Bringing Decentralized Search to Decentralized Services

Mingyu Li, Jinhao Zhu, Tianxu Zhang, Cheng Tan†, Yubin Xia, Sebastian Angel⋆, Haibo Chen
Shanghai Jiao Tong University †*Northeastern University* ⋆*University of Pennsylvania*

Abstract

Recent decentralized services such as Steemit and OpenBazaar have emerged to provide accountable social media and online e-commerce. However, despite being decentralized themselves, they still rely on a centralized or an insecure distributed search engine for end-user navigation, which makes such services vulnerable to censorship and privacy violation.

This work presents DESEARCH, the first decentralized search engine that guarantees the integrity and privacy of search results for decentralized services. DESEARCH uses trusted hardware to build a network of pipelined *lambdas*, an enclave primitive for a small piece of outsourced job. Since each lambda only has a local view, DESEARCH introduces a novel *witness* mechanism that implies the transitivity of lambdas to make this decentralized model externally verifiable. DESEARCH enables fast verification for queries and can tolerate decentralized failures. We implement a DESEARCH prototype for a decentralized social media and online market. Evaluation shows that DESEARCH can scale horizontally in a decentralized environment and support 32 million requests per day on a 234 GB real-world dataset with 82M documents.

1 Introduction

Most of today's online services—including search, social networks, and e-commerce—are centralized for reasons such as economies of scale, compatible monetization strategies, network effects, legal requirements, and technical limitations. Yet, since the birth of the Internet, there have been periods of intense interest in decentralization, including the peer-to-peer systems bonanza of the early and mid 2000s [56, 89, 106] and the current blockchain boom [38, 94, 116]. A rich set of decentralized services have appeared and are able to offer most of the functionalities that common centralized online services provide, as listed in Figure 1. Proponents of decentralization argue that centralized services often employ anti-consumer practices owing to their monopolistic positions [18, 27], and the mismatch between users' expectations and operators' incentives [43]. Further, centralized services are particularly susceptible to censorship [5, 40] (either self-imposed or coerced through technical or legal means) and collect vast amounts of user information [12, 13].

Service	Centralized	Decentralized
Currency	U.S. Dollars	Bitcoin [94]
Online Marketplace	eBay	OpenBazaar [28]
Social Media	Twitter	Steemit [39]
Video Sharing	Youtube	DTube [9]
Social Network	Facebook	Mastodon [15]
Public Storage	DropBox	IPFS [58]
Messaging	Slack	Matrix [25]
Video Conference	Zoom	Zipcall [49]
Website Hosting	WiX [46]	ZeroNet [48]
Financial Betting	Etoro [11]	Augur [2]
Supercomputing	Titan [44]	Golem [17]
Document Collaboration	Google Docs	Graphite [21]

FIGURE 1—Centralized services and decentralized alternatives.

State-of-the-art. While the idea of building fully decentralized services is alluring, developers must currently make a significant compromise: they must defer search functionality to a centralized service. For example, OpenBazaar [28] makes a strong case for a decentralized marketplace, but users must use a centralized search engine such as Bazaar Dog [45] or Duo Search [10] to discover what items are for sale in the first place. A similar compromise is made by other popular services [9, 15, 28, 39, 48]. This state of affairs is problematic because search is not an optional feature but rather a core component of these systems. Without decentralized search, the purported goals of anti-censorship is hard to attain: the search engine could trivially track users' queries, and opaquely censor particular content [4, 5, 8, 30, 40]. For example, Steemit [39] is a decentralized social media service where posts are stored on the public Steem blockchain [38], but Steemit developers can prevent users' posts from appearing on the front end [40].

Prior Proposals. Several search engines [29, 80] propose reaching consensus amongst replicas to ensure the correctness of search indexes. However, these engines rely on a central website hosted at the third party to answer queries. As a result, an end-user who visits this website has no way to validate the integrity of the displayed results, or to determine whether there are missing entries (known as "search poisoning" [32]). As an alternative, peer-to-peer-based search engines [24, 47] allow shared indexes between peers and queries can be issued to any peer (essentially implementing a distributed hash table). However, these engines do not support verifiable indexes, and allow peers to monitor clients' requests, leading to severe privacy concerns.

Challenges. Building a decentralized search engine that avoids the aforementioned shortcomings is far from trivial. First, the search engine should be able to authenticate the data source to make sure the dataset is complete and has not been fabricated by any parties. Second, the user's intention, including the query keywords and search results, should be kept private from any party. Third, the search engine should

be able to provide a proof of execution to clients that explains how the search results were generated, and why the results are complete and have integrity. Last, the search engine should have reasonable costs and scale to support many users.

DESEARCH. To address these challenges, this paper introduces DESEARCH, the first decentralized search engine that allows users to verify the correctness of their search results, and preserves the privacy of user queries. DESEARCH outsources fragments of search tasks such as crawling, indexing, aggregation, ranking, and query processing to trusted execution environments (TEEs) running on untrusted nodes that create a decentralized network, and introduces new data structures and mechanisms to meet search integrity and privacy.

First, since each executor only has a local view of the computation, DESEARCH uses *witnesses*, which are a new type of object that reflects the dataflow and establishes the correctness and completeness of results. A witness ensures that executors cannot lie about which sites they crawled, how they aggregated data, computed the index, or responded to a query. Verifying witnesses is not cheap in DESEARCH, so DESEARCH amortizes the verification cost by reusing previously checked witnesses across queries, using designated nodes—verifiers—to verify witnesses on behalf of clients.

Second, DESEARCH introduces a public storage service, called *Kanban*, which is only trusted for availability. Kanban allows nodes in the network (or "executors") to exchange intermediate information, agree on a snapshot of data in the system, manage node membership, tolerate faults, and perform data transferring and result verification. To detect rollback on Kanban data, DESEARCH summarizes an epoch-based snapshot and stores it on an append-only distributed ledger.

Finally, DESEARCH protects the privacy in the query phase with two techniques. To prevent leaks from access patterns, DESEARCH adapts an existing oblivious RAM library [99]. To resist frequency and volume side channels, DESEARCH returns the same amount of data for search queries. It does so by equalizing the lengths of all result entries. This approach does not reduce the performance or quality of the service because search engines need not display all of the content but rather a small snippet. As an analogy in the Web context, search engines like Google do not display the entirety of a Web site's content in the search result; instead, they typically display the URL of each site and a small text snippet.

We have built a prototype of DESEARCH in $5,700$ lines of C++ and $1,900$ lines of Python, and adapt it to work with Steemit (a decentralized social media service), and OpenBazaar (a decentralized e-commerce service). We deploy DESEARCH on a cluster of machines in the wide-area network that allows us to vary network latency and kill executors throughout our experiments to simulate a realistic decentralized environment. The results of our experiments show that DESEARCH scales well as more executors join the network, and can handle at least 32 million requests per day using 15 TEE-enabled machines. For each request, users can quickly verify the displayed results within 1.3 second by consulting dedicated verifiers in DESEARCH, while users themselves can verify results on their own in 20.5 minutes.

To summarize, the contributions of this paper are:

- The design of DESEARCH, the first decentralized search engine that allows any client with a TEE to join and provide search functionality for decentralized services.
- A witness mechanism that organizes verifiable proofs from short-lived executors to form a global dataflow graph. Through these witnesses, DESEARCH offers fast verification for search queries.
- A prototype of DESEARCH built for Steemit [39] and OpenBazaar [28], and an evaluation of DESEARCH's performance and scalability.

While DESEARCH enables, for the first time, scalable, verifiable and private search for existing decentralized services, it is not a viable replacement for traditional Web search engines (e.g., Google). Besides the obvious issue of scale, DESEARCH's target applications expose a single source of data (their underlying distributed ledger or proof of storage mechanism), which gives DESEARCH an anchor for its witness data structure. In contrast, the Web has no such single ledger-like mechanism, which would prevent DESEARCH from proving that all Web pages had been crawled and indexed. Nevertheless, we believe DESEARCH fills a crucial void in existing decentralized services.

2 Problem Statement

This section describes our target setting, the motivation behind our work, threat model, and potential solutions that fall short.

2.1 Decentralized Search

Consider a decentralized system setup, where many volunteers (called *executors*) together run a search engine.

Users want to perform search on *source data* using some *search algorithm*. Both the source data and the search algorithm are public and accessible to all users. For each search, a user sends *keywords* to the search algorithm and expects a *search result*—an ordered list of pointers to source data. Multiple *executors* run the search algorithm. In the context of decentralized systems, executors are owned by different and possibly adversarial entities.

After receiving the search results, users want to verify that they are correct—the results are derived from the keywords and the search algorithm on the source data. Users also want to keep their searches private. In detail, the challenge is to design a decentralized search engine that meets these goals:

- **Integrity:** the search engine should guarantee the results are generated from the desired dataset and expected algorithms. We seek two properties: (1) *Soundness*, meaning that the search process is faithfully executed, in the sense

Search Engine	Integrity		Privacy		Decentralization
	Soundness	Completness	Content	Volume	Scalability
YaCy [47]	○	○	○	○	●
Presearch [29]	●	●	○	○	●
The Graph [20]	●	●	○	○	●
IPSE [24]	●	○	○	○	●
BITE [88]	●	●	●	○	○
Rearguard [107]	●	○	●	◐	○
Oblix [91]	●	○	●	●	○
Dory [69]	●	●	●	●	○
vChain [117]	●	●	○	○	○
GEM^2-Tree [120]	●	●	○	○	●
X-Search [92]	○	○	◐	○	○
CYCLOSA [96]	○	○	○	○	●
DESEARCH	●	●	●	●	●

FIGURE 2—Comparison between prior work and DESEARCH. X-Search [92] obfuscates query keywords but does not hide them. Rearguard [107] hides the index size but not the result size. Dory [69] only considers the search index but not file retrieval.

that none of the returned results is a forgery, and their order is correct. (2) *Completeness*, meaning that no legitimate results that satisfy the search conditions are missing or hidden, which prevents (undetectable) censorship.

- **Privacy:** the search engine should prevent user keywords and search results being revealed to any third party. The system should protect the following aspects of information: (1) *Content*: the data being searched and query keywords submitted are not visible to an external observer. (2) *Volume*: any messages transferred are independent of the searched dataset and query keywords.
- **Decentralization:** A decentralized system must allow executors to freely join and leave the system. And newly joined executors must contribute meaningfully towards the system's scalability: more decentralized executors should help the search engine process more search requests.

Figure 2 shows existing decentralized or private search engines. None of them meets all three desired goals.

2.2 Motivation

Below, we describe some popular decentralized services and the problems they encounter.

Social Media. Steemit [39] is a social media platform that stores user-generated contents on the public Steem blockchain [38]. Although the raw data from the blockchain is tamper-proof, Steemit's frontend servers can manipulate the search results before delivering them to users [40]. One can think of these services as centralized curators for decentralized storage. These servers also know who searches for what, which may reveal personal interests and preferences.

Marketplaces. OpenBazaar [28] is a decentralized e-commerce marketplace built on top of a peer-to-peer network and storage. To help users search for items, OpenBazaar provides an API [6] for third-party search engines to crawl and index items. But existing search engines [3, 10, 33, 45] are opaque; they can bias results towards item listings that benefit them financially. For example, a listing owner could pay the search engine to promote its listing or hide other listings. Additionally, these engines can learn users' purchasing habits (and other information) from keywords and search histories.

In short, existing decentralized services currently lack trusted search that offers integrity and privacy. A decentralized search engine should have strict requirements on *whether* the content is faithful, *how* the results are generated, and *who* can see the final results. In Section 3, we show that DESEARCH can satisfy all of these requirements. Below, we discuss possible approaches to solve the above issue.

2.3 Some Potential Approaches

How to provide integrity and/or privacy for an execution (for example, search) has been examined broadly. We list some approaches below (see more in Section 10).

Replication such as PBFT [65] and Ethereum [116] is one approach to build systems that can guarantee execution integrity within a given number of (Byzantine) faults. However, it requires performing the computation multiple times and traditional replication protocols do not provide privacy.

Another line of work [97, 102, 112] uses cryptography—such as fully homomorphic encryption (FHE) [72], secure multi-party computation (MPC) [119], and verifiable computation (VC) [60, 95, 103]—to provide execution integrity and/or privacy. Though promising, it remains an open problem to build a system that can support a large-scale dataset like today's decentralized services.

Trusted execution environments (TEE) provide another approach to build systems that protect a sensitive execution from being tampered with or eavesdropped. However, existing TEEs either (1) have limited private memory (128 MB per node); (2) lack memory encryption [54] which makes them vulnerable to physical attacks that are realistic in a decentralized setting; or (3) are susceptible to memory tampering [53, 83]. Although newer generation of TEE [42] has expanded the enclave memory to 1TB, it relaxed its security guarantees (e.g., subject to physical replay attacks) owing to the loss of integrity tree protection. Prior distributed frameworks like VC3 [101] and Ryoan [75] address many of the above shortcomings but are not designed for a decentralized environment. Amongst related work, Dory is the most relative work which offers encrypted search over a file-sharing system. Dory on its own provides oblivious data retrieval based on given keywords, but excluding privacy protection for relevant candidatas ranking and top-k results returning, which are also the essential functions for private search.

DESEARCH uses a hybrid approach—a combination of TEEs, our epoch-based Kanban storage service, authenticated data structures (hash trees), and oblivious RAM—to address the challenges described in Section 2.1.

2.4 Threat Model

DESEARCH assumes a decentralized network where untrusted executors are operated by unknown parties, but are equipped with TEE. We assume reliable TEEs with no analog side channels [62, 111] or vulnerabilities to voltage changes [93]. We also assume the TEE manufacturers do not inject backdoors into TEEs, or shares their private keys. The remote attestation mechanism of the TEE works as intended.

In DESEARCH, executors join the system and volunteer their TEE (or they are paid or incentivized to do so, which is orthogonal). Executors can be arbitrarily malicious: they can deny services, sabotage the integrity of the inputs that go into the TEE and the outputs that come out of the TEE. They can corrupt data in unprotected memory, or load software that is not DESEARCH's into the TEE. They can even replay inputs/outputs. Moreover, executors can launch attacks by observing memory accesses and I/O patterns [63, 118] (either which memory is accessed or how much data/how many times) and DESEARCH must address them.

DESEARCH's goal is search. So we assume the source of data is correct. For concreteness, DESEARCH assumes the data is kept in a blockchain. This assumption boils down to: the blockchain has its own mechanisms to ensure the data introduced is not removed or tampered with.

The storage service, named Kanban, is fully untrusted. We only require for its availability. It can modify or forge the content of data, lie about data not being present, or drop messages between executors. At worst, Kanban can make DESEARCH unavailable but cannot cause DESEARCH to return incorrect responses to clients or learn a user's private query.

Finally, we assume a computationally bounded adversary that cannot break standard cryptographic assumptions (e.g., Diffie Hellman or RSA assumptions used in TLS).

3 System Overview

Challenges. Building a decentralized search engine that meets our requirements (§2.1) presents several challenges that stem from the decentralized environment, the limitations of today's TEEs, and the dynamic nature of search. First, decentralization requires executors to freely participate and depart (§2.1), which means that executors might be offline unexpectedly. Thus, a standard search engine design with long-running tasks and stateful components is unfavorable, as one executor's leaving can heavily impact the service.

Second, today's SGX instances have limited memory (128MB or 256MB); working sets in excess of this limit require expensive paging mechanisms. We hypothesize that the latency incurred by paging in practice will exceed what is acceptable for a user-facing service like search. To confirm this, we run a search serving component for Steemit (having 31.2GB indexes) in a single SGX instance with reasonable optimizations. It takes the instance 16 (and 65) seconds to respond to a single-keyword (and two-keyword) query. As a result, it is a necessity that the search's functionality be split into small tasks that are processed by SGX instances in parallel to achieve acceptable latency.

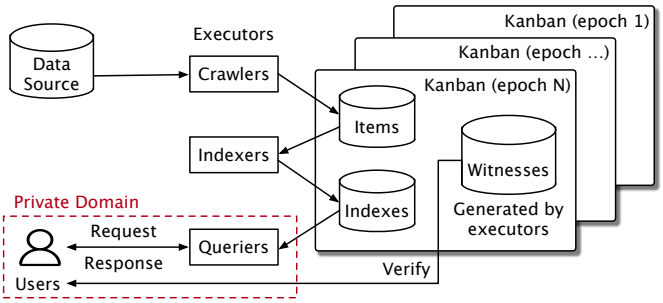


FIGURE 3—DESEARCH's architecture. DESEARCH obtains raw data from public decentralized services (e.g., Steem blockchain) as the data sources, and stores the intermediate data (i.e., items and indexes) on Kanban, a public append-only storage that creates snapshots periodically. DESEARCH executors generate witnesses along with the search pipeline. Privacy (§2.1) is offered for users in the query phase (within the dashed rectangle).

Third, search services are dynamic, and it is hard to track and verify the whole search process. In particular, a search engine is unable to plan a computation graph (like in big-data [71, 101, 112] or machine-learning [78, 86] systems) as the arrival of new source data or user queries is unpredictable. Therefore, it is unclear how to verify one executor's behavior because its "valid" data dependencies are unspecified a priori.

To address these challenges, DESEARCH decomposes a search into a pipeline of short-lived tasks where each executor is only responsible for one task at a time and on a portion of data. Executors are stateless. They fetch inputs and write outputs from and to a storage service named *Kanban*. Kanban is a cloud-based key-value store that provides high-availability, and data integrity, completeness, and freshness (§5).

To track the completion of the dynamically executing tasks within the search process, DESEARCH uses *witnesses* (§4), which are cryptographic summaries establishing the transfer of data among executors. A witness is also a proof of an executor's behaviors, which allows users to verify their search results *ex post facto*.

Figure 3 shows the architecture of DESEARCH.

Executors and Kanban. In DESEARCH, executors are categorized into four roles: crawlers, indexers, queriers, and masters (for simplicity, masters are omitted in Figure 3). The first three roles comprise a full search pipeline—*crawlers* fetch data from public sources (for example blockchain [38, 94, 116] or P2P storage [28, 58]); *indexers* construct inverted indexes; *queriers* serve users search requests.

Instead of point-to-point communication, executors in the pipeline communicate through Kanban. Kanban also stores the data (items, indexes, and witnesses) generated by executors, and provides data integrity (but not confidentiality) by periodically creating snapshots of all current state and

committing a digest of the snapshot to a public ledger (e.g., Ethereum [116] or Steem [38]). We call the time between two consecutive snapshots an *epoch*; for simplicity, we also use epochs to represent the corresponding snapshots (the data).

For privacy (§2.1), DESEARCH comprises public and private domains. Executors in the public domain access public data (like public source data) and produce shared information (like indexes). On the other hand, users' interactions with DESEARCH happen in the private domain, and their communications (for example, search requests and responses) are encrypted and kept secret.

Masters are the leader executors that serves three jobs in DESEARCH: (1) a membership management service that allows a node with TEE power to register itself to provide decentralized search; (2) a *key management service* (KMS) that allows anyone to identify if an executor is a legitimate DESEARCH member; (3) a job assignment service that coordinates the independent executors to form the search pipeline with minimal repeated work. In terms of membership, masters manage the process by which new executors join DESEARCH and authenticate that these executors are indeed TEEs running the correct software via remote attestation. If the attestation passes, masters assign a role (for example, crawler) to the newly joined executor. In terms managing the KMS, masters periodically (in the beginning of an epoch), release a list of public keys from legitimate executors so that users can verify their signatures and communicate with them. Masters also hold the *root key* that serves as the identity of the DESEARCH system, allowing the public to recognize DESEARCH. We describe how the system is bootstrapped, how new masters join, and how the root key is generated in § 9.

Workflow. DESEARCH's executors perform an ordinary search pipeline—crawling, indexing, and serving queries. Along every step of the pipeline, each executor generates a *witness*, a proof of what the executor has done and how data has been transferred. We discuss witnesses in Section 4.

To conduct a search, a user first retrieves a list of active queriers. This list is maintained by both masters and queriers: queriers update their status on the list with signed proofs specifying that they have seen the most-recent epoch; the legitimacy of the status proofs is verified by masters. Users know this by checking that the list is signed by masters.

With the active querier list, the user randomly selects one as the leader, and sends (encrypted) search keywords to the leader. The leader then seeks more peer queriers to collectively handle the request. That is, different queriers have different portions of indexes and together serve one user search request. The leader finally aggregates results by ranking on relevance, and returns to the user a list of the most relevant items. One item comprises a link (to the original content) and a content snippet that contains the searched keywords.

Together with the search results, the user also receives witnesses from queriers. The witnesses produced by the search pipeline (all of them) form a witness tree, which users can verify by starting from the witnesses received from the leader querier, traversing the tree, checking every node (witness), and confirming that the search has been correctly executed. We discuss this in Section 4.

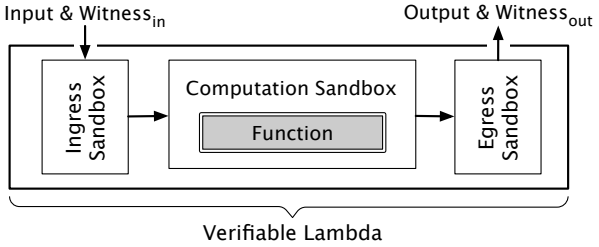


FIGURE 4—A verifiable lambda is composed of three intra-enclave sandboxes. The function box is one of: crawling, indexing, and querying. The witness$_{in}$ contains the hash for inputs and is from prior lambdas. The witness$_{out}$ is the witness generated by this lambda. Three-sandbox is is necessary for witnesses as it isolates the witness processing from the function execution. This is vital for DESEARCH because the buggy or malicious function cannot tamper with the integrity of witness.

4 Verifiable Search

In DESEARCH, search functions are outsourced to independent executors controlled by a number of individuals in a decentralized environment. Data are partitioned across executors. As a result, an executor can only have a local view of its own computation, and cannot guarantee the integrity of the entire search pipeline, even though they are fully protected with TEE. Specifically, to guarantee integrity (§2.1), there are two main challenges: (a) how to guarantee that the results are generated by the correct algorithm without being tampered with (*Soundness*), since the intermediate data are transferred amongst different executors on untrusted channels.and (b) how to ensure that the results are derived from the complete dataset without missing any data (*Completeness*).

To address the above two challenges, DESEARCH introduces *verifiable lambda* and *witness* for challenge (a), and *Kanban* with epochs for challenge (b). We detail them below.

Verifiable Lambda and Witnesses. As stated earlier (§3), DESEARCH splits a search pipeline into small tasks. Each task is executed by a basic unit, called *verifiable lambda*. The concept of a verifiable lambda (short as lambda) is inspired by serverless computing [34] (also known as Function-as-a-Service) and SGX sandboxing systems [75, 101], but the major difference is that DESEARCH's lambda requires a TEE enclave abstraction and yields a witness after every computation, allowing the intermediate data to be verified and reused.

A lambda is composed of three sandboxes (Figure 4): (1) an ingress sandbox that validates the data upon loading; (2) a computation sandbox which runs the main function and has a self-contained execution environment that does not use any external services, thus resists Iago attacks [66] (attacks in

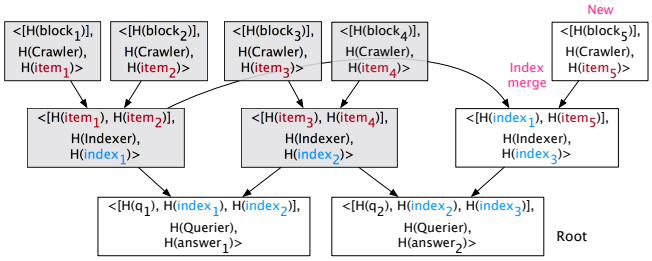


FIGURE 5—Two example witness trees. A rectangle represents a witness, edges represent data dependencies, and "H(...)" indicates hashes. This figure contains two search queries ("q_1" and "q_2"): q_1 happens first, and there is a merge update to the indexes ($index_3$ derives from $index_1$ and $item_5$), then q_2 happens. As a result, q_1 and q_2 share a subtree indicated in gray color.

which a malicious OS causes the process to harm itself); (3) an egress sandbox that generates a witness before delivering the result to the next lambda. This design isolates the actual function execution from witness processing, so that the integrity of witnesses is easy to reason about. As a technicality, there are several options for intra-enclave sandboxes, for example, hardware-based mechanism [105] and language-based isolation [113]. DESEARCH chooses the former. The notion of DESEARCH verifiable lambda can be extended to other scenarios such as the recommender system (§ 9).

A lambda's witness is a tuple:

$$\Big\langle \Big[H(in1), H(in2), \cdots\Big],\ H(func),\ H(out) \Big\rangle_{signed}$$

that mirrors how an *output* is generated by performing a *function* over a list of *inputs*. The $H(in)$ (and $H(out)$) is the hash of the input (and output) blobs, and $H(func)$ is the hash of the program binary that runs in this lambda. More generally, witnesses can be thought of as a *proof-carrying data* for the decentralized system, which explains how an output is being generated and with which piece of code. Note that a witness is signed by the lambda, and users can verify the signature using KMS (§3).

Witness-Based Verification. All witnesses from a search process form a tree, which we call a *witness tree* (see an example in Figure 5). Users can verify their search results by traversing and checking the corresponding trees, the roots of which are the witnesses users receive from queriers.

To check a single witness, a user first verifies whether the witness is signed by a legitimate executor and then checks if the hash of the executed function is as expected. This approach, combined with the integrity guarantees of the underlying TEE, ensures that the lambda which produced the witness faithfully executed the desired function.

Now consider the data transition between two adjacent lambdas in a search pipeline. The former lambda commits its output and a signed witness to Kanban; the latter lambda fetches one (or multiple) pair of data and witnesses from Kanban, checks their signatures, validates if the hash in the witness matches the data, and feeds the data to the main function. A user can verify that the inputs of a latter lambda are indeed the outputs from a former lambda by checking whether $H(in)$ of the latter equals $H(out)$ from the former.

Finally, users check if data sources are genuine by checking whether $H(in)$ in the beginning (the crawling phase) is indeed a correct summary of the original data source. Users need to download the contents from the data source and calculate their hashes, an expensive procedure (we address this performance challenge below). If all the above checks pass, users confirm that their search results are faithfully produced because all steps—crawling from the data source, the intermediate data transferring between tasks, and each task in the search pipeline—are verified to be authentic and faithfully executed.

Providing Efficient Verification. The aforementioned verification process works in principle, but in practice, a performance challenge arises. To verify a search, a user would have to download all the witnesses, check all the signatures and hashes, and examine the data source. To lighten the burden of verification on the user side, DESEARCH uses *delegated verification*: users offload parts of their verifications to some executors dedicated for verification, which we call *verifiers*.

Beyond simplifying user-side verification, delegated verification also saves work by batching and deduplicating verifications from different users. This is based on an observation that serving different search requests uses a lot of shared indexes, hence the witnesses from the shared portion can be reused (as an example, see the gray subtree in Figure 5). Because of delegated verification, verifiers have the opportunity to batch many common witnesses, which they verify once for all. Finally, users only have to verify the final step—the querier phase's witness, significantly accelerating the verification (see delegated verification's speedup in §8.2).

Search Completeness. The above verification ensures that functions are executed as expected, but there is no guarantee that these functions see all data. In fact, there is no definition of "all data" (or search completeness) from a user's perspective because newly generated data takes time to be reflected in the search results. It is therefore unspecified what data must appear in any particular search result.

To define completeness for searches, DESEARCH divides time into *epochs*, and executors write data to Kanban annotated with the current epoch (we elaborate on epochs in §5). We define a search as complete if each step (represented by witnesses) in the search pipeline (represented by a witness tree) uses *all* inputs in Kanban before the step's epoch, and each input is from the *most recent* epoch available. For example, a querier's task in epoch i is complete if the querier loads all indexes generated before epoch i, and the loaded portions of the indexes are from their latest version before epoch i.

To check the completeness of a witness, verifiers first recognize the epoch when the witness was generated, then load

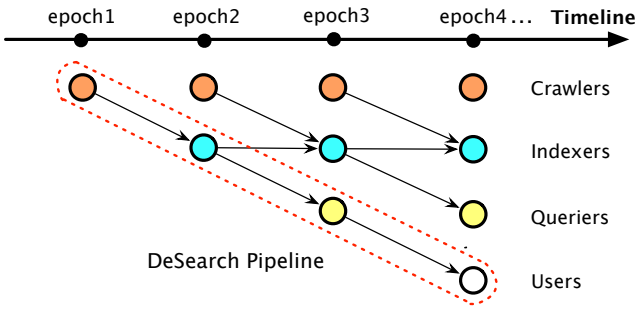


FIGURE 6—In DESEARCH, executors and users use data from the last epoch as inputs, which we call the *offset-by-one* strategy. For example, an indexer uses items of last epoch to generate new indexes (denoted by oblique arrows), and may merge indexes from the prior epoch (denoted by horizontal arrows).

the snapshot of Kanban immediately before that epoch (§5), and finally verify if the executor used all the update-to-date inputs. In practice, verifiers do not have to load the data in the snapshot. They simply load the metadata including data ids and their hashes.

5 Kanban

As mentioned in Section 4, Kanban is a storage system that provides high availability and data integrity. Kanban is hosted in public clouds for availability, but as a decentralized system, DESEARCH does not trust these cloud providers for correctness. Instead, DESEARCH creates snapshots of Kanban periodically, namely epochs, and commits digests of epochs to a public distributed ledger. This approach provides *epoch-based data integrity*: DESEARCH guarantees the integrity of all of the data included in a committed epoch.

The rest of this section will introduce Kanban's usage and guarantees, and then discuss how DESEARCH uses Kanban as a storage and coordination service.

Kanban Overview. Kanban serves two main purposes: storing data (including items, indexes, and witnesses) for the search pipeline, and allowing executors to communicate and coordinate their tasks. Kanban has simple APIs for both services.

As a data storage, Kanban exposes key-value-like APIs with `append(key, val)` and `get(key)`. Keys are constructed by the data types, chunk numbers, and epoch numbers. For example, "index-#1000-v3" represents the 1000th chunk of the index for epoch 3.

For communication, Kanban provides a mailbox for every executor with `send(mailbox, msg)` and `recv(mailbox)`. Invoking `send()` allows an executor to submit a message to a specified mailbox, and `recv()` to download messages. All messages are encrypted using the mailbox owner's public key, which can be obtained from the KMS (§3), so only owners can read message contents. There is no need for executors to know other executors' IP addresses to send them a message.

Note that both storage and communication APIs are wrappers of the canonical key-value APIs, and Kanban can easily adapt to different underlying (cloud) storage systems. Our Kanban implementation uses Redis, a popular key-value store, as the underlying storage (see §7).

Epoch-based Data Integrity. Kanban requires executors to sign their submitted data (using Ed25519) to prevent data tampering or forgeries. Still, the underlying storage can show different views of the data to different executors by omitting, fabricating, rolling back, or forking the data. Detecting such divergences often requires clients (executors in our context) to synchronize out-of-band [82, 85], which is too expensive in a decentralized environment.

DESEARCH uses a loose synchronization approach: masters periodically synchronize Kanban's states with other executors. This loose synchronization works because of two observations. First, search engines are not supposed to reflect newly generated data in search results right away because crawling and indexing takes time; as a (admittedly apples-to-oranges) comparison, Google crawls a site every 3 days or even longer [7]. Second, most of the tasks in the search pipeline are idempotent, so it is acceptable if two executors end up working on the same task. For example, it is safe for two crawlers to crawl the same data source, or two indexers to generate indexes for the same items, as the results are the same. Such duplicate work sacrifices efficiency but not correctness.

To synchronize states with other executors, a master periodically creates an incremental snapshot (an epoch) of Kanban's data storage (without mailboxes), summarizes the snapshot as a digest, and commits the digest to a public append-only ledger (for example, Ethereum [116] or Steem [38]). After the ledger accepts the digest, a new epoch is committed and is (supposedly) visible to all executors.

DESEARCH guarantees *epoch-based data integrity*: for a committed epoch, all data included in this epoch is immutable and must be visible to all executors; otherwise, verification will fail. To see how DESEARCH guarantees this, if Kanban hides data from or returns stale data to an executor, the completeness checks (§4) of this executor's witness will fail. This is because verifiers know the epoch of the witness (say epoch i) and the data this executor should have read (data in epoch $i-1$). If the witness missed any data or read some stale version, the verifier rejects.

Before using one epoch for checking completeness, verifiers must ensure that the data (represented by their ids and hashes) in one epoch is consistent with the digest on the ledger. Our current verifier implementation fetches all data ids and hashes in one epoch, calculates their digest, and compares it with the digest on the public ledger. A better solution is to organize each epoch as a hash tree [90], and verifiers can check on-demand by only re-calculating hashes for the sub-tree that they are examining.

Task Coordination by Epochs. DESEARCH's pipeline is

coordinated through Kanban, which is based on epochs. An epoch is 15 minutes by default. Executors learn the current epoch number by querying the public ledger.

Figure 6 depicts how DESEARCH works with epochs. Executors follow an *offset-by-one* strategy: they reads data from the last committed epoch. This guarantees that the DESEARCH pipeline only uses data that is already authenticated by the epoch-based digests on the ledger.

Our current implementation uses masters to assign jobs for crawlers and indexers. Masters also take charge of what data is included in each epoch. They collect all data generated in the current epoch and list them in an epoch. If an executor in epoch i fails to submit its outputs on time (before masters commit epoch i), the data will be ignored and the work is wasted. But this happens rarely as tasks are small.

Supporting Multiple Clouds. Though our current implementation only uses one cloud as Kanban's underlying storage, we plan to extend Kanban with multiple clouds for better availability, and more importantly, to lower the risk of vendor lockdown. With multiple clouds, executors write to all clouds and read from any one of them. Master executors are obligated to synchronize different clouds.

Data synchronization among clouds is challenging, which often requires running an expensive consensus protocol (like Paxos or Raft). By Kanban's epoch design, DESEARCH synchronizes clouds loosely, only when committing an epoch. And if data diverges between clouds (note that data cannot be forged, due to signatures), master executors are in charge of merging the data. The key takeaway is that DESEARCH (or rather a search service) can tolerate staleness and infrequent synchronizations, so masters have plenty of leeway to orchestrate an epoch on which all cloud agree.

6 Oblivious Search

DESEARCH guarantees search integrity (§2.1) by leveraging witnesses and SGX. One might hope that SGX would also provide privacy for searches, as SGX supports confidential computing [26] and we assume that SGX works as designed (§2.4). However, DESEARCH's design in Section 4 leaks information: an adversary can learn users' keywords without breaking any guarantees of SGX. Below, we introduce privacy violations, and then show how DESEARCH addresses these violations with ORAM and equalizing message length.

Privacy violations. To start a search, a user initiates a TLS/SSL session to a selected querier from the active list of queriers published on Kanban from masters. Although the messages are encrypted and authenticated and computations are confidential (offered by SGX), adversaries can still conduct two types of attacks (see examples in Figure 7):

First, an *executor adversary* that runs queriers can observe memory access patterns (both EPC and DRAM) to infer user keywords. Specifically, the adversary issues search requests with all possible keywords to the querier it hosts; by observing

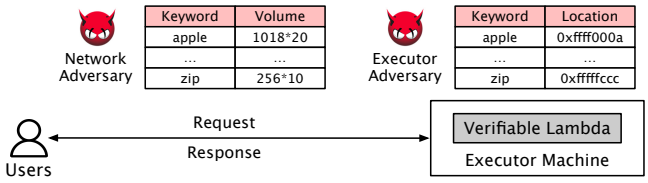


FIGURE 7—A search faces two privacy challenges: a network adversary can learn keyword information by monitoring request/response volumes, and an executor adversary can infer keywords by observing memory accesses.

memory accesses, it can construct a dictionary that maps a keyword to a memory location [118]. Consequently, when a real user sends a query, the adversary can infer the user's keywords by observing which memory locations are accessed by looking them up in the dictionary.

Second, a *network adversary* can eavesdrop on the communication between users and queriers, and among queriers (which occurs when collaboratively serving one request). By monitoring the network packet sizes, the adversary can learn information about keywords [64] because candidate lists for different keywords have different lengths, and returned items (for example, a post on Steemit) also vary in size. Similar to an executor adversary, a network adversary can also construct a dictionary that maps keywords to search response lengths.

DESEARCH + ORAM. To prevent attacks from executor adversaries, DESEARCH uses Circuit-ORAM [114], a well-studied Oblivious RAM (ORAM) protocol. To be clear, similar approaches have been explored before [88, 99]. DESEARCH's contribution is adapting the ORAM protocol to build an end-to-end decentralized search engine that achieves (relatively speaking) good performance.

Circuit-ORAM applies to a key-value store. In our case, the keys are search keywords and values are lists of item ids. Circuit-ORAM guarantees that an executor adversary cannot learn anything about the searched keywords by analyzing its memory accesses. For a keyword that does not match any item, DESEARCH performs dummy accesses. Note that Circuit-ORAM does not support concurrent accesses, so DESEARCH leverages multiple ORAM replicas for higher throughput. Specifically, DESEARCH encodes the underlying data in multiple ORAM instances, and accesses different instances to process queries. This is safe because we use ORAM exclusively for read-only workloads, and each instance is independent and has its own position map. This requires more storage space, but DESEARCH allows executors to make this trade-off.

Equalizing Message Length. Beyond ORAM, DESEARCH needs to avoid leaking information from the number of matched result items (count) and the length of each item (volume). We observe that results from search engines are highly regular: search results are displayed in multiple pages; each page contains a fixed number of items; and each item contains

DESEARCH Components	Language	Total LoC
Openbazaar Crawler	Golang	178
Steemit Crawler	Python	190
Indexer	C++	592
Querier	C++	638
Master	C++	3,126
Verifier	Python	1720
Kanban	C++	618
DESEARCH LibOS	C++	13,851
Search Engine Library (Sphinx [36])	C++	88,345
ORAM Library (ZeroTrace [99])	C++	4,838
Crypto Library (OpenSSL)	C/C++	3,437

FIGURE 8—Lines of code of each component in DESEARCH.

	Shard Size		
Metrics	**10K**	**100K**	**1M**
SGX Load Time	4s	18s	127s
ORAM Setup Time	10s	104s	1019s
Response Time	14ms	46ms	273ms
DRAM Required	13MB	31MB	296MB
SGX EPC Used	79MB	80MB	86MB

FIGURE 9—Execution time and memory usage of a DESEARCH querier under different shard sizes. SGX load time includes fetching the latest version of index/digest files from Kanban and in-enclave deserialization. ORAM setup time includes initializing two Circuit-ORAM instances, namely, inverted indexes and content digests.

a link (e.g., URL) and a small content snippet highlighting the keywords. Therefore, DESEARCH equalizes result lengths by returning a fixed number of entries for each search request, and each entry has a 256-byte summary of the original contents. Our mini-survey shows that more than 80% of search result entries from Google are within 256 bytes. DESEARCH hides the counts and volume of keywords by padding search queries to the same length and limiting the number of keywords to 32 (the maximum supported by Google).

7 Implementation

System Components. We implement our own crawlers that parse raw data from Steemit and OpenBazaar. The Steemit crawlers continue to aggregate the data appended to the Steem blockchain, and the OpenBazaar crawlers pretend to be OpenBazaar peers to obtain the online shopping items. We use the tokenizer from the Sphinx v2.0 [36]. Since we need to manage index and digest database oursevles, we build our own index management and support oblivious index management via ORAM and customized distributed searches. We use Intel SGX SDK to implement the three-sandbox verifable lambda abstraction. In particular, our current implementation uses three enclaves, which enforces hardware-level isolation between witness validation/generation and search function execution. We implement Kanban based on Redis v3.2 and deploy it on a public cloud. We use a standalone machine to simulate the append-only ledger system by providing APIs to read and write on-ledger data. The rest of DESEARCH's components are shown in Figure 8.

Parameter Tuning. We initialize the Circuit-ORAM setup for indexes with 512-byte ORAM block size, and another Circuit-ORAM instance for content summaries with 256-byte ORAM block size. We overlap their operations to minimize the latency. For the queriers, we add an inode layer as an indirection to reference the real indexes. Using two blocks as the inodes, it is enough to accommodate keywords with 64KB mapping list (the maximum length of a Steemit mapping list we found is 48KB, and that of OpenBazaar's is less than 4KB). As 99th percentile of the crawled items are shorter than 512 bytes, these raw data items are directly stored in the index block to reduce the storage overhead.

We experiment with different data setups to help determine what size of data shard is more suitable for a desktop-side querier, as shown in Figure 9. We assign 1M data items to each querier because it incurs mild memory overhead (without exceeding SGX physical memory capacity) and its response time is acceptable (within 1 second).

Finally, the Kanban epoch value is set to 15 minutes, because almost all existing distributed ledger/blockchain systems yield a new block within this period [22]. This is enough for verifiers to summarize the data snapshot on Kanban within an epoch. For shorter epochs, one could use an incremental hash function [57] to create the digest incrementally throughout the epoch instead of starting at the end of it.

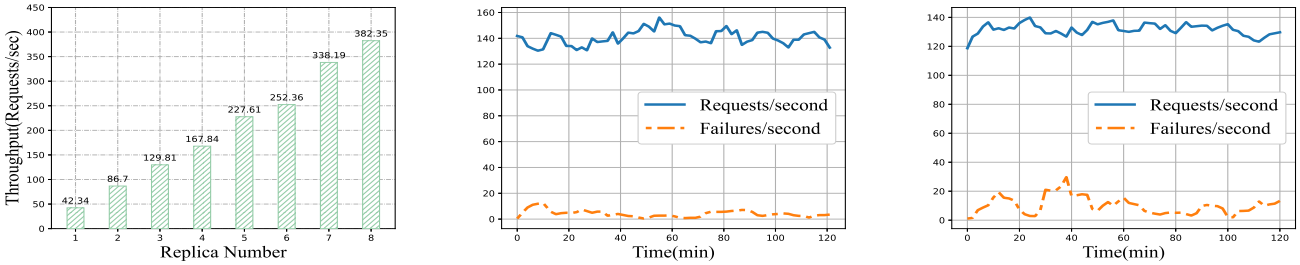
Side-Channel Defenses. DESEARCH uses the formally verified cryptographic library HACL⋆ v0.2.1 [123] which is resistant to digital (cache and timing) side channels. Particularly, we borrow its Ed25519 implementation that contains no secret-dependent memory access and branch conditions against speculative attacks [67] and branch prediction attacks [81]. We integrate ORAM into index management using a ZeroTrace [99]. For ORAM block encryption, we choose AES-NI-based AES-128-GCM. AES-NI is purportedly side-channel resistant according to Intel [76]. Finally, we use the OpenSSL library [23] with the latest patch from Intel (commit f74c8a4) to mitigate hardware vulnerabilities (e.g., power-based fault injection [111]).

Limitations. DESEARCH's current implementation only supports full-text search. The links of images, audio, or video, encoded in the texts may be hosted in other unverified servers. DESEARCH does not guarantee their integrity.

8 Evaluation

Our evaluation answers the following questions:

- What is the overall serving performance of DESEARCH when dealing with the whole dataset, in terms of end-to-end latency, throughputs, and scalability? (§ 8.1)
- How long does it take to verify a search results with respect



(a) DESEARCH scales with more replicas. (b) 20% instance failure model. (c) 50% instance failure model.

FIGURE 10—Evaluating DESEARCH w.r.t scalability and failure tolerance in a decentralized environment. In (a), each replica consist of 82 distributed instances. The failure tolerance experiments (b and c) use a 4-replica setup and collect 2-hour throughput variation and failure rates.

to soundness and completeness? (§ 8.2)
- Does DESEARCH tolerate executors joining, leaving, or failing frequently? (§ 8.3)

Experimental Setup. We deploy DESEARCH on 15 SGX-enabled machines, with 8-core Intel i7-8700 CPU at 3.20GHz, 16GB DRAM, 128MB processor reserved memory (≈ 94MB EPC), on Ubuntu 16.04 LTS, Linux kernel 4.13.0, and microcode version 0xd6. We run enough querier instances on each machine without reaching the physical memory limit (no swap space used), and each instance is seen as an executor.

To build a decentralized DESEARCH network, we also use cloud-based virtual machines to add more executors. We run DESEARCH using simulation mode and add the latency obtained from the microbenchmark (see § 8.1, Overhead Analysis). To simulate a geo-distributed environment, we use the global ping statistics [16] to emulate the latency amongst executors. It takes around 237ms on average to establish a private session using Kanban's mailbox (using executor's public key). Our evaluation does not include this overhead, since a user or an executor can reuse a session to avoid this costly overhead coming from frequent session re-establishment.

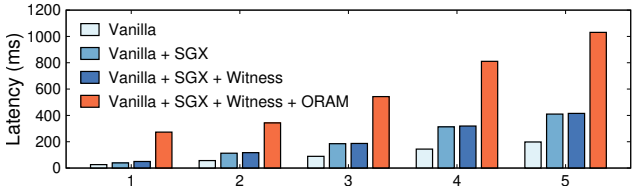
8.1 Serving Performance

Search on Steemit Dataset. Steemit [39] is a decentralized blogging and social media service built upon the Steem blockchain [38], which rewards users with the cryptocurrency Steem Dollars (SBD) for publishing and curating content. We run 50 DESEARCH crawlers on SGX-enabled desktops to constantly update Kanban with the latest posts from the Steem blockchain. As the time of evaluation, we altogether fetched 81,681,388 posts, distributed across 952 epochs (nearly 10 days) leading to around a 234GB Steemit dataset. Simultaneously, we run 50 indexer instances to build indexes using the data from Kanban. It takes 108s for an indexer to fetch 100K items from Kanban, and 88s to generate the corresponding indexes. The indexers further remove 659 stopwords based on Google stopword list [41], and therefore yields a total of 294K keywords for an inverted index. All of this results in witnesses totaling 21.31GB.

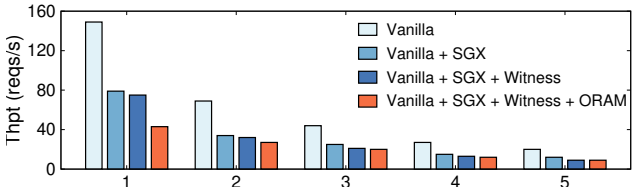
To offer search for the complete Steemit dataset (234GB), we run 82 instances of DESEARCH queriers, each managing 1 million data items (namely, Steemit posts) with witness generation and ORAM enabled. We measure the end-to-end latency by randomly choosing queriers to collect all results, and sending 10,000 single-keyword search requests to saturate the queriers. The average end-to-end latency is 205ms, and the worst end-to-end latency is 706ms. This latency is reasonable for a search engine. The throughput is 43 requests/sec (this corresponds to 3.7M requests/day). Although the measured throughput is relatively low, we expect that a decentralized network can achieve higher throughput as more executors join (see "Scalability" below).

Search on OpenBazaar Dataset. OpenBazaar (OB) [28] is a decentralized e-commerce platform where individuals can freely trade goods with no interference from a middleman. The OB frontend provides an API [6] for a customized search engine to update the shop contents by crawling IPFS [58], an append-only storage for OB's shopping lists. Because OB allows sellers to remove listings that are no longer available (e.g., sold out or discontinued), we do not crawl the stale listings from IPFS. At the time of evaluation, it has an average of 21K listings on sale per day, and hence we only use one of DESEARCH's executors to support the search. The latency for searching OB's dataset is 37ms, while the throughput is 53 requests/sec. Everyone who has an SGX machine can reuse the indexes on Kanban, build the search for OB in about 3 minutes, and quickly support other peers in need.

Overhead Analysis. How does trusted hardware, the witness mechanism, and oblivious protections affect the search performance? To answer this question, we measure the overhead breakdown of one querier instance managing 1M data items. For each experiment, we run `ab` for 5min and search the most popular keywords in the dataset, which can be considered as the worst-case scenario for DESEARCH. Figure 11 shows the average latency and average throughput for multi-keyword searches. The latency is proportional to the number of search keywords as it requires fetching more rounds of index blocks from the Circuit-ORAM server and finding intersecting re-



(a) The average latency of multi-keyword searches.



(b) The average throughput of multi-keyword searches.

FIGURE 11—Latency and throughput with the overhead broken out with increasing numbers of search keywords. The test uses top 5 frequently used keywords for search, hence showing the worst-case performance. Vanilla is a native DESEARCH querier with indexes for 1 million data items without using SGX protection. SGX represents the SGX LibOS-based isolation; Witness represents the lambda confinement; ORAM represents the Circuit-ORAM construction.

sults.

Putting a querier into SGX with DESEARCH's lambda enforcement adds 107% overhead in latency and 102% overhead in throughputs. This is because each network I/O requests must issue an ocall (a context switch between SGX enclave and user-space) in DESEARCH's egress sandbox. In the future, we can optimize DESEARCH with asynchronous ocalls as studied in SCONE [55]. The witness generation consists of hashing the lambda's inputs and outputs with SHA-256, and signing this witness with Ed25519. The witness generation only imposes 1% overhead. Finally, we find that the dominating factor is Circuit-ORAM, which incurs 148% $\sim$ 446% performance overhead. While this overhead is considerable, it gives strong privacy guarantees against the executor adversary that controls the lambda.

Historical statistics [19] show that 71.3% search queries do not exceed four keywords. Four-keyword ORAM-based searches in DESEARCH incur 811ms latency, which is acceptable in a human interactive process.

Scalability. To evaluate if DESEARCH can scale horizontally, we measure the overall throughput while increasing the number of DESEARCH replicas, each of which consist of 82 Steemit queriers. As shown in Figure 10a, the throughput of the deployed network increases linearly with the number of replica servers. By supporting 380 requests/sec, DESEARCH can handle at least 32 million requests per day.

According to Alexa traffic analysis [1], Steemit has 30K daily users, and each of them has at least 2 visits per day. Because of the fact that Steemit is one of the top 5 popular decentralized services [37], this means that our current DESEARCH implementation is able to support up to 540 different decentralized services concurrently.

	Native	Verifier-based	Speedup
Soundness			
Witness Download	437s	–	
Signature Verification	273s	–	
Completeness			
Witness Tree Verification	512s	–	
Final-Phase Verify			
Verifier Interaction	0s	1.0s	
Signature Verification	0s	0.3s	
Total Time	1231s	1.3s	947x

FIGURE 12—User-side verification cost comparison between native and verifier-based delegated verification.

8.2 Verification Cost

A user verifies the final results holding:

- the root digests retrieved from the ledger,
- the public keys of all executors from DESEARCH masters,
- several witnesses from queriers and the rest of the related witness from Kanban.

The first two can be prefetched and therefore can be deemed an offline cost. Figure 12 shows the user-side costs of different verification choices for a single-keyword search.

In the native verification, a user verifies the search results on her own. To verify soundness, it takes 437s to download witnesses from Kanban, and 273s to check the signatures using Ed25519. For completeness, the user first ensures the witnesses from Kanban are consistent with the digests from the ledger, and then breadth-first traverses the connected witness tree from search results to indexes to items. It takes 521s to finish this verification. The verifier code can be installed locally to automate this process.

In the verifier-based verification, a user sends the received witnesses to verifiers. The optimization opportunity is that queriers share indexes in common within an epoch, and hence the verification effort can be reused. The user only needs to verify the witnesses in the private domain, which are the 82 witnesses from queriers. It only takes 1.0s to interact with a verifier and 0.3s for hash and signature verifications.

8.3 Failure Tolerance

To simulate failure, we use an online/offline list to maintain each executor's status. Statistically speaking, we assign 80% executors with online time more than one epoch (15 min) and 20% staying online for less than one epoch. If a querier does not respond in time, the client issues a retry after a timeout (3s). Dead executors are removed from the active list when the epoch is updated, and they will come back alive and re-join DESEARCH network again. If the remaining online executors fail to comprise a complete dataset, the client will receive a failure. This test use 4 replicas, where each instance still

handles one million data items.

We collect the variation of throughput and failure rates under different situations using 2 hours of experiments. In Figure 10b, we observe DESEARCH's throughput varies between 130 requests/sec and 160 requests/sec, and the failure rate is below 10 per second. Figure 10c shows another extreme situation where 50% of executors fail within one epoch. We observe the throughput still remains around 130 requests/sec while the failure rates rise to 15 per second on average.

9 Discussion

Incentive Model. Like existing decentralized systems [94, 116], DESEARCH relies on volunteers to offer service availability. To encourage TEE owners to join in DESEARCH, we hence discuss a possible incentive model, inspired by Teechain [84] that harnesses TEE as a deposit bank, where a cryptocurrency owner can securely transfer tokens from a blockchain to an offline TEE enclave. Our observation is that currently Steem Dollars (SBD) flow from readers to mostly popular post authors. Our incentive model, instead, transfers SDB from readers $\rightarrow$ queriers $\rightarrow$ indexers $\rightarrow$ crawlers $\rightarrow$ authors, where anyone who uses data from others *pays* for the data usage. Our witness is a natural evidence for this usage. The more TEE computing power an executor contributes, the more cryptocurrency it gains as rewards. Introduing such an incentive model also helps mitigate the DoS attacks as the attack has to pay the executor.

System Bootstrap. To bootstrap DESEARCH, any executor with TEE can become a master, as long as it downloads the code of master and runs the code within the TEE enclave. The master's initialization will generate a pair of public/private keys. The former is recorded on the ledger for public availability, and the latter is kept within the enclave, being DESEARCH's root key. Any party can use TEE remote attestation mechanism [77] to verify the genuineness of the root key. The master then registers itself in the active list on Kanban. More executors (say, 21) are required to join in and be assigned masters to start the epoch. Executors that join later will be finally assigned search lambdas to provide the search service.

Storage and Network Cost. DESEARCH assumes a public Kanban that provides large storage and high bandwidth. Using Amazon's 32GB-RAM EC2 T2 instance price as of November 2020, DESEARCH costs about $10 US dollars per day. However, the cost can be amortized over 30K users per day if we assume enough users are using DESEARCH (§ 8.1), for a total of less than 0.033 cents per user each day.

Extending DESEARCH to Other Functionalities. The architecture of DESEARCH is not limited to the search scenario. The nature of the stateless design in DESEARCH lambda allows it to be extended to more decentralized services. For example, it is intuitive to extend our architecture for verifiable recommendation for OpenBazaar, the online shopping case. A user can verify that the advertisements are chosen based on his interest, without infiltrating any opaque intention. To protect user's privacy, such a decentralized recommender system would require more care such as differential privacy.

Resistance on the Supply-chain Attack. To resist recent notorious supply chain attacks (e.g., SolarWinds Attack [35]), DESEARCH should provide ene-to-end verifiablity from source code to lambda image. Thanks to Reproducible Builds [31], the correctness of DESEARCH results can be audited via the trail of updates of the lambda software. It is possible to place a compiler (bootstrap trust via Diverse Double-Compiling [115] against Trusting Trust attacks [108]) inside an verifiable lambda, and use the generated witness to verify the code of the lambda.

Other Use Cases. DESEARCH, being an infrastructure, can be used as "watchdog" for other search services. In the "covert adversar" threat model, the adversary can act maliciously provided that it is not being caught (otherwise would have significant financial loss or legal implications). Users can continue to use an existing search engine, and cross-validate the results with a system like DESEARCH in the background and off the critical path, or potentially as a spot check rather than on every search query. This presupposes that the centralized search engine is willing to public algorithms for crawling, indexing, ranking, etc.

Higher Throughputs. Achieving higher throughputs for decentralized systems is an exciting topic. The current throughput of an DESEARCH executor is limited due to the use of Circuit-ORAM, which inherently lacks concurrency support. We also tried array-based ORAM, such as Square-Root ORAM (SQRT-ORAM) [109]. Our experience with SQRT-ORAM does incur little slowdown to throughputs, but its reshuffling phase introduces 11.8s downtime for every 2000 requests for a querier. To hide this downtime, PRO-ORAM [109] suggests using multiple instances. However, this would incur more than 12GB memory overhead. We think it is not suitable for a client-side machine. Our lesson is that although our implementation is limited by Circuit-ORAM, the decentralized replicas actually contribute to acceptable throughputs (see Figure 10a).

10 Related Work

Decentralized Search Engines. There have been efforts in building decentralized search services. YaCy [47] is a peer-to-peer distributed search project since 2004. It enables decentralized indexes generation and allows these indexes to be shared between peers. It implicitly assumes honest peers, which does not hold true in a realistic decentralized environment. Presearch [29] leverages a blockchain to provide decentralized search. Its design choice inherits blockchains downsides (e.g., wasteful replicas, privacy breaches). The Graph [20] is an indexing protocol to search over decentralized stor-

age (e.g., Ethereum and IPFS). Similar to DESEARCH, The Graph utilized a volunteer network; however, it lacks privacy for the search process, and relies on determinism for result verification. By contrast, DESEARCH is the first decentralized search engine that addresses the challenges of both verifiability and privacy. Dory [69] is a decentralized oblivious search system atop the file system. Dory on its own does not return the files, nor rank results, both of which are crucial to the real search. DESEARCH provides privacy for keyword-based data retrieval, ranking and returning results to end users.

Verifiable Search Engines for Decentralized Services. There is an increasing interest in providing verifiable search for decentralized services, due to the diversified usages of decentralized applications (see Figure 1). For verifiable blockchain searches, vChain [117] adopts authenticated data structures (ADS) while GEM^2-Tree [120] explores on-chain indexes. Compared with DESEARCH, vChain and GEM^2-Tree only support range-based searches, whereas DESEARCH provides general-purpose full-text search and offers verifiability via witness-based dataflow tracking. IPSE [24] provides search over a public decentralized storage—IPFS [58] and allows users to validate the correctness of the returns results via hash-based content addressability. Freenet [14] is an anonymous file sharing network (similar to BitTorrent, providing hash-based file verifiability) but is not search oriented. One should know the file's seed to retrieve the file, but not necessarily search all files that contain a particular content. BITE [88] offers trustworthy Bitcoin transaction searches by deploying an enclave at a Bitcoin full node. Although IPSE, Freenet and BITE can provide correct results for users (*soundness* in DESEARCH), they lack *completeness* for the final results, which is critical for search because missing results can make the search service vulnerable to censorship [40]. In comparison, DESEARCH witnesses also guarantee search completeness with respect to the latest committed epoch.

Private Search Engines with TEE. TEE is a hot topic for providing private search. To hide users' search intention, X-Search [92] utilizes a cloud-side TEE proxy while Cyclosa [96] adopts browser-side TEE proxies. X-Search and Cyclosa are metasearch engines (a proxy between users and a search engine) which reveals query keywords and results to search providers, whereas DESEARCH is a complete search engine that provides privacy by combining TEE and ORAM. A long line of prior works (Opaque [122], OCQ [70], ZeroTrace [99], Oblix [91], Obliviate [50], POSUP [74], etc.) have explored TEE and ORAM combination to protect search process privacy. Similar efforts have been made by combining symmetric searchable encryption (SSE) and TEE (e.g., Rearguard [107]), or private information retrieval (PIR) and TEE (e.g., SGX-IR [104]). DESEARCH can benefit from these private search techniques to better protect decentralized search services.

Secure Big-data Systems with TEE. Many systems have leveraged TEE for big-data computation [59, 75, 101] and big-data analytics [70, 98, 122]. VC3 [101] secures a map-reduce framework with TEE, and Opaque [122] protects SQL queries for Spark SQL. Different from VC3 and Opaque, DESEARCH is facing a dynamic computation graph. As the number of alive executors change over time, DESEARCH applies witnesses to verify the correctness of the ad-hoc tasks. Ryoan [75] provides distributed sandboxes for private data computation. Compared with Ryoan, DESEARCH offers publicly verifiable witnesses for reusable intermediate data and effectively reduces the verification cost (see § 8.2).

Other TEE-based Systems for Decentralized Services. TEE have been used to build a data feed service for blockchain (or oracle) [121], off-chain smart contracts [68], Bitcoin fast payment channel [84], and online-service sharing [87]. They differ from DESEARCH in their purpose and functionality.

TEE and Serverless Computing. DESEARCH's verifiable lambda notion is inspired by today's serverless computing. DESEARCH extends this notion from centralized cloud-based computing to decentralized computing. S-FaaS [51], T-FaaS [61], Clemmys [110] are TEE-based serverless systems that protect serverless workloads with TEE. Instead, DESEARCH utilizes epoch-based Kanban and TEE-generated witnesses to maintain and verify the state of the (stateless) TEE lambdas.

New Decentralized Systems. Recently, some new architectures of decentralized systems have been proposed to address the limitations (low-throughput, resource waste, lack of privacy) of conventional decentralized ledgers or blockchain. Algorand [73] and Blockene [100] propose new consensus protocols to achieve a high-throughput. Omniledger [79] and Protean [52] introduce sharding to scale out the blockchain. Similarly, DESEARCH shards executors to different roles, and offloads states to Kanban to achieve high scalability.

11 Conclusion

DESEARCH is the first decentralized search engine to support existing decentralized web applications, while guaranteeing verifiability owing to its verifiable *witness* mechanism and offering end-to-end privacy for query keywords and search results. DESEARCH achieves good scalability and minimizes fault disruptions through a novel architecture that decouples the decentralized search process into a pipeline of verifiable lambdas and leverages a global and highly available Kanban storage service to exchange messages between lambdas. We implement DESEARCH on top of Intel SGX machines and evaluate it on two decentralize systems: Steemit and OpenBazaar. Our evaluation shows that DESEARCH can scale linearly with the number of workers (executors) and can achieve the stringent subsecond latency required for a search engine to be widely usable.

Acknowledgments

We sincerely thank the anonymous reviewers (including at OSDI 2020) for their helpful and constructive comments,

and our shepherd Marko Vukolic who greatly helps improve this paper. We also thank Sajin Sasy for discussing about ORAM. This work was supported in part by National Key Research and Development Program of China (No. 2020AAA0108502), China National Natural Science Foundation (No. 61972244, U19A2060, 61925206, 61732010). Sebastian Angel was funded by NSF Award 2045861 and DARPA contract HR0011-17-C0047.

References

[1] Alexa - competitive analysis, marketing mix and traffic. https://www.alexa.com/siteinfo/steemit.com.

[2] Augur. https://www.augur.net/.

[3] Blockstamp openbazaar explorer. https://bazaar.blockstamp.market/.

[4] Cam4 live-streaming adult site exposed 7tb records publicly. https://www.2-spyware.com/cam4-live-streaming-adult-site-exposed-7tb-records-publicly.

[5] Censorship by google. https://en.wikipedia.org/wiki/Censorship_by_Google.

[6] Content discovery on OpenBazaar. https://openbazaar.org/blog/decentralized-search-and-content-discovery-on-openbazaar/.

[7] Crawl stats report - search console help - google support. https://support.google.com/webmasters/answer/9679690.

[8] Data-enriched profiles on 1.2b people exposed in gigantic leak. https://threatpost.com/data-enriched-profiles-1-2b-leak/150560/.

[9] Dtube. https://d.tube.

[10] Duo search is a search engine for openbazaar. https://bitcoinist.com/duo-search-is-a-search-engine-for-openbazaar/.

[11] Etoro. https://stocks.etoro.com/.

[12] Facebook is illegally collecting user data, court rules. https://thenextweb.com/facebook/2018/02/12/facebook-is-illegally-collecting-user-data-court-rules/.

[13] Facebook suspends cambridge analytica for misuse of user data, which cambridge denies. https://www.cnbc.com/2018/03/16/facebook-bans-cambridge-analytica.html.

[14] Freenet. https://ipfs-search.com/#/search.

[15] Giving social networking back to you - the mastodon project. https://joinmastodon.org/.

[16] Global ping statistics. https://wondernetwork.com/pings.

[17] Golem network. https://golem.network/.

[18] Google faces antitrust investigation by 50 us states and territories. https://www.theguardian.com/technology/2019/sep/09/google-antitrust-investigation-monopoly.

[19] Google search statistics and facts 2020. https://firstsiteguide.com/google-search-stats/.

[20] The graph is an indexing protocol for querying networks like ethereum and ipfs. https://thegraph.com.

[21] Graphite docs. https://www.graphitedocs.com/.

[22] How many bitcoins are mined everyday? https://www.buybitcoinworldwide.com/how-many-bitcoins-are-there/.

[23] Intel® software guard extensions ssl. https://github.com/intel/intel-sgx-ssl.

[24] Ipfs search. https://freenetproject.org/.

[25] Matrix - an open network for secure, decentralized communication. https://matrix.org/.

[26] Microsoft azure confidential computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/.

[27] The monopoly-busting case against google, amazon, uber, and facebook. https://www.theverge.com/2018/9/5/17805162/monopoly-antitrust-regulation-google-amazon-uber-facebook.

[28] OpenBazaar. https://openbazaar.org/.

[29] Presearch is a decentralized search engine, powered by the community. https://www.presearch.io/.

[30] Report: 267 million facebook users ids and phone numbers exposed online. https://www.comparitech.com/blog/information-security/267-million-phone-numbers-exposed-online/.

[31] Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code. https://reproducible-builds.org/.

[32] search poisoning. https://whatis.techtarget.com/definition/search-poisoning.

[33] Searchbizarre. https://searchbizarre.com/.

[34] Serverless computing. https://en.wikipedia.org/wiki/Serverless_computing.

[35] The solarwinds orion breach, and what you should know. https://blogs.cisco.com/security/the-solarwinds-orion-breach-and-what-you-should-know.

[36] Sphinx: Open source search server. http://sphinxsearch.com/.

[37] State of the dapps. https://www.stateofthedapps.com/rankings.

[38] Steem. https://steem.com/.

[39] Steemit. https://steemit.com/.

[40] Steemit censoring users on immutable social media blockchain's front-end. https://cryptoslate.com/steemit-censoring-users-immutable-blockchain-social-media/.

[41] Stop words - words ignored by search engines. https://www.link-assistant.com/seo-stop-words.html.

[42] Supporting intel sgx on multi-socket platforms. https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions/supporting-sgx-on-multi-socket-platforms.html.

[43] A timeline of facebook's privacy issues — and its responses. https://www.nbcnews.com/tech/social-media/timeline-facebook-s-privacy-issues-its-responses-n859651.

[44] Tital-advancing the era of accelerated computing. https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/.

[45] Welcome to bazaar dog, your scrappy open bazaar search

provider. https://www.bazaar.dog/.
[46] Wix.com: Free website builder | create a free website. https://www.wix.com/.
[47] Yacy - decentralized search engine. https://yacy.net/.
[48] Zeronet. https://zeronet.io/.
[49] Zipcall.io free browser based video calling for everyone. https://zipcall.io/.
[50] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. Obliviate: A data oblivious filesystem for intel sgx. In *NDSS*, 2018.
[51] Fritz Alder, N. Asokan, Arseny Kurnikov, Andrew Paverd, and Michael Steiner. S-faas: Trustworthy and accountable function-as-a-service using intel SGX. In *Proceedings of the ACM Cloud Computing Security Workshop (CCSW)*, 2019.
[52] Enis Ceyhun Alp, Eleftherios Kokoris-Kogias, Georgia Fragkouli, and Bryan Ford. Rethinking general-purpose decentralized computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, 2019.
[53] AMD. AMD Secure Encrypted Virtualization (SEV). https://developer.amd.com/sev/.
[54] ARM. Arm TrustZone Technology. https://developer.arm.com/ip-products/security-ip/trustzone.
[55] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Daniel O'Keeffe, Mark L Stillwell, et al. Scone: Secure linux containers with intel sgx. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.
[56] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM Conference*, 2009.
[57] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 1997.
[58] Juan Benet. Ipfs - content addressed, versioned, p2p file system. *ArXiv*, abs/1407.3561, 2014.
[59] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2017.
[60] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2013.
[61] Stefan Brenner and Rüdiger Kapitza. Trust more, serverless. In *Proceedings of the ACM International Conference on Systems and Storage (SYSTOR)*, 2019.
[62] Jo Van Bulck, M. Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, F. Piessens, M. Silberstein, T. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *USENIX Security Symposium*, 2018.
[63] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *Proceedings of the USENIX Security Symposium*, 2017.
[64] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2015.
[65] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1999.
[66] Stephen Checkoway and Hovav Shacham. Iago attacks: why the system call API is a bad untrusted RPC interface. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.
[67] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. Sgxpectre attacks: Leaking enclave secrets via speculative execution. 2019.
[68] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.
[69] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. Dory: An encrypted search system with distributed trust. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.
[70] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Oblivious coopetitive analytics using hardware enclaves. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2020.
[71] Tien Tuan Anh Dinh, Prateek Saxena, Ee-Chien Chang, Beng Chin Ooi, and Chunwang Zhang. M2r: Enabling stronger privacy in mapreduce computation. In *Proceedings of the USENIX Security Symposium*, 2015.
[72] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.
[73] Y. Gilad, Rotem Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2017.
[74] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and A. A. Yavuz. Hardware-supported oram in effect: Practical oblivious search and update on very large dataset. *Proceedings on Privacy Enhancing Technologies*, 2019:172 – 191, 2018.
[75] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: a distributed sandbox for untrusted computation on secret data. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.
[76] Intel. Intel® 64 and IA-32 Architectures Software Developer Manuals. https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html.
[77] Intel. Strengthen Enclave Trust with Attestation. https://software.intel.com/content/www/us/en/develop/topics/software-guard-

extensions/attestation-services.html.

[78] Zhihao Jia, Oded Padon, James J. Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[79] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[80] Ziliang Lai, Chris Liu, Eric Lo, Ben Kao, and Siu-Ming Yiu. Decentralized search on decentralized web. *ArXiv*, abs/1809.00939, 2019.

[81] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *Proceedings of the USENIX Security Symposium*.

[82] Jinyuan Li, M. Krohn, David Mazières, and D. Shasha. Secure untrusted data repository (sundr). In *OSDI*, 2004.

[83] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. Exploiting unprotected I/O operations in amd's secure encrypted virtualization. In Nadia Heninger and Patrick Traynor, editors, *Proceedings of the USENIX Security Symposium*, 2019.

[84] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter R. Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[85] Prince Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. In *OSDI*, 2010.

[86] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. Kungfu: Making training in distributed machine learning adaptive. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[87] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. Delegatee: Brokered delegation using trusted execution environments. In *Proceedings of the USENIX Security Symposium*, 2018.

[88] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. BITE: bitcoin lightweight client privacy using trusted execution. In *Proceedings of the USENIX Security Symposium*, 2019.

[89] P. Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS*, 2002.

[90] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Proceedings of the International Cryptology Conference (CRYPTO)*, 1987.

[91] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[92] Sonia Ben Mokhtar, Antoine Boutet, Pascal Felber, Marcelo Pasin, Rafael Pires, and Valerio Schiavoni. X-search: revisiting private web search using intel SGX. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, 2017.

[93] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel sgx. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[94] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008.

[95] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2013.

[96] Rafael Pires, David Goltzsche, Sonia Ben Mokhtar, Sara Bouchenak, Antoine Boutet, Pascal Felber, Rüdiger Kapitza, Marcelo Pasin, and Valerio Schiavoni. CYCLOSA: decentralizing private web search through sgx-based browser extensions. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[97] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[98] Do Le Quoc, Franz Gregor, Jatinder Singh, and Christof Fetzer. Sgx-pyspark: Secure distributed data analytics. In *International World Wide Web Conference (WWW)*, 2019.

[99] Sajin Sasy, Sergey Gorbunov, and Christopher W. Fletcher. Zerotrace : Oblivious memory primitives from intel SGX. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[100] Sambhav Satija, Apurv Mehra, Sudheesh Singanamalla, Karan Grover, Muthian Sivathanu, Nishanth Chandran, Divya Gupta, and Satya Lokam. Blockene: A high-throughput blockchain over mobile devices. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[101] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[102] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[103] Srinath T. V. Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[104] Fahad Shaon and Murat Kantarcioglu. Sgx-ir: Secure information retrieval with trusted processors. In *DBSec*, 2020.

[105] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel SGX. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Op-*

erating Systems (ASPLOS), 2020.

[106] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM Conference*, 2001.

[107] Wenhai Sun, Ruide Zhang, Wenjing Lou, and Y. Thomas Hou. REARGUARD: secure keyword search using trusted hardware. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 2018.

[108] Ken Thompson. Reflections on trusting trust. *Commun. ACM*, 1984.

[109] Shruti Tople, Yaoqi Jia, and Prateek Saxena. PRO-ORAM: practical read-only oblivious RAM. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2019.

[110] Bohdan Trach, Oleksii Oleksenko, Franz Gregor, Pramod Bhatotia, and Christof Fetzer. Clemmys: towards secure remote execution in faas. In *Proceedings of the ACM International Conference on Systems and Storage (SYSTOR)*, 2019.

[111] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

[112] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. Conclave: secure multi-party computation on big data. *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2019.

[113] Huibo Wang, Pei Wang, Yu Ding, Mingshen Sun, Yiming Jing, Ran Duan, Long Li, Yulong Zhang, Tao Wei, and Zhiqiang Lin. Towards memory safe enclave programming with rust-sgx. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[114] Xiao Wang, T.-H. Hubert Chan, and Elaine Shi. Circuit ORAM: on tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2015.

[115] David A. Wheeler. Countering trusting trust through diverse double-compiling. In *ACSAC*, pages 33–48. IEEE Computer Society, 2005.

[116] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151, 2014.

[117] Cheng Xu, Ce Zhang, and Jianliang Xu. vchain: Enabling verifiable boolean range queries over blockchain databases. In *Proceedings of the ACM SIGMOD Conference*, 2019.

[118] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[119] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1982.

[120] Ce Zhang, Cheng Xu, Jianliang Xu, Yuzhe Tang, and Byron Choi. GEM^2-Tree: A gas-efficient structure for authenticated range queries in blockchain. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2019.

[121] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2016.

[122] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca A. Popa, Joseph Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[123] Jean Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. Hacl*: A verified modern cryptographic library. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.