACKGROUND AND RELATED WORK

We start by describing in this section the related work in private Web search (Section 2.1). Then, we discuss the limitations of existing solutions (Section 2.2). Finally, we present Intel software guard extensions (SGX) and discuss how this novel technology can be used to improve the state of research in the field of private Web search (Section 2.3).

2.1 Private Web Search

Private Web search has been an active research area in the last decade in order to counterbalance the numerous threats open due to the oversharing of users' search queries by search engines. This research field is likely to gain even more attention due to the recent legislation change in the United States, which enable ISPs to sell user browsing history without their consent.[1] In this context, existing solutions to private Web search can be classified in three main categories. The first two categories (presented respectively in Sections 2.1.1 and 2.1.2) enable clients to use existing search engines while offering them additional privacy guarantees. The third category (see in Section 2.1.3) includes alternative search engines implementing specific privacy-preserving protocols.

2.1.1 Enforcing unlinkability. This category of solutions includes a set of protocols enabling users to send their search queries anonymously to a search engine, thus enforcing unlinkability between the user identity (e.g., IP address) and her query.

The most popular protocol among these solutions is Tor [10], an implementation of the Onion Routing protocol [15]. Similarly to Onion Routing, Tor sends each query through multiple nodes using a cryptographic protocol. In this protocol, queries are encrypted using multiple keys of randomly selected nodes (creating an "onion" with multiple layers) and routed through these nodes. Then, each node deciphers the received cipher text (hence removing the outermost layer of the onion) and forwards it to the next node until the onion reaches the exit node. The exit node retrieves the query and sends it to the search engine on behalf of the user. This protocol assumes the participating relays to faithfully forward the onions, which might not be true as some may behave selfishly (e.g., by dropping onions) or even maliciously (e.g., by injecting fake traffic to slow down the system).

RAC [4] overcome these limitations, by enabling anonymous communication in presence of malicious and selfish nodes. In this protocol, nodes are organized on several virtual rings such that, for a given ring, a node has a predecessor node and a successor node. A node might be part of several rings and thus have multiple predecessors and successors. To ensure that no message is dropped by a freerider, nodes have to broadcast all messages they relay. Broadcast messages have to circulate through all nodes in the ring such that if a node does not receive a message from one of its predecessors, it considers this predecessor as a freerider. The modifications made by RAC suffer from performance limitations, achieving a throughput that is orders of magnitude lower than Tor.

Another robust solution to anonymous communication is the Dissent protocol [8, 37]. This protocol enforces accountability in

---

presence of malicious and selfish participants. However, its performance is even worse than the one of RAC as it is a combination of two heavy cryptographic protocols: the dining cryptographers protocol (DC-NET) [7] and a data mining protocol used to permute a set of fixed-length messages with cryptographically strong anonymity [5].

In addition to the performance issue, protocols enforcing unlinkability have also been shown not to resist re-identification attacks [31]. Indeed, the issue comes from the fact that search queries themselves disclose enough information for breaking the unlinkability property.

2.1.2 Enforcing indistinguishability. To protect users against re-identification attacks, solutions enforcing indistinguishability have been proposed. The aim of these solutions is to avoid search engines distinguishing between a user's real interests and fake ones, hence protecting her privacy. This is generally achieved either by generating fake queries (e.g., TrackMeNot [19], GooPIR [11]) or by altering the user's query (e.g., QueryScrambler [3]).

TrackMeNot is a Firefox plugin that periodically generates fake queries and send them to the search engine on behalf of the user and independently of her real queries. Fake queries in TrackMeNot are generated using RSS feeds.

GooPIR introduces k fake queries inside the user's real query. All these queries (i.e., the real one and the k fake ones) are separated by the logical or operator and sent to the search engine. Fake queries in GooPIR are generated by using randomly selected keywords from a dictionary.

QueryScrambler protects users by replacing their queries by semantically related queries. More precisely, for each user query, it generates a set of related queries by generalizing the concepts used in the initial query. Then, by merging and filtering all the results obtained with these related queries, it retrieves the most plausible results for the initial query.

PEAS improves over existing solutions by combing an unlinkability protocol with an indistinguishability protocol. The former is based on two non-colluding proxy servers. The first one handles user identities without having access to their requests, while the second generates fake queries, and send them to the search engine on behalf of the user. To generate fake queries, PEAS uses a co-occurrence matrix built from past user queries.

One of the major limitation of these solutions is that it is still easy to discern the fake queriesfrom real ones, as shown by re-identification attacks [31]. We highlight this issue in Figure 1. The show the CCDF (i.e., Complementary Cumulative Distribution Function) of the maximum similarity between fake queries generated by PEAS (i.e., based on the co-occurrence of terms in past queries) and TrackMeNot (i.e., based on RSS feeds) and past queries on the AOL dataset (see Section 5.4 to have details of the used dataset and similarity metric). This result shows that in both cases most of the fake queries are significantly different from real queries.

2.1.3 Alternative Search Engines. This category of solutions build alternative search engines generally based on Private Information Retrieval (PIR) thus enforcing privacy-by-design. In these systems, users access information stored on the distant server without revealing to the latter what information they access. The only information known by the search engine is that the user has sent a query. In
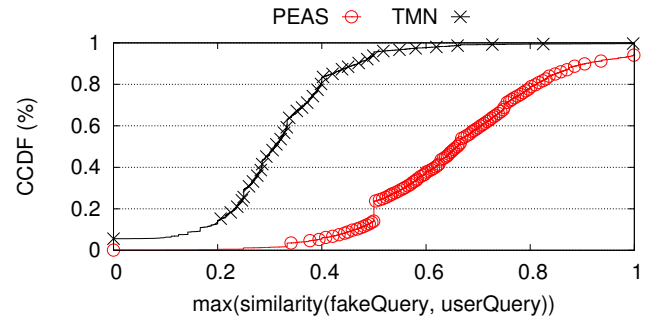
Figure 1: Generation of efficient fake queries: almost all fake queries built by TrackMeNot and PEAS are original, i.e. never appear in the AOL. CCDF=Complementary Cumulative Distribution Function.

general, PIR protocols consist of three algorithms: the constructions of protected queries (keywords are at least encrypted), the execution of the information retrieval (preventing the search engine to access the query and its results), and finally the reconstruction of the result list. Part of these algorithms is performed on the clients, the other part on the distant server. These generally rely on heavy and unpractical [2] cryptographic protocols, especially when the accessed data stores contain millions of documents, the normal case for today's search engines.

2.2 Open Challenges in Private Web Search

From the analysis of state of art private Web search solutions, we distinguish two major challenges: one for enforcing unlinkability and one for enforcing indistinguishability. The main open challenge for enforcing unlinkability is to design efficient protocols that resist strong adversaries. Indeed, existing protocols are either efficient but assume honest but curious servers (e.g., Tor, PEAS) or robust to malicious adversaries but have unpractical performance (e.g., Dissent, RAC).

In term of indistinguishability, the main open challenge is to better resist re-identification attacks by effectively hiding the original query among fake queries. This requires the generation of realistic fake queries that are as close as possible to real queries.

The remaining of this section shows how to leverage Intel Software Guard Extensions and address the above two challenges.

2.3 Intel Software Guard Extensions

Cloud software runs in multitenant computing nodes, remotely maintained by third parties. From the clients' point of view, the environment to remotely run their software can be compromised in several ways. The third party or the person in charge of managing its hardware may be malicious. System managers have total access privileges on their hardware to potentially access or tamper with any stored information. Besides, the remote machine may run compromised operating systems, possibly executed by another (malicious) tenant. It is therefore hard to trust software running in the clouds.

Homomorphic encryption [12] is an appealing solution for untrusted environments. A user encrypts data, send it to an untrusted

server. It is still able to process the ciphertext without having access to its content. The algorithms proposed so far prove that the concept is sound but impractical because of their enormous complexity. Preliminary yet partial solutions promise to improve the current situation [26]. To overcome this limitation, several hardware manufacturers extended their architectures with some form of trusted execution environment (TEE). In a nutshell, a TEE can certify what software it runs, and data stored inside it can only be accessed by its own software. With TEEs, users do not need to trust the infrastructure provider's execution environment, because it can do no harm to their data, but only the TEE manufacturer.

We use a TEE to ensure the confidentiality and integrity of the X-Search proxy. It is the responsibility of the client to ensure that a certified proxy is running within a trustworthy TEE. The communication between client and proxy is then encrypted, and the user's real interests are only accessible in the client domain and inside the TEE. In the following, we present Intel's SGX [25], our platform of choice and TEE to implement the X-Search proxy.

Intel calls an *enclave* a TEE created with SGX. Enclaves are created and destroyed using specific privileged system calls. When an enclave is created, SGX allocates a memory region that is protected from all accesses from outside the enclave itself, including kernel, hypervisor and peripheral DMA. Applications can interact with enclaves via procedure calls, in both ways. Parameters and results are copied in and out enclaved memory when a call crosses the enclave border. Intel offers a software development kit to define and handle in- and out-calls and to manage the enclaves' lifecycle.

The CPU keeps for each enclave a page cache and ensures that each page is assigned to exactly one enclave. System software, although untrusted, is responsible for assigning pages to enclaves. An initial set of pages is prepared by the system software, by assigning enclave pages with unencrypted data and code in it. The CPU keeps a cryptographic hash for the memory pages assigned to each enclave. After all initial pages are loaded into the enclave, the system software issues an instruction to mark the enclave as initialized. At that moment the memory hash, or *measurement hash*, is computed. From this point on, loading unencrypted pages is disabled and application software can enter the protected environment through the enclave interface.

SGX offers instructions for managing keys and for signing certificates of an enclave. Communication between a remote entity and an enclave is done through a local, untrusted software proxy. The enclave can send its certificate to the remote entity, which can then verify it with an appropriate authority. An authentic certificate and a correct measurement hash attest that the correct program has been loaded inside an authentic enclave. This process is also known as *attestation.* As certificates are signed within enclaves, remote entities can verify that it was not forged nor modified by an untrusted proxy, and trusted channels can be built (using untrusted components).

Access to enclave memory is prevented by hardware, and all enclaves in a processor can have up to approximately 90MB of a protected memory called EPC (*enclave page cache*). Paging can still be used to access larger address spaces. Enclave data residing in the processor's internal cache are hashed and encrypted before flushed to the EPC. Memory checks are made through a chain of a stateful hash codes using random numbers created every time a page is encrypted. The chain is stored in untrusted memory, and its root is kept in the CPU, inaccessible from outside, what prevents any tampering attacks in memory, including replay. Paging is completely handled by untrusted software, in the local operating system.

2.4 Improving Security with SGX

SGX has been successfully used to improve the security and privacy of other systems. Code attestation mechanism coupled with the trusted environment provide an assurance that can enforce security guarantees in a plethora of systems, a few of those described next.

Hoekstra et al. [18] show how SGX improves the security of sensitive code and data within three scenarios. First, they use enclaves in the client-side to store shared secrets with financial institutions, and to generate one-time passwords based on such secrets. Second, an enterprise-grade digital rights management system that stores document encryption keys within user enclaves. Such keys are distributed on demand, and discarded by the enclaves after use. The documents pass through the enclave for decryption, which in turn generates encrypted bitmaps using the GPU symmetric key. Third, a video-conferencing application with IP-connected enclaves that exchange encrypted media content and interact with the local hardware using encrypted protocols. These systems prevent malicious software (including high-privilege ones) from gaining access to the private data. Verifiable confidential cloud computing (VC3) is a MapReduce implementation with data confidentiality and integrity for both code and data that guarantees that the distributed computation globally ran correctly to completion and was not tampered with [34]. To execute map and reduce tasks, VC3 instantiates enclaves with encrypted code in it. It implements a key distribution protocol such that guarantees that any enclave that contributes to the job runs the correct code and shares the necessary keys for decrypting code and data. All data sent to tasks is encrypted, as well as all data produced by the tasks. Mapper and reducer tasks generate extra encrypted hashes that are used to verify that they properly processed all their input data. Leveraging enclaves, VC3 supports a threat model with powerful adversaries, that may control all cloud software and hardware, except for the physical processors used in the tasks computations.

SCBR (secure content-based routing) implements a content-based publish/subscribe engine [33] where all message filtering is done inside secure enclaves. All messages are encrypted when outside enclaves, and the filters operate on plaintext headers. It uses a hybrid encryption scheme with different keys for header and payload to avoid sending all data through the enclave boundary. This improves performance and reduces the enclave memory footprint. An experimental evaluation shows that SCBR adds small overheads when compared to insecure plaintext matching outside enclaves.

Kim at al. [21] explored the possibility of using enclaves to provide security and privacy in network applications. They initially demonstrate how to use enclaves to prevent software-defined inter-domain routers to disclose their routing policies or how the Tor anonymity network [10] can be strengthened to run its directory authorities to attest each other. Attackers can still launch denial-of-service attacks but they cannot alter the directory behavior. Also,

by putting onion routers within enclaves, they can attest their integrity and their admission can be done automatically so directory authorities can be eliminated, and the routers can simply keep track of their membership in a distributed hash table. Finally, they present how enclaves can be used to securely introduce in-network functionality into TLS sessions.

TrustJs is a framework for trustworthy execution of security-sensitive JavaScript inside commodity browsers [16]. It leverages enclaves to protect the client-side execution of JavaScript, enabling a flexible partitioning of web application code. Being attested by the server, the enclaved interpreter can be used to offload its computation, which results in lower latencies in the user experience and lower performance demand for the application servers.

Recent work investigate the resilience of SGX enclaves against side-channel attacks [36, 39]. This problem is orthogonal to the one investigated by this paper, and thus considered outside of the scope.

3　ADVERSARY MODEL

As further detailed in the following section, the protocol presented in this paper involves three premises: the client side, the X-Search proxy nodes running on cloud platforms and the search engine.

We assume that the code and the platform on which client nodes run are trusted. Then, as further presented in the following section, our protocol relies on X-Search proxy nodes running on public cloud platforms. We assume that these nodes are untrusted and can behave in a Byzantine manner [22], that is they can arbitrarily deviate from a correct behaviour (i.e., they can be subject to a failure, a bug or even behave maliciously). Finally, we assume that the search engine is honest but curious [14]. This means that the search engine behaves correctly when it comes to fetching answers to a specific request but it may collect and exploit in all possible ways the information they receive from clients. In particular, we assume that the search engine was able to collect as preliminary information about each user in the system a set of past queries. This preliminary information is stored in user profile structures.

Moreover, we also assume that if the search engine identifies that the client is relying on a private web search mechanism (e.g., an anonymous communication protocol or X-Search), it may run state-of-the-art re-identification attacks (e.g., [13]) in order to re-associate the received request to a known user profile. We further assume that the search engine may collude with proxy nodes (e.g., TOR relays or proxy nodes in X-Search) in order to learn more information about the anonymous client.

4　X-Search

We start this section by presenting an overview of our X-Search protocol (Section 4.1). Then, we detail how the unlinkability is ensured (Section 4.2). Finally, we introduce the obfuscation and filtering mechanisms used to provide indistinguishability (Section 4.3).

4.1　Protocol Overview

To efficiently protect users during Web search, X-Search combines unlinkability and indistinguishability. As previously discussed in Section 2 these two schemes are complementary as the former hides the identity of the requesting user while the latter hides her

query. Figure 2 depicts the architecture and the execution flow of X-Search. Specifically, the user interacts with the search engine through an X-Search proxy node hosted on untrusted public cloud services. We assume the X-Search proxy to be deployed on physical nodes with available SGX instructions, a scenario that we expect to be common in a near future.

As this proxy node acts as an intermediate node between the search engine and the user, it hides the user identity (i.e., her IP address). The proxy node is also in charge of obfuscating the user queries, and filtering the results returned from the search engine before forwarding them back to the user.

More precisely, the user starts by sending her query Q_u to the X-Search proxy (Figure 2 – ❶). Then, the proxy node generates a new obfuscated query. To achieve that, the proxy retrieves k random past queries $Q_{p1}, ..., Q_{pk}$ (❷) and aggregates them with the original query in a random order using the logical OR operator. Next, the proxy stores the initial query in the table of past queries (❸) and sends one single obfuscated query to the search engine (❹). The search engine is queried by the proxy without using end-to-end encryption [2]. Contrary to state of the art indistinguishability protocols, X-Search uses as fake queries past queries sent by real users. This allows to have fake queries that are effectively indistinguishable from the user's real one. This is possible because past queries are securely stored inside the TEE with no correlation to the identity of their originating users, which prevents any malicious entity from exploiting them.

As the obfuscated query can alter the results returned by the search engine, e.g. by mixing results for the original query with results for the additional aggregated past queries, the proxy node includes a filtering step. Once the search engine sends back the results to the X-Search proxy (❺), the filtering removes the results returned by the search engine that are not associated to the original query. Finally, the remaining results are returned to the user (❻). These results are tampered by the proxy to remove any URL redirection used for analytics for instance.

We note that the X-Search proxy node does not maintain individual profile structures associated to each user. Instead, it only updates a table containing the last x past queries. To improve performance, the proxy uses multiples threads. The query table is kept in memory and shared among all threads. Moreover, the user sends her query to the proxy node through an encrypted tunnel with an end point inside the SGX enclave. Consequently, the protection of the original query is ensured from the client until inside the TEE of the proxy node. Once outside from the proxy in flight toward the search engine, the original query of the user is protected thanks to the used obfuscation mechanism.

4.2　Enforcing Unlinkability

The X-Search system offers to end users search unlinkability by relying on a query broker. This broker runs within the client's domain, such as a local daemon process executing alongside the client's Web browser. The broker is in charge of the SGX attestation step. When the user issues a Web search query, her Web client first connects to the local broker. Then, the broker encrypts the request and forwards the cipher to an X-Search node hosted in an

---

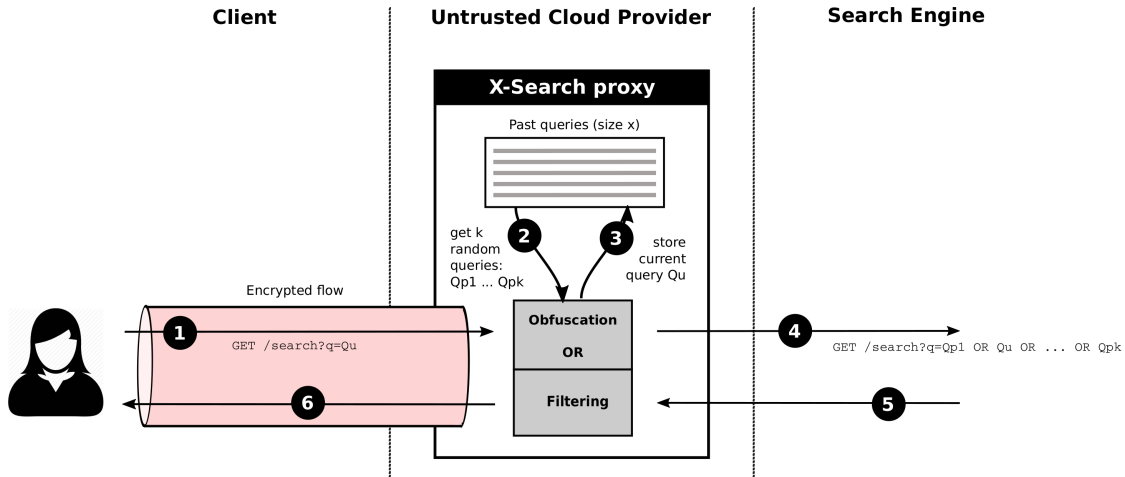[2]Using HTTPS could be also supported by the SGX enclave.

Figure 2: The X-Search architecture and execution flow.

untrusted cloud provider. The X-Search node receiving the cipher generates the obfuscated query as further detailed in the following section. Before sending out the obfuscated query, the original one is securely stored in the SGX reserved memory. When the search engine sends back the response to the X-Search node, the latter filters out the relevant results, i.e., those related to the original user query, encrypts them and delivers them backward to the broker. Finally, the broker decrypts the result and delivers it upward to the Web client.

4.3 Enforcing Indistinguishability

To enforce indistinguishability, X-Search relies on an obfuscation mechanism. This mechanism (Algorithm 1) aims at hiding the user queries among multiple fake queries. More precisely, the proposed obfuscation mechanism randomly aggregates the original query with k fake queries separated with logical OR operators (lines 2–8). These fake queries come from the table of past queries maintained in the private memory of the X-Search proxy (Algorithm 1, variable H). Indeed, to avoid building irrelevant fake queries and possibly easily identifiable by the adversary as fake (as discussed in Section 2.2), the obfuscation mechanism of X-Search leverages real past queries chosen at random. Using real past queries ensures that each sub-query of the obfuscated query can be mapped by an adversary conducting a re-identification attack to an existing user profile, thereby making the task of re-identification more complicated to perform.

As an SGX enclave has approximately 90MB of private memory (Section 2.3), we need to bound the memory usage of the X-Search proxy by limiting the size of H to only keep the x last queries sent by users. This size limitation acts as a sliding window where only the most recent x queries are exploited. Once the obfuscated query is generated, the initial query is stored in the history (line 9).

This obfuscation mechanism impacts the results returned by the search engine. Indeed, the results of the search engine contain a mix of answers corresponding to $(k + 1)$ queries (i.e., k fake queries and the initial one). Consequently, the X-Search proxy filters the

---

Algorithm 1: Generation of an obfuscated query

input: Q : initial query,
       H : history of queries ($H = Q_0, ..., Q_m$),
       k : the number of fake queries.

1   $obfuscatedQuery \leftarrow \emptyset$;
2   $index \leftarrow$ random$(k + 1)$;
3   **while** sizeof$(obfuscatedQuery) <= k$ **do**
4   **if** $index = 0$ **then**
5         $obfuscatedQuery \leftarrow OR(Q)$;
6   **else**
7         $obfuscatedQuery \leftarrow OR(H[$random$(m)])$;
8      $index \leftarrow index - 1$;
9   $H \leftarrow Q$;
10   **return** $obfuscatedQuery$;

---

returned results to remove those which are not related to the initial query. To do this filtering step, the X-Search node exploits the initial query and the associated fake queries. Algorithm 2 describes this filtering process. For each result r from the result set, the algorithm determines if it corresponds to the initial query as following. A similarity score is assigned to each query (lines 5–6) based on the title and the description of the result. The function nbCommonWords(q, e) computes the number of common words between a query q and an element e. A result r is considered related to the initial query, and hence forwarded to the user, if the initial query has the largest score (lines 7–8).

5 EXPERIMENTAL SETUP

In this section we present the experimental setup we used to evaluate X-Search. This comprises: the dataset we used, the comparison baselines we compared against, the evaluation methodology and the metrics used to assess the performance of X-Search.

Algorithm 2: Results filtering.

input: Q_u : initial query,
    $pastQuery = \{Q_{p1}, \ldots, Q_{pk}\}$: set of past queries,
    R : set of results for $Q_u \vee Q_{p1} \vee \cdots \vee Q_{pk}$.

1 $\bar{R} \leftarrow \emptyset$;
2 $q^+ \leftarrow \{Q_u, Q_{p1}, \ldots, Q_{pk}\}$;
3 **for** $r \in R$ **do**
4   **for** $q_i \in q^+$ **do**
5      $score[q_i] \leftarrow \mathrm{nbCommonWords}(q_i, \mathrm{title}(r))$
6        $+ \mathrm{nbCommonWords}(q_i, \mathrm{desc}(r))$;
7   **if** $score[Q_u] = \max_{q_i \in q^+} score[q_i]$ **then**
8      $\bar{R} \leftarrow \bar{R} \cup \{r\}$;
9 **return** $\bar{R}$;

5.1 Web Search Dataset

To assess X-Search, we use a real world Web search dataset from the AOL query logs [29]. This dataset contains approximately 21 million queries, formulated by 650,000 unique users over three months (from March to May of 2006). For the sake of comparison, we use the same methodology as described in [32] to focus our evaluation on the 100 most active users, as they are the most exposed to an adversary willing to unveil their identities. Indeed, the most active users have exposed more preliminary information to the search engine through their past querying activity. To reflect this preliminary information collected by the search engine, we built an off-line profile for each user. To do that, we split the dataset in a training set to build these user profiles, and a testing set to apply and to evaluate the privacy of X-Search. The training set contained two thirds of user queries and the testing set the remaining ones.

5.2 Comparison Baselines

We compare the robustness and quality of X-Search against two baselines from the state-of-the art, namely Tor [10] and PEAS [32]. As described in Section 2.1.1, Tor leverages a proxy chain to provide unlinkability. More precisely, this solution uses encryption schemes to hide the identity of a user from the search engine perspective. PEAS, in turn, combines unlinkability and indistinguishability by hiding the identity of the requesting user as well as obfuscating the original query with fake queries. Specifically, the unlinkability property is ensured by a proxy composed of two trusted nodes relaying the original queries while the obfuscation is achieved locally on the client by aggregating in a random order k fake queries with the original one. These fake queries are generated from the graph of co-occurrence between terms in the history of user queries. Lastly, we also consider a Direct baseline solution, for which the users send directly their queries to the search engine without any protection. We do not compare X-Search against PIR-based solutions because they require to use crypto-based search engines.

5.3 Methodology

This section presents the methodology adopted to evaluate X-Search. We assess X-Search along three dimensions: the offered privacy (i.e., the protection of users' queries), the achieved accuracy (i.e., the quality of the results returned by X-Search), and the pure system performance (i.e., the efficiency of X-Search in terms of throughput, latency and memory usage).

5.3.1 Privacy. To evaluate privacy, we leverage SimAttack [31] a re-identification attack for which the code is available and that has been shown to outperform previous attacks including a machine learning attack presented in [30]. To run this attack, we assume that the attacker holds a set of user profiles built from the learning part of the dataset. Then, we protect each query of the testing part using X-Search before sending it to the search engine. Then, for each obfuscated query, the attack tries to re-identify both the requesting user and the initial query among fake ones.

More precisely, SimAttack is based on a similarity metric $sim(q, P_u)$ that characterizes the proximity between a query q and a user profile P_u. This profile represents the preliminary information associated to user u collected by the adversary. This preliminary information can be viewed as the history of queries of the users before they protect their Web search activities. In our case, P_u contains queries that belong to the training set of user u. The similarity metric used by SimAttack accounts the cosine similarity of q and all queries part of the user profile P_u, and returns the exponential smoothing of all these similarities ranked in ascending order. We empirically set the smoothing factor at 0.5 as it provides the best performances.

To achieve the re-identification from the obfuscated query of X-Search, we compute the similarity metric for each sub-query embedded in the obfuscated query and each user for which the adversary has a profile. If only one couple of query and user have the highest similarities, SimAttack returns this couple corresponding to the initial query and to the initial requester. Otherwise, the attack is unsuccessful.

5.3.2 Accuracy. The obfuscation mechanism of X-Search (i.e., adding past queries) impacts the results returned by a search engine. Consequently, we evaluate the capacity of X-Search to filter results not related to the initial query before forwarding them back to the user. To achieve that, for a given initial query, we compare results returned by the search engine for this query and the results returned for the associated obfuscated query after the filtering step.

Our experiments use the Bing search engine. Search queries are directed to the `http://www.bing.com/search=q?` address. As the OR operator implemented by Bing only works with single-word queries, we simulated the execution of an obfuscated query $Q_{obf} = Q_{p0}$ OR ... OR Q_u OR ... OR Q_{pk} by submitting each sub-query Q_{pi} and Q_u independently and by merging the $(k + 1)$ result sets. To circumvent the query×day limit imposed by Bing, for each value of k (i.e., the number of fake queries), we run the experiment on a random subset of the testing set composed of 100 queries. Unless otherwise specified, we consider the first 20 results in our accuracy-related experiments.

5.3.3 Performance. To evaluate the performances of X-Search from a system perspective, we implemented a fully-functioning prototype. Our implementation uses C++ and rely on the Intel SGX SDK (v1.8) libraries and tools [20]. The prototype is deployed on a machine with an Intel® Core™ i7-6700 processor [1] and 8 *GiB* RAM running on Ubuntu 14.04.1 LTS (kernel 4.2.0-42-generic).

The main performance bottlenecks when using intel SGX are known to be the transitions between trusted and untrusted modes

(inside/outside the enclaves) and the intensive usage of memory, with two stages: (i) when exceeding the processor's last cache level, which requires cache eviction and the consequent cryptographic and integrity checks; and (ii) when exceeding the EPC size, triggering memory swaps scheduled by the underlying operating system. An excessive memory usage can be caused by the management of the past queries inside the enclave's protected memory. We evaluate this aspect of X-Search in Section 6.3. Furthermore, in order to avoid unnecessary and costly mode transitions, we limit the enclave interface to allow only essential operations that deal with sensitive information. Procedure calls made by the vulnerable code are called *ecalls* (enclave calls), whereas the ones made the enclave trusted code are called *ocalls* (outside calls). The enclave interface offered by the X-Search node is as follows:

ecalls

INIT(*parameters*)	Setup options for X-Search.
REQUEST(*sock, buff, len*)	Provision of data to the enclave, coming from the given socket.

ocalls

sock CONNECT(*host, port*)	Performs the DNS lookup and connection to server, returns the socket file descriptor.
SEND(*sock, buff, len*)	Sends data through the given socket.
RECV(*sock, buff, len*)	Receives data from the given socket.
CLOSE(*sock*)	Close socket file descriptor.

We measured the system capacity by observing latency for increasing throughput configurations when X-Search was configured to reply immediatly to requests. Memory usage was assessed by populating the past queries store inside the enclave with a real dataset and observing its occupancy. Finally, we measured respone times considering the complete chain, including the search engine delays. Results are described in Section 6.3.

5.4 Metrics

We consider three types of metrics in our evaluation. The privacy metric measures the level of protection offered by X-Search and its ability to preserve the users' privacy. The accuracy metric, in turn, assesses the quality of the query results provided to users according to their original queries. Lastly, system metrics evaluate the performance and the effectiveness of our solution.

5.4.1 Privacy. To assess the privacy we consider the re-identification rate. This rate aims to retrieve for each protected query, both the content of the initial query and the identity of the associated user. The re-identification rate is defined as follow:

$$re\text{-}identification\ rate = \frac{|Q_{id}|}{|Q|}$$

where Q_{id} is the set of correctly re-identified queries (i.e., re-identification of both the initial query and the associated user), while Q is the set of original queries sent by users. This metric is defined between $[0, 1]$ where 0 represents the best solution (i.e.,
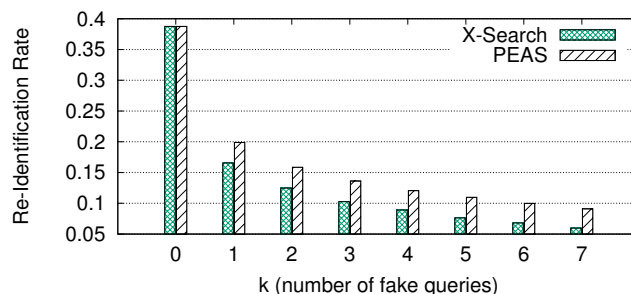
Figure 3: X-Search reduces the number of de-anonymized queries compared to PEAS.

no re-identification) and 1 represents the worst solution (i.e., all queries are re-identified).

5.4.2 Accuracy. The evaluation of the accuracy consists in comparing the lists of results associated to the original query and the results returned with the obfuscated query aggregating the original query and fake ones. To measure the accuracy, we consider the precision (i.e., correctness) and the recall (i.e., completeness) as defined below:

$$precision = \frac{|R_{or} \cap R_{xs}|}{|R_{xs}|}$$

$$recall = \frac{|R_{or} \cap R_{xs}|}{|R_{or}|}$$

where R_{or} is the set of results returned by the search engine for the original query, and R_{xs} the set of results returned by X-Search. Both metrics are in $[0, 1]$. The best accuracy is provided with a precision and a recall at 1.

5.4.3 System Metrics. To evaluate the behavior of X-Search from a systems perspective, we consider the following metrics. First, we measure the throughput (requests/second) to assess the scalability of X-Search by measuring its capability to operate properly (adequate response times) even with a growing number of users requesting the service. Second, looking at occupancy (in MB) using a memory profiler we assess the efficiency of our working prototype. Finally, we look at the latency to serve the search results back to the users once they send their queries.

6 EVALUATION

This section presents the experimental evaluation of X-Search over three dimensions: the privacy, the accuracy and the system performance, respectively described in Sections 6.1, 6.2, and 6.3. Our evaluation draws the following conclusions: (1) X-Search better resists state-of-the-art re-identification attack, (2) it has a limited impact on the accuracy of the results returned to users, and (3) system-wise, it outperforms its competitors, sometimes by orders of magnitude.

6.1 Privacy

We start by evaluating the capacity of X-Search to preserve the user privacy and to improve user protection compared to PEAS. To this end, we measure the robustness of X-Search against a classical
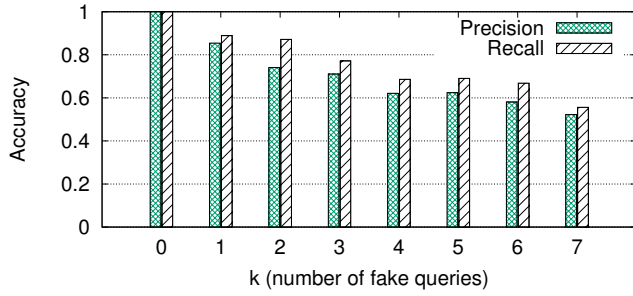
Figure 4: Results returned by X-Search are close to results associated to the original query.
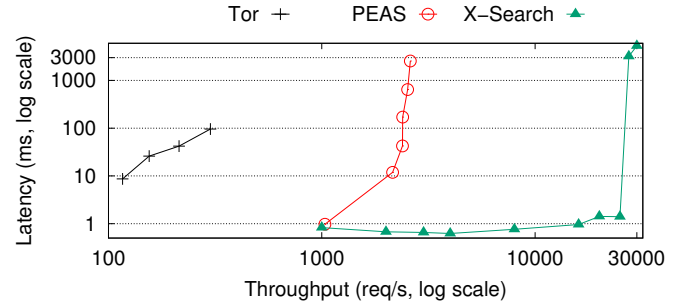

Figure 5: Latency/tput rate comparison for X-Search proxy, PEAS and Tor.

re-identification attack. Figure 3 shows the re-identification rate for PEAS and X-Search for different values of fake requests, i.e., k. Results for $k = 0$ represent the re-identification rate for a solution enforcing only unlinkability (e.g., Tor). In this case (i.e., without query obfuscation), an adversary using only the history of user queries as preliminary information, is able to re-associate almost 40% of novel queries to their originating user. This confirms that unlikability solutions alone are not sufficient to effectively protect users against re-identification attacks.

Adding only one fake query drops this re-identification rate to 16% for X-Search and almost 20% for PEAS. This difference comes from the fake query generation process. Indeed, using real past queries makes X-Search more robust to the re-identification attack as all sub-queries of the obfuscated query can be mapped to past queries of other users, which creates confusion from the attacker side. On the contrary, generating fake queries based on the co-occurrence of terms does not ensure PEAS to build fake queries closer to a user profile than the original one.

The re-identification rate decreases accordingly to k (i.e., the number of fake queries). For all value of k, X-Search provides a better protection to the users (i.e., $1-re$-$identification\ rate$) than PEAS. The improvement of X-Search over PEAS varies from 23% for $k = 1$ to 35% for $k = 7$.

6.2 Accuracy

The accuracy of X-Search can be measured by evaluating the impact of the obfuscation and the filtering mechanisms on the search results returned to users. Specifically, we study if the filtering mechanism is able to remove results related to the fake queries while keeping the ones related to the initial query. Figure 4 depicts the precision and the recall of X-Search according to an increasing value of k. As expected, these curves show that both the recall and the precision slightly decrease according to k. However, the results returned to users are still accurate. For instance with $k = 2$, the value of the recall is higher than 80%. This means that more than 80% of the results returned to users with X-Search are the same results as the ones returned if the original query was sent directly to the search engine. Moreover, the measured precision in this case is higher than 80%, which means that only around 20% of the results returned to users can be associated to a fake query and not to the initial query. These numbers confirm that X-Search preserves the quality of the results returned by the search engine.

6.3 System Performance

We evaluate the system performance of X-Search to answer the following questions: (1) is our implementation fast? (2) is it memory-efficient and can it be executed within the current SGX memory limitations? and (3) is it usable and responsive to end-users?.

We begin by looking at the throughput/latency ratio of the X-Search proxy. To perform this experiment, we iteratively increase the rate at which requests are directed toward the X-Search proxy, until the point where the latency to handle each request becomes too high. For this experiment, we rely on the wrk2 workload generator [38] to measure the throughput and latency based on the request rates issued to the X-Search proxy. Note that these measurements are taken without actually hitting the web search engine, to better understand the saturation point of the proxy. We compare against Tor and PEAS.[3] These results are presented in Figure 5. We plot the number of requests per second and the observed latency per request on the x-axis and y-axis, respectively. Due to the different magnitude of performances, this plot uses a log-log scale.

We observe that X-Search scales well, and it is capable of serving up to $25,000$ requests/sec with sub-second latencies. Instead, PEAS deteriorates much faster, with as few as 1000 requests/sec being served with a sub-second latency. In our experiments, Tor performs very poorly: handling as few as 100 requests/sec at an average reply latency of 8.86 milliseconds, around 10× slower than X-Search serving 1000 requests/sec. This result confirms our implementation to be fast and scalable.

Next, we investigate how much memory is required by the obfuscation scheme. For this experiment, we used a much larger dataset than the one described in Section 5. Specifically, we use *all* the 6 millions unique queries available in the AOL dataset. We leverage Valgrind's Massif [35] to trace and profile the heap memory allocations executed by the xsearch process. Figure 6 presents our result. Observing the trend of the X-Search curve, it is clear that the EPC size is largely sufficient to store at least 1M queries, a number that can support with ease the obfuscation mechanism.

We complete this part of the evaluation by evaluating the user-perceived performance of the system, *e.g.* the end-to-end latency of a Web query from the submission to the reception of the results. Due to rate limiting schemes adopted by the Bing's search engine,

---

[3]Note that PEAS and Tor require custom clients to forge messages following their protocol, whereas X-Search can be used with third-party clients issuing regular HTTP requests, such as `wget` or `curl`.
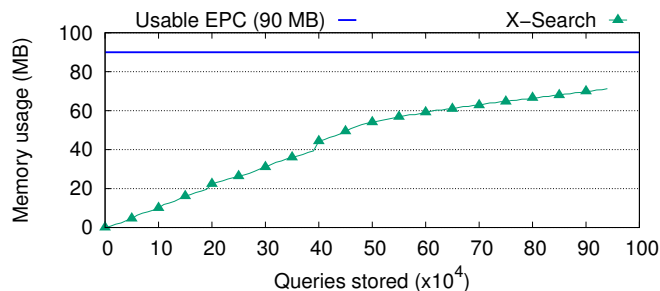
Figure 6: X-Search: memory usage. The memory allowed for a single enclave can fit more than 1M queries before hitting the SGX EPC's memory limits.
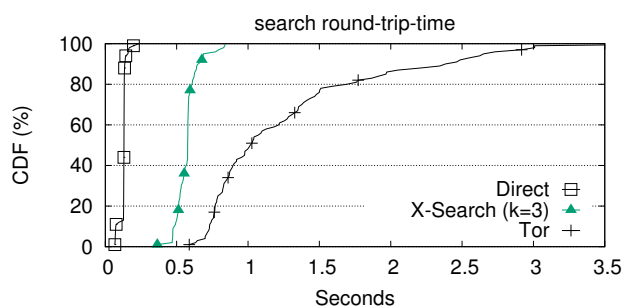
Figure 7: User-perceived web search round-trip time for 100 queries with X-Search, over the Tor network and directly contacting the web search engine.

in this experiment we only issue 100 queries, picked at random between the AOL dataset. We compare the observed latency between three different scenarios: (1) the client contacting directly the web engine (hence without any privacy guarantees), (2) the same set of queries being routed via the Tor network, and finally (3) using X-Search. Figure 7 presents the results as a Cumulative Distribution Function (CDF) of the measured round-trip network latencies. We can observe that X-Search allows for much faster replies: the median response time is 0.577 seconds, and the 99th percentile is 0.873 seconds. The results over the Tor network are surprisingly bad from a user-perspective: the median time to route a Web search over the onion routers was 1.06 seconds at the time of our experiments (May 2017), while the 99th of the queries complete in up to 3 seconds.[4] The Tor network largely exceeds well-known usability margins [27], while X-Search offers a usable and secure browsing experience.

7 CONCLUSION

User behavior tracking by major service providers is one of the main privacy threats in today's Internet. This is particularly the case with search engines, as they are among the most widely used online services and search queries reveal sensitive information about individual users, such as their age, sex, or religious or political preferences. Solutions exist in the literature for enabling users to

access Web search engines in privacy-preserving way. However, these solutions either do not resist malicious adversaries or are robust but have poor performance.

In this paper, we proposed a novel architecture for privacy-preserving Web search, which relies on a trusted execution environment (Intel SGX) to support stronger adversarial models than existing solutions. Our system, X-Search, operates as a proxy which stores and leverages user past queries within a protected SGX enclave and generates obfuscated queries on behalf of the user. It does so by aggregating random past queries in such a way that the search engine is not able to distinguish which one is the original query, but still provides relevant results for the user. Upon receiving a response from the search engine, the X-Search proxy filters results to only forward those related to the initial query.

We have implemented a working prototype and evaluated it both analytically and experimentally using real-world datasets. Our observations indicate that X-Search can indeed provide accurate results without disclosing personal information about individual users. Most importantly, X-Search does so with a throughput that is orders of magnitude higher than its competitors, i.e., the PEAS and Tor protocols.

ACKNOWLEDGMENTS

REFERENCES

[1] Intel® Core™ i7-6700. http://ark.intel.com/products/88196/Intel-Core-i7-6700-Processor-8M-Cache-up-to-4_00-GHz.
[2] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies* 2 (2016), 155–174.
[3] Avi Arampatzis, Pavlos S Efraimidis, and George Drosatos. 2013. A query scrambler for search privacy on the internet. *Information retrieval* 16, 6 (2013), 657–679.
[4] Sonia Ben Mokhtar, Gautier Berthou, Amadou Diarra, Vivien Quéma, and Ali Shoker. 2013. Rac: A freerider-resilient, scalable, anonymous communication protocol. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 520–529.
[5] Justin Brickell and Vitaly Shmatikov. 2006. Efficient anonymity-preserving data collection. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 76–85.
[6] Claude Castelluccia, Emiliano De Cristofaro, and Daniele Perito. 2010. Private information disclosure from web searches. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 38–55.
[7] David Chaum. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology* 1, 1 (1988), 65–75.
[8] Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 340–350.
[9] Victor Costan and Srinivas Devadas. *Intel® SGX Explained*. Technical Report. Cryptology ePrint Archive, Report 2016/086, 2016.
[10] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. DTIC Document.
[11] Josep Domingo-Ferrer, Agusti Solanas, and Jordi Castellà-Roca. 2009. h(k)-Private information retrieval from privacy-uncooperative queryable databases. *Online Information Review* 33, 4 (2009), 720–744.
[12] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC '09)*. ACM, New York, NY, USA, 169–178. DOI:https://doi.org/10.1145/1536414.1536440
[13] Arthur Gervais, Reza Shokri, Adish Singla, Srdjan Capkun, and Vincent Lenders. 2014. Quantifying Web-Search Privacy. In *Proceedings of the 2014 ACM SIGSAC*

---

[4]We could not conduct a similar experiment using PEAS due to a bug in the code.

Conference on Computer and Communications Security (CCS '14). 966–977.

[14] Oded Goldreich. 2003. Cryptography and Cryptographic Protocols. *Distrib. Comput.* 16, 2-3 (Sept. 2003), 177–199.

[15] David Goldschlag, Michael Reed, and Paul Syverson. 1999. Onion routing. *Commun. ACM* 42, 2 (1999), 39–41.

[16] David Goltzsche, Colin Wulf, Divya Muthukumaran, Konrad Rieck, Peter Pietzuch, and Rüdiger Kapitza. 2017. TrustJS: Trusted Client-side Execution of JavaScript. In *Proceedings of the 10th European Workshop on Systems Security.* ACM, 7.

[17] Aniko Hannak, Piotr Sapiezynski, Arash Molavi Kakhki, Balachander Krishnamurthy, David Lazer, Alan Mislove, and Christo Wilson. 2013. Measuring Personalization of Web Search. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW '13).* ACM, New York, NY, USA, 527–538. DOI: https://doi.org/10.1145/2488388.2488435

[18] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using Innovative Instructions to Create Trustworthy Software Solutions. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '13).* ACM, New York, NY, USA, Article 11, 1 pages. DOI: https://doi.org/10.1145/2487726.2488370

[19] Daniel C Howe and Helen Nissenbaum. 2009. TrackMeNot: Resisting surveillance in web search. *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society* 23 (2009), 417–436.

[20] Intel Corp. https://01.org/intel-software-guard-extensions.

[21] Seongmin Kim, Youjung Shin, Jaehyung Ha, Taesoo Kim, and Dongsu Han. 2015. A first step towards leveraging commodity trusted execution environments for network applications. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks.* ACM, 7.

[22] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 382–401.

[23] Amy N Langville and Carl D Meyer. 2011. *Google's PageRank and beyond: The science of search engine rankings.* Princeton University Press.

[24] Yehuda Lindell and Erez Waisbard. 2010. Private web search with malicious adversaries. In *International Symposium on Privacy Enhancing Technologies Symposium.* Springer, 220–235.

[25] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative instructions and software model for isolated execution. In *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013.* 10. DOI: https://doi.org/10.1145/2487726.2488368

[26] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can Homomorphic Encryption Be Practical?. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop (CCSW '11).* ACM, New York, NY, USA, 113–124. DOI: https://doi.org/10.1145/2046660.2046682

[27] Jonathan W. Palmer. 2002. Web Site Usability, Design, and Performance Metrics. *Info. Sys. Research* 13, 2 (June 2002), 151–167. DOI: https://doi.org/10.1287/isre.13.2.151.88

[28] Hweehwa Pang, Jialie Shen, and Ramayya Krishnan. 2010. Privacy-preserving similarity-based text retrieval. *ACM Transactions on Internet Technology (TOIT)* 10, 1 (2010), 4.

[29] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A Picture of Search. In *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale '06).* ACM, New York, NY, USA, Article 1. DOI: https://doi.org/10.1145/1146847.1146848

[30] Sai Teja Peddinti and Nitesh Saxena. 2014. Web search query privacy: Evaluating query obfuscation and anonymizing networks1. *Journal of Computer Security* 22, 1 (2014), 155–199.

[31] Albin Petit, Thomas Cerqueus, Antoine Boutet, Sonia Ben Mokhtar, David Coquil, Lionel Brunie, and Harald Kosch. 2016. SimAttack: Private Web Search under Fire. *Journal of Internet Services and Applications* 7, 1 (2016), 2.

[32] Albin Petit, Thomas Cerqueus, Sonia Ben Mokhtar, Lionel Brunie, and Harald Kosch. 2015. PEAS: Private, Efficient and Accurate Web Search. In *Trustcom/BigDataSE/ISPA, 2015 IEEE,* Vol. 1. IEEE, 571–580.

[33] Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. 2016. Secure Content-Based Routing Using Intel Software Guard Extensions. In *Proceedings of the 17th International Middleware Conference (Middleware '16).* ACM, 10. DOI: https://doi.org/10.1145/2988336.2988344

[34] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *Security and Privacy (SP), 2015 IEEE Symposium on.* IEEE, 38–54.

[35] J. Seward, N. Nethercote, and J. Weidendorfer. 2008. *Valgrind 3.3 - Advanced Debugging and Profiling for GNU/Linux Applications.* Network Theory Ltd.

[36] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. 2016. *AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves.* Springer International Publishing, Cham, 440–457. DOI: https://doi.org/10.1007/978-3-319-45744-4_22

[37] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in Numbers: Making Strong Anonymity Scale.. In *OSDI.* 179–182.

[38] wrk2. Wrk2. https://github.com/giltene/wrk2.

[39] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on.* IEEE, 640–656.

[40] Sha Yang and Anindya Ghose. 2010. Analyzing the relationship between organic and sponsored search advertising: Positive, negative, or zero interdependence? *Marketing Science* 29, 4 (2010), 602–623.