BlindChain: Keeping Query Privacy in Blockchain Out of Sight

Jingxian Cheng[1], Qin Jiang[2], Ao Wang[3], Saiyu Qi[3]✉, Ke Li[3], Zhangjie Fu[2], and Yong Qi[3]

[1] Chang'an University, Xi'an, China
[2] Nanjing University of Information Science & Technology, Nanjing, China
[3] Xi'an Jiaotong University, Xi'an, China
✉ saiyu-qi@xjtu.edu.cn

Abstract. Blockchain allows full nodes and light nodes coexist to support users with limited resources. Any query from light nodes must be forwarded to full nodes that maintain the entire blockchain data. However, outsourcing queries to full nodes may invade users' privacy. Most existing works of privacy-preserving blockchain query systems focus on deploying Software Guard Extensions (SGX) for every full node or resorting to expensive cryptographic tools such as ORAM, incurring heavy communication and computation overheads. In this paper, we propose a blockchain system *BlindChain* that can protect query privacy of light nodes and verify correctness of query results with lightweight overhead. *BlindChain* integrates Function Secret Sharing (FSS) into blockchain, requiring only a small part of full nodes to be configured with SGX. We first introduce an SGX-based garrison to meet the trust model requirements of the FSS in the context of blockchain. Then, we design a novel authenticated data structure for *BlindChain* to verify result correctness. Finally, we propose an optimization technique based on parallel processing to improve the query performance. We demonstrate the privacy-preserving property of *BlindChain* through rigorous security analysis. Experiment results show that *BlindChain* outperforms the previous works in terms of both computation and communication overheads.

Keywords: Query Privacy, Blockchain, Function Secret Sharing, SGX.

1 Introduction

Due to the success of cryptocurrencies such as Bitcoin and Ethereum, blockchain has gained overwhelming momentum in recent years. The decentralization and provenance features of blockchain make it be exploited in many scenarios, such as international settlement, security trading, healthcare, Internet of Things, etc. [1][2]. To query data stored on a blockchain, a user can join the blockchain as a full node that stores the whole blockchain data (including block headers and data records). In this way, the user can process queries locally. However, an infinite gap between the limited resources of the user and the heavy bandwidth and storage overheads required to join as a full node is of great difficulty to cross. To address this issue, a common approach is to make the users act as light nodes which only need to store block headers that are a small fraction of the blockchain. Light nodes outsource queries to a serving full node. The full node returns query results with verification objects (VO) which includes the proof for the query user to verify the correctness of the results.

First Author and Second Author contribute equally to this work.

However, outsourcing queries to a full node raises the sharp concerns about query privacy. A single full node is untrusted and may infer sensitive user information from these queries. For example, cryptocurrencies as one of the most important applications of blockchain, the leakage of critical information such as user identity violates the security goals of decentralized cryptocurrencies. Although the pseudo-anonymity guarantees unlink-ability between the real-world identities and on-chain addresses of light nodes, full nodes still can infer the real-world identities of light nodes through the frequent access [3]. Existing blockchain systems leverage Bloom filters to fetch data records. In this way, the on-chain addresses can be hidden with the help of the false-positive ratio. However, recent work [4] has shown that Bloom filters cannot fully protect light node privacy.

To tackle the above issues, some SGX-based privacy-preserving blockchain query systems such as BITE [5] and EncELC [6] are proposed. These schemes need prevalent deployment of trusted hardware by configuring SGX for every full node, which is not always practical in all scenarios. In another research trend, PIR-based scheme BIT-PIR [7] employs cryptographic primitives to protect query privacy, which hide access information of light nodes but cannot protect metadata such as access frequency. In addition, OblivChain [8] utilizes multi-server architecture to enable oblivious query scheme for blockchain systems. Obviously, the use of off-chain service providers partly violates the "decentralization" principle of blockchain. Moreover, the core idea of above schemes is processing queries inside SGX enclaves by scanning the index or resorting to expensive cryptographic tools such as ORAM, incurring heavy communication or computation overheads.

In this paper, we propose BlindChain, a lightweight blockchain query system to protect privacy of light nodes and verify correctness of query results. BlindChain integrates a cryptographic primitive, Function Secret Sharing (FSS) [9], into blockchain by configuring SGX for a small part of full nodes. FSS allows a user to split a function into succinct shares such that any strict subset of the shares reveals nothing about the function. Using FSS to protect query privacy does not require additional handling of access pattern and search pattern leakage, thus incurring lightweight computation and communication costs. Intuitively, a light node can divide each query into function shares using FSS and send them to different full nodes. The query privacy can be hidden unless all the full nodes collude. The light node can recover the query result according to the responses from all the full nodes. In this way, most queries can be answered at low overhead in a single network round trip. We find two key observations that support integrating FSS into blockchain. First, blockchain is naturally coupled with FSS as a multi-replicas architecture. Second, the consensus protocol of the blockchain ensures the consistency of multi-replicas, which means every full node of the blockchain holds an identical copy of the blockchain data.

However, we still encounter several challenges when deploying FSS in the blockchain. First, FSS needs at least one honest server to meet the trust model requirements. Although more than 51% of the full nodes in the blockchain are honest, a light node cannot avoid distributing all the function shares to malicious full nodes, as the number of function shares is generally far less than 49% of the total number of the full nodes. Ensuring that at least one full node receiving the function shares is honest remains a critical challenge. Second, FSS just assumes that the input data of a query is correct and does not ensure query authentication. A malicious full node can easily jeopardize query result by tampering the inputted blockchain data before starting the query. Ensuring that full nodes evaluate function shares on inputs corresponding to the correct data records is thus another key challenge. Third, as an append-only structure, applying FSS to blockchain incurs linear overhead with the number of blocks contained in the

Table 1. Comparison with related works

Scheme	Privacy-Preserving	Authentication	Decentralization	Lightweight	Methods
BITE[5]	✓	×	✓	×	SGX+ORAM
EncELC [6]	✓	×	✓	×	SGX+ORAM
BIT-PIR [7]	✓	×	✓	×	PIR
OblivChain [8]	✓	✓	×	×	ORAM
BlindChain	✓	✓	✓	✓	FSS+SGX

query region. Mitigating the performance limitations of running FSS on blockchain systems is an additional critical challenge.

To tackle these challenges, we design several new techniques in BlindChain to adjust FSS to be suitable in the context of blockchain. First, we introduce an SGX-based garrison to meet the trust model requirements of the FSS. The garrison consists of a group of guard nodes, which is a small part of the full nodes in blockchain. BlindChain only requires each guard node in the garrison to equip SGX enclave, while BITE and EncELC ask all full nodes to equip SGX. To conduct a query, a light node selects a query group which contains at least one guard node in the garrison from all full nodes and issues function shares of the query to all nodes in the group one for each. Such a design ensures that at least one of the full nodes that receive the function shares does not collude with other nodes.

Second, we augment each block with a novel authenticated data structure (ADS). The ADS is designed based on the combination of FSS and blockchain and embedded into each block header. According to the ADS, the selected guard node returns VO which includes proof for light nodes to verify the correctness of query results. Third, we propose an optimization technique by applying the execution phase in a parallel manner to leverage the modern multi-core processors, thus reducing the heavy overhead that is linear with the number of queried blocks. Combining these techniques together, BlindChain can be used to protect query privacy in blockchain while verifying result correctness with lightweight overhead. The comparison of the related works is shown in Table 1.

To summarize, this paper has the following contributions:

- We propose a novel blockchain system BlindChain that can protect query privacy of light nodes and verify correctness of query results with lightweight overhead.
- In BlindChain, we first integrate FSS into blockchain to reduce hardware configuration requirements and achieve lightweight overhead. We set a small part of full nodes as an SGX-based garrison to meet the security requirement of FSS in the context of blockchain.
- We further design a novel ADS to ensure the authentication of query results in BlindChain and an optimization technique with parallel processing for improving the query performance.
- We implement a prototype and conduct extensive experiments to validate the performance of BlindChain. Results demonstrate that BlindChain outperforms the previous works in terms of both computation and communication overheads.

The rest of this paper is organized as follows: In Section 2, we review related works before outlining system overview of BlindChain in Section 3. Section 4 describes BlindChain's design. Section 5 gives the security analysis of our scheme and the performance evaluation is shown in Section 6. Ultimately, we conclude in Section 7

2 Related Work

FSS for private queries. Boyle et al. introduce Function Secret Sharing (FSS) [9] and extend it in the later works [10][11]. FSS provides a way for additively secret-sharing a function.

With the help of FSS, there is a large body of research which aims to protect query privacy. Splinter[12] uses FSS to allow users to make a variety of private queries on a public database. The Splinter user splits each query into multiple shares and sends each share to a different cloud server that holds a copy of the data. DORY [13] applies FSS to an encrypted search system, which splits trust between multiple servers and hides search access patterns. Durasift[14] uses FSS with MPC across n servers to support arbitrary boolean expressions of at most $n-3$ keywords. Floram [15] and Bunn et al. [16] use FSS to implement distributed-trust ORAM which provides private reading and writing. Waldo [17] is a time-series database built on top of FSS, which enables multi-predicate and arbitrary aggregates. Furthermore, Waldo supports three-party honest-majority setting. These works deploy FSS in traditional cloud environment and are therefore inapplicable to decentralized, trustless blockchains.

Secure Query on Blockchain. The research related to secure queries on blockchain can be broadly divided into two categories. The first category is verifiable query schemes to ensure query integrity. vChain[18] is the first work to address the query integrity issue for blockchain by including an accumulator-based ADS into the block header. Shao et al.[19] propose a query authentication scheme for blockchain by combining MB-tree with SGX. LVQ[20] is a lightweight verifiable query scheme by integrating the Bloom filter Merkle tree with the Sorted Merkle tree. However, these efforts do not consider query privacy protection. The second category is privacy-preserving query schemes to hide sensitive information of light nodes. BITE[5] and EncELC[6] prevalently deploy trusted execution enclaves on every full node to protect query privacy of Bitcoin and Ethereum, respectively. BITE and EncELC both control output leakage by padding query results and utilize expensive oblivious techniques (e.g., ORAM) to ensure oblivious transaction retrievals. BIT-PIR [7] is a customized private information retrieval scheme to match the Bitcoin transaction query scenario. However, PIR-based schemes can only hide data access information for a single light node but cannot protect file metadata such as access frequency. OblivChain [8] provides privacy-preserving oblivious query services for blockchain light nodes, which introduces read-only ORAM to improve performance. Nevertheless, OblivChain utilizes multiple off-chain servers as a proxy, which partly weakens decentralized advantages of blockchain. BlockOPE[21] is a privacy-preserving order-oriented query scheme for permissioned blockchain by integrating order-preserving encryption with blockchain. In summary, these works protect query privacy with the help of expensive cryptographic tools, thus incurring heavy performance overhead.

3 System Overview

3.1 High-level design

Fig.1 shows the system model of BlindChain, which consists of four types of parties: (1) Miner: full nodes that can append new blocks, construct query index and generate ADS; (2) Query User: light nodes that delegate private queries to full nodes; (3) Service Provider (SP): full nodes that provide data storage and query services for query users; (4) Garrison: a group of guard nodes that are a small part of full nodes equipped with SGX.

A blockchain consists of multiple blocks, each of which contains temporal data records. In the remained of this paper, we take cryptocurrency as an example to illustrate BlindChain, which can be simply transplanted into other types of blockchain applications. The records of cryptocurrency are a set of transactions $\{tx_1, ..., tx_n\}$. Each transaction tx_i is in the form of $\langle t_i, W_i \rangle$, where t_i is the timestamp of tx_i and W_i is the keywords set appeared in tx_i. A query user could issue a time-window keyword query $q = \langle [t_s, t_e], w \rangle$, where t_s is the start time and

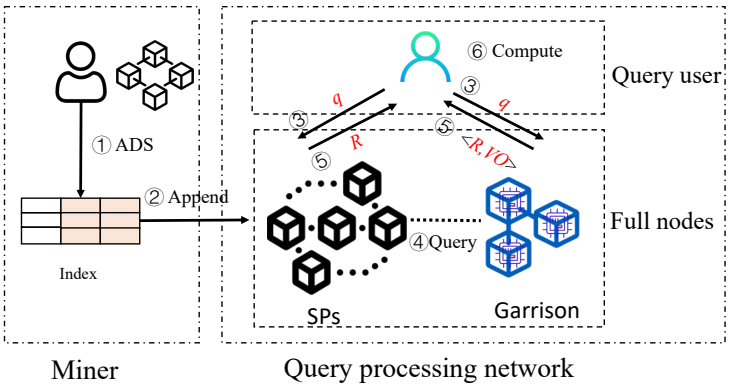


Fig. 1. BlindChain system model

t_e is the end time of a time period, and w is the queried keyword. Transactions satisfying $\{tx_i = \langle t_i, W_i \rangle | t_i \in [t_s, t_e] \wedge w \in W_i\}$ should be returned as query result of q. BlindChain allows full nodes to learn the ID of the block being accessed, but nothing about the retrieved transactions, thus protecting queried keyword w.

Fig.1 depicts the steps of a specific query. Miners first construct query index and ADS for each block in the mining process (Step ①). Then miners append new blocks to the blockchain (Step ②). A query user splits query q into multiple shares and sends at least one share to the SGX enclave of a guard node in the garrison, while sending the other shares to different SPs (Step ③). As the full nodes of the blockchain, all selected SPs and the guard node hold entire blockchain data so that they can evaluate respective shares (Step ④), and send the result back to the query user (Step ⑤). Meanwhile, the guard node also returns a verification object VO which includes proof for query user to verify the correctness of the query result. Finally, the query user can compute and verify the final query result by assembling the responses (Step ⑥).

For a query $q = \langle [t_s, t_e], w \rangle$, the time period $[t_s, t_e]$ typically covers multiple blocks. Full nodes can serially process each block, which incurs linear overhead with the number of blocks covered by the time period. To address this issue, we divide blocks into intervals and apply the execution phase in a parallel manner to boost the query performance. By integrating the paralleling technique to the above design, the query user obtains and verifies query results with lightweight overhead while hiding the privacy to the selected full nodes.

3.2 Threat Model

We follow a wildly adopted blockchain threat model [22]. Specifically, we assume that the blockchain works functionally, i.e., the majority of the miners ($> 51\%$) in the blockchain system are honest and will faithfully produce blocks and maintain the blockchain. We further assume that the blockchain network is strongly synchronized. All full nodes will sync new blocks and all light nodes will sync new block headers in time. We consider attacks on the permissionless blockchain, such as selfish mining [23] and eclipse attacks[24], are out of the scope of this paper. We consider the full node is untrusted, who may return partial transactions or incorrect transactions and spy upon query privacy for gathering users' sensitive information. A query user who wants to obtain data records on the blockchain is considered honest and is connected to the blockchain for syncing valid block headers.We assume that the hardware security enforcement of Intel SGX cannot be break.

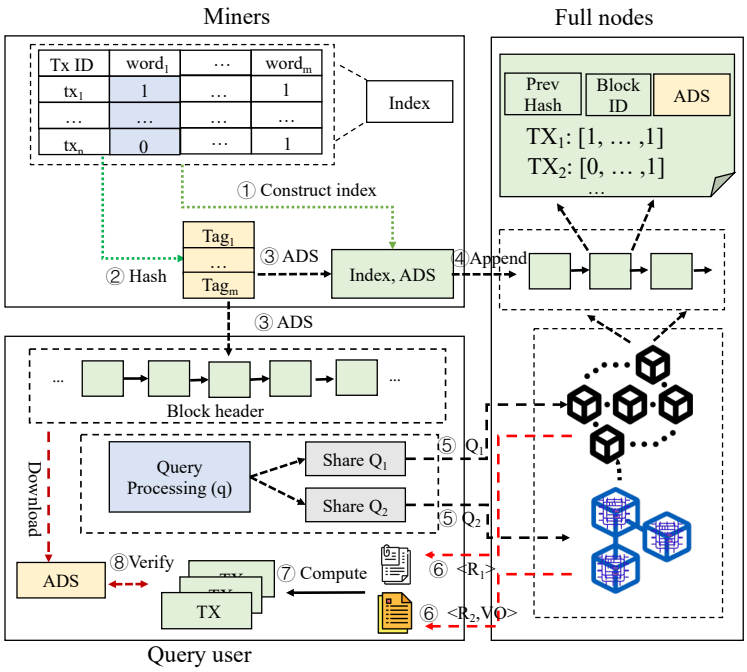


Fig. 2. Workflow of BlindChain ($l = 2$)

4 BlindChain Design

We depict the workflow of BlindChain in Fig.2 and present its details including index construction, basic query processing, verifiable query processing and parallel processing.

4.1 Index construction

In the mining process, miners construct an index for each block according to the keywords sets of the transactions in the block. As shown in Fig.2, if a block b contains n transactions and m different keywords, then the index of b can be constructed as an $n \times m$ bitmap table where each element is a single bit. The single bit in the ith row and jth column is set to "1" if the transaction tx_i contains keyword w_j, and "0" otherwise.

Based on the index, a preliminary query scheme can be realized. To query a keyword w, the query user must find all transactions where the indexes corresponding to w are set to "1". The query user can achieve this by retrieving from a full node the column corresponding to w. If the ith entry of the column is set to "1", then the query user marks transaction tx_i as a query result. However, directly using bitmap table to construct an index may incur heavy storage overhead if a block contains a large number of keywords and transactions. In addition, the evaluation of the large index will lead to linear computation overhead with the number of columns. We design a compression index based on Bloom filter to improve performance. In the mining process, miners replace the bitmap in each row with the Bloom filter of the keywords set of the corresponding transaction. The usage of Bloom filter provides compression while preserving column alignment.

4.2 Basic query processing

Using the above index, a single full node can provide service for query users by returning the column corresponding to the queried keyword w. In this way, the full node learn the keyword w, thus leaking the privacy of the query user. Therefore, BlindChain needs to hide the retrieved columns from the full node to protect query privacy. We leverage Distributed Point Functions (DPFs) which is a type of FSS to allow the query user to retrieve columns corresponding to w while hiding keyword information from the full nodes. We use the bitmap index in the remained of this paper for ease of illustration. All described schemes can be applied to the Bloom filter based index through a simple adjustment.

FSS allows a query user to split a function f into succinct function shares and send shares to different full nodes which are services hosting a copy of dataset. The query user can combine the evaluations at a given point x to recover the result, that is, $f(x) = \Sigma_{i=1}^{l} f_i(x)$. Any strict subset of the shares does not reveal anything about the function f. There are two basic FSS constructions: point functions and interval functions, which take the following forms:

- Point function

$$f_{a,b}(x) = \begin{cases} b, \, x = a \\ 0, \, \text{otherwise} \end{cases} \tag{1}$$

- Interval function

$$f_{a,b}(x) = \begin{cases} 1, \, a \leq x \leq b \\ 0, \, \text{otherwise} \end{cases} \tag{2}$$

In both functions, FSS keeps the parameters a and b private: an adversary can tell that it was given a share of point or interval function, but cannot observe a and b.

DPFs are a type of FSS that supports point functions. An l-party DPF scheme is defined by the following algorithms:

- $\mathsf{DPF.Gen}(a, b) \rightarrow (K_1, ..., K_l)$: Outputs keys $K_1, ...K_l$ that allow the l full nodes to jointly evaluate the point function $f_{a,b}(x)$.
- $\mathsf{DPF.Eval}(K_i, x) \rightarrow y_i$: Given the key K_i and a point x as input, full node i outputs value y_i.

When a query user needs to privately evaluate a point function f where $f(a) = b$ on some input x, the query user runs $\mathsf{DPF.Gen}(a, b)$ to generate keys and then distributes K_i to full node i for all l full nodes. The full node i runs $\mathsf{DPF.Eval}(K_i, x)$ and returns the result y_i to the query user. The query user can reconstruct $f(x)$ by computing $f(x) = \Sigma_{i=1}^{l} \oplus y_i$.

In BlindChain, query privacy is preserved by leveraging DPFs, allowing query users to interact with multiple full nodes rather than relying on a single full node. When querying a keyword w in a block b, the query user requests the column in the index of block b corresponding to w. Assuming this column is the j-th column of the index, the query user generates l shares of a point function that evaluates to 1 for the j-th column and 0 for all other columns. These function shares are distributed among l selected full nodes, with each node receiving one share. Each full node evaluates its function share across all columns of the index, computes the logical AND of the DPF evaluation with the column contents, and returns the XOR of the results to the query user. By aggregating the responses from all l full nodes, the query user reconstructs the j-th column of the index.

To ensure security in DPFs, at least one honest server is required. While it is assumed that more than 51% of the full nodes in a blockchain are honest, a query user cannot entirely avoid selecting malicious full nodes, particularly since the parameter l is typically much smaller than

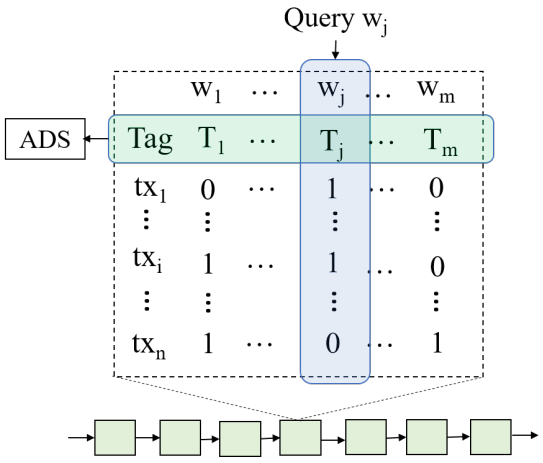


Fig. 3. ADS design

49% of the total number of full nodes. We leverage trusted hardware to address this limitation. BlindChain introduces a mechanism where a subset of full nodes, referred to as the garrison, is selected through the blockchain's consensus protocol. Each node in the garrison is equipped with an SGX enclave, acting as a guard node responsible for maintaining security. When distributing function shares, the query user is required to select at least one guard node from the garrison and send one share to its SGX enclave, while distributing the remaining shares to multiple other SPs. The SGX-equipped guard node is inherently resistant to collusion with other SPs. Consequently, even if an adversary compromises all SPs except the guard node, they cannot reconstruct the keyword w, thereby preserving query privacy.

Enabling private queries for a single block is not sufficient to meet practical query requirements. For a query $q = \langle [t_s, t_e], w \rangle$, the time period $[t_s, t_e]$ typically covers multiple blocks. Each full node can serially query every block within the time period. Specifically, the query user generates function shares according to w and l. Then the query user sends l shares to all selected SPs and the guard node. Each full node evaluates its share on the index of every block one by one and returns corresponding results. To ensure that the query results of the same block can be matched by the query user, full nodes store the query results of all blocks in a queue according to the block ID. Finally, each full node returns the queue to the query user.

4.3 Verifiable query processing

The basic processing assures query privacy by splitting the query into l shares and distributing these shares across different full nodes, including at least one guard node. However, malicious full nodes may compromise result correctness by evaluating DPF function shares over tampered indexes. Additionally, while the SGX enclave deployed on a guard node guarantees execution correctness, it cannot promise the correctness of the index provided by the guard node. To address these challenges, we design an ADS to enable query users to verify the correctness of query results. Miners generate an ADS for each block and embed it into the block header. The primary purpose of the ADS is to ensure that all selected full nodes evaluate function shares on columns corresponding to the correct index. Based on the ADS, the SGX enclave on a guard node can return a VO, which the query user can use to confirm whether the query results were derived from the correct index. This mechanism strengthens the reliability and trustworthiness of query results in BlindChain.

Algorithm 1: Full nodes protocols

Function ADSConstruct(b) (by miners):
  $index[n][m]$ = get_index(b)
  $bn \leftarrow$ blockID of b
  for $i = 1 : m$ **do**
    $s = \emptyset$
    for $j = 1 : n$ **do**
      $s = s||index[j][i]$
    $T_i = \mathsf{hash}(bn||w_i||s)$
  $ADS_b = \mathsf{hash}(T_1||...||T_m)$
  Store ADS_b in the block header of b
Function QuerySP(b, K) (by the SP):
  $index[n][m]$ = get_index(b)
  $bn \leftarrow$ blockID of b
  $r = 0$
  for $i = 1 : m$ **do**
    $s = \emptyset$
    for $j = 1 : n$ **do**
      $s = s||index[j][i]$
    $r = r \oplus (\mathsf{DPF.Eval}(K, i) \wedge s)$
  return $\langle r, bn \rangle$
Function QueryGN(b, K) (by the guard node):
  The guard node equipped with enclave TE:
  $index[n][m]$ = get_index(b)
  $bn \leftarrow$ blockID of b
  TE.**Load**($index[n][m]$)
  $r = 0$
  for $i = 1 : m$ **do**
    $s = \emptyset$
    for $j = 1 : n$ **do**
      $s = s||index[j][i]$
    $r = r \oplus (\mathsf{DPF.Eval}(K, i) \wedge s)$
  VO = TE.**Load**($\{T_1, ..., T_m\}$)
  return $\langle r, VO, bn \rangle$

Fig.3 shows the design of ADS for a block b that maintains an $n \times m$ bitmap table as an index. In the mining process, miners generate an additive tag for every column when constructing the index. Miners cannot generate a tag for the row because query users must retrieve individual columns rather than entire rows. The tag of each column must be unique to be valid. Therefore, the tag is computed over not only the column vector, but also the block ID bn and the keyword corresponding to the column. Specifically, for a column i corresponding to keyword w_i, miners generate a tag $T_i = \mathsf{hash}(bn||w_i||index[1][i]||...||index[n][i])$, where $index[n][m]$ represents the bitmap index table. After getting tags of all columns, miners construct ADS_b for block b by hashing tags of all columns: $ADS_b = \mathsf{hash}(T_1||...||T_m)$. ADS_b is stored in the block header of b.

A guard node runs the DPF share over the columns of the index of block b and sends the query result with a VO back to the query user. The returned VO should be able to prove that the column corresponding to the queried keyword w inputted to the DPF share is correct. Because of the protection of query privacy, the guard node does not know which column corresponds to w. Thus, the guard node returns $\{T_1, ...T_m\}$ as VO, which can be used to verify that all columns as the input of the DPF function shares are correct.

The query user obtains the final result R which is a column vector containing n elements by assembling responses from all full nodes. To verify the correctness of R, the query user checks whether the tag for every column is correct. Specifically, the query user first obtains $T_w = \mathsf{hash}(bn||w||R)$ and checks whether the VO contains the same value as T_w. Then the query user computes $H = \mathsf{hash}(T_1||...||T_m)$ and compares it with ADS_b stored in the block header. If all the checks are passed, the query user can confirm that all columns as the input

Algorithm 2: Light nodes protocols

Function QueryUser(b, w, l)**:**
  GenKeyShare(b, w, l)
  for $i = 1 : l - 1$ **do**
    $\langle r_i, bn\rangle = SP_i$.QuerySP$(b, K_i)$
  $\langle r_l, VO, bn\rangle$ = QueryGN(b, K_l)
  RecoVerify $(\{r_1, r_2, ..., r_l\}, VO, bn)$

Function GenKeyShare(b, w, l)**:**
  // get the column of w in the index of block b
  a = get_column(b, w)
  DPF.Gen$(a, 1) \rightarrow (K_1, ..., K_l)$
  Distributing $K_1, ..., K_{l-1}$ to differnet SPs $\{SP_1, ..., SP_{l-1}\}$ and K_l to a guard node
  return $\{K_1, ..., K_l\}$

Function RecoVerify $(\{r_1, r_2, ..., r_l\}, VO, bn)$**:**
  $R = \Sigma_{i=1}^{l} \oplus r_i$
  T_w = hash$(bn||w||R)$
  flag=True
  for $i = 1 : m$ **do**
    if $T_w == T_i$ **then**
      flag=False
      break
  if *flag* **then**
    return "verification failed"
  H = hash$(T_1||...||T_m)$
  if $H == ADS_{bn}$ **then**
    return "verification succeed"
  else
    return "verification failed"

of the DPF function shares are correct, thus ensuring the correctness of the query result. The whole query and verification process for a single block b is shown in Algorithm 1 and 2.

4.4 Parallel query processing

For a query $q = \langle [t_s, t_e], w\rangle$, the time period $[t_s, t_e]$ typically covers multiple blocks. Serially processing each block will result in linear overhead with the number of blocks. We propose an optimization technique to improve the query performance by leveraging a well-designed parallel algorithm.

Based on the structure of blockchain and the number of CPU P in the system, we define the length of block interval as $\lceil \frac{BN}{P} \rceil - 1$, where BN is the number of blocks in the query region $[t_s, t_e]$. There are P intervals maintained in the full nodes. Each CPU C processes one interval ID_C by invoking the query function which retrieves the query result in the interval $[bn_1, bn_2]$. We can use the CPU whose identifier ID_C is 0 as the manager to collect the query results using **MPI** (Massage Passing Interface) function *MPI_ Rev*. Each CPU can send its query result to the manager using function *MPI_ Send*. Specifically, if the type of the full node T is the service provide SP, the result R contains the block ID bn and the query result r. Otherwise, the result R contains bn, r and a verification object VO.

5 Security Analysis

We leverage the simulation paradigm of Secure Multi-Party Computation (SMPC) to formalize the security guarantees of BlindChain. Specifically, we define an ideal-world in which the information available to the adversary serves as the basis for characterizing the permissible information leakage of BlindChain. In this ideal-world, light nodes interact with an ideal functionality $\mathcal{F}$. For each query issued by a light node, $\mathcal{F}$ processes the request and provides a

well-defined subset of information to the adversary (denoted as a simulator $\mathcal{S}$ in the ideal-world). The subset of information that $\mathcal{F}$ reveals to $\mathcal{S}$ defines the information leakage allowed by BlindChain.

We additionally define a real-world that models the BlindChain protocol. In the real-world, light nodes interact directly with an adversary, denoted as $\mathcal{A}$, who processes requests and obtains certain information. We will demonstrate that for ideal functionality $\mathcal{F}$ in the ideal-world, there exists a simulator $\mathcal{S}$ such that $\mathcal{A}$ cannot distinguish whether it is operating in the real-world or interacting with the simulator $\mathcal{S}$. This indistinguishability ensures that any attack feasible by $\mathcal{A}$ in the real-world can also be replicated by $\mathcal{S}$ in the ideal-world. Consequently, the adversary's capabilities are constrained to those permitted by the defined security guarantees.

We model the blockchain and the consensus protocol as ideal functionalities, denoted as $\mathcal{F}_{BC}$ and $\mathcal{F}_{consensus}$, respectively, as their underlying mechanisms are well-studied and thoroughly analyzed in prior research. This abstraction allows us to concentrate on the specific components and innovations of the BlindChain protocol.

The ideal-world functionality $\mathcal{F}$ takes as input the identities of the full nodes that are compromised or may become compromised. This information is subsequently forwarded to the simulator $\mathcal{S}$, enabling it to dictate the execution accordingly. In contrast, the real-world setting models the execution of the BlindChain protocol. In this framework, we denote the guard node as P, which also interacts with $\mathcal{F}$ to handle requests. The ideal functionality $\mathcal{F}$ is formally defined as follows: Given the function $\mathsf{QueryUser}(b, w, l)$, $\mathcal{F}$ processes the query and reveals some information to $\mathcal{S}$. $\mathcal{F}$ builds index for the transactions in block b and returns the transactions that match the queried keyword w. Then, $\mathcal{F}$ sends the request type, b and $\mathcal{L}(\mathcal{F}_{BC}.\mathsf{getTrans}(b))$ to $\mathcal{S}$.

Definition 1. *Let λ be the security parameter. Let Π be the BlindChain protocol. Let $\mathcal{A}$ be an adversary that outputs a single bit. For any set of compromised full nodes ω chosen by $\mathcal{A}$, let $\boldsymbol{Real}_{\Pi,\mathcal{A},\omega}(1^\lambda)$ be the random variable denoting $\mathcal{A}$'s output when interacting with the real-world. For a simulator $\mathcal{S}$, an adversary $\mathcal{A}$ that outputs a single bit, and the set of compromised full nodes ω, let $\boldsymbol{Ideal}_{\mathcal{S},\mathcal{A},\omega}(1^\lambda)$ be the random variable denoting $\mathcal{A}$'s output when interacting with the ideal-world.*

We say that a protocol Π securely evaluates the ideal functionality $\mathcal{F}$ defined above if there exists a non-uniform algorithm $\mathcal{S}$ probabilistic polynomial-time in λ such that for every non-uniform adversary $\mathcal{A}$ probabilistic polynomial-time in λ that outputs a single bit, for every set of compromised full nodes ω chosen by $\mathcal{A}$, the probability ensemble of $\boldsymbol{Real}_{\Pi,\mathcal{A},\omega}(1^\lambda)$ over λ is computationally indistinguishable from the probability ensemble of $\boldsymbol{Ideal}_{\mathcal{S},\mathcal{A},\omega}(1^\lambda)$ over λ, that is:

$$|Pr[\boldsymbol{Real}_{\Pi,\mathcal{A},\omega}(1^\lambda) = 1]| - |Pr[\boldsymbol{Ideal}_{\mathcal{S},\mathcal{A},\omega}(1^\lambda)]| \leq negl(\lambda) \tag{3}$$

Theorem 1. *Suppose that in BlindChain, the light nodes sync new block headers in time, the hash function is collision-resistant, the DPF promises the security guarantees and the executions inside SGX enclave are integrity-proven. For every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$, for every set of compromised full nodes ω chosen by $\mathcal{A}$, the probability ensemble of $\boldsymbol{Real}_{\Pi,\mathcal{A},\omega}(1^\lambda)$ over λ is computationally indistinguishable from the probability ensemble of $\boldsymbol{Ideal}_{\mathcal{S},\mathcal{A},\omega}(1^\lambda)$ over λ.*

Proof. We consider a sequence of four hybrid setups. $\mathcal{H}_0$ is equivalent to the real-world model, and $\mathcal{H}_3$ is equivalent to the ideal-world model. In a true hybrid argument, only one operation can be modified at a time.

Hybrid $\mathcal{H}_0$. This is exactly the real-world model.

Hybrid $\mathcal{H}_1$. This is the same as $\mathcal{H}_0$, except that we replace the adversary $\mathcal{A}$ with the simulator $\mathcal{S}$. When $\mathcal{A}$ sends a message to $\mathcal{F}_{BC}$, $\mathcal{S}$ forwards the message to $\mathcal{F}_{BC}$, obtains the response and forwards it to $\mathcal{A}$, as if $\mathcal{A}$ communicated with $\mathcal{F}_{BC}$ directly. $\mathcal{S}$ acts simply as a relay, shuttling data back and forth between $\mathcal{A}$ and $\mathcal{F}_{BC}$. Therefore, $\mathcal{A}$ cannot distinguish $\mathcal{H}_0$ from $\mathcal{H}_1$.

Hybrid $\mathcal{H}_2$. This is the same as $\mathcal{H}_1$, except that we introduce the ideal function $\mathcal{F}$. $\mathcal{F}$ just relays messages back and forth between the real-world model and the simulator $\mathcal{S}$. $\mathcal{F}$ acts as another intermediate relay, thus $\mathcal{A}$ cannot distinguish $\mathcal{H}_1$ from $\mathcal{H}_2$.

Hybrid $\mathcal{H}_3$. This is the same as $\mathcal{H}_2$, except that we randomly sample columns rather than running $\mathsf{DPF.Gen}()$ on the indexes corresponding to the queried keyword. From the security of DPFs, the adversary $\mathcal{A}$ cannot distinguish the columns chosen randomly from the indexes corresponding to the queried keyword, so $\mathcal{A}$ cannot distinguish between $\mathcal{H}_2$ and $\mathcal{H}_3$. Notice that $\mathcal{H}_3$ is identical to the ideal-world that at least one full node (the guard node) is honest.

In addition, as the hash function hash is collision-resistant, the probability of an adversary $\mathcal{A}$ to input a tampered index to the DPF shares while passing the verification protocol is negligible. Specifically, this implies that $\mathcal{A}$ is able to construct a false column vector which is not equal to the column vector corresponding to the queried keyword w, but the tag T^* for the false column vector can be used to reconstruct the correct ADS stored in the block header, which contradicts to the collision-resistant of the hash function. Actually, the success probability of an adversary to produce a tag T^* such that

$$Pr[T^* \neq T_w \wedge \mathsf{hash}(T_1||...||T^*||...||T_m) = ADS] \leq \varepsilon \tag{4}$$

where ε is a negligible value, T_w is the tag for the column corresponding to w and m is the number of the index columns. It promises that the verification protocol is unforgeable.

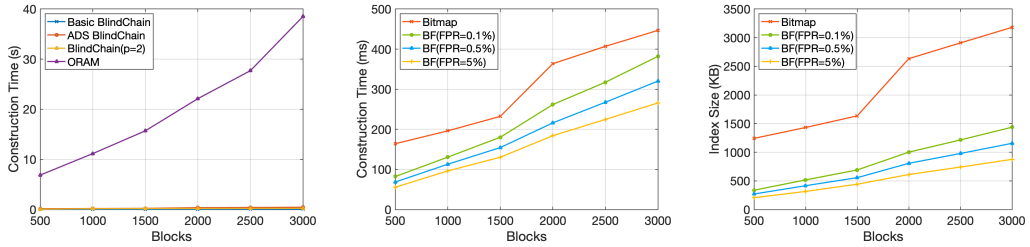
6 Performance Evaluation

We implement BlindChain in about 3000 lines of C++. We first realize a two-party DPF using a similar implementation in [12] and evaluate its performance in Section 6.1, 6.2 and 6.3. We then implement a multi-party DPF scheme based on [9] and demonstrate the scalability of BlindChain from two-party DPF to multi-party DPF in Section 6.4. BlindChain is decoupled from the underlying blockchain system so that it can be easily integrated into existing blockchains, such as Ethereum, Fabric and Bitcon. The full node is installed on a machine with Intel Core i7-10700 2.9GHz CPU and 80GB RAM, running on Ubuntu 18.04. The guard node is equipped with Intel SGX Linux SDK. The light node is installed on a machine with Intel Xeon E3-1231 v3 CPU and 8GB RAM, running on Ubuntu 18.04.

We use a real transaction dataset that is downloaded from the Ethereum blockchain database, which contains 96000 blocks with almost 1 million transaction records. Each transaction can be represented as $\langle$t, value, from, to$\rangle$, where value is the amount of Ether transferred, from and to is the address of sender and receiver respectively and t is the timestamp of the transaction. The sender/receiver addresses and transaction value are fetched as keywords.

We implement the following four systems and compare their performance during the indexing and querying phase.

- **ORAM baseline:** ORAM is the underlying technique of previous works[5][6][8], which protects query privacy at the cost of significant overhead.
- **Basic BlindChain:** Full nodes process query using DPF as described in Section 4.2. Basic BlindChain does not contain ADS construction, thus it only guarantees query privacy but does not verify result correctness.



(a) Index construction time of four systems.

(b) Construction time of different indexes.

(c) Size of different indexes.

Fig. 4. Index cost

- **ADS BlindChain:** Adding ADS construction to Basic BlindChain as described in Section 4.3. ADS BlindChain illustrates the overhead of the ADS to ensure the result correctness.
- **BlindChain:** Parallel processing blocks on the basis of ADS BlindChain. BlindChain is the prototype that combines all of our proposed techniques.

6.1 Index Cost

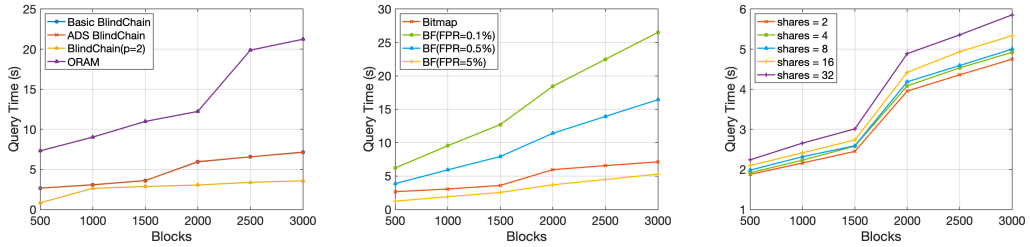
We evaluate the performance of constructing index as shown in Fig.4. We first compare the index construction time of the four systems. As shown in Fig.4(a), the index construction time increases linearly when the number of queried blocks grows. The ADS construction brings slightly increased time overhead, which is the cost for verifying the result correctness. BlindChain yields up to 100× improvement in index construction time compared with ORAM baseline. For example, BlindChain constructs indexes for 3000 blocks in less than 0.3s, while that of the ORAM baseline requires about 38s.

We then compare the index construction time and storage cost of the bitmap index and Bloom filter based index. We adjust the false positive rate (FPR) of the Bloom filter to 0.1%, 0.5% and 5%, respectively. Fig.4(b) and Fig.4(c) show the time cost and storage cost, respectively. It is obvious that the Bloom filter based indexes reduce index construction time and compress index size. In addition, improving FPR reduces the construction time and storage cost of the Bloom filter based index.

6.2 Query Cost

We evaluate the query performance for full nodes of the four systems. We omit the performance evaluation on the light node side, since their operations in our design, i.e., query generation and result verification, are very lightweight. Fig.5(a) depicts the query latency. We can observe that the query latency increases almost linearly when the number of queried blocks grows. The performance of ADS BlindChain is slightly less than that of Basic BlindChain, which indicates that ADS design to ensure result correctness brings negligible extra overhead to query performance. The BlindChain with the degree of parallelism $p = 2$ reduces the query latency by roughly a factor of p. BlindChain is up to 10× faster than the ORAM baseline.

Fig.5(b) compares the query performance of schemes based on different indexes which are bitmap index and Bloom filter based index. The bitmap index performs better than the Bloom filter based indexes with FPR 0.1% and 0.5%, but slower than it with FPR 5%. Using the

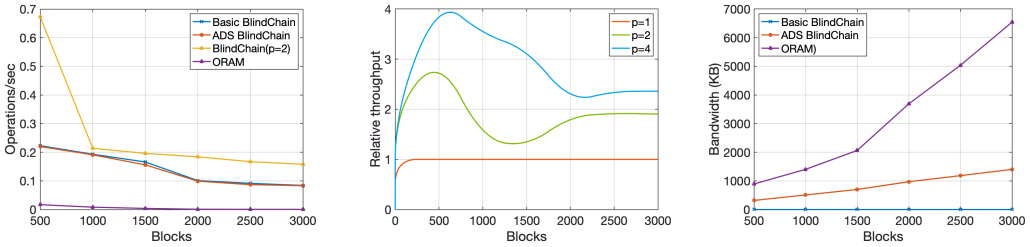


(a) Query performance of four systems.

(b) Query performance of different indexes.

(c) Query performance of multi-party FSS schemes.

Fig. 5. Query cost



(a) Throughput of four systems.

(b) Effect of parallelism (p denotes the degree of parallelism) on throughput

(c) Bandwidth for BlindChain in comparison to ORAM baseline.

Fig. 6. Throughput and bandwidth

Bloom filter based index, full nodes need to apply DPF to multiple columns rather than one column of the bitmap index, thus bringing increased time cost. However, the size of the Bloom filter based index with FPR 5% is small enough, resulting in a lower time cost than the bitmap index.

6.3 Throughput and bandwidth

We evaluate the throughput and bandwidth of the four systems and depict results in Fig.6. Fig.6(a) shows the throughput result. Basic BlindChain and ADS BlindChain have nearly the same throughput. BlindChain with $p = 2$ improves the throughput by roughly a factor of p. We can observe that BlindChain achieves almost 300× higher throughput than the ORAM baseline. We further evaluate throughput of BlindChain for different degrees of parallelism p. Fig.6(b) shows the relative throughput. Parallelism improves the throughput by roughly a factor of a little lower than p for a large number of queried blocks.

We compare the query bandwidth of Basic BlindChain, ADS BlindChain and ORAM baseline. BlindChain has the same bandwidth overhead as ADS BlindChain because the parallel technique does not affect bandwidth cost. As shown in Fig.6(c), the bandwidth scales linearly with the number of blocks. The bandwidth of Basic BlindChain is negligible. ADS BlindChain increases bandwidth cost, most of which comes from VO, but it is still much smaller than the ORAM baseline.

6.4 Multi-party FSS scheme

Fig.5(c) shows the comparison of query performance by varying the number of function shares in DPF. The experiment is done on the ADS BlindChain. We set the number of function shares to 2^n, where the integer n varies from 1 to 5. The result shows that the query latency increases almost linearly when the number of queried blocks grows for both two-party and multi-party DPF schemes. In addition, increasing the number of function shares brings acceptable query time overheads. For example, the scheme with 32 function shares increases 23% query latency compared to the scheme with 2 function shares. In conclusion, BlindChain is scalable from two-party DPF to multi-party DPF.

7 Conclusion

In this paper, we propose a novel blockchain system BlindChain which protects query privacy of light nodes and verifies correctness of query results with lightweight overhead. First, we integrate FSS into blockchain by introducing an SGX-based garrison to meet the trust model requirements of FSS in the context of blockchain. Second, we design a novel ADS for BlindChain to enforce result verification. Third, we propose an parallel optimization technique to boost query performance. Experimental results show that BlindChain outperforms the previous works in terms of both computation and communication overheads.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China under Grant 62302229, the National Natural Science Foundation of China under Grant 62172328, Natural Science Basic Research Program of Shaanxi(Program No.2024JC-JCQN-67), the Shaanxi Province QinChuangYuan "Scientist + Engineer" Team Building Project No. 2022KXJ-054.

References

1. Yanchao Zhu, Zhao Zhang, Cheqing Jin, Aoying Zhou, and Ying Yan. Sebdb: Semantics empowered blockchain database. In *IEEE ICDE*, pages 1820–1831, 2019.
2. Chengjun Cai, Yifeng Zheng, and Cong Wang. Leveraging crowdsensed data streams to discover and sell knowledge: A secure and efficient realization. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 589–599. IEEE, 2018.
3. Karl Wüst, Sinisa Matetic, Moritz Schneider, Ian Miers, Kari Kostiainen, and Srdjan Čapkun. Zlite: Lightweight clients for shielded zcash transactions using trusted execution. In *International Conference on Financial Cryptography and Data Security*, pages 179–198. Springer, 2019.
4. Arthur Gervais, Srdjan Capkun, Ghassan O Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 326–335, 2014.
5. Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. Bite: Bitcoin lightweight client privacy using trusted execution. In *USENIX Security 19*, pages 783–800, 2019.
6. Chengjun Cai, Lei Xu, Anxin Zhou, Ruochen Wang, Cong Wang, and Qian Wang. Encelc: Hardening and enriching ethereum light clients with trusted enclaves. In *IEEE INFOCOM 2020*, pages 1887–1896. IEEE, 2020.
7. Yankai Xie, Qingtao Wang, Ruoyue Li, Chi Zhang, and Lingbo Wei. Private transaction retrieval for lightweight bitcoin clients. *IEEE Transactions on Services Computing*, 2023.

8. Yifang Zhang, Mingyue Wang, Fangda Guo, Xidi Qu, Yu Guo, and Shengling Wang. Oblivchain: Enabling oblivious queries for blockchain light clients with malicious security. In *International Conference on Database Systems for Advanced Applications*, pages 3–19. Springer, 2024.
9. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.
10. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.
11. Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 871–900. Springer, 2021.
12. Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, 2017.
13. Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. {DORY}: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119, 2020.
14. Brett Hemenway Falk, Steve Lu, and Rafail Ostrovsky. Durasift: A robust, decentralized, encrypted database supporting private searches with complex policy controls. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 26–36, 2019.
15. Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523–535, 2017.
16. Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient 3-party distributed oram. In *International Conference on Security and Cryptography for Networks*, pages 215–232. Springer, 2020.
17. Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468. IEEE, 2022.
18. Cheng Xu, Ce Zhang, and Jianliang Xu. vchain: Enabling verifiable boolean range queries over blockchain databases. In *ACM SIGMOD*, pages 141–158, 2019.
19. Qifeng Shao, Shuaifeng Pang, Zhao Zhang, and Cheqing Jing. Authenticated range query using sgx for blockchain light clients. In *International Conference on Database Systems for Advanced Applications*, pages 306–321. Springer, 2020.
20. Xiaohai Dai, Jiang Xiao, Wenhui Yang, Chaofan Wang, Jian Chang, Rui Han, and Hai Jin. Lvq: A lightweight verifiable query approach for transaction history in bitcoin. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 1020–1030. IEEE, 2020.
21. Zhihao Chen, Qingqing Li, Xiaodong Qi, Zhao Zhang, Cheqing Jin, and Aoying Zhou. Blockope: Efficient order-preserving encryption for permissioned blockchain. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1245–1258. IEEE, 2022.
22. Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.
23. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.
24. Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on {Bitcoin's}{peer-to-peer} network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.