A New Privacy-Preserving Searching Model on Blockchain

Meiqi He, Gongxian Zeng, Jun Zhang, Linru Zhang, Yuechen Chen, SiuMing Yiu

The University of Hong Kong
{mqhe,gxzeng,jzhang3,lrzhang,ycchen,smyiu}@cs.hku.hk

Abstract. It will be convenient for users if there is a market place that sells similar products provided by different suppliers. In physical world, this may not be easy, in particular, if the suppliers are from different regions or countries. On the other hand, this is more feasible in the virtual world. The Global Big Data Exchange in Guiyang, China, which provides a market place for traders to buy and sell data, is a typical example. However, these virtual market places are owned by third parties. The security/privacy is a concern in addition to the expensive service charges. In this work, we propose a new privacy-preserving searching model on blockchain which enables a decentralized and secure virtual search-and-match market place. The core technical contribution is a new searchable encryption scheme for blockchain. We adopt the similarity preserving hash and leverage smart contracts to protect the system from the forgery attack and double-rewarding attack. We formally prove the security and privacy of our protocol, and evaluate our scheme on the private net of Ethereum platform. Our experimental results show that our protocol can work efficiently.

Keywords: security and privacy, privacy-preserving searching, blockchain

1 Introduction

As a customer, we all have the experience of looking for a product or a service. It would be convenient if there is a market place where we can find multiple suppliers of the same or similar product/service. In physical world, there are examples of this market place in different parts of the world (e.g. there is a building in Hong Kong selling wedding accessories, a tea street in Beijing, China selling different kinds of tea, and an Italian region in New York with many Italian restaurants etc.). However, it is not easy to have one that allows suppliers from different regions or countries to participate, except those organized by a third party (e.g. governments, trade organizations) which only open for a short period of time as the traders need to physically attend the event. On the other hand, it is more feasible to have such a market place in the virtual world. However, these virtual market places are usually owned by third parties. The security and privacy may be a major concern to the suppliers and customers in addition to expensive membership fees and service charges.

To further motivate our study, let us consider the following remarkable application: love matching service. There are multiple service providers. In order to use the actual service to find potential candidates for dating, in most cases, a customer needs to pay membership fees and other service charges. If we want to increase the chance of having a match, one may need to join multiple providers. The privacy of the customers (e.g. customers may not want others to know their criteria for choosing a partner) will totally rely on service providers. A more convenient and secure scenario is as follows. We have a market place allowing traders (service providers or product suppliers, we refer them as data owners in the rest of the paper) and customers to join. The market place is not owned by any third party. In our new model, customers can issue (or broadcast) a search query in this platform. Data owners can make a profit by finding a match for the customers. Instead of having the customer to join multiple providers or check the products from every supplier, now the customer only needs to issue one query, all data owners in the platform can help to locate the appropriate product/service for him. Ideally, the criteria in the search query is not revealed to the data owner to protect the privacy of the customers. The customers are only charged if matches are found. To realize this virtual market place, in this paper, we plan to explore a new "search-and-match" model on blockchains. The use of blockchains has the fundamental benefit that it does not require a trusted third party nor a centralized authority while it can provide a transparent and trusted platform for trading with low transaction fees (and service charges). Note that this platform can be extended for many applications such as data trading, job matching, finding rental apartment, and searching matched marrow.

The abstract problem: We model the problem as a privacy-preserving keyword matching problem on blockchains. The data of a data owner is represented as documents (e.g. product descriptions). Each document is characterized by a set of keywords. The user (or customer) query is in the form of a keyword. A data owner can earn a reward from a user for each document that matches the keyword in the query. Since all transactions occur in blockchain, which is transparent, the platform needs to satisfy the following basic requirements to guarantee the privacy of transactions.

1. The query is hidden from all data owners and other users. But then, the same query can be used by all data owners to search their documents.
2. The documents returned by a data owner are revealed only to the user who issued the query.
3. Miners of the blockchain (see Section 2 for a description of miners) are able to verify if the returned document indeed contains the keyword of the query, but they are not able to reveal the content of the document.

Besides the basic requirements, we also consider the following risks/attacks.

1. Double-rewarding attack: Greedy data owner may try to generate duplicate/similar documents to get double rewards once he knows which document can match the query.

2. Forgery attack: Only the data owner of the document that matches a query can reply to the query and get the reward.
3. Fair exchange: A customer cannot skip paying the reward as promised after receiving the document.

Highlights of our solutions and contributions: At a first glance, the problem is similar to the traditional searchable encryption (SE) problem [6, 7, 10, 13, 20], which allows a data owner to outsource a dataset onto a server allowing other customers to search it with a token while preserving the privacy of both the data and the query. However, there are major differences between our problem and the traditional SE problem. In traditional SE, the token for the customer is generated by the data owner. In our case, the customer will generate the token without any help from the data owner as the token needs to be used by multiple data owners. For traditional SE, the search is done by a third party, while in our case the search is done by the owner while the results need to be checked by a miner. Documents provided by different data owners are encrypted using different keys, i.e., the same encrypted query is required to search multiple documents encrypted by different keys.

To solve this problem, we propose a new multi-key searchable encryption scheme in the public key setting that enables a user to provide one search token, but allows multiple data owners to search documents encrypted with different keys. To achieve this, we are motivated by the multi-key searchable encryption scheme proposed in [18] to let each data owner transform the token to match its own documents. Each document is parsed into a set of distinct keywords and a *proof* for each keyword will be produced that can be verified by a miner. Note that [18] cannot be used directly in our case as [11, 21] have shown that [18] has a leakage problem in keyword access pattern (for details, see Section 4). We thus propose new encryption and matching methods in our scheme. In particular, to eliminate this leakage problem, we enhance our encryption method so that the data owner can update the ciphertext of the keyword every time the corresponding document has been replied to a query.

To make sure that others cannot reply to a query if the document does not belong to him, we embed the owner's public key into the encryption of the document. For the problems of forgery and double-rewarding attacks, we propose the followings. We adopt smart contracts to regulate the behaviors of owner and user. We use hash value to record the existence of documents while creating. Miners of the blockchain will only consider the document created before query. In brief, contracts, hash values and search tokens are all treated as transactions to be recorded into blockchain as undeniable proof. To avoid dishonest data owners conducting double-rewarding attack, we adopt TLSH [15], which is a similarity preserving hash used in digital forensics. In this way, data owners are required to provide the similarity hash values of their documents when replying a query and miners can detect duplication before mining it into blockchain[1]

[1] Note that in case two honest data owners have the same document or two very similar documents, it depends on the probability, the one whose document is confirmed by the miner first will get the reward.

In summary, the contributions of our paper are listed as follows.

- We propose a new privacy-preserving searching model for searching over encrypted data. The model can be used on blockchains. To realize this model, we design a novel multi-key searching encryption scheme.
- We identify several possible attacks in the new model and provide solutions to prevent these attacks, e.g. we carefully design smart contracts to handle the forgery attack and adopt similarity preserving hash to deal with double-rewarding problem.
- We formally prove the security of our scheme and evaluate our protocol in the private net of the existing Ethereum platform. Experimental results show that our protocol can run efficiently.

2 Background

2.1 Blockchain and smart contract

Please refer to [14] for an overview of blockchain. Here, we want to highlight an important role in blockchain, which is *miner*. They are responsible for verifying and adding new transactions, creating a new block by solving a puzzle (referred as the Proof of Work (PoW)) and receive some benefit in return. Once a block is added to the chain, it is extremely difficult for anyone to modify it, the correctness of the network can be guaranteed. In our case, all queries and replies are treated as transactions. Miners will be responsible to check if a reply from a data owner actually contains the keyword of the query before the reply can be added into the chain.

Smart contract is supported in some blockchain based platforms (*e.g.* Ethereum [4]). It is widely used in many complicated functions [8,9,12,17]. Smart contracts are computer programs that act as agreements where the terms of the agreement can be preprogrammed with the ability to be executed and enforced. With the decentralized setting, smart contracts are public to all users in the network. Once deployed, it is difficult to modify the contract, even by the creator, due to the Proof of Work. We leverage smart contracts in our construction to make sure that all parties would follow the defined actions.

2.2 Basics of multi-key searchable encryption (MKSE)

The multi-key keyword matching scheme used in our system consists of the following steps.

- *Setup*: Generate the system parameters.
- *Encryption*: For each file, the data owner creates an encrypted keyword index so that it can be used in matching a query.
- *Retoken*: For each file, data owner computes a value for the user based on his public key if he is authorized to search the data. This protocol works as an access control so that token from the authorized user can be transformed to match the encrypted index.

- *Documents encryption*: Data owner encrypts the documents.
- *Search token*: User generates the query token for a keyword.
- *Match*: Data owner searches the query over the encrypted index.
- *Documents decryption*: User can decrypt the result documents if he pays for it.

2.3 Bilinear Map

Let G_1, G_2, G_T be groups of prime order p and $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map. g_1, g_2 are the generators of G_1, G_2. The map satisfies the following properties:

1.Computable: given $x \in G_1, y \in G_2$ there is a polynomial time algorithms to compute $e(x, y) \in G_T$.

2.Bilinear: for any integers $a, b \in [1, p]$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.

3.Non-degenerate: if g_1, g_2 are the generators of G_1, G_2 then $e(g_1, g_2)$ is a generator of G_T.

2.4 Similarity preserving hash

Hash functions are well-known and commonly used for proving integrity and file identification. Traditional hashes (MD5, SHA-1, SHA-2) are used to check if a file has been modified or tampered (even one bit change). However, for some other applications such as identifying new versions of documents and software, locating variants of malware families, finding similar infringing copies, deduplication on storage system, we cannot use traditional hashes. Similarity-preserving hash [1–3, 15] was developed to handle these applications. In our case, we will adopt this technique to help miners to check duplicate/similar documents and only accept the first one to avoid the user double-paying for the same similar document.

Similarity-preserving hash aims at detecting similarity between objects by creating a digest in a way that similar objects will produce similar digests. When comparing two digests, a score related to the amount of content shared between them is given. There are several criteria to reflect the effectiveness and efficiency of different hash functions: 1) Efficiency: efficiency includes the comparison efficiency and space efficiency. 2) Sensitivity and robustness: sensitivity refers to the granularity at which an algorithm can detect similarity; robustness is a metric of how an algorithm can be in the midst of noise and plain transformations such as insertion/deletion.

The final decision on selecting an algorithm depends on the applications. In our case, we use the TLSH [15] algorithm for approximate matching as we pay more attention to the sensitiveness and robustness. Sdhash [3] and mvhash-B [1] suffer from active manipulation or anti-blacklisting. According to the authors [15], TLSH is more robust. It is reported that file can be deliberately modified by an adversary using randomization so that Ssdeep and Sdhash may fail, but TLSH still has a high chance to identify similar files. Additional experiments [16] showed that TLSH can detect strings which have been manipulated with adversarial intentions.

3 System overview

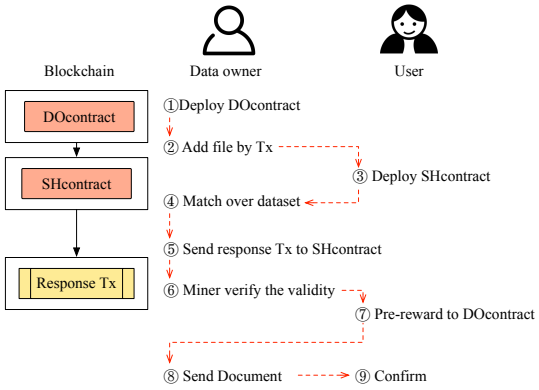


Fig. 1. System model

In Fig 1, we outline the architecture of our model. We summarize the challenges within these steps and present our ideas of how to resolve them in the following sections.

3.1 Method to ensure fair exchange

To ensure fair data exchange, we regulate the allocation of deposits and rewards by issuing DOcontract. It works as a fair intermediary to ensure that the processing of exchanging money with data between an owner and a user is carried out properly like a safe remote purchase between a seller and a buyer. To publish the search token and collect the search results, we design the SHcontract. The followings are the two smart contracts we proposed:

1) DOcontract: Smart contract to exchange data with money

To guarantee that no one can cheat for money during the trading, we have the following consideration. First, when data owners initialize the contract, they have to transfer some deposit to the contract. Before each response, the data owner are required to pledge an amount of money from the funding of the contract. Therefore, we define the contract state to control the procedure of contract initialization. Specifically, when the DOcontract is sent to the blockchain, the state is set to *Created* and will change after each operation. As long as the data owner transfers the deposit, the contract state will be set to *Active*. The other functions can be executed if and only if the contract is active. Also, similar idea can be used to make sure they have pledge when response and the user has frozen the rewards before sending file. We restrict the access control of each function to guarantee that no one can cheat for money. The contract are designed to have the following functions.

- **Funding:** The owner must pledge a certain amount of money as an initial fund so that it can be used as deposit when answering search queries.
- **Deposit:** After sending response transactions, data owner pledges a certain amount of fund so that if they fail to provide valid document after response, the user who searches can get the deposit.
- **Reward:** This function allows the user to send the rewards to the contract. The money is *locked* until user receives the document from data owner.
- **Terminate:** As long as the owner would like to terminate the contract and there is no search work in process, he can activate the function to get back the funding in the contract.

2) SHcontract: Smart contract to maintain search results

This smart contract is deployed by the user to collect and maintain the search results. SHcontract publishes the information about the query including search token, rewards for each response and maximum response number.

Data owners that would like to answer the query can interact with the contract by sending a response transaction. The function **Response** embedded in this contract requires the validation proof as input involving:

- Proof information indicating the correctness of matching;
- Similarity hash of the document.
- Block number and transaction index that contain the hash record.

Sending transactions to call a function in smart contract will embed the input arguments in the transaction. With this property, miners can easily extract and verify the proof provided by the data owner and determine whether to include it into block. And the SHcontract can receive and accept the response only when the miners accept and include the transaction.

3.2 Method to resist double-rewarding

As introduced before, in our model, malicious data owner can produce a set of identical or similar documents to gain the rewards multiple times. We refer this as double-rewarding attack. To deal with the problem above, we propose to use duplication detection techniques in digital forensics called similarity-preserving hash to identify identical and similar documents to decrease the probability of having double-rewarding attack. In our design, data owners are required to include the similarity hash values in the response transaction. The miners who want to include this transaction have the responsibility to do the verification and then the SHcontract can receive and record the response after mining is completed. During the verification, miners compare the distance between this response with all the previous accepted responses. To generate and compare the hash values, we use the TLSH algorithm [15]. TLSH uses the following 4 steps to construct the digest.

- Process the input using a sliding window to populate an array of bucket counts;

- Calculate the quartile points;
- Construct the digest header values based on the quartile points, the length of the file and a checksum;
- Construct the digest body by generating a sequence of bit pairs, which depend on each bucket's value in relation to the quartile points.

4 Construction with SE

In this section, we show how to search on blockchain with a specific keyword search scheme. We first highlight our design technique for this part.

Hiding keyword access pattern: Recently, the problem of leakage-abuse attack over searchable encryption has been widely investigated [5, 11, 21]. In [5], Cash *et al.* presented a characterization of the leakage profiles of in-the-wild searchable encryption products and SE schemes in the literature. In [11], Grubbs *et al.* presented many ways of attacking a software framework named Mylar [19] whose building blocks include a SE scheme protocol [18]. However, most of the attacks only leverage implementation issues of Mylar and are not related to the cryptographic protocols. In [21], the authors also mentioned that the leakage abuse attack in [11] could not be extended to cover any application using the underlying SE scheme and gave a deeper analysis to the MUSE (multi-user SE, also known as MKSE) scheme, in particular for [18]. Their work showed that in the case of MUSE, there is a new leakage problem named keyword access pattern and can result in serious attack if some users can collude with the server. In their following work [22], they proposed a secure and scalable MUSE scheme without this leakage. However, their scheme is based on two non-colluding parties which cannot be used in our setting. In our model, given that blockchain is a public network which is transparent to any users, without a careful design, publishing index ciphertext is likely to reveal such leakage so that attacks using keyword access pattern leakage are easier to be conducted. It is essential to get rid of the leakage of keyword access pattern. We thus require the data owner to update the ciphertext of the keyword after response, with low computational overhead, so that there will not be keyword access pattern leakage.

Unforgeable identity: Another innovative elements in our keyword matching scheme is that, each data owner is linked with a secrete parameter x and publish $y = g_2^x$ as unforgeable identity. And x is involved in the index encryption protocol, y is included in the matching scheme. In this way, even if other user obtains your proof information, they cannot make use of it to gain profit. Let us consider the following case. Data owner A has a pre-knowledge about B's database that their data set have similar background. Then A can stay passive and observe B's responses. Upon B's releasing a proof for response, A will copy it and answer the query as well. Without our unforgeable identity x, y, the *proof* definitely can match the query, and A has high possibility to gain rewards without search.

4.1 System setup

1) At the beginning of setup, $\mathsf{Param}(\lambda)$ is called to input the security parameter and output system parameters.

> $\underline{\mathsf{Param}(\lambda)}$: Input the security parameter λ. Let G_1, G_2, G_T be groups of prime order p and $e : G_1 \times G_2 \to G_T$ be a bilinear map. g_1, g_2, g_T are the generators of G_1, G_2 and G_T respectively. Let $H_1 : \{0,1\}^* \to G_1$ be a collision resistant hash function. Let $sp = (p, G_1, G_2, G_T, g_1, g_2, H_1)$ be the system parameters.

2) User who wants to publish search queries calls $\mathsf{SearchKey}(sp)$ to generate a pair of public key and secrete key pk, sk. Keep sk as a secret key and publish pk.

> $\underline{\mathsf{SearchKey}(sp)}$: User chooses a random number $\alpha \in \mathbb{Z}_p^*$ and computes $pk = g_2^\alpha$. Keep $sk = \alpha^{-1} \mod p$ as a secret key and publish pk.

3) Data owner deploys a DOcontract and generates a secret parameter $x \in \mathbb{Z}_p^*$ and publish $y = g_2^x$ as unforgeable identity which we mentioned before.

4.2 Add file

1) Data owner parses document d into distinct keywords: $\{w_1, \cdots, w_m\}$. For each keyword, Data owner runs $\mathsf{Enc}(d, w_i)$ and outputs (k_{d,w_i}, c_{d,w_i}).

> $\underline{\mathsf{Enc}(d, w_i)}$:
>
> 1) Data owner chooses a new key for his new files: $k_d \in \mathbb{Z}_p^*$.
> 2) Data owner chooses a random number $t \in \mathbb{Z}_p^*$, and computes $r = g_2^t$.
> 3) Data owner chooses a new key for w_i: k_{d,w_i}, and computes $s = (k_d k_{d,w_i} - xh)t^{-1} \mod p$, where $h \in \mathbb{Z}_p^*$ is a random number.
> 4) $c_{d,w_i} = (H_1(w_i)^h, H_1(w_i)^s, r)$.
> 5) Output (k_{d,w_i}, c_{d,w_i}).

2) Data owner chooses new keys for his new files and if the file is authorized to user u with public key pk, Data owner will compute $\mathsf{Delta}(d, pk, k_d)$=$\Delta_{d,u}$.

> $\underline{\mathsf{Delta}(d, pk, k_d)}$:
>
> 1) Data owner computes $\Delta_{d,u} = (pk)^{k_d}$.
> 2) Output $\Delta_{d,u}$.

3) Finally, DO encrypts the document d using $\mathsf{SKE}(d)$ and outputs the ciphertext.

$\underline{\mathsf{SKE}(d, k_d)}$:

1) Encrypt d by computing $C_d = d \oplus f(ID(d), k_d)$, where f is a pseudorandom function, $ID(d)$ is the identity of the document.
2) Output C_d.

4) Data owner publishes a transaction including the new hash value $hash\_new = \mathsf{SHA\text{-}1}(hash\_old||C_d)$. $hash\_old$ denotes the hash value of the data set before uploading this document.

4.3 Keyword Search

1) As long as the user u wants to search for a keyword w, he computes the search token $tk_w = H_1(w)^{sk}$. and setup a SHcontract to collect the response.
2) Data owner follows $\mathsf{Match}(tk_w, k_{d,w_i}, c_{d,w_i})$ to checks the equality and gives the output.

$\underline{\mathsf{Match}(tk_w, k_{d,w_i}, c_{d,w_i})}$:

1) Data owner parses c_{d,w_i} into three parts as (u_1, u_2, u_3).
2) Check if $e(u_1, y)e(u_2, u_3)$ equals to $e(tk_w, \Delta_{d,u}^{k_{d,w_i}})$.
3) If so, output 'match', if not, output 'no'.

4) Data owner sends a response transaction to the SHcontract with similarity hash of this file and $proof = (\Delta_{d,u}^{k_{d,w_i}}, c_{d,w_i}, BlockNum, Index)$, where $BlockNum$ and $Index$ are the block number and transaction index that contains the hash value when uploading. After posting the $proof$, Data owner updates c_{d,w_i} by running $\mathsf{Enc}(d, w_i)$ with new parameters.
5) Finally, Data owner is required to give deposit in the DOcontract.

4.4 Response verification

After the response transaction is sent into the mining pool, the miners can get the $proof$ and do verifications to determine whether to include it into his blocks.
1) The response verification mainly consists of 3 steps:

- Check if $e(tk_w, \Delta_{d,u}^{k_{d,w_i}}) = e(H_1(w_i)^h, y)e(H_1(w_i)^s, r)$.
- Analyze the hash value in transaction (using $BlockNum$ and $Index$ to locate) to test if the new hash value is equal to $\mathsf{SHA\text{-}1}(hash\_old||C_d)$.

– Check the similarity with previous responses.

2) If the transaction passes the verification, miners will include it into his block so that the response is published on the blockchain and SHcontract will receive the transaction.

4.5 File retrieval and decryption

1) After receiving the response, the user will call **Reward()** function in DOcontract to pledge the rewards, then the user computes $C'_d = C_d \oplus f(ID(d), sk)$ and sends it to the data owner to decrypt.
2) Data owner computes $C''_d = C'_d \oplus f(ID(d), k_d)$ and sends it back to the user. User can obtain the plaintext of d by $C''_d \oplus f(ID(d), sk) = d$.
3) As long as the deal is completed successfully, Data owner will get back the deposit and rewards, otherwise, the user will resume the rewards and get the deposit.

5 Security Analysis

We define the privacy of our keyword matching scheme using the simulation paradigm in searchable encryption, following [6,7], that is based on the notion of *leakage*. We first clarify the difference between keyword matching in our setting and previous SE setting. In previous SE scheme, data owners outsource the index and encrypted database onto a cloud server where the server is believed to have the most knowledge over the protocol and encrypted dataset. However, in our setting, there is no centralized server while all queries and responses are published on the blockchain which is a globally visible ledger. The users in this blockchain are supposed to be the ones who can obtain all the leakage and be curious to the dataset of the data owners and queries of the other users. Also, there is no "beginning leakage", including the length of the index and the size of the database compared to the previous SE scheme. All the leakage are query-revealed. In order to characterize the leakage in our scheme, we give the following definitions:

Definition 1. *(Leakage function $\mathcal{L}$) Given a search input w, $\mathcal{L}$=$\{\mathsf{ap}(w, D)$,$\mathsf{sp}(w)$, $\mathsf{rl}(w)$, $tk_u(w)$, $proof = (\Delta_{d,u}^{k_{d,w_i}}, c_{d,w_i}$, BlockNum, Index), $C_d\}$, where $\mathsf{ap}(w, D)$ denotes the access pattern, $\mathsf{sp}(w)$ denotes the search pattern and $\mathsf{rl}(w)$ represents the number of documents matching each query. $tk_u(w)$ is the search token post by user u for keyword w, and $proof$ is leaked in each response, where d is the matched document.*

Theorem 1. *Our keyword match scheme has leakage profile $\mathcal{L}$ against the adversary if there exits a polynomial-time simulator $\mathcal{S}$, that for all polynomial-time adversary $\mathcal{A}$, the output of real execution and simulated execution are computationally indistinguishable.*

Due to the page limit, the formal proof is included in the appendix A.

6 Experiments

While the construction of the previous sections gives an overview of our model and approach, we have yet to describe how our techniques integrate with existing blockchain platform. In this section, we show the evaluation results of our scheme on the private net of Ethereum platform, which is known as an open-source blockchain-based distributed computing platform and supports smart contract. We set up the go-ethereum from https://github.com/ethereum/go-ethereum, which is the official golang implementation of the Ethereum protocol. We built private Ethereum chain on a single server node with Intel(R) Core(TM) i5-3570 CPU, using single core processor.

6.1 Ethereum platform

The basic functions are provided by the Ethereum, such as creating an account and sending transactions. Besides, it provides Ethereum Virtual Machine (EVM), which is part of the block verification protocol and can run the functions defined in the contracts. From the view point of developers, a contract has a specific address on the Ethereum blockchain, where we can store the contract code and data. Thus, we only need to send the Ethereum-specific binary format code onto the chain. When doing practically Turing complete computation, we can pass messages (contained in a transaction) to the contracts, which is exactly what we need. To reach consensus, all nodes in Ethereum would go through the transactions listed in the blocks and runs codes in the EVM. The Ethereum protocol charges a fee per computational step that is executed in a contract or transaction to prevent deliberate attacks and abuse on the Ethereum network. Every transaction is required to include a gas limit and a fee that it is willing to pay per gas. If the total amount of gas used by the computational steps spawned by the transaction, including the original message and any sub-messages that may be triggered, is less than or equal to the gas limit, then the transaction is processed.

6.2 Simulation Design

To measure the performance of our scheme, we deployed the DOcontract and SHcontract and implemented the verification protocol including the process of similarity comparison, hash checking and checking if the response matches the query. Also, since the new response transactions require verification before being mined into a block, we studied the impact of such kind of transactions on the normal Ethereum network.

For the two smart contracts, the functions overview has been introduced in Section 3.1. In this section, we address the specific challenges that we come up when we deploy them in Ethereum. Due to page limitation, we will not show the detailed pseudocode of DOcontract and SHcontract here.

As a matter of fact, in live Ethereum, the number of transactions a miner decide to include into his block depends on many factors. For instance, each

miner will set a minimum gas unit before mining. Only transactions with gas above this level will be accepted. To simplify the experiment and simulate our protocol in the same standards, we assume that the miners all adopt greedy algorithm and the transaction with higher transaction fee has priority. In other words, they would mine a block that contains as many transactions as possible and the transaction that offers higher gas fee would be firstly to be included in a block.

In our experiment, the contracts are developed in Solidity language and the verification is implemented with Go language. We use the "crypto" package of golang and package "bn256" to implement the bilinear group.

6.3 Metrics

We test our protocols mainly according to the following criteria:

- *Response transaction verification time* is the average running time, over 1000 tests, required to verify a response transaction. Since the verification step is set before the miners mine the transaction into blocks, if the verification is not efficient, it will slow the miner down in comparison with mining normal transactions. We will also include the time for every sub-step. (You can refer the verification algorithm to 4.4 and one more step (*i.e.* check deposit) introduced in the above simulation design.)
- *Transaction waiting time* in blockchain denotes the interval between a transaction's creation and its inclusion in a block. It is an important measurement when designing a blockchain application. We want to know that how much influence our response transactions have on the average waiting time of all types of transactions and on the average waiting time of current existing normal transactions. We simulate the relationship between the waiting time and the instantaneous transactions number.

For different blockchain platforms, they have different block interval (i.e. the time to generate a new block), e.g. 10 minutes for Bitcoin and 12 seconds for Ethereum. Thus, we count the verification time by block numbers so that it is easier for readers to assess our scheme across different platforms.

6.4 Results

From Table 1, we can see that it takes about 50 ms to verify a response transaction and most of time is spent on performing bilinear map computation. From the websites that have the statistical performance of current live Ethereum (e.g. ETH Gas Station[2], Etherscan[3], etherchain[4]), we can know that each block usually contains about 70 transactions. Thus, it would be about 0.35 seconds to verify these transactions even all of them are our response transactions. We

[2] https://ethgasstation.info/index.php
[3] https://etherscan.io/
[4] https://etherchain.org/

have known that, the block interval for Ethereum is around 12 seconds and 10 minutes for Bitcoin, thus, the verification time is negligible compared to the mining time.

Verification Steps	Sub-Steps	Time
Read Tx	Get Tx from pending pool	$361.206\mu s$
	Analyze data from Tx	$11.987\mu s$
Verification details	Check Match	52.234123 ms
	Check similarity (100 times)	$6.4\mu s$
	Get Tx with new hash record	$500.109\mu s$
	Extract new hash from Tx	$58.483\mu s$
	Get Tx with old hash record	$376.23\mu s$
	Extract old hash from Tx	$300.333\mu s$
	Check file hash	$350.162\mu s$
Check deposit	Check state in DOcontract	$547.93\mu s$
		Total: 54.746963 ms

Table 1. Verification Time

Transaction	Size (Bytes)	Estimated gas usage
Tx to deploy DOcontract	4825	735044
Tx to deploy SHcontract	1543	268391
Response Tx	970	120190

Table 2. Transaction input size

Algorithm	Token	Delta	Enc	ReToken	Match
Time (ms)	2.2528	5.0599	9.8592	23.0500	31.0791

Table 3. Running time of keyword search scheme

For the reason that the contracts implement lots of functions in our scheme and require many parameters to ensure the protocol works fairly and correctly from setup to completion, it can be inferred that the response transactions in our scheme are larger than the normal transfer transactions. To make it clearer, we further investigate the contracts size, transaction size and gas usage in Ethereum. The results are shown in Table 2.

The transactions to deploy smart contracts are one-time consuming so we pay more attention to the response transactions, which need 4825 Bytes for each

one in Table 2. At the time of writing this paper, the average block size in Ethereum is 16192 Bytes so the response transactions would occupy much space in a block. Also, a normal transaction uses about 43000 units gas on average but our response transaction costs about 120190 units gas. If comparing to an ether transfer transaction, which costs only 21000 units gas, the gas usage of our response transaction is about 6 times. However, from etherchain [4], we know that, the current gas limit for each block is about 6800000 units, thus it has enough resource for our application while one block would contain less transactions and it takes longer waiting time when dealing with our response transactions. We do the following experiments to evaluate how much influence our response transactions have on the average waiting time of all types of transactions and on the average waiting time of current existing normal transactions.

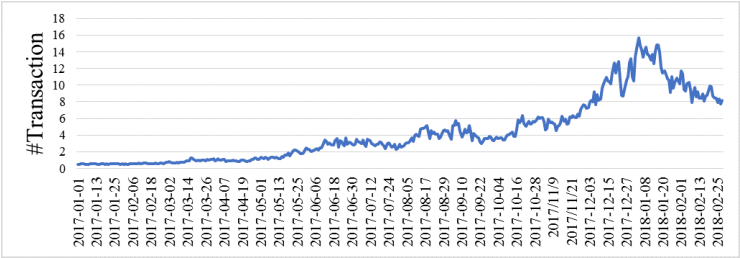
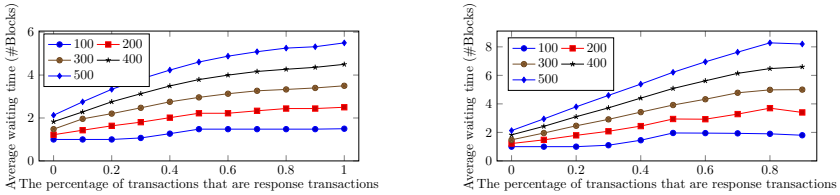


Fig. 2. Number of transactions per second



(a) Average waiting time of all types of transactions

(b) Average waiting time of normal transactions

Fig. 3. Waiting time

The evaluation on waiting time is displayed in Figure 3(a) and Figure 3(b). From Figure 3(a), we can know that the average waiting time of all transactions increases with the increase of the percentage of response transactions among all transactions. Even there are 500 response transactions, the waiting time is no more than 6 blocks from Figure 3(a). And from Figure 3(b), we can see

that the lines go up first and drop later, of which the reason is that we adopt greedy algorithm for miners and there are less normal transactions when the percentage is higher. In fact, in live Ethereum, we can estimate that the peak value of transactions generated within a block interval is about 100 to 200. From etherchain [4], the number of transactions generated per second since 01/01/2017 is plotted in Figure 2. It is clearly that, even the quantity increases rapidly, the average level recently is about 7.4 transaction per second (88.8 per block interval). Under this background, the two figures present quite good results, that is, the average waiting time is about 2 to 3 blocks. In other words, our response transactions can integrate well with the existing blockchain-base network.

Last but not least, we evaluate the running time of algorithms in the keyword search scheme using go language. The results are summarized in Table 3. The time is averaged over 1000 tests with randomly generated keywords. We can conclude that the scheme has a modest overhead.

7 Conclusions

In this work, we propose and formulate a new model for privacy-preserving searching on blockchains. We present a keyword search scheme for searching over text data. We focus on single keyword search. It is desirable to extend our scheme to handle multiple keyword search with boolean operators, approximate matching for the keywords, and non-text files.

Acknowledgement: This project is partially supported by a RGC Project (CityU C1008-16G) funded by the HK Government.

References

1. Breitinger, F., Astebøl, K.P., Baier, H., Busch, C.: mvhash-b-a new approach for similarity preserving hashing. In: IT security incident management and it forensics (IMF), 2013 seventh international conference on. pp. 33–44. IEEE (2013)
2. Breitinger, F., Baier, H.: Similarity preserving hashing: Eligible properties and a new algorithm mrsh-v2. In: International conference on digital forensics and cyber crime. pp. 167–182. Springer (2012)
3. Breitinger, F., Baier, H., Beckingham, J.: Security and implementation analysis of the similarity digest sdhash. In: First international baltic conference on network security & forensics (nesefo) (2012)
4. Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform. URL https://github. com/ethereum/wiki/wiki/% 5BEnglish% 5D-White-Paper (2014)
5. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 668–679. ACM (2015)
6. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In: NDSS. vol. 14, pp. 23–26 (2014)

7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security **19**(5), 895–934 (2011)
8. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Symposium on Self-Stabilizing Systems. pp. 3–18. Springer (2015)
9. Delmolino, K., Arnett, M., Kosba, A.E., Miller, A., Shi, E.: Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. IACR Cryptology ePrint Archive **2015**, 460 (2015)
10. Goh, E.J., et al.: Secure indexes. IACR Cryptology ePrint Archive **2003**, 216 (2003)
11. Grubbs, P., McPherson, R., Naveed, M., Ristenpart, T., Shmatikov, V.: Breaking web applications built on top of encrypted data. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1353–1364. ACM (2016)
12. Heilman, E., Baldimtsi, F., Goldberg, S.: Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In: International Conference on Financial Cryptography and Data Security. pp. 43–60. Springer (2016)
13. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 965–976. ACM (2012)
14. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
15. Oliver, J., Cheng, C., Chen, Y.: Tlsh–a locality sensitive hash. In: Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth. pp. 7–13. IEEE (2013)
16. Oliver, J., Forman, S., Cheng, C.: Using randomization to attack similarity digests. In: International Conference on Applications and Techniques in Information Security. pp. 199–210. Springer (2014)
17. Poon, J., Dryja, T.: The bitcoin lightning network (2015)
18. Popa, R.A., Zeldovich, N.: Multi-key searchable encryption. IACR Cryptology ePrint Archive **2013**, 508 (2013)
19. Popa, R.A., Stark, E., Helfer, J., Valdez, S., Zeldovich, N., Kaashoek, M.F., Balakrishnan, H.: Building web applications on top of encrypted data using mylar. In: NSDI. pp. 157–172 (2014)
20. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. pp. 44–55. IEEE (2000)
21. Van Rompay, C., Molva, R., Önen, M.: A leakage-abuse attack against multi-user searchable encryption. Proceedings on Privacy Enhancing Technologies **2017**(3), 168–178 (2017)
22. Van Rompay, C., Molva, R., Önen, M.: Secure and scalable multi-user searchable encryption (2018)

A Proof of Theorem 1

Proof. The simulator $\mathcal{S}$ is given leakage $\mathcal{L}$ to simulate the view of the adversary via imitating the real protocol. We generate the proof string $proof' = ((\Delta_{d,u}^{k_{d,w_i}})', (c_{d,w_i})', BlockNum, Index)$ and $(C_d)'$ as follows:

1. Simulating $(\Delta_{d,u}^{k_{d,w_i}})'$: Given $\mathcal{L}$, $\mathcal{A}$ choose a composite random key k_s that $k_s = k_{s_1} \cdot k_{s_2}$, and compute $(\Delta_{d,u}^{k_{d,w_i}})' = (\Delta_{d,u}^{k_{d,w_i}})^{k_s}$.

2. Simulating $(c_{d,w_i})'$: For each keyword query w_i, $(c_{d,w_i})'$ is consist of three parts.

- Simulating $(H(w_i)^h)'$: compute $(H(w_i)^h)' = (H(w_i)^h)^{k_s}$.
- Simulating $(H(w_i)^s)'$: compute $(H(w_i)^s)' = (H(w_i)^s)^{k_{s_1}}$.
- Simulating r': compute $r' = r^{k_{s_2}}$.

3. Simulating $(C_d)', BlockNum', Index'$: Use the same value in the response of real execution.

It follows by construction that response with $proof'$ will also match the search token tk_w if $proof$ does because:

$$
\begin{aligned}
e(tk_w, (\Delta_{d,u}^{k_{d,w_i}})') &\stackrel{?}{=} e((H(w_i)^h)', y)e((H(w_i)^s)', r') \\
\text{Left} &= e(H(w_i), (\Delta_{d,u}^{k_{d,w_i}})^{k_s}) \\
&= e(H(w_i), g_2)^{k_d k_{d,w_i} k_s} \\
\text{Right} &= e((H(w_i)^h)', y)e((H(w_i)^s)', r') \\
&= e(H(w_i)^{hk_s}, g_2^x)e(H(w_i)^{sk_{s_1}}, g_2^{tk_{s_2}}) \\
&= e(H(w_i), g_2)^{xhk_s + stk_{s_1}k_{s_2}} \\
&= e(H(w_i), g_2)^{(st+xh)k_s}
\end{aligned}
$$

Therefore, Left = Right if $proof$ matches tk_w.

We now claim that no polynomial-size distinguisher can distinguish between the distributions $proof'$ and $proof$. Note that in the simulation above, $\Delta_{d,u}^{k_{d,w_i}}$ and $(\Delta_{d,u}^{k_{d,w_i}})'$ as well as all the components in c_{d,w_i} and $(c_{d,w_i})'$ can be regarded as the problem to distinguish between g^{ab} and g^{abc}. For example, c_1 in c_{d,w_i} equals to $H(w_i)^h = g_1^{ah}$ and c_1' in $(c_{d,w_i})'$ equals to $H(w_i)^{hk_s} = g_1^{ahk_s}$. Then, we make the following Lemma 1.

Lemma 1. *If g^{ab} and g^{abc} are indistinguishable from random numbers in the same groups respectively, then g^{ab} and g^{abc} are indistinguished from each other.*

Since g^{ab} and g^{abc} are of the same structure, we only need to prove the distinguishability of anyone of them. For contradiction, we assume that there is a PPT adversary $\mathcal{D}$ that distinguishes g^{ab} and $R \stackrel{\$}{\leftarrow} G$, then we show how to construct a PPT reduction $\mathcal{B}$ that can use Exp to break the DDH assumption. breaks DDH.

Experiment

- Given a (multiplicative) cyclic group G of order p, and with generator g.
- $\mathcal{B}$ receives (g^a, g^b, g^{ab}) and (g^a, g^b, R), $R \stackrel{\$}{\leftarrow} G$. $\mathcal{B}$ passes g^{ab} and R to $\mathcal{D}$.
- $\mathcal{B}$ guesses the same as $\mathcal{D}$.

Finally, $(C_d), BlockNum, Index$ and $(C_d)', BlockNum', Index'$ are identical, therefore, no polynomial-size distinguisher can distinguish between the outputs of real execution and simulated execution.