Searching an Encrypted Cloud Meets Blockchain: A Decentralized, Reliable and Fair Realization

Shengshan Hu, Chengjun Cai, Qian Wang, Cong Wang, Xiangyang Luo, Kui Ren, and Minghui Li

Abstract

Enabling search directly over encrypted data is a desirable technique to allow users to effectively utilize encrypted data outsourced to a remote server like cloud service provider. So far, most existing solutions focus on an honest-but-curious server, while security designs against a malicious server have not drawn enough attention. It is not until recently that a few works address the issue of verifiable designs that enable the data owner to verify the integrity of search results. Unfortunately, these verification mechanisms are highly dependent on the specific encrypted search index structures, and fail to support complex queries. There is a lack of a general verification mechanism that can be applied to all search schemes. Moreover, no effective countermeasures (*e.g.*, punishing the cheater) are available when an unfaithful server is detected.

In this work, we explore the potential of smart contract in Ethereum, an emerging blockchain-based decentralized technology that provides a new paradigm for trusted and transparent computing. By replacing the central server with a carefully-designed smart contract, we construct a decentralized privacy-preserving search scheme where the data owner can receive correct search results with assurance and without worrying about potential wrongdoings of a malicious server. To better support practical applications, we introduce fairness to our scheme by designing a new smart contract for a financially-fair search construction, in which every participant (especially in the multi-user setting) is treated equally and incentivized to conform to correct computations. In this way, an honest party can always gain what he deserves while a malicious one gets nothing. Finally, we implement a prototype of our construction and deploy it to a locally simulated network and an official Ethereum test network, respectively. The extensive experiments and evaluations demonstrate the practicability of our decentralized search scheme over encrypted data.

I. Introduction

Due to the increasing storage and computation resource demands, today's organizations demonstrate a strong tendency to outsource their data to remote servers like cloud service providers. Since the outsourced data may contain sensitive information, the data owners usually opt to encrypt their data, *e.g.*, financial transactions, medical records, before outsourcing to the server. This in turn hinders the data utilization such as search operations frequently performed.

Since the pioneering work [1] on searchable encryption, much effort has been devoted to designing effective and efficient mechanisms to enable search over encrypted data. In most existing works, the remote server is modeled as an honest-but-curious entity [2], [3] who never tries to deviate from the prescribed protocol. In reality, however, a malicious server may return partial answers or even non-matching documents (*e.g.*, due to random failures). More seriously, any security breach and insider attacker may illegally gain access to alter the computations performed over the data. This could happen when a successful malware infection (*e.g.*, email attachments, infected P2P media) on one host gives an attacker a high access authority. To address these concerns, security designs against a malicious server are urgently needed to facilitate the wide application of encrypted data search.

Recently, a few works have been focused on designing verifiable privacy-preserving search schemes where a data owner is able to verify the integrity of search results. Nevertheless, their verification techniques (*e.g.*, using MAC [4] or hash table [5]) are highly dependent on specific search schemes, and for now only support simple query expressions such as single-keyword search. How to generically impose verifiability on the existing multifarious search schemes that support expressive queries and complex data structures (*e.g.*, similarity search [6] or graph data [3], [7]) remains unclear. More importantly, all existing verifiable search schemes focus on detecting cheating behaviours, with no effective coping countermeasures (*e.g.*, punishing the cheater) followed up, which greatly hinders the wide adoption of their schemes. Taking the universal pay-per-use model for example, in the worst case the data owner may end up with getting an incorrect result while losing money, whereas the malicious server earns money with cheating. This obviously demands a more reliable realization of search schemes over encrypted data with not only verifiability to detect misbehaviours of a malicious server, but also a built-in fairness mechanism to protect the interests of the data owner.

To address the above concerns, we first observe that the main reason of possible cheating is that the centralized server is too powerful without being supervised. Therefore, we propose to resort to smart contract, a newly emerging blockchain-based decentralized computing paradigm in Ethereum [8] where all operations are transparent and reliable. Getting rid of a central server, outsourcing search queries to smart contract yields a correct and immutable result, and requires no further verifications by the data owner. It thoroughly eliminates our misgivings about a malicious adversary as long as the security of Ethereum is guaranteed. To realize this goal, we, for the first time, propose a decentralized privacy-preserving search scheme Π, utilizing the popular decentralized platform: the smart contract in Ethereum. To give an exemplary instantiation, we build Π on the classic inverted index based searchable symmetric encryption schemes [5], [9], and design the corresponding smart contract to circumvent various barriers (*e.g.*, gas limitation) in Ethereum. We emphasize that our framework is a general one, and many other solutions supporting complex expressiveness (*e.g.*, similarity search) and structured data (*e.g.*, graph) fit for our setting as well and can be altered likewise to have their decentralized counterparts.

To further bridge the gap between theoretical viability and practical concerns of privacy-preserving search schemes, we observe that the key incentive mechanism behind such applications in the real world is that the data owner wants to outsource computation-expensive jobs to a worker (*i.e.*, the remote server in cloud settings), and in return the worker obtains a certain amount of monetary compensation accordingly. Therefore, we novelly use smart contract to create a fair reciprocal mechanism, where the data owner receives correct search results as long as he honestly pays the money, and the worker earns the money as long as he faithfully follows the protocol. Moreover, we consider a more complex scenario, *i.e.*, the multi-user setting [2], [10], where the data owner may tend to make a profit by publishing and sharing the database with legitimate users. We thus propose a fair privacy-preserving search scheme Π_{fair}, where a new smart contract is designed to trace monetary rewards, including transaction fees, among involved parties in the multi-user setting. It ensures that the data owner gets paid as long as he reveals the transcript which allows the other users to search the database, and the other users can get correct search results as long as they pay the money.

In summary, we make the following key contributions:

- By leveraging the smart contract in Ethereum, we, for the first time, propose a decentralized privacy-preserving search scheme Π to guarantee that the data owner receives correct search results and has no need to perform verifications in the face of a malicious adversary.
- Based on Π, we novelly introduce the notion of fairness and propose a fair privacy-preserving search scheme Π_{fair} such that each participant, especially in the multi-user setting, is fairly treated and incentivized to conform to correct computations.
- We implement a prototype of our design and deploy it to a locally simulated network and an official Ethereum test network, respectively. Extensive experiments and evaluations are conducted to demonstrate the practicability of decentralized search schemes over encrypted data.

II. Background

A. Smart Contract in Ethereum

Ethereum is a new promising blockchain-based decentralized computing platform [8], [11]. Its security is maintained by a cryptographic chain of puzzles (or blocks). Miners in the Ethereum network validate and approve transactions while mining new blocks. Mining a new block by successfully solving a designated cryptographic puzzle rewards the miners with newly-created cryptocurrency and thus incentivizes them to mine more blocks. The correctness of the network is guaranteed by this incentive mechanism. In general, Ethereum provides us with two appealing properties:

- *Consensus.* The entire network agrees on the rules to verify each transaction and block. The data stored and computations executed on Ethereum must be consistent across miners and cannot be modified or denied.
- *Transparency.* Ethereum is a public network. All the stored data and executed computations are transparent to any users.

Therefore, Ethereum acts as a trusted base which is *trusted for correctness and availability, but not for privacy*.

Smart Contracts in Ethereum are applications with a state stored in the blockchain. They can facilitate, verify, and enforce the process of a contract. Each smart contract, identified by a special address, consists of script code, a currency balance, and storage space in the form of a key/value store. Once created and deployed to Ethereum, the contract's code cannot be modified forever even by its creator.

Gas System is a fantastic feature in smart contract. It is designed to mitigate Denial-of-Service (DoS) attack on the Ethereum network. Specifically, the contract script is compiled into Ethereum opcodes and stored in the blockchain. Each opcode will cost a certain pre-defined amount of *gas* [8]. When initiating a smart contract through sending a transaction, the sender has to specify the available $\mathsf{gasLimit}$ that supports for execution, and the corresponding $\mathsf{gasPrice}$ that the sender is willing to pay for each unit of gas. The transaction will get included in the blockchain successfully only when the balance of the sender is larger than $\mathsf{gasLimit} \times \mathsf{gasPrice}$.

B. Cryptographic Tools

In our constructions, we make use of variable-input-length pseudo-random functions (PRFs) which are polynomial-time computable functions that cannot be distinguished from random functions by any probabilistic polynomial-time adversary. Formal definitions of PRFs can be found in [12].

III. Preliminaries

A. System Overview

In Fig. 1, we outline the architecture of our design. Our scheme consists of the following three algorithms:

- $\mathsf{Setup}(\mathsf{DB})$: *The data owner takes as input a database* DB *and outputs a tuple* $(\mathsf{EDB}, K, \delta)$ *where* EDB *is the encrypted database,* K *is the secret key, and* δ *is the data owner's state.*

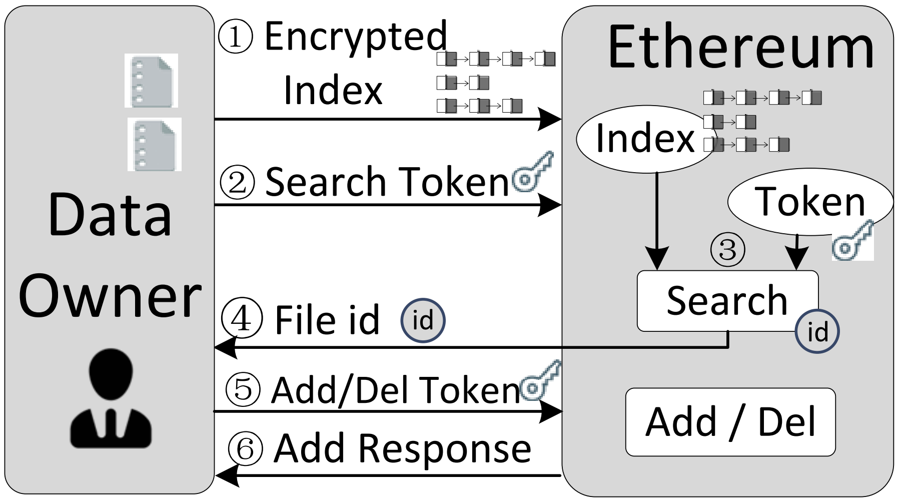


Fig. 1. A system overview for our scheme Π.

- $\mathsf{Search}(K, \delta, w; \mathsf{EDB})$: *The data owner takes as input the secret key K, its state δ, and a search word $w \in \{0,1\}^*$, and the smart contract takes as input the encrypted database EDB. The smart contract outputs a set of identifiers while the data owner has no output.*
- $\mathsf{Update}(K, \delta, \mathsf{op}, \mathsf{id}, \mathsf{W}_{\mathsf{id}}; \mathsf{EDB})$: *The data owner takes as input the key K, the state δ, an operation $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$, a file identifier id, and a set W_{id} of distinct keywords, and the smart contract takes as input EDB. These inputs represent the actions of adding or deleting a file with identifier id.*

For ease of presentation, operations on the data documents are not shown in the framework since the data owner could easily employ the traditional symmetric key cryptography to encrypt the data and then outsource encrypted data to any decentralized file storage network like InterPlanetary File System (IPFS).

B. Notations

A database $\mathsf{DB} = (\mathsf{id}_i, \mathsf{W}_i)_{i=1}^d$ is a list of identifier-keyword pairs where $\mathsf{id}_i \in \{0,1\}^l$ and $\mathsf{W}_i \subseteq \{0,1\}^*$. The set of keywords of the database DB is $\mathsf{W} = \cup_{i=1}^d \mathsf{W}_i$. The set of documents containing a given keyword $w \in \mathsf{W}$ is denoted by $\mathsf{DB}(w) = \{\mathsf{id}_i | w \in \mathsf{W}_i\}$. We will always set $m = |\mathsf{W}|$ and $N = \sum_{w \in \mathsf{W}} |\mathsf{DB}(w)|$ to be the number of distinct keywords and the total number of keyword-document pairs, respectively.

C. Design Goals

Fairness. The fairness for privacy-preserving search is the key new property introduced in this paper. Our core observation is based on the financial nature of the incentive mechanism behind such applications in the real world. Loosely speaking, our notion of fairness guarantees:

- The data owner receives correct search results as long as he pays for his search jobs that are delegated to a worker, while the worker earns money as long as he honestly follows the protocol.
- In the multi-user setting, in addition to the above requirements, the other users receive correct search results as long as they pay for both their search jobs that are delegated to a worker and the access to the data owner's data. The data owner earns money as long as he reveals the transcript (*e.g.*, search token).

In summary, fairness guarantees that each party involved is incentivized to do correct computations. If a party deviates from the protocol, then he gets nothing.

Soundness. This property basically indicates that the server will get caught if it tries to deviate from the protocol. Usually the existing works achieve this objective by letting the data owner conduct a series of verifications. In this paper, we extend this notion to claim that the received search results are reliable and correct definitely, and thus no verification is needed on the data owner.

Confidentiality. We should protect the confidentiality of data files or the query keywords from the adversary. Besides, we aim to provide another strong security notion: forward privacy, indicating that the adversary does not learn if the newly-added document contains a keyword that has been searched before.

IV. Design Challenges and Countermeasures

Intuitively, any existing privacy-preserving search schemes can be directly adapted to decentralized environment by replacing the central server with smart contracts. Unfortunately, some innovative features that guarantee the robustness and security of smart contracts become obstacles instead in this adaption. In this section we present some main design challenges and summarize the countermeasures at a high level.

A. Gas System

Gas Limitation. In Ethereum, each transaction that calls a function of a smart contract has an upper bound of consumed gas, called $\mathsf{gasLimit}$ as described in Section II-A. Each operation, including sending/storing data and executing computations, has a fixed gas cost. This restricts the designed function to have extremely limited computation steps and storage. Therefore, to make the privacy-preserving search over a large database feasible, we are motivated to divide the database into smaller ones and manage them individually. The general idea is as follows. In the setup phase where a large encrypted index is built, we partition the encrypted index into several blocks and upload them to the contract with sufficient transactions such that each transaction consumes less gas than $\mathsf{gasLimit}$. To ensure correctness, the contract needs to align the data together in order to return all matched results.

Gas Availability. In the smart contract, each transaction is also associated with a $\mathsf{gasPrice}$ that specifies the money the sender is willing to spend to purchase the gas. It is required that the user who initiates the transaction has an account balance larger than the gas cost for executing the transaction. Otherwise the transaction will abort intermediately while the consumed gas cannot be refunded. Thus we should be very careful with the contract design with regard to gas cost. On the one hand, it is critical to ensure that each functionality (*e.g.*, Search, Update) in the contract incurs lower gas cost than the sender's account balance. On the other hand, we should keep pace with gas consumption in each phase such that fairness can be guaranteed eventually. The need for gas availability indicates our protocol design in Section VII.

B. The Verifier's Dilemma

In Ethereum, miners are required to check the validity of transactions. However, verifying transactions may become significantly expensive when there are abundant and complex expressions in the smart contract. For rational miners, they are thus incentivized to skip the verification of the expensive transactions so as to stay ahead in the race to mine the next block. This phenomenon is called the *verifier's dilemma* [13]. To mitigate this attack, we are motivated to reduce the computation burden on the contract as much as possible.

Our first observation is that smart contracts support dictionary data type, and the main computation overhead lies in the search phase. In light of this, we make use of dictionary to store the encrypted index (*i.e.*, EDB) that leads the search time complexity to be $O(d_w)$, where d_w is the number of times that the keyword w has been historically added to the database.

Our second optimization is the utilization of packing method inspired by [9]. Specifically, we can pack multiple plaintexts and encrypt the output to obtain one ciphertext with the same size. The search result is thus in blocks instead of individuals. Besides, packing also helps us circumvent the above *Gas Limitation* since it greatly reduces the storage cost. We note that although [9] claimed to use the packing method as well, it didn't describe how to implement it explicitly. Our detailed construction can be found in Section V.

V. Our Decentralized Construction

In this section, we construct a decentralized privacy-preserving search scheme Π that achieves soundness as well as confidentiality. Π is built upon existing pioneering inverted index frameworks (such as [5], [9]). We will show that our design rationale can also be applied to other encrypted search schemes with expressive queries or complex data types in Section VIII.

A. Basic Construction

In Fig. 2, we give a formal description for Π. For simplicity, let $F : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, $G : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^*$ be two pseudo-random functions (Note that there should be different PRFs for different input keys). We use $||$ to denote the concatenation operation. "$\lfloor\cdot\rfloor$" is a floor function, and "$|\cdot|$" denotes the number of elements in a list. For a dictionary data type, it includes two algorithms: Add and Delete. And we use term Get to fetch the specified data item in a dictionary. For example, given a dictionary data type γ and an input label l, $\mathsf{Get}(\gamma, l)$ outputs the corresponding item $d||r$ and parses it into d and r.

In the Setup phase, the data owner divides the $\mathsf{DB}(w)$ into $\alpha + 1$ blocks, with each block of p entries. Here p is a system parameter chosen by the data owner. We use concatenation to pack multiple file identifiers into one. To ensure confidentiality, the bit length of $\widetilde{\mathsf{id}}$ should be less than that of the security parameter λ. Therefore, we have $p \leq \frac{\lambda}{l}$, where l is the bit length of the file identifier. Note that before uploading the database, the list L should be placed in lexicographic order. Otherwise it will leak information about the order in which the input was processed. To avoid exceeding $\mathsf{gasLimit}$, we partition the encrypted database into n blocks and send them to the contract one by one with n different transactions. At the contract side, they are

```
Setup($\mathsf{DB}$):
  1) The data owner initializes an empty list $\mathsf{L}$, and an empty dictionary $\sigma$, and samples three keys $K, K^A, K^D \xleftarrow{\$} \{0,1\}^\lambda$.
  2) For each keyword $w \in \mathsf{W}$:
     a) $K_1 \leftarrow F(K, 1||w)$; $K_2 \leftarrow F(K, 2||w)$;
     b) Set $\alpha \leftarrow \lfloor \frac{|\mathsf{DB}(w)|}{p} \rfloor, c \leftarrow 0$, where $p$ denotes the number of file identifiers that can be packed.
     c) Divide $\mathsf{DB}(w)$ into $\alpha + 1$ blocks. Pad the last block to $p$ entries if needed.
     d) For each block in $\mathsf{DB}(w)$:
        - $\widetilde{\mathsf{id}} \leftarrow \mathsf{id}_1||\mathsf{id}_2||...||\mathsf{id}_p$; $r \xleftarrow{\$} \{0,1\}^\lambda$; $d \leftarrow \widetilde{\mathsf{id}} \oplus G_{K_2}(r)$; $l \leftarrow F(K_1, c)$; $c++$.
        - Add $(l, d, r)$ to the list $\mathsf{L}$ in lex order.
  3) Set $\mathsf{EDB} = \mathsf{L}$; Partition $\mathsf{EDB}$ into $n$ blocks $\mathsf{EDB}_i$ for $1 \leq i \leq n$, and send them to the smart contract.
  4) The smart contract initializes two empty dictionaries $\gamma$ and $\gamma^A$, and an empty list $\mathsf{ID}_{\mathsf{del}}$.
  5) For each received $\mathsf{EDB}_i$, the smart contract parses each entry in $\mathsf{EDB}_i$ into $(l, d, r)$, and adds each $(l, d||r)$ to $\gamma$.

Search($K, K^A, K^D, w$):
  1) $K_1 \leftarrow F(K, 1||w)$, $K_2 \leftarrow F(K, 2||w)$, $K_1^A \leftarrow F(K^A, 1||w)$, $K_2^A \leftarrow F(K^A, 2||w)$, $K_1^D \leftarrow F(K^D, w)$.
  2) The data owner sets $c \leftarrow 0$, and estimates $R$ and $\mathsf{step}$.
  3) For $i = 0$ to $R$:
     a) Send search token $ST = (K_1, K_2, K_1^A, K_2^A, K_1^D, c)$ to the smart contract; Set $c \leftarrow c + \mathsf{step}$.
  4) The smart contract asserts that the estimated gas cost is lower than the balance, and then:
     a) For $i = 0$ until $\mathsf{Get}$ returns $\perp$ or $i \geq \mathsf{step}$:
        - $l \leftarrow F(K_1, c)$; $d, r \leftarrow \mathsf{Get}(\gamma, l)$; $\widetilde{\mathsf{id}} \leftarrow d \oplus G_{K_2}(r)$; $c++$; $i++$.
        - Parse $\widetilde{\mathsf{id}}$ into $(\mathsf{id}_1, \cdots, \mathsf{id}_p)$; Assert $\mathsf{id}_j \notin \mathsf{ID}_{\mathsf{del}}$ $(1 \leq j \leq p)$ and save $\mathsf{id}_j$ to the state.
     b) Assert $\gamma^A$ has not been searched.
     c) For $c = 0$ until $\mathsf{Get}$ returns $\perp$:
        - $l \leftarrow F(K_1^A, c)$; $d, r \leftarrow \mathrm{Get}(\gamma^A, l)$; $\mathsf{id} \leftarrow d \oplus G_{K_2^A}(r)$; $c++$;
        - Assert $\mathsf{id} \notin \mathsf{ID}_{\mathsf{del}}$ and save $\mathsf{id}$ to the state.

Add($K, K^A, K^D, \mathsf{id}, \mathsf{W}_{\mathsf{id}}$) :
  1) The data owner initializes an empty list $\mathsf{L}^A$, and then:
     a) For each keyword $w \in \mathsf{W}_{\mathsf{id}}$:
        - $K_1 \leftarrow F(K, 1||w)$; $K_2 \leftarrow F(K, 2||w)$; $K_1^A \leftarrow F(K^A, 1||w)$; $K_2^A \leftarrow F(K^A, 2||w)$; $K_1^D \leftarrow F(K^D, w)$.
        - $r \xleftarrow{\$} \{0,1\}^\lambda$; $c \leftarrow \mathsf{Get}(\sigma, w)$; If $c = \perp$ then $c \leftarrow 0$; $l \leftarrow F(K_1^A, c)$; $d \leftarrow \mathsf{id} \oplus G_{K_2^A}(r)$; $\mathsf{id}_{\mathsf{del}} \leftarrow F(K_1^D, \mathsf{id})$.
        - Add $(l, d, r, \mathsf{id}_{\mathsf{del}})$ to $\mathsf{L}^A$ in lexicographic order.
     b) Send $\mathsf{L}^A$ to the contract.
  2) The smart contract initializes an empty list $\mathsf{re}$ of size $|\mathsf{L}^A|$, and parses each tuple of $\mathsf{L}^A$ into $(l, d, r, \mathsf{id}_{\mathsf{del}})$, set $i \leftarrow 0$.
  3) For each tuple in $\mathsf{L}^A$:
     a) if $\mathsf{id}_{\mathsf{del}} \in \mathsf{ID}_{\mathsf{del}}$, then $\mathsf{re}[i] \leftarrow 1$ and delete $\mathsf{id}_{\mathsf{del}}$ from $\mathsf{ID}_{\mathsf{del}}$, else $\mathsf{re}[i] \leftarrow 0$ and add $(l, d||r)$ to $\gamma^A$; $i++$.
  4) The data owner reads $\mathsf{re}$ from the smart contract, and then:
     a) For $i = 0$ to $|\mathsf{re}|$:
        - if $\mathsf{re}[i] = 0$ then fetch the $i$-th keyword $w$ in $\mathsf{W}_{\mathsf{id}}$; $c \leftarrow \mathsf{Get}(\sigma, w)$; c++; Insert $(w, c)$ into $\sigma$.

Delete($K^D, \mathsf{id}, \mathsf{W}_{\mathsf{id}}$):
  1) The data owner initializes an empty list $\mathsf{L}^D$, and then:
     a) For each keyword $w \in \mathsf{W}_{\mathsf{id}}$:
        - $K_1^D \leftarrow F(K^D, w)$, $\mathsf{id}_{\mathsf{del}} \leftarrow F(K_1^D, \mathsf{id})$; Add $\mathsf{id}_{\mathsf{del}}$ to $\mathsf{L}^D$ in lex order.
  2) Send $\mathsf{L}^D$ to the contract.
  3) The smart contract adds $\mathsf{id}_{\mathsf{del}}$ to $\mathsf{ID}_{\mathsf{del}}$ for each element $\mathsf{id}_{\mathsf{del}}$ in $\mathsf{L}^D$:
```

Fig. 2. Construction of our decentralized privacy-preserving search scheme Π.

received iteratively and placed together using the dictionary data type. Similarly, the search process will be completed with R transactions, each of which returns step items at most. Here n, R and step are public system parameters and experimentally determined.

In the Search phase, for each query, the data owner sends a transaction containing the search token to the designated smart contract. Note that each contract has a unique address in Ethereum. With the search token and previously stored index, the smart contract executes search algorithms and saves the search results (*i.e.*, file identifiers) to its state, which is known publicly including the data owner.

B. Supporting Dynamic Updates

Π supports dynamic updates as well. In the Add phase, we encrypt file id without using packing. This is because encrypting several plaintexts into one ciphertext makes it hard for the contract to identify which file-keyword pair has been previously deleted, *i.e.*, whether it exists in the set $\mathsf{ID}_{\mathsf{del}}$. In addition, in reality changes often happen with only one or several documents at one time. Update incurs much less gas cost than the *Gas Limitation.* Therefore, individually dealing with file id satisfies the system requirements for update operations. For the protocol on the smart contract, we remark that transaction triggering functions in the smart contract doesn't return any results. Execution of any function only changes its state that is permanently stored on Ethereum. We implement our scheme by saving search results into the state and later reading them on the data owner side.

C. Forward Privacy

Forward privacy is an important security design goal in the literature. It means that the adversary does not learn if the newly-added document contains a keyword that has been searched before. Inspired by recent progress [5], Π can be easily extended to achieve forward privacy. The key idea is to use trapdoor permutation to make the search token unlinkable to the update token. Specifically, when generating a label for the c-th entry in $\mathsf{DB}(w)$, instead of using a counter c that increases itself, we use a trapdoor permutation π in a way that $\beta_c = \pi_{sk}^{-1}(\beta_{c-1})$ and set the label as $l = F(K, \beta_c)$ where β_0 is a randomly-chosen integer. Then on the smart contract, it can only compute $\beta_{c-1} = \pi_{pk}(\beta_c)$ with the public key in polynomial time, but not β_{c+1} since it has no secret key. Therefore, the $(c+1)$-th entry newly-added to $\mathsf{DB}(w)$ without having been searched cannot be deduced from previously-leaked search token β_c. This variant has the same communication complexity with Π, and the computation overheads on the data owner and the contract increase a little caused by permutation computation.

VI. Security Proof

Soundness: It is straightforward to see that our scheme Π achieves soundness as long as the security of Ethereum is guaranteed. This is because if the smart contract is correctly executed on Ethereum, the search results will be stored as contract states permanently and publicly. Each miner in the Ethereum network can verify the data. The *consensus* property of Ethereum ensures the correct execution of each search operation.

Confidentiality: To prove confidentiality, we follow the real-ideal simulation paradigm [9] and first proceed with the formal definition of three stateful leakage functions $\mathcal{L} = (\mathcal{L}_1,\ \mathcal{L}_2,\ \mathcal{L}_3)$ considered in our construction.

- (*Leakage function* $\mathcal{L}_1$). Given an initial input DB, $\mathcal{L}_1(\mathsf{DB}) = \sum_{w\in\mathsf{W}}\lceil\frac{|\mathsf{DB}(w)|}{p}\rceil$. Meanwhile, it initializes a counter $i = 0$, an empty list Q, a set ID containing all the identifiers in DB, and saves them as the state.
- (*Leakage function* $\mathcal{L}_2$). Given a search input w, $\mathcal{L}_2(\mathsf{in}) = \{\mathsf{sp}(w,Q), \mathsf{DB}(w), \mathsf{AP}(w,Q,\mathsf{ID}), \mathsf{DP}(w,Q,\mathsf{ID})\}$, where $\mathsf{sp}(w,Q)$ denotes the search pattern, $\mathsf{AP}(w,Q,\mathsf{ID})$ (resp. $\mathsf{DP}(w,Q,\mathsf{ID})$) denotes the add (resp. deletion) pattern of the keyword w with respect to Q and ID, all of which are defined below. Meanwhile, it increases i and appends (i, search, w) to Q.
- (*Leakage function* $\mathcal{L}_3$). Given an add update input $(\mathsf{id}, \mathsf{W}_{\mathsf{id}})$, $\mathcal{L}_3(\mathsf{in}) = \{\mathsf{add}, |\mathsf{W}_{\mathsf{id}}|, (\mathsf{sp}(w,Q), \mathsf{ap}(\mathsf{id},w,Q),$ $\mathsf{dp}(\mathsf{id},w,Q)) : w \in \mathsf{W}_{\mathsf{id}}\}$, where $\mathsf{ap}(\mathsf{id},w,Q)$ (resp. $\mathsf{dp}(\mathsf{id},w,Q)$) denotes the add (resp. deletion) pattern of id, w with respect to Q, both of which are defined below. Meanwhile, it increases i, appends $(i, \mathsf{add}, \mathsf{id}, \mathsf{W}_{\mathsf{id}})$ to Q and adds id to ID. For a delete update input, the only difference is that $\mathcal{L}_3(\mathsf{in})$ outputs del instead of add as the first component. Finally, if any of the search patterns was non-empty, then it also outputs id.

Theorem 1. *If G and F are pseudo-random, then our scheme Π is $\mathcal{L}$-secure against non-adaptive attacks.*

Proof Sketch: We describe a polynomial-time simulator $\mathcal{S}$ such that for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the outputs of its real execution $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda)$ and simulated execution $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\lambda)$ are computationally indistinguishable. The simulator $\mathcal{S}$ is given leakage $\mathcal{L}$ to simulate the view of the adversary via imitating the real protocol. For example, to simulate the initial EDB, $\mathcal{S}$ first chooses the keys $\widetilde{K_1}, \widetilde{K_2}, \widetilde{K_1^A}, \widetilde{K_2^A}, \widetilde{K_1^D}$ for each search at random with repetitions specified by the search pattern. For all file ids associated with each search keyword w (*i.e.*, $\mathsf{id} \in \mathsf{DB}(w)$), $\mathcal{S}$ computes l, d and r as specified in real Setup (using $\widetilde{K_1}$ and $\widetilde{K_2}$ as K_1 and K_2), adds each pair (l,d,r) to a list L, and then adds random pairs to L (still maintained in lexicographic order) until it has $\sum_{w\in\mathsf{W}}\lceil\frac{|\mathsf{DB}(w)|}{p}\rceil$ total elements, and finally creates a dictionary $\widetilde{\gamma}$. Similarly, $\mathcal{S}$ is able to simulate the search/add/delete queries from leakage functions $\mathcal{L}$. Our theorem thus holds due to the

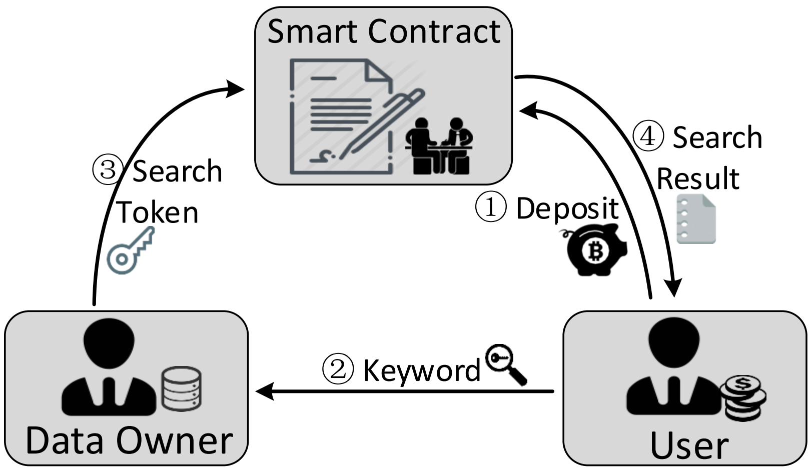


Fig. 3. System model for our fair design in the multi-user setting.

pesudo-randomness of F and G. Π can be easily extended to achieve security against *adaptive* attacks by making use of random oracle [9]. Formal proof is omitted here due to space limitation. □

VII. Enabling Fairness

In this section, we show how to use smart contract to construct a fair privacy-preserving search scheme based on Π. Our definition of fairness is a variant from that in secure multiparty computation [14], and is inspired from recent works on *financial fairness* [15]. We propose to achieve the goal that each party is financially incentivized to do correct computations, and a dishonest party earns nothing if cheating in the end.

A. Single-user Setting

Our primary observation for fairness in privacy-preserving search schemes is that the key incentive mechanism behind such applications in reality is letting the worker (*i.e.*, the server in cloud settings) get monetary rewards in return for processing the delegated search jobs. In light of this, all existing schemes may suffer from the situation where a malicious worker can deviate from the protocol while still earning money if the data owner has paid first. On the other hand, a greedy data owner also tries to make the worker perform search job without paying first. As a result, the worker may gain nothing for his effort.

Our basic scheme Π, on the other hand, has naturally guaranteed fairness. This is because whatever operations (*e.g.*, Search,Update) the data owner wants to conduct, he has to pay cryptocurrency called Ether to the worker (*i.e.*, the miner in Ethereum) to purchase *gas*. And the smart contract is automatically executed on each miner and sets the state with correct operation results, which can be read by the data owner.

We stress that existing verifiable search schemes [4], [5], [16] do not support fairness. They usually let the data owner pay first before the worker handles the query and then verify whether the results are correct. In this case, however, the worker can end up with earning the money while deviating the protocol. This is particularly unsatisfactory in an environment where financial issues are involved.

B. Multi-user Setting

In the multi-user setting, where the data owner allows a third party (*i.e.*, other authorized users) to search the database [2], [10], things get much more complicated. This is due to the fact that the user is another mutually untrusted party, and we need to ensure each party is fairly treated. Specifically, we need to ensure: 1) the data owner gets paid if the user searches the database; 2) the user gets correct search results if he has paid the money. To this end, we modify the protocol Π to construct our fair privacy-preserving search scheme Π_{fair}.

Overview. Fig. 3 gives an overview of Π_{fair}. Since everything on the smart contract is public, using existing techniques such as broadcast encryption [10] to add and revoke users is not applicable because they require the worker to store a private key. Thus we use the straightforward extension as indicated in [2]: the data owner receives the user's query, and generates the corresponding search tokens as if himself is searching the database.

Notation. Table I presents the primary notations for Π_{fair}. We use \$ to denote the cryptocurrency (*i.e.*, Ether). $\$\mathcal{B}_{\mathsf{owner}}$ and $\$\mathcal{B}_{\mathsf{user}}$ are unique Ethereum account balances of the data owner and the user, respectively. The data owner sets a price $\$\mathsf{offer}$ for each search, and the user makes a deposit $\$\mathsf{deposit}$ for each search. The gas cost for executing search $\mathsf{G}_{\mathsf{srch}}$ is a system constant. The gas limit for search $\mathsf{GL}_{\mathsf{srch}}$ and the price for each unit of gas $\$\mathsf{gasPrice}$ are specified by the data owner.

TABLE I
NOTATIONS FOR Π_{FAIR}.

$\$\mathcal{B}_{\mathsf{owner}}$	Balance of the data owner.
$\$\mathcal{B}_{\mathsf{user}}$	Balance of the user.
$\$\mathsf{deposit}$	Deposit currency by the user.
$\$\mathsf{gasPrice}$	Price for each unit of gas.
$\$\mathsf{offer}$	Price for each search offered by the data owner.
$\mathsf{GL}_{\mathsf{srch}}$	Gas limit for calling Search() function.
$\mathsf{G}_{\mathsf{srch}}$	Gas cost for calling Search() function.

Fair scheme Π_{fair}: Protocol on smart contract.

```
FSetup():
  1) Do Setup() as in $\Pi$.
  2) The data owner sets a price $\$\mathsf{offer}$ for each search.
  3) The user makes a deposit $\$\mathsf{deposit}$ from $\$\mathcal{B}_{user}$.
  4) The user sets a time limitation $T_1$.

FSearch(ST):
  1) Assert that the transaction sender is the data owner.
  2) Assert current time $T < T_1$.
  3) Assert $\$\mathsf{deposit} > \mathsf{GL}_{\mathsf{srch}} \times \$\mathsf{gasPrice} + \$\mathsf{offer}$.
     a) Call $\mathsf{Search}(ST)$.
     b) Set $\$\mathsf{cost} \leftarrow \$\mathsf{offer} + \mathsf{G}_{\mathsf{srch}} \times \$\mathsf{gasPrice}$.
     c) Send $\$\mathsf{cost}$ to $\$\mathcal{B}_{\mathsf{owner}}$.
     d) Set $\$\mathsf{deposit} \leftarrow \$\mathsf{deposit} - \$\mathsf{cost}$.
     e) Send $\$\mathsf{deposit}$ to $\$\mathcal{B}_{\mathsf{user}}$.
  4) Assert current time $T > T_1$.
     a) Send $\$\mathsf{deposit}$ to $\$\mathcal{B}_{\mathsf{user}}$.
```

Fig. 4. Fair decentralized privacy-preserving search scheme.

Contract Design. Fig. 4 shows the contract design for Π_{fair}. The $\mathsf{Search}()$ part of Π_{fair} is exactly the same with Π. For simplicity here we only use it as a subroutine, and use ST to denote the received search token and omit other details.

To ensure fairness, we set a time limitation T_1 specified by the user. Within T_1, the data owner is able to send the search token to the contract and earn money. Beyond T_1, the search request by the user gets expired and the user's deposit will be refunded. Note that one of the important conditions that allows the search is that the deposit should be larger than the price of the data owner's offer adding with the gas cost for executing $\mathsf{Search}()$ function. This is because the query transaction is initiated by the data owner, and the gas cost will be deducted from the data owner's account $\$\mathcal{B}_{\mathsf{owner}}$. This is unfair since the data owner triggers query transaction in place of the user. Therefore, the user has to pay extra fees to cover the gas consumption. When the search terminates, the cost is sent to $\$\mathcal{B}_{\mathsf{owner}}$. In particular, the remaining deposit should be refunded to the user account $\$\mathcal{B}_{\mathsf{user}}$ immediately to prevent the data owner from running away with the deposit, by repeatedly sending the query transaction with the same search token.

In our construction, the data owner can also send the search token to the user off-line and let the user himself initiate the search transaction. In this case, the contract needs a slight modification: $\$\mathsf{deposit}$ only needs to be larger than $\$\mathsf{offer}$, and $\$\mathsf{cost}$ is set to be equal to $\$\mathsf{offer}$. We recommend, however, transmitting and recording search token through smart contract since it makes any cheating evident, and the contract can financially punish the data owner if he doesn't reveal the search token.

VIII. Generalization of Our Framework

Recent works on privacy-preserving search have focused on increasing their expressiveness such as supporting similarity search [6] or developing structured encryptions like graph encryption [3], [7]. All of them are also bothered with a serious security challenge: a malicious central server can output partial or even incorrect results whenever it wants. To address this concern, these works can be tuned into our decentralized setting likewise as long as the design challenges proposed in Section IV are well addressed. The most intuitive observation of this extension is that smart contract actually provides us with a trusted and transparent "server". The main obstacle lies in dealing with various limitations of gas system in smart contract. Our proposed

TABLE II
EVALUATION DATABASE SIZES.

DB name	(ω, id) pairs	distinct keywords	EDB
DB1	$100,763$	$22,673$	5.4MB
DB2	$300,617$	$54,980$	14.1MB
DB3	$500,567$	$75,924$	21.3MB
DB4	$1,000,141$	$123,912$	39MB

TABLE III
EVALUATIONS IN TESTRPC.

DB name	Setup			Search			Update		
	D.O. time	S.C. time	#Tx	D.O. time	S.C. time	#Tx	D.O. time	S.C. time	#Tx
DB1	$9s$	$23min$	350	$\approx 1ms$	$7s$	1	$\approx 1ms$	$10s$	1
DB2	$15s$	$66min$	1126	$\approx 1ms$	$8s$	1	$\approx 1ms$	$10s$	1
DB3	$18s$	$114min$	1703	$\approx 1ms$	$10s$	1	$\approx 1ms$	$10s$	1
DB4	$23s$	$949min$	3101	$\approx 1ms$	$16s$	1	$\approx 1ms$	$10s$	1

several countermeasures (*e.g.*, dividing the encrypted index and conquering them individually, packing multiple identifiers) throw light on how to address this issue. Once constructed using smart contract, soundness is naturally guaranteed and there is no need to concern itself with a malicious server anymore. In addition, our designed smart contract in Fig. 4 provides a template for all the multi-user settings to achieve fairness. Based on this, we are able to refine various complex conditions in the contract according to specific real circumstances. For instance, in a scenario where the data owner would like to share his photos with friends [10], the price $\$\mathsf{offer}$ can be set to be 0 letting others freely search the database.

IX. IMPLEMENTATION AND EVALUATIONS

We implement a prototype of Π using a bit more than 5000 lines of code, including the test program. We instantiate the data owner on a machine with 16GB of RAM, 4 Intel cores i7-3770, running Ubuntu 16.04.2. The smart contract is deployed to a local simulated network *TestRPC* and also an official Ethereum test network *Rinkeby*, respectively. The data owner side and the smart contract are written in Python and using Solidity in combination with Javascript as the intermediate interactive language, respectively. We don't particularly present the results for Π_{fair} since it has comparable performance with Π in terms of various evaluation metrics (*e.g.*, time cost), and Π_{fair} can be easily implemented by imposing a series of control conditions on Π, without introducing notable costs.

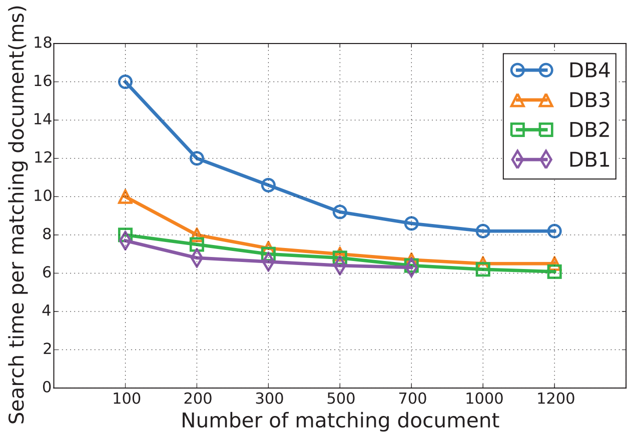
A. Implementation Details

We implement PRF and random oracles using HMAC-SHA256. Since Ethereum currently does not support HMAC instantiation, we follow the standard construction of HMAC [12] and implement HMAC-SHA256 using Python and Solidity, respectively. To avoid exceeding $\mathsf{gasLimit}$, in the setup phase the encrypted database EDB is divided into n subsets and sent to the smart contract with n transactions. Due to the time-varying nature of $\mathsf{gasLimit}$, we experimentally include 70 entries from the list L in each transaction and set the pack number to be $p = 8$. In addition, each search query is also completed with R transactions at most, each of them returns $\mathsf{step} = 47$ items at most. In our experiments, $R = 4$ satisfies our requirements. We use datasets derived from Enron emails which are a collection of plain text files. We extract a subset of emails and select increasing subsets from the original subset as document collections with different numbers of (w, id) (*i.e.*, keyword/identifier) pairs. The key attributes of these datasets are summarized in Table II.

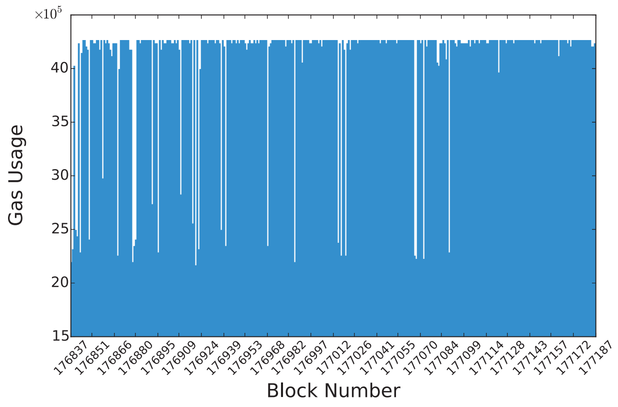
B. Experiments on Simulated Network

To demonstrate the scalability and unique performance characteristics of our design, we first use *TestRPC* to construct a simulated Ethereum network locally. *TestRPC* is initialized with the default configuration, which is much like real Ethereum environment except for that its block time for mining is set to be 0. This allows us to focus on the performance of search part on smart contract, irrespective of time-consuming mining process and complex network circumstances (*e.g.*, broadcast latency, transaction mining delay) in Ethereum.

Table III presents an overview of time costs and transaction numbers for each phase on different datasets. Here D.O. and S.C. represent the time costs on the 'Data Owner' and 'Smart Contract', respectively. #Tx stands for the number of transactions needed to complete the corresponding phase. Search time is evaluated by returning 100 matched documents. Update overheads are given by adding and deleting a file, the size of which is chosen to incur only one transaction. In the setup phase, different from existing centralized search schemes where the data owner side dominates the efficiency, the time cost on smart contract is much higher than that on the data owner. This is because storing EDB is completed with thousands of transactions, with each transaction costing 4 seconds on average to be manipulated.

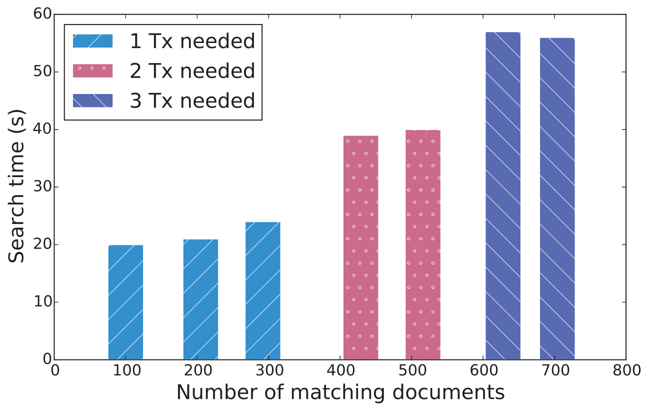


(a) Search time per matching document in TestRPC.

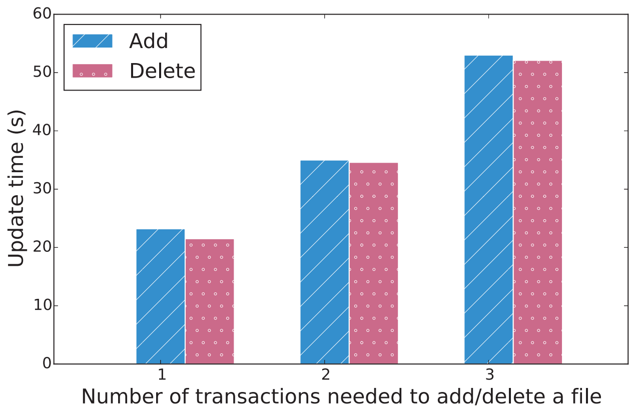


(b) Setup: Gas usage of each mined block in Rinkeby.

Fig. 5. Efficiency evaluations in TestRPC and gas usage in Rinkeby.



(a) Search time vs. the number of matching documents



(b) Update time vs. the number of transactions.

Fig. 6. Efficiency evaluations in Rinkeby.

To show the core algorithm, Fig. 5(a) presents the search time per found document varying with the number of matching records. We report average run times over 30 trials. The first thing we can notice is that a larger result set yields a lower search overhead (on a per matching document basis). We explain that by the constant cost of loading past mined blocks from disk into memory before each search runs. This also explains our second observation: the larger the dataset, the slower the search algorithm is. This is because a larger number of mined blocks leads to a longer time for loading.

C. Experiments on Official Test Network

To show the practicability of our scheme, we deploy Π to the official Ethereum test network *Rinkeby* that mimics the real production network. Due to the limited balance, we only conduct experiments on the smallest database DB1. Our account and contract addresses in Rinkeby are

- 0x7aef688b95a1bee573d464766b3a6c0470b9b57b.
- 0xecE97a98Da7f5DBECcb81E772dD04710e676Aa96.

To illustrate the impact of mining process on the efficiency, we record the block number of each transaction generated in our setup phase and the corresponding gas usage, as shown in Fig. 5(b). In summary, it consists of 350 transactions, each of which is mined into one block with block number ranging from $176,837$ to $177,187$. The average block time for mining is $15s$, resulting in $88min$ to complete the entire setup phase. This again explains why the time cost of setup is dominated by the smart contract, instead of the data owner like in existing centralized search schemes. Besides, the average gas usage for a transaction is $4,201,232$. Currently 1 gas costs about 1.8×10^{-8} Ether, at the exchange rate of 89 USD at the time of writing. So each transaction costs about 0.076 Ether (or 6.7 USD). At first glance, such cost may seem a little expensive for ordinary users for ordinary searches, however, Π still applies to many real-world scenarios. For instance, personal medical profiles are of significant importance for everyone. Getting a correct and complete medical history of a patient enables the doctor to make a precise disease diagnosis and health evaluation. In these demanding cases, it is worth spending more for the sake of retrieving highly reliable data.

Fig. 6(a) shows the total time needed to perform a search, given a search token (we neglect the cost of generating a search token since it is a small constant in microseconds). Each point is the mean of 10 executions. It clearly demonstrates the performance bottleneck of Π. To be specific, we can see that the search time grows with the increase of the number of matching documents. But the sharp growth lies in the increase of the transaction number needed to complete the search step. It indicates that the time cost of mining each transaction dominates the overhead for each search. On the contrary, search algorithm has a faint impact on the efficiency. Generally the time cost of mining process is dynamically adjustable. When the blockchain environment scales to allow a higher gas limitation or a faster mining process, our search efficiency increases as well.

A similar situation occurs in Fig. 6(b) which describes time costs varying with the number of transactions needed to add/delete a file. By choosing different sizes of files, we have update completed with different numbers of transactions. It again shows that the mining process for each transaction is the dominant factor on the efficiency.

X. Conclusion

Traditional privacy-preserving search schemes rely on a central server to manipulate search jobs. In this work, we resort to blockchain technologies and construct a decentralized design aiming at addressing malicious adversary. Different from existing verifiable schemes, our search results are correct and immutable, and no verifications are needed on the data owner side. Besides, we make use of smart contract to construct a fair privacy-preserving search scheme where each party, in the multi-user setting particularly, is fairly treated and incentivized to do correct computations. Experimental results obtained on our prototype demonstrate the practicability of our scheme.

Acknowledgment

Qian and Kui's researches are supported in part by National Natural Science Foundation of China (Grant Nos. U1636219, 61373167, 61772236), and Outstanding Youth Foundation of Hubei Province (Grant No. 2017CFA047). Cong's work is supported in part by the Research Grants Council of HK under Project CityU C1008-16G, and the Innovation and Technology Commission of HK under ITF Project ITS/168/17. Xiangyang's research is supported in part by National Key Research and Development Program (Grant No. 2016QY01W0105). Qian Wang is the corresponding author.

References

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of S&P*. IEEE, 2000, pp. 44–55.
[2] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proc. of CCS*. ACM, 2013, pp. 875–888.
[3] Q. Wang, K. Ren, M. Du, Q. Li, and A. Mohaisen, "Secgdb: Graph encryption for exact shortest distance queries with efficient updates," in *Proc. of FC*. Springer, 2017, pp. 79–97.
[4] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage." in *Proc. of NDSS*, vol. 71, 2014, pp. 72–75.
[5] R. Bost, "$\sigma o \varphi o \varsigma$: Forward secure searchable encryption," in *Proc. of CCS*. ACM, 2016, pp. 1143–1154.
[6] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, pp. 1–1, DOI: 10.1109/TDSC.2016.2 593 444, 2016.
[7] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "Grecs: graph encryption for approximate shortest distance queries," in *Proc. of CCS*. ACM, 2015, pp. 504–517.
[8] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.
[9] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation." in *Proc. of NDSS*, vol. 14. Citeseer, 2014, pp. 23–26.
[10] A. Kiayias, O. Oksuz, A. Russell, Q. Tang, and B. Wang, "Efficient encrypted keyword search for multi-user data sharing," in *Proc. of ESORICS*. Springer, 2016, pp. 173–195.
[11] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE TII, DOI: 10.1109/TII.2017.2786307*, 2017.
[12] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2014.
[13] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proc. of CCS*. ACM, 2015, pp. 706–719.
[14] G. Asharov, "Towards characterizing complete fairness in secure two-party computation," in *Proc. of TCC*. Springer, 2014, pp. 291–316.
[15] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. of IEEE S&P*. IEEE, 2016, pp. 839–858.
[16] R. Bost, P.-A. Fouque, and D. Pointcheval, "Verifiable dynamic symmetric searchable encryption: Optimality and forward security." *IACR Cryptology ePrint Archive*, vol. 2016, p. 62, 2016.