Authenticated Range Query Using SGX for Blockchain Light Clients

Qifeng Shao[1,2], Shuaifeng Pang[1], Zhao Zhang[1(✉)], and Cheqing Jing[1]

[1] School of Data Science and Engineering, East China Normal University, Shanghai, China
{shao,sfpang}@stu.ecnu.edu.cn, {zhzhang,cqjin}@dase.ecnu.edu.cn
[2] School of Software, Zhongyuan University of Technology, Zhengzhou, China

Abstract. Due to limited computing and storage resources, light clients and full nodes coexist in a typical blockchain system. Any query from light clients must be forwarded to full nodes for execution, and light clients verify the integrity of query results returned. Since existing authenticated query based on Authenticated Data Structure (ADS) suffers from significant network, storage and computing overheads by virtue of Verification Objects (VO), an alternative way turns to Trust Execution Environment (TEE), with which light clients have no need to receive or verify any VO. However, state-of-the-art TEE cannot deal with large-scale application conveniently due to limited secure memory space (i.e, the size of enclave in Intel SGX is only 128MB). Hence, we organize data hierarchically in both trusted (enclave) and untrusted memory and only buffer hot data in enclave to reduce page swapping overhead between two kinds of memory. Security analysis and empirical study validate the effectiveness of our proposed solutions.

Keywords: Blockchain · Authenticated query · MB-tree · Intel SGX

1 Introduction

Blockchain, the core technology of Bitcoin [9], is a decentralized, trustless, tamper-proof and traceable distributed ledger managed by multiple participants. Specifically, by integrating P2P protocol, asymmetric cryptography, consensus algorithm, hash chain structure and so forth, blockchain can achieve trusted data sharing among untrusted parties without the coordination of any central authority.

All nodes of blockchain are usually classified as full node and light node. Full node holds a complete copy of block data, while the light node (also called light client) only stores block header or other verification information due to limited storage resource. Queries from a light node are forwarded to full nodes for execution, but the integrity of query results returned from full node needs to be authenticated by light client itself.

Current blockchain systems have limited ability to support authenticated queries of light clients. Most of them are merely suitable for digital currency field,

Y. Nah et al. (Eds.): DASFAA 2020, LNCS 12114, pp. 306–321, 2020.
https://doi.org/10.1007/978-3-030-59419-0_19

e.g., Simple Payment Verification (SPV) in Bitcoin can only answer queries of transaction existence. To the best of our knowledge, no present system is able to handle range query and verification accordingly, like selecting transactions satisfying "2019-11 $\leq Timestamp \leq$ 2019-12". With the popularization of blockchain technology among traditional industries, the desire to support various authenticated queries becomes stronger.

In this study, we focus on authenticated range query, a representative task. The authenticated range query can be tracked back to outsourcing databases, where clients delegate data to remote database servers and initiate database query. Although the signature chaining [10] and Authenticated Data Structure (ADS), e.g., Merkle Hash Tree (MHT) [8] and Merkle B-tree (MB-tree)[3], are widely adopted to guarantee the correctness and completeness of query results, neither fits for blockchain since the former triggers tremendous signature computing cost and the latter raises the cost of VO, e.g., Merkle cryptographic proofs. Since computational cost of signature chain is determined by hardware and may not diminish in a short time, we propose a solution of blockchain based on ADS. Nevertheless, applying existing ADS to blockchain is quite challenging.

- The full node returns query results along with cryptographic proofs, known as VO. On the client side, the splicing and authentication of these VO require significant network and computing resources.
- As updates lead to hash computing and signature costs, traditional ADS solutions assume that the databases they're serving have fixed or fewer updates, which is not applicable to the case of blockchain since blocks are appended to the blockchain periodically.

Consequently, a new solution that assures the integrity of query results and further effectively reduces the costs of verification and maintenance needs to be devised.

Recently, the emergance of trusted hardware (Trusted Execution Environment, TEE) that supports secure data accessing offers a promising direction of designing range query authentication schemes. For example, Intel Software Guard Extensions (SGX) [7] can protect code and data from disclosure or modification, and enforce the security level of application. SGX allows to create one or more isolated contexts, named enclaves, which contain segments of trusted memory. Hence, to guarantee integrity and confidentiality, sensitive codes are installed in enclave and run on untrusted machines.

The special region of isolated memory reserved for enclave is called Enclave Page Cache (EPC). Currently, EPC has a maximal size of 128 MB, of which only 93 MB are utilizable for applications. EPC page faults occur when the code accesses the pages outside of enclave. Page swapping is expensive, because enclave memory is fully encrypted and its integrity also needs to be protected. Two built-in wrapper codes, *ecall* and *ocall*, respectively invoke enter and exit instructions to switch the execution context. These two codes add overhead of approximately 8,000 CPU cycles, compared to 150 cycles of a regular OS system call [12]. Though the emergence of SGX solves the secure remote computing

problem of sensitive data on untrusted servers, the performance implications of SGX remain an open question. When applying Intel SGX to blockchain, effective optimization strategies must be considered.

In summary, this paper proposes an efficient query authentication scheme for blockchain by combining ADS-based MB-tree with Intel SGX. To the best of our knowledge, it is the first step toward investigating the problem of query authentication with SGX over blockchain. Our main contributions are as follows.

- An efficient SGX-based query authentication scheme for blockchain is proposed, with which light clients have no need to receive or verify any VO.
- A solution of integrating MB-tree with SGX is devised in view of the space limitation of enclave memory. Only frequently-used MB-tree nodes are cached in enclave.
- To reduce the cascading hash computing cost brought by item-by-item updates on MB-tree, a hybrid index consisting of an MB-tree and a skip list in enclave is provided.
- We conduct empirical study to evaluate the proposed techniques. Experimental results show that the efficacy of the proposed methods.

The rest of the paper is organized as follows. Section 2 reviews existing works. Section 3 introduces the problem formulation. Section 4 presents our solution of authenticated range query with SGX. The batch update is discussed in Sect. 5. The security analysis is presented in Sect. 6. The experimental results are reported in Sect. 7. Section 8 concludes this paper.

2 Related Work

To the best of our knowledge, no existing work explores authenticated range queries using SGX for blockchain. In the following, we briefly review related studies and discuss relevant techniques.

Query Authentication over Traditional Database. Query authentication has been extensively studied to guarantee the results' integrity against untrusted service providers. There are two basic solutions to ensure correctness and completeness, signature chaining [10] and ADS. For signature chaining, requiring every tuple being signed, the servers using aggregated signatures always return only one signature regardless of the result set size, and the client can process aggregate verification. Signature chaining features small VO size and communication cost, but it cannot scale up to large data sets because of high cost of signing all tuples. For ADS, MHT [8] and MB-tree [3] are widely adopted. MHT solves the authentication problem for point queries. MB-tree combines MHT with B^+-tree to support authenticated range queries. As MB-tree enables efficient search as B^+-tree and query authentication as MHT, our proposed solution uses this approach. MB-tree has also been studied to support authenticated join [14] and aggregation queries [4]. These works are more about outsource databases, insufficient for the case of blockchain.

Query Authentication over Blockchain. SPV, introduced by Satoshi Nakamoto [9], can only verify if a transaction exists in the blockchain or not. Hu et al. [2] leverage smart contract for trusted query processing over blockchain, focusing on file-level keyword searching without investigating the indexing issue. To support verifiable query over blockchain, Xu et al. [13] propose an accumulator-based ADS scheme for dynamic data aggregation over arbitrary query attribute, but blockchain clients need to receive and verify VO. Zhang et al. [15] present a gas-efficient scheme to support authenticated range query by utilizing multiple MB-trees.

Blockchain with Intel SGX. Present blockchain systems mainly perform software-based cryptographic algorithms to ensure the trusty of data. The appearance of trusted hardware, Intel SGX, opens up new possibility to enhance integrity and confidentiality of blockchain. Town Crier [16], an authenticated data feed system between existing web sites and smart contracts, employs SGX to furnish data to Ethereum. Ekiden [1] enables efficient SGX-backed confidentiality-preserving smart contracts and high scalability. BITE [6] leverages SGX on full nodes, and serves privacy-preserving requests from light clients of Bitcoin. Although these existing studies harmonize blockchain and SGX, none of them explore query authentication with SGX.

3 System Overview

Architecture. Figure 1 elucidates our system model that consists of a full node and a light client. The queries from the light client are forwarded to the full node for processing. The full node must prove it executes queries faithfully and returns all valid results, since query results may be maliciously tampered with. Traditional solutions organize data with MB-tree, and provide light clients with both query results and cryptographic proofs (VO) for further authentication. In our case, however, a big VO, especially when processing range queries, may be beyond the processing capacity of light clients like mobile devices. Consequently, our system is equipped with Intel SGX, providing integrity and confidentiality guarantees on untrusted full nodes. The query results are returned to clients through a secure channel. Clients can trust these query results without receiving or verifying any VO. Considering limited enclave memory, we organize data hierarchically in trusted memory (enclave) and untrusted memory, where skip list and MB-tree are adopted in trusted memory and untrusted memory respectively. Skip list in enclave buffers appended blocks. It merges block data into MB-tree periodically once exceeding the predefined threshold. It is worthy to note that a hot cache, residing in enclave, caches the frequently-used MB-tree nodes. These nodes will no longer be verified in future queries. More details are discussed in Sects. 4 and 5.

Adversary Model. In this study, we assume that there is no specific affiliation between light clients and full nodes. The full node is treated as a potential adversary since no participant in the blockchain network trust others. To address such

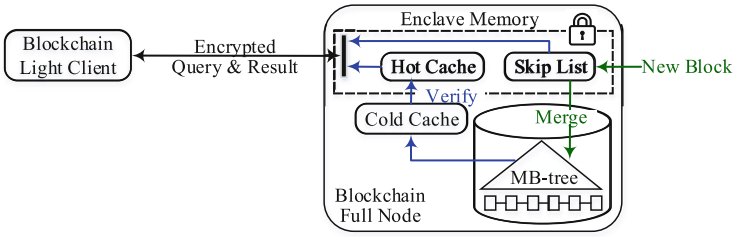


Fig. 1. System architecture.

a threat and free the clients from trivial verification process, we apply TEE-based Intel SGX to process query with integrity assurance. Since enclave memory is limited, we employ an authenticated index structure, MB-tree, outside of enclave to guarantee data integrity. Even though an adversary may compromise OS and other privileged softwares on a full node, it cannot break the hardware security enforcement of Intel SGX. With our hardware-based model, clients can trust the correctness and the completeness of query results under the following criteria.

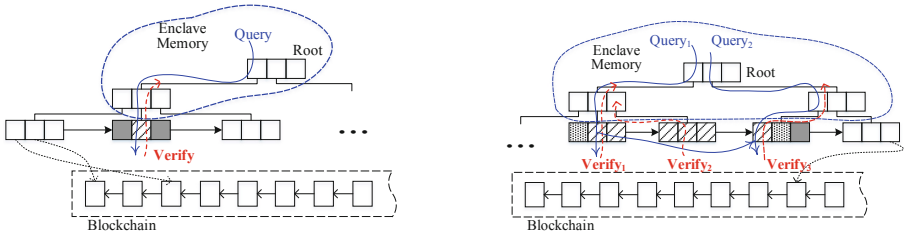
- **Correctness.** All results that satisfy the query conditions haven't been tampered with.
- **Completeness.** No valid result is omitted regarding the range query.

4 Authenticated Range Query with SGX

SGX can protect code and data from disclosure or modification. Hence, an ideal solution to guarantee query results' integrity is to install the entire storage engine and execute all queries in enclave, which eliminates computing and network overheads induced by VO in traditional solutions. However, the memory limitation of enclave makes it inadequate to handle large scale applications. In this study, we design a mechanism to organize data hierarchically in untrusted and trusted memory. Meanwhile, the data in untrusted memory is organized as MB-tree and the frequently-accessed internal nodes are cached in enclave as trusted checkpoints. A skip list, maintained in trusted memory, buffers newly appended block data. Once the capacity of the skip list reaches a threshold, a merge operation is launched from skip list to MB-tree.

4.1 MB-tree in SGX

In this solution, the root node of MB-tree is always resident in enclave, and the rest are loaded into enclave according to the query request during runtime. After verifying the Merkle proofs of a node, it is believed to be trusted and used for searching. The frequently-used nodes are cached in enclave to implement authenticated query cheaply, and the rest nodes are outside enclave to alleviate the size limitation of enclave.



(a) Point query with SGX. (b) Range query with SGX.

Fig. 2. Query and verify on MB-tree with SGX.

In MB-tree, each node contains $f-1$ index keys and f pointers to the child nodes, where f is the fanout. Like MHT, each pointer is augmented with a corresponding digest. In the leaf node, each digest is a hash value $h = H(r)$, where r is the record pointed by pointer entry. In the internal node, each digest is a hash value $h = H(h_1||...||h_f)$, where $h_1, ..., h_f$ are the hash values of the child nodes. Then, recursively up to the root node, the contents of all nodes in the entire tree are reflected to the root node by hash. Since the root node involves digest about every node, the entire tree data can be verified based on the root node. Therefore, attackers cannot modify or replay any value in the tree. MB-tree can be created either from scratch or based on existing data. The enclave on an untrusted full node is firstly authenticated through remote attestation of Intel SGX. Once passing the remote attestation, we provision the root node into the enclave through a secure channel. When the MB-tree maintaining thread in enclave receives a new block, transactions are extracted and verified based on the verification rules of blockchain before updating the corresponding index items.

We now illustrate the query and authentication process on an MB-tree with SGX. Figure 2(a) demonstrates authenticated point queries on MB-tree in enclave. The query process is the same as the traditional one, during which, accessing nodes from root to leaf, appending hashes of sibling nodes to VO and returning the query results. SGX can perform all the authentication works, so light clients don't have to receive or verify any VO. Since enclave may cache previously verified nodes, when computing Merkle proofs from bottom to up, i.e., verification path, the authenticating process can be early terminated once encountering a node located in enclave, as Fig. 2(a) illustrates (the red dashed arrow).

Different from point queries, authenticated range queries ensure the correctness and completeness of results at the same time. Thus, the left and right boundaries of results should also be included in VO for further completeness authentication. As demonstrated by red dash arrows in Fig. 2(b), the results of range query involve multiple consecutive leaf nodes, i.e., a number of verification paths. The authenticating process goes like this: the leftmost and rightmost leaf nodes are responsible for computing the node digest by considering query results,

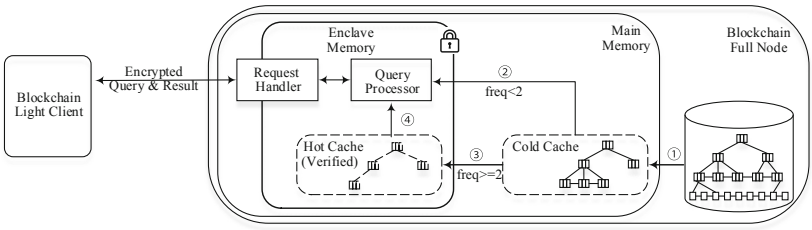


Fig. 3. Cache architecture of query processing.

sibling hashes and boundaries synthetically, while the leaf nodes located between them only compute the digest based on all results in the node. Same as point query, a verified node in enclave can cut short the verification path and further reduce the computational cost. Specifically, when all the leaf nodes covered by the results are in enclave, there is no need to perform any verification. Thus, SGX simplifies and improves the query authentication.

4.2 Cache Architecture of Query Processing

To improve accessing efficiency of MB-tree, we design three-level storage architecture, including disk storage, cold cache and hot cache. Figure 3 details the cache architecture. Disk storage at the lowest level persists the entire MB-tree. Cold cache, on untrusted memory, caches the MB-tree nodes to reduce I/O cost. Hot cache, on trusted enclave memory, only caches frequently-used and verified MB-tree nodes to alleviate verifying cost. We integrate these two types of caches and design an efficient cache replacement strategy.

When applying simple and efficient LRU cache replacement algorithm to MB-tree in enclave, burst accesses and sequential scans will make enclave read in nodes only accessed once. These nodes will not be swapped out of enclave in a short time, which lowers the utilization of the enclave memory. Motivated by the LRU-2 cache replacement algorithm [11] that keeps around the last 2 reference times for each page to estimate evicted page, we propose a replacement algorithm, Hierarchical Least Recently Used (H-LRU), for the two-level cache architecture composed of hot cache and cold cache. H-LRU considers more of the reference history besides the recent access for each node and addresses the problems of correlated references.

As shown in Algorithm 1, when a node of MB-tree is accessed for the first time, it's read out from the disk and buffered in the cold cache, thus avoiding the I/O cost in the subsequent accesses. When such node is accessed again, if that's been quite a while since the last access, i.e., uncorrelated reference, it is promoted to the hot cache, thus eliminating the verifying cost in the future queries. Algorithm 1 uses the following data structure.

– **HIST(n,t)** denotes the history of reference times of node n, discounting correlated references. $HIST(n, 1)$ denotes the last reference, $HIST(n, 2)$ the second to the last reference.

Algorithm 1: H-LRU

```
   Input: Node n, Time t                          /* n is referenced at time t */
 1 if n ∈ HotCache then
 2     if isUncorrelated(n, t) then
 3         move n to the head of HotCache;
 4 else if n ∈ ColdCache then
 5     if isUncorrelated(n, t) then
 6         if HotCache.isFull() then
 7             move the tail out of HotCache;
 8         add n to the head of HotCache;
 9 else                                              /* n is not in memory */
10     if ColdCache.isFull() then                 /* select replacement victim */
11         min ← t;
12         foreach Node i ∈ ColdCache do
13             if t − LAST(i) > CR_Period&&HIST(i, 2) < min then
                /* CR_Period: Correlated Reference Period */
14                 victim ← i;                     /* eligible for replacement */
15                 min ← HIST(i, 2);
16         move victim out of ColdCache;
17     add n to ColdCache;
18     HIST(n, 2) ← HIST(n, 1);
19     HIST(n, 1) ← t;
20     LAST(n) ← t;

21 function IsUncorrelated(n, t)
22     flag ← FALSE;
23     if t − LAST(n) > CR_Period then        /* an uncorrelated reference */
24         HIST(n, 2) ← LAST(n);
25         HIST(n, 1) ← t;
26         LAST(n) ← t;
27         flag ← TRUE;
28     else                                     /* a correlated reference */
29         LAST(n) ← t;
30     return flag;
```

– **LAST(n)** denotes the time of the last reference to node n, which may be a correlated reference or not.

Simple LRU may replace frequently referenced pages with pages unlikely to be referenced again. H-LRU moves hot nodes to the enclave memory and permits less referenced nodes to stay in normal memory only. When hot cache is full, it evicts the least recently used node. When cold cache is full, it evicts the node whose second-most recent reference is furthest in the past. Our algorithm takes both the recentness and frequency into consideration and avoids the interference

of related references, so as to improve the utilization of the enclave memory and achieve better performance.

5 Batch Updates

For MB-tree, whichever leaf node is updated, its digest will be propagated up to the root node, which incurs significant computational cost. If the entire subtree to be updated is cached in enclave, and after a certain period, the digest changes caused by multiple updates are merged and written back to the root node for one time, the update cost of MB-tree will be significantly reduced. In addition, different from traditional databases that are randomly updated at any time, the characteristic of blockchain that periodically submits transactions by block is very suitable for the scenario of batch updates undoubtedly.

Since only the signed root node is trusted in traditional MB-tree, its digest changes must be propagated to the root node immediately when any leaf node is updated. When dealing with frequent updates, such a pattern will surely downgrade system performance. With SGX, all nodes cached by enclave are verified and trusted as mentioned before. The propagation of digest changes can end at an internal node located in enclave memory.

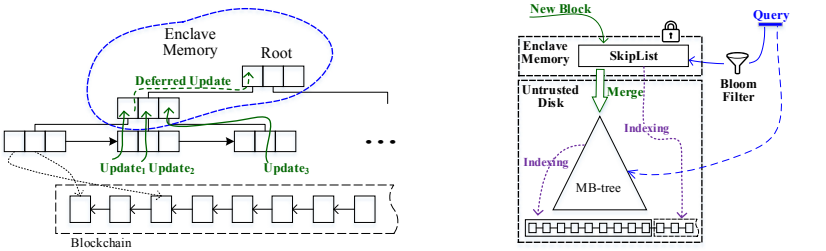
As shown in Fig. 4(a), $Update_1$, $Update_2$, and $Update_3$ respectively represent three update operations on different leaf nodes. When combined with SGX, the parent node of three updates is verified and trusted, so they just need to propagate the digest to the parent node. When the parent node is swapped out by the replace algorithm, or its structure changes due to splitting or merging, the updated digest reflecting three changes will be propagated to the root node.

5.1 Batch Updates with Hybrid Index

In addition to the cost of propagating digest, if the updated MB-tree occurs node splitting and merging, it will further amplify the update cost. The lock operation for updating node will block the query and limit concurrency. To alleviate the cost of MB-tree update, previous works generally adopt batch update that defers the installation of a single update and waits for a batch of updates to process at specific time intervals. For blockchain, it accumulates multiple update transactions to a block according to the time interval or the number of transactions and submits them by blocks, so the blockchain is more suitable for batch update.

This paper presents a dual-stage hybrid index architecture. As shown in Fig. 4(b), it allocates additional trusted space in enclave to create a skip list buffering the incoming new blocks. Compared to B^+-tree, red-black tree and other balanced trees, skip list is more suitable for memory index and has no additional rebalancing cost. Our hybrid index is composed of skip list and MB-tree, where the skip list located in enclave, is used to index newly appended block, and the MB-tree, located in disk, is used to index historical block. Query processor searches the skip list and MB-tree at the same time, and then merges the results returned by skip list and MB-tree to get the full results. A bloom filter atop of the skip list is added to improve query efficiency.

5.2 Merge



(a) Deferring digest updates. (b) Dual-stage hybrid index.

Fig. 4. Batch update and merge.

The main purpose of applying merge mechanism in this paper is to utilize batch updates to alleviate the cost of the digest propagation in MB-tree. Blockchain updates data by block periodically, which is different from existing databases, so its merging algorithm and merging strategy are different from previous designs.

There are two solutions for batch updates in MB-tree: full rebuild and delta update. Full rebuild merges and reorders existing leaf nodes of MB-tree with new data and rebuilds the entire MB-tree. Delta update adds new sorted data to the MB-tree in batch. When full rebuild is applied to MB-tree, it will incur considerable cost to recompute the digest of the entire MB-tree and block queries for a long time. Therefore, delta update is selected as our merging algorithm, as shown in Algorithm 2.

The batch update algorithm is efficient because it performs searching and propagates digest updates only once for all keys belonging to the same leaf. When advancing down the tree, the algorithm applies the lock-coupling strategy of nodes, which means only the leaf node and its parent are locked exclusively. The parent is kept locked until all child nodes have been updated and the digest changes from them have been applied.

To find the leaf for a search key, the procedure *Search* starts from the root and advances down along a path by using search key. If an internal node is full or half full, the split or merge operation is triggered accordingly.

Since our system has hot and cold caches for query processing, and updates are processed in batch to reduce MB-tree update cost, we treat the skip list as a write buffer that continuously accumulates new blocks from the blockchain network and move the entire data out of skip list at one time.

Further, it is necessary to determine the specific threshold about how many blocks are buffered for one merge. If the number of buffered blocks is too small to form a considerable sequence length, the MB-tree update cost will not be effectively reduced. Considering the query time in skip list, too many buffered blocks will take longer to query and process the merge.

Algorithm 2: Batch Updates

```
   Input: Node root, Transaction[] txs        /* txs is sorted in skip list */
 1 i ← 1;
 2 parent ← root;                                  /* searching from root */
 3 X-LOCK(parent);
 4 while i ≤ txs.length do
 5     leaf ← Search(parent, txs[i].key);
 6     repeat
 7         if txs[i].op = INSERT then
 8             insert (txs[i].key, txs[i].poniter, txs[i].digest) into leaf;
 9         else
10             delete (txs[i].key, txs[i].pointer, txs[i].digest) from leaf;
11         i ← i + 1;
12     until all txs belonging to leaf have been inserted/deleted OR parent has
         either n − 1 children when deleting or 2n children when inserting;
13     if parent is not in enclave then
14         verify parent and move it into enclave;
15     updateDigest(leaf, parent);  /* propagating digests to parent only */
16     UNLOCK(leaf);
17     if txs[i].key is in the range of parent and parent has either n − 1 or 2n
         children OR txs[i].key is not in the range of parent then
18         updateDigest(parent, root);       /* propagating digests to root */
19         UNLOCK(parent);
20         parent ← root;                          /* re-searching from root */
21         X-LOCK(parent);
22 UNLOCK(parent);

23 function Search(parent, k)
24     node ← getChildNode(parent, k);     /* node is the child of parent */
25     X-LOCK(node);
26     while node is not a leaf node do
27         if node contains 2n keys then
28             split(parent, node);
29         else if node contains n − 1 keys then
30             merge(parent, node);
31         UNLOCK(parent);
32         parent ← node;                       /* making node as new parent */
33         node ← getChildNode(parent, k);
34         X-LOCK(node);
35     return node;
```

6 Security Analysis

In this section, we perform security analysis. Our basic security model of range query authentication is secure, if the underlying hash function is collision-resistant and security enforcement of SGX cannot be broken.

Tampering Attack. As the frequently-accessed nodes of MB-tree and the entire skip list are resident in enclave, attackers cannot tamper with them. Although, the other nodes of MB-tree located in normal memory can be tampered by adversaries, the integrity of query results returned can been authenticated by the verified nodes in enclave. Using the query results and VO, query processor reconstructs the digests up to the root, and compares the root digest against that in enclave. Considering a node being successfully tampered with, there exist two MB-trees with different nodes but the same root digest. This implies a successful collision of the underlying hash function, which leads to contradiction.

Rollback Attack. In rollback attack, the untrusted node can replace the MB-tree with an old version, which makes the clients read stale results. A trusted monotonic counter can protect the latest version of MB-tree. To detect and defend the rollback attack, we use SGX monotonic counter service or rollback-protection system such as ROTE [5] to guarantee the freshness of query results.

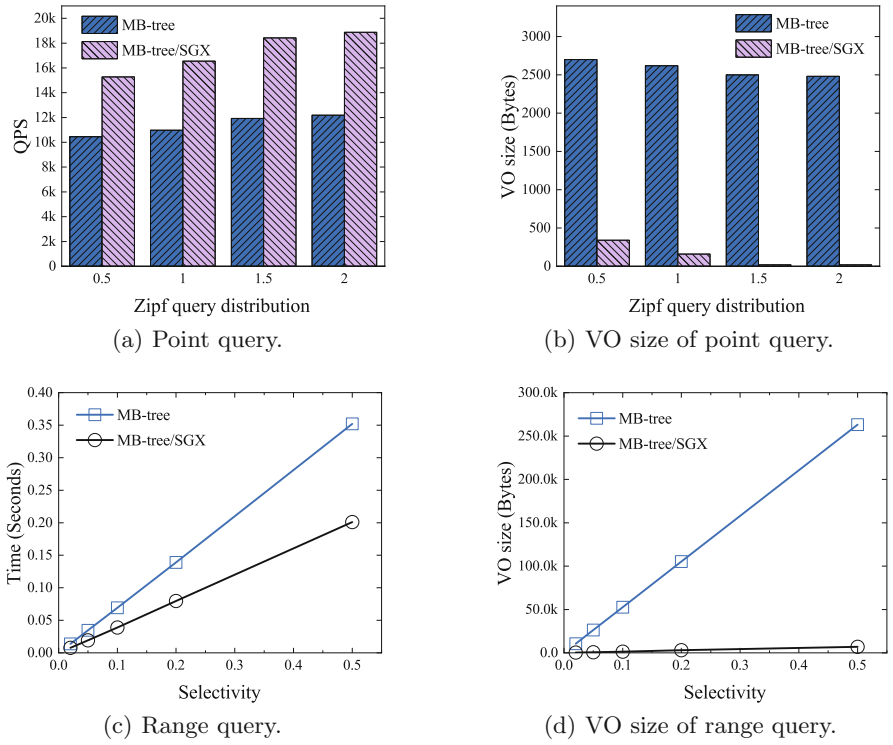
Untrusted Blockchain Data. In our solution, the SGX on an untrusted full node performs all verification for clients, yet the untrusted full node can deliver incorrect or incomplete blocks, even not send the latest block to the enclave. To protect against such compromise, a client needs to acquire the latest block hash from other sources, compares the block hash to that from the SGX, and deduces if the results are integrity or not.

7 Implementation and Evaluation

In this section, we evaluate the performance of our proposed scheme that integrates MB-tree and Intel SGX, including authenticated query, cache architecture and batch updates.

7.1 Experimental Setup

We use BChainBench [17], a mini benchmark for blockchain database, to generate synthetic blockchain dataset that consists of 1 million transactions, of which each key has 8 bytes and value has 500 bytes. We implement and construct an MB-tree with 2 KByte page size. For each node of MB-tree, both the key and pointer occupy 8 bytes and the digest uses 20 bytes, which makes each node index 56 entries ($\lfloor(2048 - (8 + 20))/(8 + 8 + 20)\rfloor$=56). Initially, our MB-tree is stored on disk, except that a copy of the root node is located in enclave. All experiments were conducted on a server, which is equipped with a 32 GB RAM and an Intel Core i7-8700k CPU @2.70Hz, and runs Ubuntu 16.04 OS with Intel SGX Linux SDK and SGXSSL library.



(a) Point query. (b) VO size of point query.

(c) Range query. (d) VO size of range query.

Fig. 5. Query performance and VO size.

7.2 Query Performance

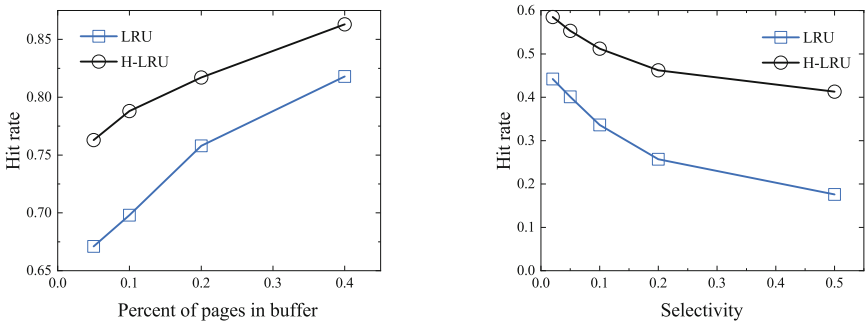
Figure 5(a) manifests the performance of point query in Zipfian distribution. With the increment of skew parameter, the throughput of MB-tree in SGX is about 1.6 times more than traditional MB-tree, because the frequently-used MB-tree nodes in enclave cut short the verification path. In Fig. 5(b), the VO size of MB-tree in SGX decreases by one or two order of magnitude. For traditional MB-tree, the query authentication is evaluated on the server, so the performance of light clients on mobile devices becomes worse when the verification is performed locally. For MB-tree in SGX, the verification is accomplished by SGX on full node, so that light clients avoid receiving and processing VO.

Figure 5(c) demonstrates the performance of range query in Zipfian distributions. The executing time of MB-tree in SGX is merely 60% of the traditional MB-tree when the selectivity is set to 50%. In Fig. 5(d), the reduction of VO size is more remarkable, since range query owns much more verification information than point query. Tens of kilobytes VO exhibit significant network overhead for lights clients, especially for mobile devices.

7.3 Cache Performance

Figure 6(a) reports the performance of H-LRU and LRU. We run 100,000 point queries, and report cache hit rate, i.e., the number of accessing MB-tree nodes located in hot cache to the number of accessing all nodes. We change the cache size from 5% to 40% of the cache size of the highest hit rate. H-LRU provides about 10% improvement over traditional LRU. The performance boost is higher with smaller cache size.

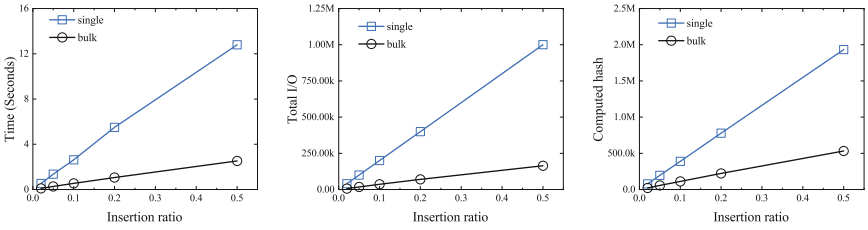
In Fig. 6(b), we randomly mix some range queries in point queries, which will start scan operations occasionally. We set the probability of starting a range query to 0.1, i.e., one-tenth of the generated queries are range queries. We vary the selectivity based on the cache size of the highest hit rate. The experiments confirm that H-LRU is more adaptable than LRU.



(a) Hit rate vs. cache size. (b) Hit rate vs. selectivity.

Fig. 6. The effect of the H-LRU.

7.4 Update Performance



(a) Total update time. (b) Total I/O. (c) Number of re-hashing

Fig. 7. Update performance.

Figure 7 presents the performance of batch update. Due to insufficient space, batch update only consists of a number of uniform insertions, ranging from 1% to 50% of the total blockchain data size. When the insertion ratio reaches 50%, the update time and the number of re-hashing are diminished by about 4 times, and I/O cost is reduced by about 6 times. It is because MB-tree is bulk-loaded with 70% utilization, and batch insertions quickly lead to many split operations, which creates a lot of new nodes. Although most of improvements are contributed by reducing of I/O cost, our batch update algorithm avoids hash computing being propagated to the root node and reduces the lock operations. Traditional MB-tree requires one expensive signature re-computation for every update. In order to show only the update performance of the tree, we did not consider that factor.

8 Conclusion

We explore the problem of authenticated range queries using Intel SGX for blockchain light clients. The main challenge lies in how to design an authenticated query scheme with memory-limited enclave. We propose a solution by integrating MB-tree with SGX, which caches frequently-used MB-tree nodes in trusted enclave and the rest nodes in untrusted memory. An efficient cache replacement algorithm, H-LRU, is devised for the two-level cache architecture, which considers more of the reference history to improve enclave memory utilization. To reduce the cascading hash computing brought by updates on MB-tree, we provide a hybrid index consisting of an MB-tree and a skip list in enclave, which buffers multiple new blocks in skip list and regularly merges them to MB-tree in batch. Security analysis and empirical results substantiate the robustness and efficiency of our proposed solution. In future, we plan to extend our idea to process other authenticated queries, such as join and aggregation.

Acknowledgment. This research is supported in part by National Science Foundation of China under grant number U1811264, U1911203, 61972152 and 61532021.

References

1. Cheng, R., et al.: Ekiden: a platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 185–200. IEEE (2019)
2. Hu, S., Cai, C., Wang, Q., Wang, C., Luo, X., Ren, K.: Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 792–800. IEEE (2018)
3. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proceedings of the 2006 International Conference on Management of Data, pp. 121–132. ACM (2006)
4. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Authenticated index structures for aggregation queries. ACM Trans. Inf. Syst. Secur. (TISSEC) **13**(4), 32 (2010)

5. Matetic, S., et al.: ROTE: rollback protection for trusted execution. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 1289–1306 (2017)
6. Matetic, S., Wüst, K., Schneider, M., Kostiainen, K., Karame, G., Capkun, S.: BITE: bitcoin lightweight client privacy using trusted execution. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 783–800 (2019)
7. McKeen, F., et al.: Innovative instructions and software model for isolated execution. HASP@ISCA **10**(1) (2013)
8. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
9. Nakamoto, S., et al.: Bitcoin: a peer-to-peer electronic cash system (2008)
10. Pang, H., Tan, K.L.: Authenticating query results in edge computing. In: Proceedings of the 20th International Conference on Data Engineering, pp. 560–571. IEEE (2004)
11. Robinson, J.T., Devarakonda, M.V.: Data cache management using frequency-based replacement, vol. 18. ACM (1990)
12. Weisse, O., Bertacco, V., Austin, T.: Regaining lost cycles with hotcalls: a fast interface for SGX secure enclaves. ACM SIGARCH Comput. Archit. News **45**(2), 81–93 (2017)
13. Xu, C., Zhang, C., Xu, J.: vChain: enabling verifiable Boolean range queries over blockchain databases. In: Proceedings of the 2019 International Conference on Management of Data, pp. 141–158. ACM (2019)
14. Yang, Y., Papadias, D., Papadopoulos, S., Kalnis, P.: Authenticated join processing in outsourced databases. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, pp. 5–18. ACM (2009)
15. Zhang, C., Xu, C., Xu, J., Tang, Y., Choi, B.: Gemˆ2-tree: a gas-efficient structure for authenticated range queries in blockchain. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 842–853. IEEE (2019)
16. Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.: Town crier: an authenticated data feed for smart contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 270–282. ACM (2016)
17. Zhu, Y., Zhang, Z., Jin, C., Zhou, A., Yan, Y.: SEBDB: semantics empowered blockchain database. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1820–1831. IEEE (2019)