DCert: **Towards Secure, Efficient, and Versatile Blockchain Light Clients**

Yang Ji
yangji@comp.hkbu.edu.hk
Hong Kong Baptist University
Kowloon, Hong Kong

Cheng Xu
chengxu@comp.hkbu.edu.hk
Hong Kong Baptist University
Kowloon, Hong Kong

Ce Zhang
cezhang@comp.hkbu.edu.hk
Hong Kong Baptist University
Kowloon, Hong Kong

Jianliang Xu
xujl@comp.hkbu.edu.hk
Hong Kong Baptist University
Kowloon, Hong Kong

ABSTRACT

Light clients have been widely used in blockchain systems to support lightweight nodes by synchronizing and verifying block headers only. However, there are two major limitations with the current light client design. First, with the ever increasing blockchain size, the cost for light clients to process and store all the block headers would soon become prohibitively high. Second, only simple queries can be supported by light clients due to the limited functionality of block headers. To address these issues, in this paper, we propose DCert, a novel decentralized certification framework, to enable *superlight* clients with *constant* storage and state validation costs. The main idea is to leverage a trusted enclave (e.g., Intel SGX) to recursively certify the entire history of the blockchain. With DCert, the blockchain integrity can be easily validated by superlight clients with a secure certificate. Furthermore, to support rich verifiable queries on light clients, DCert can be extended to certify authenticated indexes for different types of queries on an as-needed basis. While DCert is compatible with existing blockchain systems, its security is guaranteed by the trusted enclave. Our benchmark-based empirical study shows that DCert incurs a small certification overhead, yet it is capable of supporting efficient verifiable queries with a constant storage size of 2.97 KB and a constant bootstrapping time of 0.14 ms.

CCS CONCEPTS

• **Security and privacy → Hardware-based security protocols**; **Distributed systems security**.

KEYWORDS

Trusted Hardware; Light Clients; Blockchains; Authenticated Index

ACM Reference Format:
Yang Ji, Cheng Xu, Ce Zhang, and Jianliang Xu. 2022. DCert: Towards Secure, Efficient, and Versatile Blockchain Light Clients. In *23rd ACM/IFIP International Middleware Conference (Middleware '22), November 7–11, 2022, Quebec, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3528535.3565250

1 INTRODUCTION

Recent years have witnessed the rise of blockchain as a revolutionary technique to empower cryptocurrencies and decentralized applications [23, 30]. It is an append-only ledger that records the transactions that are agreed upon by mutually untrusted nodes. The data recorded on the blockchain is immutable and tamper-resistant due to the hash-chaining design and the utilization of distributed consensus protocols. A typical blockchain network consists of three types of nodes: *full node*, *miner*, and *light client*. A full node stores the complete blockchain data, including block headers, transactions, and states. A miner is a special full node that can also propose new blocks. A light client keeps track of only block headers to ensure the blockchain integrity and verify specific transaction/state data retrieved from full nodes. The light client design enhances the decentralization and robustness of the blockchain since more nodes with limited storage resources can also participate in the blockchain network.

Although a light client has a much smaller storage requirement than a full node, its storage overhead is still considerably high, especially when the blockchain grows longer and longer. For example, an Ethereum light client must store at least 7.93 GB of block headers as of September 2022.[1] Such a significant storage requirement may prevent some resource-limited clients (e.g., mobile or IoT devices) from joining the network. Moreover, a newly joined client will be required to check all the block headers to validate the integrity of the blockchain, which results in a long bootstrapping time. For example, it takes around an hour to synchronize and validate Ethereum's block headers for a light client, which is quite long compared with Ethereum's relatively short block interval (around 15 seconds).

Recently, with the boom of decentralized applications, there is an increasing demand for users to query blockchain data for analytics. For example, the BigQuery system from Google Cloud allows users to search historical Bitcoin data [1]. However, since the query service is out of the security guarantee of the blockchain system, the result integrity is not guaranteed, i.e., the query service provider

[1]Ethereum has reached 1.56×10^7 blocks by September 2022 according to https://etherscan.io/blocks and the size of a block header is at least 508B [30].

may tamper with query results intentionally or unintentionally. There have been some works that studied verifiable queries over blockchain databases, which allow light clients to query historical blockchain data with integrity assurance [24, 27, 33]. However, the limitation of these existing works is that they require additional authenticated information to be built into the block structure for query result verification, which is incompatible with existing blockchains. Moreover, such a built-in approach only works for pre-defined query types and is not able to support new query types in an on-demand manner. It remains a challenge to design a verifiable query processing solution that supports efficient and versatile queries for light clients without modifying the underlying blockchains.

To address the above issues, in this paper, we design a decentralized certification framework, called DCert, which is compatible with existing blockchain systems and achieves constant-cost blockchain integrity validation for light clients. The main idea is to employ a trusted-hardware (e.g., Intel SGX) assisted full node to recursively certify a block if and only if the following conditions are successfully validated: (i) the integrity of the metadata in the current block header, (ii) the integrity of state transitions from the previous block to the current block, and (iii) the integrity of the previous block's certificate. DCert allows light clients to validate the blockchain integrity by only checking and keeping the latest block header and its certificate. Moreover, some novel designs are proposed for certificate construction to mitigate the performance overhead brought by the trusted hardware.

To support verifiable queries for light clients, a typical method is to build an *authenticated index* over historical blockchain data, which can provide proofs to attest to query integrity. To ensure the integrity of the authenticated index, we propose an augmented certificate that includes enough authentication information of the authenticated index (e.g., the root digest). Similar to the block certificate construction, the trusted hardware certifies the integrity of the authenticated index of the current block. With the augmented certificate and the proof of query results, light clients can verify the results' integrity. Since an authenticated index is often designed for a specific type of query, multiple authenticated indexes are usually needed to support different types of queries. However, as the construction of an augmented certificate binds the block certification and authenticated index certification together, it has a high construction overhead when having multiple authenticated indexes. To tackle this, we propose a hierarchical certificate that separates the block certification and authenticated index certification to reduce the certificate construction overhead while supporting versatile verifiable queries.

To summarize, our contributions made in this paper are as follows:

- We propose DCert, a novel decentralized certification framework that is compatible with existing blockchain systems and achieves constant-cost blockchain integrity validation for light clients.
- We design an SGX-based certificate construction algorithm and develop several optimizations to boost the performance.
- We further propose an augmented certificate and a hierarchical certificate to support efficient and versatile verifiable queries for light clients.
- We conduct extensive experiments to validate the performance of DCert. The results show that it takes constant cost (0.14 ms time and 2.97 KB storage) for light clients to validate the blockchain. Meanwhile, the construction time of all the proposed certificates is within 500 ms, which is negligible compared to the block interval.

The rest of this paper is organized as follows. Section 2 reviews the background. Section 3 gives a system overview of DCert. Section 4 presents the detailed certificate construction and verification algorithms. In Section 5, we discuss how to construct the augmented and hierarchical certificates for verifiable queries. Section 6 presents the security analysis and the experimental results are reported in Section 7. Section 8 discusses the related work. Finally, we conclude our paper in Section 9.

2 BACKGROUND

In this section, we give some background knowledge necessary for introducing our design, including blockchain basics and Intel SGX enclaves.

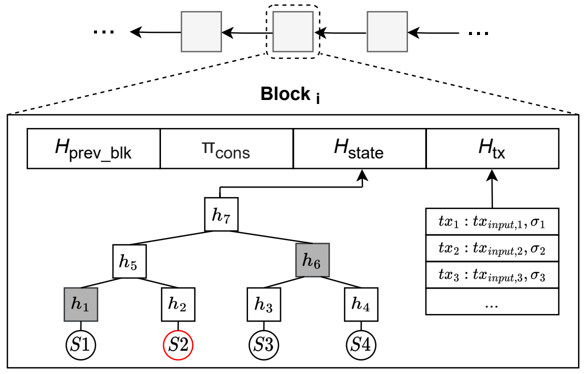


Figure 1: Block Data Structure

2.1 Blockchain Basics

Blockchain consists of a sequence of chained blocks that record transactions with a set of global states. As shown in Fig. 1, the header of each block consists of four fields: (i) H_{prev_blk}, which is the hash of the previous block; (ii) π_{cons}, which is a consensus related data constructed by the miner; (iii) H_{state}, which is the root hash of the Merkle Hash Tree (MHT) constructed upon the global states; (iv) H_{tx}, which is the root hash of the MHT built upon the block transactions. In addition to the block header, each block also stores all the transactions, the global states, and their corresponding MHTs.

MHT is a hierarchical data structure for ensuring data integrity with a collision-resistant hash function. Specifically, an MHT is a hash tree constructed in a bottom-up manner. Figure 1 shows an example of an MHT of four states S_1 to S_4. Each leaf node stores the hash of the indexed object, while each internal node stores a hash computed from its two child nodes (e.g., $h_5 = H(h_1||h_2)$, where "$||$" is the concatenation operation). In the general blockchain context, MHT is often used to efficiently prove the existence of a transaction or a global state, given its root hash included in the block header. For example, in Fig. 1, in order to prove the existence of the state S_2, h_1 and h_6, which are the sibling hash values along its search path, are returned as the proof. One can easily verify the existence of S_2

by reconstructing the root hash using the proof and comparing it with the one in the block header (H_{state} in Fig. 1). Apart from being used in the block structure, MHT has also been extended to various database indexes to construct *authenticated indexes* for different queries. For example, to support authenticated queries in relational databases, MHT can be extended to the multi-way Merkle B-tree (MB-tree), which follows the B+-tree structure and augments each index entry with a corresponding hash [19].

As mentioned in Section 1, miners are responsible for proposing new blocks. Specifically, the transactions are first collected from the blockchain network, and their validity is checked using the senders' public keys. Then, every transaction will be executed by the miner, and the global states, as well as their MHT, will be updated accordingly. Finally, the miner can publish a block once it successfully finds π_{cons} according to the consensus protocol. Full nodes simply observe the blockchain network. When a new block arrives, the full node will check the validity of all metadata in the block header and the validity of all transactions. Then, the transactions will be re-executed for checking the integrity of global states against H_{state}. Once all the checking passes, the full node can append the new block to its ledger. Light clients, different from miners and full nodes, only validate and keep block headers to validate the blockchain integrity due to limited resources.

2.2 SGX

A fundamental component of our design is Intel Software Guard Extensions (SGX) [9], a prominent implementation of trusted hardware. It enhances the security of remote programs and data with the assurance of integrity and privacy. Remote users can place their sensitive information into a protected address area called *enclave*. Data and code inside the enclave are isolated from the outside environment, including privileged processes running at higher levels, e.g., operating systems.

SGX enables a remote user to verify the programs running inside the enclave through a proving process called *remote attestation*. Once the enclave is initialized, the CPU will generate a hash value known as *measurement* to uniquely identify the program inside the container. By calling the trusted enclave, it can produce a *quote* including the measurement and user data, which is cryptographically signed by the hardware-protected key. The Intel SGX attestation service (IAS) is then responsible for verifying this quote and producing an attestation report for further verification. Assuming the trustworthiness of the IAS, this attestation report can prove the integrity of the enclave program execution.

Despite the strong security guarantees, the current SGX-enabled CPU restricts the enclave memory to 128 MB, whereof only 93 MB are available for enclave applications. Once an application overuses the enclave memory, the SGX kernel module will call the in-memory paging between the inside and outside of the enclave [7], which involves costly operations like encryption. Another performance issue is from the enclave transitions, called *Ecall* and *Ocall*. External applications can enter the enclave through the *Ecall* function and exit the enclave through the *Ocall* function. Analyzed in [20, 25, 26, 28, 29], frequent enclave calls can degrade the program performance significantly. Therefore, a well-designed enclave program should minimize its enclave memory usage and the frequency of enclave transitions.

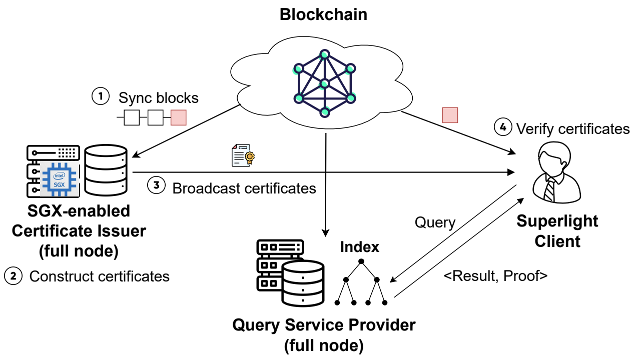


Figure 2: System Architecture

3 DCert OVERVIEW

In this section, we provide an overview of DCert, a decentralized certification framework that achieves constant-cost blockchain integrity validation for light clients. We focus on general-purpose blockchain systems with smart contract capacities, e.g., Ethereum.

3.1 Design Goals

We aim to achieve the following design goals in DCert:

- **Achieving constant-cost validation of blockchain integrity.** As discussed in the previous sections, for light clients, it is essential to keep the cost of blockchain integrity validation to be constant in terms of both time and space complexity.
- **Compatible with existing blockchain systems.** DCert should be compatible with existing blockchains, which implies no changes to underlying systems.
- **Supporting versatile verifiable queries.** It is important to allow light clients to perform various types of verifiable queries over blockchain data with efficient performance.

3.2 System Model

To meet the design goals, DCert introduces new variants of blockchain full nodes. Figure 2 shows an overview of the DCert system model, which consists of the following types of nodes apart from typical blockchain nodes.

- **SGX-enabled Certificate Issuer (CI).** It is a full node equipped with the SGX enclave, responsible for certifying block integrity and authenticated index integrity for blockchain validation and query result verification, respectively.
- **Query Service Provider (SP).** It constructs and maintains authenticated indexes upon blockchain data to support verifiable queries with efficient performance.
- **Superlight Client.** It is a variant of the traditional light client. The major difference is that superlight clients only keep and validate the latest block and its certificate since they are enough for validating the blockchain integrity. Meanwhile, superlight clients are also allowed to perform versatile verifiable queries over blockchain data.

Threat Model and Assumptions. Following a widely adopted blockchain threat model [18], we assume the underlying blockchain is trusted for its integrity and availability. The CIs and SPs in the

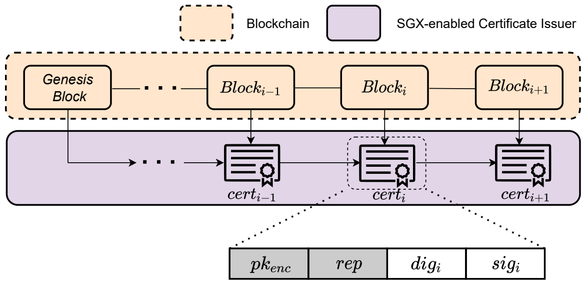


Figure 3: Certificate Construction

DCert framework are considered to be untrusted since the processes of certificate construction and query processing are beyond the security guarantee of the blockchain system. The CIs may construct forged certificates, and the SPs may return tampered or incomplete results, intentionally or unintentionally. On the other hand, we assume that the SGX enclave is trusted for its integrity and privacy, as many works do [3, 8, 10, 21]. We also assume that superlight clients are honest since they are end-users and will not do any malicious behaviors against themselves. It's worth noting that, as an additional service, DCert does not affect the decentralization and security of the underlying blockchain.

3.3 Solution Overview

The core idea of DCert is to leverage the SGX enclave to construct a certificate for each block (see Fig. 3), to prove the integrity of the block and the state transitions from its previous block (except for the genesis block). Such a recursive block certificate design allows a superlight client to use the latest block and its certificate to validate the blockchain integrity with a constant cost. Furthermore, to support verifiable queries on superlight clients, authenticated indexes can be used to provide proofs to attest to the results' integrity. The above block certificate can be extended to certify the authenticated indexes associated with each block.

A straightforward method of certifying a block is that the SGX enclave signs the block information using the hardware-protected key so that a client can run *remote attestation* to validate the signature of the block. However, the remote attestation is quite costly owing to the need of interaction with the Intel IAS. Therefore, we consider using a hierarchical PKI design [37] that decouples the block signing and the remote attestation process. Specifically, during the initialization, the SGX enclave program generates a key pair, (sk_{enc}, pk_{enc}). The secret key sk_{enc} is kept inside the SGX enclave to sign the block information in a secure manner and the public key pk_{enc} will be made public for future validation by superlight clients. After the enclave initialization, an attestation report rep is generated to attest to the enclave program execution and the generated public key.

When constructing a block certificate, the SGX enclave program first checks whether the block is correctly transited from its previous block. If so, it then constructs a block certificate $cert_i$, containing four elements: $\langle pk_{enc}, rep, dig_i, sig_i \rangle$. Here, pk_{enc}, rep are the public key and the attestation report mentioned above. The third element, dig_i, is the digest of the block, containing enough information for validating the block integrity (e.g., block height, consensus proof, and root digests of associated authenticated indexes). The last element sig_i is the signature of dig_i signed by sk_{enc}. When validating the block certificate, a superlight client can use rep to validate pk_{enc}, and further use pk_{enc}, sig_i to validate the digest dig_i concerning with the block integrity. Note that the certificate construction and validation process in DCert do not need to modify the structure or protocol of the underlying blockchain, which is compatible with existing blockchain systems.

Certification Workflow. As shown in Fig. 2, the certification workflow in DCert generally consists of the following steps:

① As a blockchain full node, the CI keeps synchronizing the global states and prepares necessary information for constructing certificates using the SGX enclave.

② Upon receiving the latest block, the CI calls the SGX enclave program to construct the certificate for this block and its associated authenticated indexes.

③ After certificate construction, the CI broadcasts the certificate to the blockchain network.

④ Finally, the superlight client can validate the blockchain integrity and authenticated index integrity in a constant cost using the published certificate, so as to support verifiable queries.

Challenge. The above solution overview shows the certificate construction and the certification workflow in DCert. A simple approach for the certificate construction is to implement all the logic inside the SGX enclave. However, this approach suffers from significant system overhead owing to the limitation of the enclave memory and the high-cost operations (i.e., *Ecall* and *Ocall*) between the inside-enclave and outside-enclave programs, which makes the certificate construction impractical. To tackle this performance issue, we design an efficient SGX-based certificate construction algorithm as well as some optimizations for constant blockchain integrity validation in the next section.

4 CERTIFICATE CONSTRUCTION AND VALIDATION

In this section, we discuss how to construct block certificates for blockchain integrity validation in DCert. We start by presenting our overall idea. Then, we elaborate on our certificate construction algorithms outside and inside the enclave on the CI. Finally, we show how superlight clients validate blocks using certificates.

4.1 Overall Idea

To construct the certificate for a new block blk_i, a naive method is to maintain all state data inside the enclave and update the states based on the transactions contained in blk_i. However, this method is impractical due to the large size of the state data (e.g., over 920 GB for Ethereum as of September 2022[2]) and the limited memory of the enclave (i.e., 93 MB). To tackle this, inspired by the stateless enclave design [14], we propose to apply *verifiable computing* to execute part of the certificate construction program outside the enclave so that the amount of data to be loaded into the enclave can be minimized. More specifically, first, the outside-enclave program computes a read set $\{r\}_i$, a write set $\{w\}_i$, and their Merkle proofs π_i^r, π_i^w based on the previous block blk_{i-1}'s states for updating the new block blk_i's states. It is worth noting that the Merkle proofs

[2]https://etherscan.io/chartsync/chaindefault

Algorithm 1: Certificate Construction Program **(CI)**

```
1  Function gen_cert($blk_{i-1}, cert_{i-1}, blk_i, pk_{enc}, rep$)
       Input: Previous block $blk_{i-1}$ and its certificate $cert_{i-1}$, new
              block $blk_i$, enclave-generated public key $pk_{enc}$,
              attestation report $rep$;
2      $\langle \{r\}_i, \{w\}_i \rangle \leftarrow$ comp_data_set($blk_{i-1}, blk_i$);
3      $\pi_i \leftarrow$ get_update_proof($blk_{i-1}, \{r\}_i, \{w\}_i$);
       /* Enter the enclave                                  */
4      $sig_i \leftarrow$ ecall_sig_gen($blk_{i-1}, cert_{i-1}, blk_i, \pi_i$);
       /* Exit the enclave                                   */
5      $H(hdr_i) \leftarrow$ get_blk_digest($blk_i$);
6      $dig_i \leftarrow H(hdr_i)$;
7      $cert_i \leftarrow \langle pk_{enc}, rep, dig_i, sig_i \rangle$;
8      return $cert_i$;
```

Algorithm 2: Signature Generation Program **(inside the enclave)**

```
1  Function ecall_sig_gen($blk_{i-1}, cert_{i-1}, blk_i, \pi_i$)
       Input: Previous block $blk_{i-1}$ and its certificate $cert_{i-1}$, new
              block $blk_i$, update proof $\pi_i$;
2      $\langle height_{i-1}, hdr_{i-1}, hdr_i \rangle \leftarrow (blk_{i-1}, blk_i)$;
3      if $height_{i-1} = 0$ then
4          assert $H(blk_{i-1}) = H_{genesis}$;
5      else
6          assert cert_verify_t($hdr_{i-1}, cert_{i-1}$);
7      assert blk_verify_t($blk_{i-1}, blk_i, \pi_i$);
8      $sk_{enc} \leftarrow$ load_sk();
9      return Sign($sk_{enc}, H(hdr_i)$);
10 Function blk_verify_t($blk_{i-1}, blk_i, \pi_i$)
11     $\langle H_{i-1}, \pi_i^{cons}, H_i^s, H_i^{tx}, \{tx\}_i, height_i \rangle \leftarrow blk_i$;
12     $\langle \{r\}_i, \pi_i^r, \pi_i^w \rangle \leftarrow \pi_i$;
13     $\langle height_{i-1}, hdr_{i-1}, H_{i-1}^s \rangle \leftarrow blk_{i-1}$;
14     assert $H_{i-1} = H(hdr_{i-1})$ and $height_i = height_{i-1} + 1$;
15     verify_cons($\pi_i^{cons}$);
16     verify_hash($H_i^{tx}, \{tx\}_i$);
17     verify_mht($H_{i-1}^s, \pi_i^r, \{r\}_i$);
18     for each $tx$ in $\{tx\}_i$ do
19         verify($tx$);
20         $\{w\}_{tx} \leftarrow$ execute($tx, H_{i-1}^s, \{r\}_i$);
21         $\{w\}_i \leftarrow \{w\}_i \cup \{w\}_{tx}$;
22     verify_mht($H_{i-1}^s, \pi_i^w, \{w\}_i$);
23     assert $H_i^s =$ update($\pi_i^w, \{w\}_i$);
24     return true;
25 Function cert_verify_t($hdr, cert$)
       Input: Block header $hdr$, certificate $cert$
26     $\langle pk_{enc}, rep, dig, sig \rangle \leftarrow cert$;
27     assert $rep$ is signed by the IAS;
28     assert the current enclave program's measurement equals
         $rep$'s;
29     assert $pk_{enc}$ matches $rep$'s public key signature;
30     verify_sig($sig, dig, pk_{enc}$);
31     assert $dig = H(hdr)$;
32     return true;
```

π_i^r and π_i^w are used to validate the read set $\{r\}_i$ and the set of neighboring nodes related to $\{w\}_i$ with reference to the states in blk_{i-1}, so that $\{w\}_i$ can be correctly committed in blk_i. Then, an update proof π_i consisting of $\{\{r\}_i, \pi_i^r, \pi_i^w\}$, along with blk_i, the previous block blk_{i-1} and its certificate $cert_{i-1}$, will be passed into the enclave. After verifying the integrity of the update proof using blk_{i-1} and $cert_{i-1}$, the inside-enclave program replays the new transactions to validate the digest of blk_i and constructs a new certificate $cert_i$.

4.2 Certificate Construction Algorithm

Algorithm 1 shows the detailed certificate construction algorithm, including the outside-enclave program that pre-processes the block data and the inside-enclave program for generating the certificate signature. The outside-enclave program first executes the transactions in blk_i and computes its read set $\{r\}_i$ and write set $\{w\}_i$ upon the states in blk_{i-1} (Line 2). Then, it generates the update proof π_i of $\{r\}_i$ and $\{w\}_i$ against the Merkle tree of the previous states in blk_{i-1} (Line 3). The update proof π_i will be used by the enclave program to check the integrity of $\{r\}_i$ and update the state tree.

After pre-processing the block data, the CI calls the inside-enclave program ecall_sig_gen via *Ecall* to generate the signature sig_i of the block certificate $cert_i$ (Line 4). The enclave-generated public key pk_{enc} and the attestation report rep (obtained during the enclave initialization as discussed in Section 3.3) are then placed into $cert_i$. Furthermore, the hash of the block header $H(hdr_i)$ is also included as dig_i in $cert_i$ for ensuring the block integrity.

The inside-enclave program ecall_sig_gen is essential in the block certificate construction algorithm. Algorithm 2 describes its procedure. It accepts three types of inputs: (i) the old block information, including the previous block blk_{i-1} and its certificate $cert_{i-1}$, (ii) the new block blk_i, and (iii) the update proof π_i of the read and write sets. If the previous block blk_{i-1} is the genesis block, the program simply checks whether blk_{i-1} matches the hard-coded genesis block digest $H_{genesis}$ (Line 4), because the genesis block is deterministic and does not need a certificate as proof. Otherwise, the program will call the function cert_verify_t to check the validity of the previous block blk_{i-1} using the certificate $cert_{i-1}$ (Line 6), including the following aspects: (i) rep is correctly signed by the IAS; (ii) the current enclave program and the public key pk_{enc} matches the attestation report rep; (iii) the digest in $cert_{i-1}$ is valid against the signature in $cert_{i-1}$; and (iv) the digest of blk_{i-1} matches the digest in $cert_{i-1}$.

After validating blk_{i-1}, the program proceeds to call the function blk_verify_t, which verifies the validity of the current block blk_i (Line 7). Function blk_verify_t first checks the previous hash H_{i-1} and the block height $height_i$ of blk_i against blk_{i-1} (Line 14). Then, it checks the consensus proof π_i^{cons} (e.g., the *nonce* value satisfies the desired difficulty in Proof of Work [23]) (Line 15). It also checks the hash H_i^{tx} of the transactions (Line 16). The program next verifies the integrity of the read set $\{r\}_i$ using its Merkle proof π_i^r and the state root H_{i-1}^s in block blk_{i-1} (Line 17). If $\{r\}_i$ passes the verification, the program executes all the transactions in blk_i based on $\{r\}_i$ and generates the write set $\{w\}_i$ (Lines 18-21). The write proof π_i^w is first checked with respect to $\{w\}_i$ and then is used to compute the updated state root for blk_i (Lines 22-23). If the updated state root matches H_i^s in blk_i, it means the state transitions from blk_{i-1} to blk_i are correct. Finally, after all tests are passed, the enclave signs a signature sig_i for the block digest $H(hdr_i)$ (Lines 8-9).

Example. *Figure 4 shows an example of block certificate construction. Suppose that the state tree of block blk_1 is shown in the top-left*

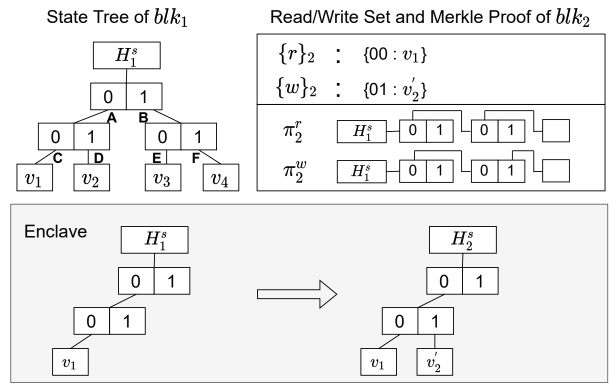


Figure 4: An Example of Chain State Update

part of the figure. Given a new block blk_2, the outside-enclave program generates its read set $\{r\}_2 = \{00 : v_1\}$ and write set $\{w\}_2 = \{01 : v_2'\}$. Correspondingly, the read proof π_2^r includes $\{h_B, h_D\}$. The write proof π_2^w consists of $\{h_B, h_C, h_D\}$. To ensure the integrity of the data input from the outside, the enclave program verifies the integrity of $\{r\}_2$ by reconstructing the hash root $H(H(H(v_1)||h_D)||h_B)$ and comparing it with H_1^s in blk_1. Similarly, the write proof integrity is verified by reconstructing the hash root $H(H(h_C||h_D)||h_B)$ and comparing it with H_1^s in blk_1. After executing all the transactions of blk_2, the write set $\{w\}_2$ is obtained. Next, the enclave program computes the updated state root of blk_2 by $H\left(H\left(h_C||H(v_2')\right)||h_B\right)$ and checks whether the new state root matches H_2^s in blk_2. If so, it means the state transitions from blk_1 to blk_2 are correct. Finally, the signature sig_2 can be computed.

4.3 Blockchain Integrity Validation

Next, we describe how a superlight client validates the blockchain integrity with the certificate issued by the CI. As shown in Algorithm 3, the superlight client starts by validating the block information with the certificate (Lines 2–7), using a procedure similar to blk_verify_t in Algorithm 2. Note that a consensus protocol needs to check (i) the consensus proof and (ii) the chain selection in case of blockchain forks. While the consensus proof has been validated by the block certificate, we still need to check that the block conforms with the system's chain selection rule (Line 8). For example, Bitcoin adopts the longest chain rule; thus, the superlight client always selects the validated block with the largest block height as the latest block, which corresponds to the longest chain.

It is worth noting that the superlight client needs to check an attestation report only once for the same enclave. Only if the superlight client switches to the certification service of another CI, a new attestation report needs to be examined.

5 EXTENSION TO VERIFIABLE QUERIES

In this section, we discuss how to extend our block certificate to support verifiable queries for superlight clients in DCert. We first briefly introduce the overview of our design. Then, we elaborate on the detailed certificate schemes, including the augmented certificate and the hierarchical certificate. Finally, we give a case study to show how our certification framework supports verifiable queries.

Algorithm 3: Blockchain Integrity Validation **(Superlight Client)**

```
1 Function validate_chain($hdr_i, cert_i$)
      Input: New block header $hdr_i$, certificate $cert_i$
2     $\langle pk_{enc}, rep, dig_i, sig_i \rangle \leftarrow cert_i$;
3     assert $rep$ is signed by the IAS;
4     assert $rep$'s measurement matches the certificate-construction
        enclave program;
5     assert $pk_{enc}$ matches $rep$'s public key signature;
6     verify_sig($sig_i, dig_i, pk_{enc}$);
7     assert $dig_i = H(hdr_i)$;
8     assert $hdr_i$ follows the system's chain selection rule;
9     return true;
```

5.1 Design Overview

To support verifiable queries for superlight clients, the SP needs to construct and maintain an authenticated index over the blockchain data and then provide both query results and integrity proofs for result verification. It is worth noting that DCert can support any queries where authenticated query processing algorithms are available (e.g., simple blockchain queries such as range/keyword queries [24] and complex queries such as aggregations [32]). However, since the authenticated index is built off the chain, the blockchain system cannot guarantee its integrity. To solve this problem, we propose an augmented certificate to extend our block certificate, which can efficiently prove the integrity of the authenticated index off the chain without any on-chain modifications. More specifically, the CI utilizes the enclave to validate the index updates with the block state transitions from the previous block. Only if passing the validation, the CI can construct a new augmented certificate for authenticated index integrity. On the SP side, it processes queries and returns query results as well as the corresponding integrity proofs to superlight clients. Then, superlight clients can verify the correctness of query results with the help of the augmented certificate issued by the CI and the integrity proofs generated by the SP.

The augmented certificate scheme binds the block certification logic with the authenticated index certification logic by invoking the *Ecall* function once. By following the design rationale in Section 2.2, this scheme achieves good performance for a single authenticated index. Nevertheless, with the increasing number of query types, the CI needs to maintain more authenticated indexes with the index certification, which incurs high construction overhead. To alleviate this problem, we further propose a hierarchical certificate scheme that separates the block certification and the authenticated index certification to reduce the certificate construction overhead.

5.2 Certificate Construction Schemes

To prove that the authenticated index is correctly updated by new blocks, we propose new certificate schemes to certify the digest of the authenticated index with respect to the underlying block states. This implies that the certificate needs to verify the following three perspectives: (i) the underlying block states are correct; (ii) the write data for the index update generated by the new block is complete and correct; and (iii) the new authenticated index digest is correctly updated by the write data based on its previous digest. In what follows, we present two schemes of constructing the certificate

```
Algorithm 4: Augmented Certificate Construction (CI)
Input: Previous block $blk_{i-1}$, previous augmented certificate
       $cert_{i-1}^{idx}$, previous index digest $H_{i-1}^{idx}$, new block $blk_i$, new
       index digest $H_i^{idx}$, auxiliary data $aux$;
/* Enter the enclave */
1  $\langle \pi_i, \pi_i^{idx} \rangle \leftarrow aux$;
2  $\langle height_{i-1}, hdr_{i-1}, hdr_i \rangle \leftarrow (blk_{i-1}, blk_i)$;
3  if $height_{i-1} \neq 0$ then
4      assert cert_verify_t$(hdr_{i-1}||H_{i-1}^{idx}, cert_{i-1}^{idx})$;
5  else
6      assert $H_{i-1}^{idx} = H_{genesis}^{idx}$;
7  assert blk_verify_t$(blk_{i-1}, blk_i, \pi_i)$;
8  $\{w\}_i^{idx} \leftarrow$ get_index_write_data$(blk_i)$;
9  verify_mht$(H_{i-1}^{idx}, \pi_i^{idx}, \{w\}_i^{idx})$;
10 assert $H_i^{idx}$ = update$(\pi_i^{idx}, \{w\}_i^{idx})$;
11 $sk_{enc} \leftarrow$ load_sk();
12 $sig_i \leftarrow$ Sign$(sk_{enc}, H(hdr_i||H_i^{idx}))$;
/* Exit the enclave */
13 $dig_i \leftarrow H(hdr_{i-1}||H_{i-1}^{idx})$;
14 $cert_i^{idx} \leftarrow \langle pk_{enc}, rep, dig_i, sig_i \rangle$;
15 return $cert_i^{idx}$;
```

using the enclave.

5.2.1 Augmented Certificate Construction. The augmented scheme is to directly append the authenticated index update procedure to the block certificate construction algorithm and augment its certificate information with the index digest H_i^{idx}. For efficiency reasons, the augmented certificate construction algorithm also follows the Merkle-proof-based approach to update the authenticated index inside the enclave. That means the enclave only handles the low-cost proof verification operations inside the enclave and receives the auxiliary data from the outside. Algorithm 4 describes the detailed process. Besides the inputs for the block certificate construction, it accepts three more inputs: (i) the previous index digest H_{i-1}^{idx} with its augmented certificate $cert_{i-1}^{idx}$, (ii) the current index digest H_i^{idx}, and (iii) the auxiliary data aux, which includes the write proof π_i^{idx} for updating the authenticated index. Line 7 applies the validation logic of the block certificate construction. As for the untrusted write proof π_i^{idx}, it first checks the integrity of this Merkle proof against the previous index root H_{i-1}^{idx} and the index write data $\{w\}_i^{idx}$ retrieved from blk_i (Line 9). Afterwards, it updates a new root digest of the authenticated index using $\{w\}_i^{idx}$ and π_i^{idx}. If the re-computed root is not identical to H_i^{idx}, the program will abort immediately (Line 10). Otherwise, after all tests are passed, the enclave signs the certificate information $(hdr_i||H_i^{idx})$ by using sk_{enc}.

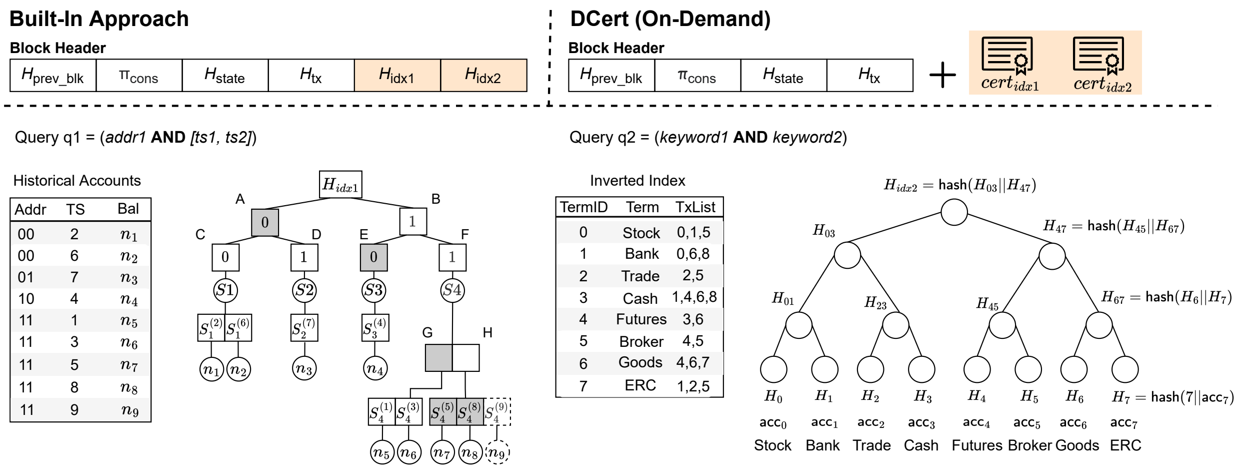
5.2.2 Hierarchical Certificate Construction. Despite the concise design of the augmented certificate construction scheme, it has a performance issue when dealing with multiple authenticated indexes for different query types. Observing Algorithm 4, we can find that each time a new augmented certificate is constructed for a new authenticated index, the enclave replays the block validation function blk_verify_t, which incurs expensive repetitive computations. To tackle this problem, we propose a hierarchical certificate scheme

```
Algorithm 5: Hierarchical Certificate Construction (CI)
Input: Previous block $blk_{i-1}$, new block $blk_i$, sets of index
       integrity information $\{H_{i-1}^{idx}, cert_{i-1}^{idx}, H_i^{idx}\}_{idx}$, auxiliary
       data $aux$;
/* Get the block certificate */
1  $cert_i \leftarrow$gen_cert$(blk_{i-1}, cert_{i-1}, blk_i, pk_{enc}, rep)$;
2  for each index set in $\{H_{i-1}^{idx}, cert_{i-1}^{idx}, H_i^{idx}\}$ do
       /* Ecall into the enclave again */
3      $\langle \{r\}_i, \pi_i^r, \pi_i^{idx} \rangle \leftarrow aux$;
4      $\langle height_{i-1}, hdr_{i-1}, hdr_i \rangle \leftarrow (blk_{i-1}, blk_i)$;
5      if $height_{i-1} \neq 0$ then
6          assert cert_verify_t$(hdr_{i-1}||H_{i-1}^{idx}, cert_{i-1}^{idx})$ ;
7      else
8          assert $H(blk_{i-1}) = H_{genesis}$;
9          assert $H_{i-1}^{idx} = H_{genesis}^{idx}$;
10     assert cert_verify_t$(hdr_i, cert_i)$ ;
11     $\{w\}_i^{idx} \leftarrow$ get_index_write_data$(blk_i)$;
12     verify_mht$(H_{i-1}^{idx}, \pi_i^{idx}, \{w\}_i^{idx})$;
13     assert $H_i^{idx}$ = update$(\pi_i^{idx}, \{w\}_i^{idx})$;
14     $sk_{enc} \leftarrow$ load_sk();
15     $sig_i \leftarrow$ Sign$(sk_{enc}, H(hdr_i||H_i^{idx}))$;
       /* Exit the enclave */
16     $dig_i \leftarrow H(hdr_{i-1}||H_{i-1}^{idx})$;
17     $cert_i^{idx} \leftarrow \langle pk_{enc}, rep, dig_i, sig_i \rangle$;
18     $\{cert\}_{idx}$.insert$(cert_i^{idx})$;
19 return $\{cert\}_{idx}$;
```

(Algorithm 5) to separate the block certification and the authenticated index certification. When a new block blk_i is received, the CI first calls the function gen_cert to get the block certificate $cert_i$ (Line 1). Based on this block certificate $cert_i$ and multiple sets of index integrity information, the certificates of the authenticated indexes will be constructed one by one (Line 2 to Line 18). During the construction, it first checks the correctness of the previous states of the index H_{i-1}^{idx} through the previous certificate $cert_{i-1}^{idx}$. Then, it updates the digest of the authenticated index as the augmented certificate scheme does. The difference is that, due to the prior construction of $cert_i$, the hierarchical certificate program can directly verify blk_i using its $cert_i$, instead of re-executing it again (Line 10). Finally, the hierarchical certificate with the certificate information $(hdr_i||H_i^{idx})$ is signed and inserted into the hierarchical certificate set $\{cert\}_{idx}$.

5.3 Query Processing and Verification

In DCert, superlight clients keep track of the certificates issued by the CI. They can send query requests to the SP. The SP will process queries upon the authenticated index, and return the query results and the corresponding integrity proofs. To verify the query results, superlight clients first asserts that the authenticated index used by the SP is correct by checking its Merkle root against $(hdr_i|H_i^{idx})$ in the certificate. Afterwards, superlight clients can check the query results against the index's digest using the corresponding integrity proofs.


Built-In Approach
Block Header
H_{prev_blk} π_{cons} H_{state} H_{tx} H_{idx1} H_{idx2}
DCert (On-Demand)
Block Header
H_{prev_blk} π_{cons} H_{state} H_{tx} +
$cert_{idx1}$ $cert_{idx2}$
Query q1 = (*addr1* **AND** *[ts1, ts2]*)
Historical Accounts
H_{idx1}
A B C D E F G H
0 1 0 1 0 1
S1 S2 S3 S4
$S_1^{(2)}$ $S_1^{(6)}$ $S_2^{(7)}$ $S_3^{(4)}$ $S_4^{(1)}$ $S_4^{(3)}$ $S_4^{(5)}$ $S_4^{(8)}$ $S_4^{(9)}$
n_1 n_2 n_3 n_4 n_5 n_6 n_7 n_8 n_9
Query q2 = (*keyword1* **AND** *keyword2*)
Inverted Index
$H_{idx2} = \text{hash}(H_{03}||H_{47})$
H_{03}
$H_{47} = \text{hash}(H_{45}||H_{67})$
H_{01} H_{23} H_{45}
$H_{67} = \text{hash}(H_6||H_7)$
H_0 H_1 H_2 H_3 H_4 H_5 H_6 $H_7 = \text{hash}(7||\text{acc}_7)$
acc_0 acc_1 acc_2 acc_3 acc_4 acc_5 acc_6 acc_7
Stock Bank Trade Cash Futures Broker Goods ERC

Addr	TS	Bal
00	2	n_1
00	6	n_2
01	7	n_3
10	4	n_4
11	1	n_5
11	3	n_6
11	5	n_7
11	8	n_8
11	9	n_9

TermID	Term	TxList
0	Stock	0,1,5
1	Bank	0,6,8
2	Trade	2,5
3	Cash	1,4,6,8
4	Futures	3,6
5	Broker	4,5
6	Goods	4,6,7
7	ERC	1,2,5

Figure 5: Case Study of Verifiable Queries

5.4 Case Study

We now show how our certificate design supports verifiable queries without modifying the underlying blockchain structure. Figure 5 presents a detailed case study with two types of queries: historical query over blockchain accounts [24] and conjunctive keyword query over blockchain transactions [12]. Here, we first discuss why the existing approaches [24, 27, 33] fail to support efficient verifiable queries. Then, we will elaborate on how our DCert supports these queries.

Limitation of Existing Approaches. The top-left part of Fig. 5 shows the built-in approach adopted by the existing works. To answer historical queries, LineageChain [24] designs an authenticated deterministic skip list and integrates this index into each block of the chain. Similarly, vChain [33] and vChain+ [27] also add the root digests of specifically designed authenticated indexes to the block headers. Despite the query integrity assurance, such built-in approaches forcibly modify the block structure of the underlying blockchain, which is incompatible with existing blockchain systems. It is also difficult to support new query types in an on-demand manner.

Our Approach. Different from the previous works, our DCert does not integrate the authenticated indexes with the underlying blockchain structure. Instead, DCert allows the CI to issue certificates for superlight clients to track the updates of the authenticated indexes. With this, the SP can provide flexible integrity-assured query services using the authenticated indexes.

As shown in the lower-left part of Fig. 5, a two-level tree index can be employed for searching historical versions of every account. In this two-level tree, the upper level is a Merkle Patricia Trie [30] that stores the hashes of the account addresses, and the lower level is a Merkle B-tree [19] that records the time-stamped states of each account. The leaf node of the lower tree is the state value with a specific timestamp, denoted as $S_k^{(v)}$, where v is the timestamp and k is the account key. For example, $S1$ has the account key 00 and contains the states with two timestamps: $S_1^{(2)}$ and $S_1^{(6)}$.

Instead of augmenting the block structure with the index's digest H_{idx1}, DCert leverages the decentralized certificate to attest to the authenticated index. Suppose that the state value n_9 (dashed line) is inserted with the account key 11 and timestamp 9. During the certificate construction, the CI prepares a write proof π_{idx1} for the enclave's. It consists of $\{h_A, h_E, h_G, h(S_4^{(5)}), h(S_4^{(8)})\}$. For verification purposes, the enclave can reconstruct the root hash of the index $H\left(h_A||H\left(h_E||H\left(h_G||H\left(h(S_4^{(5)})||h(S_4^{(8)})\right)\right)\right)\right)$ using π_{idx1} and compare it with the previous root hash H_{idx1}. If they match, π_{idx1} is valid. Finally, the digest is further updated to H'_{idx1} by following $H\left(h_A||H\left(h_E||H\left(h_G||H\left(h(S_4^{(5)})||h(S_4^{(8)})||H\left(n_9\right)\right)\right)\right)\right)$, and the enclave signs H'_{idx1} with the corresponding block information hdr_i to construct a new certificate $cert_{idx1}$. By signing the index digest H'_{idx1} with its underlying block, the certificate $cert_{idx1}$ guarantees the authenticated index integrity.

With the certificate $cert_{idx1}$, a superlight client can send query requests to the SP. Consider a historical query with account key 11 over the time window [2, 3]. The SP computes and sends back the query result $S_4^{(3)}$ and its corresponding integrity proof $\{h_A, h_E, h_H, h(S_4^{(1)})\}$. In the verification phase, the superlight client can first use the certificate $cert_{idx1}$ to validate the index root hash and then verify the correctness of the query result by reconstructing the index root hash $H\left(h_A||H\left(h_E||H\left(H\left(h(S_4^{(1)})||h(S_4^{(3)})\right)||h_H\right)\right)\right)$ using the integrity proof.

Next, consider keyword queries over blockchain transactions. For example, a query $q = [Stock \text{ AND } Bank]$ wants to find all the transactions that contain the keywords "Stock" and "Bank". To efficiently support such keyword queries, an inverted index [12] can be maintained by the SP and the CI. When needed, a new certificate $cert_{idx2}$ can be constructed by the enclave of the CI by signing the digest H_{idx2} of this authenticated index (shown in the right-lower part of Fig. 5). Based on this certificate, superlight clients can perform verifiable queries by following the query scheme proposed in [12].

6 SECURITY ANALYSIS AND DISCUSSION

This section performs a security analysis on our proposed block certificate and verifiable query schemes. We start by presenting a formal security definition of the block certificate.

Definition 1 (Block Certificate Security). We say a decentralized certificate scheme is secure if, given a blockchain network $\mathcal{B}$, it is impossible for any polynomial-time adversary to construct a valid block certificate $cert$ for the block blk with the following conditions: (i) blk is an invalid block; or (ii) blk does not satisfy the chain selection rule according to network consensus.

The above definition ensures that the chance for a malicious CI to persuade a verifier with an invalid latest block is negligible. We now show that our proposed block certificate satisfies the desired security requirement.

Theorem 1. *Our proposed block certificate scheme is secure with respect to Definition 1 if the underlying cryptographic primitives and the trusted hardware are secure.*

Proof. We prove this theorem by contradiction.

- Case 1: The block blk is an invalid block. In this case, if an adversary can forge a block certificate $cert_i$ for an invalid block blk_i, then there must exist a polynomial-time adversary $\mathcal{A}$ who either (1) can persuade the enclave program to accept an invalid previous block, (2) can persuade the enclave program to accept invalid blockchain states, or (3) can penetrate the security isolation of the SGX enclave to interfere its program isolation. The first case is impossible because the enclave program will validate the certificate $cert_{i-1}$ for block blk_{i-1}. Thanking to the recursive nature, the enclave program can establish the validity of each block tracing all the way back to the genesis block. The second case is also impossible as the CI will construct Merkle proofs to attest to the block state inputs. A successful forgery of the Merkle proofs would mean a collision in the underlying cryptographic hash function, which contradicts our assumption. Finally, the last case directly contradicts our assumption that the trusted hardware is secure.
- Case 2: The block blk does not satisfy the chain selection rule. Since the client will check whether the block metadata such as the block height satisfies the system's chain selection rule, this case is impossible.

□

We next give a formal security definition of the verifiable query scheme and analyze its security guarantee.

Definition 2 (Verifiable Query Security). We say a verifiable query is secure if, given a blockchain $\mathcal{B}$, it is impossible for any polynomial-time adversary to construct a valid integrity proof π along with a valid certificate $cert$ for a tampered or incomplete query result R of a query Q.

The above definition guarantees that the chance for an adversary to convince the user of a tampered or incomplete result is negligible. We now show that our proposed algorithms satisfy the desired security requirements.

Theorem 2. *Our proposed verifiable query schemes are secure with respect to Definition 2 if (i) the underlying cryptographic primitives and the trusted hardware used to construct the index certificate cert are secure; and (ii) the underlying verifiable query algorithm used to construct the integrity proof π is secure.*

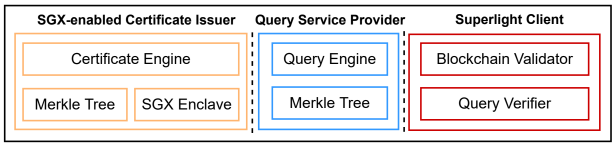


Figure 6: System Modules of DCert

Proof. The integrity of the query result depends on the integrity of the query index and the underlying verifiable query algorithm. Here, our index certificate $cert$ attests to the integrity of the query index. Since its construction is similar to that of block certificate, the security proof follows that of Theorem 1. Given a validated query index, a tampered or incomplete result means that the adversary is able to forge the integrity proof π, which contradicts our assumption that the underlying verifiable query algorithm is secure.

□

The above analysis proves the security of our proposed block certificate and verifiable query schemes. It is worth noting that DCert currently implements the signature generation program (Algorithm 2) using Intel SGX, which relies on a centralized TEE authority, i.e, Intel. Although the decentralization of the underlying blockchain is independent to that of DCert, one may wish to avoid relying solely on Intel. To this end, we note that the DCert can be deployed using any other TEE implementations such as ARM TrustZone, RISC-V MultiZone, and AMD Platform Security Processor.

7 IMPLEMENTATION AND EVALUATION

We implement a prototype of DCert in Rust programming language, with around 8,000 lines of codes. The enclave implementation uses Apache Teaclave SGX SDK[3]. We instantiate an SGX-enabled certificate issuer on a machine equipped with SGX-enabled CPU Intel i7-7567U @ 3.50GHz, 32GB RAM, running Ubuntu 20.04LTS. For each of the SP and the superlight client, a desktop computer with Intel Core i7-10710U 1.10GHz CPU and 16 GB RAM is used.

7.1 Implementation

As shown in Fig. 6, we implement the essential modules of DCert based on an Ethereum prototype, including (i) *certificate engine*, which is built based on the Rust Ethereum Virtual Machine[4] and is responsible for constructing block and index certificates; (ii) *Merkle tree*, which offers the functionalities of manipulating authenticated indexes; (iii) *SGX enclave*, which is a secure enclave with the pre-defined interface; (iv) *query engine*, which processes verifiable queries over blockchain data and generates proofs to attest to the integrity of query results; (v) *blockchain validator*, which enables superlight clients to validate the blockchain integrity; and (vi) *query verifier*, which verifies the integrity of query results with the block and index certificates.

7.2 Experiment Setup

We use Blockbench [11] to evaluate the performance. It offers both micro-benchmarks consisting of DoNothing (denoted as DN),

[3]https://github.com/apache/incubator-teaclave-sgx-sdk
[4]https://github.com/rust-blockchain/evm

Parameters	Value
# of generated blocks	10^3, 10^4, $\mathbf{10^5}$, 10^6
# of transactions per block	25, **50**, 75, 100
Workload	DN, KV, CPU, **SB**, IO

Table 1: System Parameters

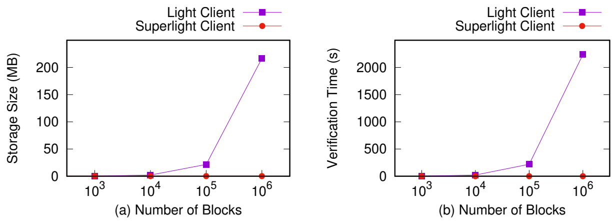


Figure 7: Bootstrapping Cost vs. # Blocks

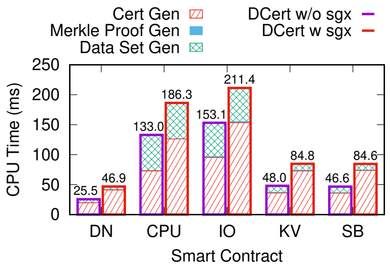


Figure 8: Performance vs. Smart Contract

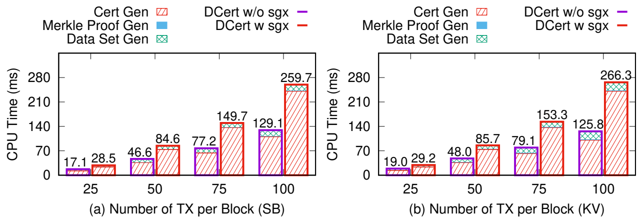


Figure 9: Performance vs. Block Size

CPUHeavy (denoted as CPU), and IOHeavy (denoted as IO), and macro benchmarks consisting of KVStore (denoted as KV) and SmallBank (denoted as SB). For the experiments of certificate construction, we initially deploy 500 smart contracts and then continuously invoke these smart contracts until there are more than 100k blocks in the ledger. We randomly generate 100k sender accounts for sending transaction requests in each experiment. For the verifiable query experiments, we create 500 key-value tuples and then continuously issue update transactions until there are 10k blocks in the ledger. Then, we perform historical account queries with different time windows. Table 1 lists all the system parameters used in the experiments, where the default settings are highlighted in boldface.

7.3 Evaluation Metrics

The following metrics are used to evaluate our proposed framework: (i) bootstrapping costs, including the storage size and chain validation time of superlight clients; (ii) time of block certificate construction of the CI, including the pre-processing program outside and certificate construction program inside the enclave; (iii) time of augmented and hierarchical certificate construction; and (iv) verifiable query performance, including the query processing time and the proof size.

7.4 Evaluation Results

7.4.1 Bootstrapping Cost. We start with evaluating the performance of blockchain bootstrapping for superlight clients. From Fig. 7a, we can see that the storage requirement of the traditional light client increases significantly with the growth of the chain length. In contrast, the superlight client in DCert maintains a constant storage size of 2.97 KB (including the latest block and its certificate). Figure 7b further shows the comparison from the perspective of chain validation time. Notably, the superlight client only takes a constant bootstrapping time of 0.14 ms, while the traditional light client suffers from the continually increasing validation time.

7.4.2 Certificate Construction Cost. Figure 8 reports the performance of certificate construction and its performance overhead incurred by the enclave. We break down the certificate construction time into two components: an untrusted pre-processing program outside the enclave (read/write set and Merkle proof generation) and trusted certificate generation program inside the enclave.

It can be seen that the inside-enclave operations account for the majority of the cost. We observe that the enclave only brings about at most 1.8x performance degradation, which would not dramatically affect the construction of block certificates. As for the pre-processing operations outside the enclave, we can find that the time of Merkle proof generation is too small to be visible in the figure. The cost of the read/write set generation mainly depends on the complexity of smart contracts. Since the enclave overhead is mainly induced by the state tree updates, the complex smart contracts (CPU, IO) incur a longer transaction execution time and weaken the impact of the enclave overhead.

7.4.3 Impact of Block Size. We vary the block size (i.e., the number of transactions) to evaluate its impact on the performance of the block certificate construction. We test the two macro benchmarks, namely KVStore (KV) and SmallBank (SB). Figure 9 shows their certificate construction time and related breakdown costs. With the input of more transactions, the certificate engine takes more time to execute transactions and prepare their Merkle proofs for the enclave. We can find that the performance overhead incurred by the enclave is also increasing. This is mainly because the size of the read/write set and its Merkle proofs passed into the enclave increases with the number of transactions, which degrades the performance of the enclave. Even so, the overall performance is within a reasonable range and would not much affect the DCert certification service.

7.4.4 Augmented and Hierarchical Certificate Construction. Figure 10 shows that the overall performance of augmented and hierarchical certificate construction when varying the number of authenticated indexes. With the increase of authenticated indexes, there is a substantial increase of the augmented certificate construction time. In contrast, the hierarchical scheme only yields a slight increase. This is mainly because the augmented scheme re-executes the chain certification program each time adding a new authenticated index, while the hierarchical scheme can efficiently certifying the corresponding block using the prepared block certificate. When there is only one authenticated index, the augmented scheme per-

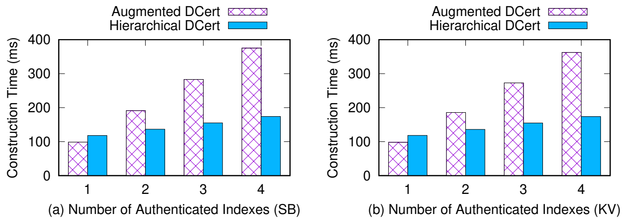


Figure 10: Performance vs. # Authenticated Indexes

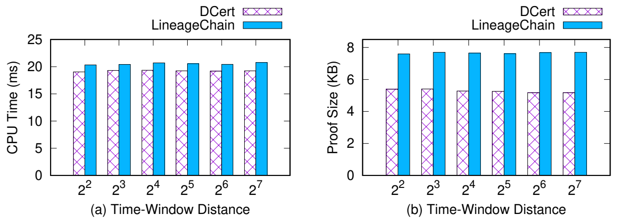


Figure 11: Query Performance vs. Time-Window Distance

forms slightly better than the hierarchical one. This is because the hierarchical scheme separates the block certification logic and the index certification logic, which incurs one more *Ecall* function call.

7.4.5 Verifiable Query Performance. Finally, we evaluate the performance of the historical queries over blockchain accounts. We implement two indexes for such historical queries: one is the two-level tree with the lower tree of Merkle skip list proposed in LineageChain [24], and the other is the two-level tree of DCert presented in Fig. 5. Figure 11 shows the query latency with the increasing time-window distance from the latest block. As can be seen, our DCert achieves a better performance of query processing and a smaller size of integrity proofs than LineageChain in all cases. This is because with the versatile feature of DCert, we are able to use a more efficient two-level index structure (i.e., Merkle Patricia Trie and Merkle B-tree) to process and verify the queries.

8 RELATED WORK

In this section, we briefly review several related studies on blockchain light client validation, verifiable blockchain queries, and blockchain system based on trusted hardware.

8.1 Blockchain Light Client Validation

Due to the linear complexity of block headers, the current blockchain light clients suffer from comparably heavy storage and communication overheads. To tackle this problem, several studies proposed sublinear light clients based on the consensus protocol [6, 16, 17]. Kiayias et al. [16] proposed the proofs of proof of work (PoPow) and reduced the linear complexity to a logarithmic one for light clients. Later, an improved version, non-interactive PoPoW (NIPoPoW) [17], is proposed to mitigate the potential attacks caused by PoPow. Afterwards, to further improve the applicability of NIPoPoW, Flyclient [6] is proposed for chains of varied difficulty by using an optimal probabilistic block sampling protocol and Merkle Mountain Range (MMR) commitments. However, the complexity for these light clients is still comparably high. In comparison, our proposed DCert provides constant complexity without changing the consensus protocol of the underlying blockchain.

Some existing studies are proposed to reduce the bootstrapping cost of light clients to a constant level by utilizing recursive block validation [5, 15]. Specifically, they utilized the cryptography-based verifiable computation scheme, i.e., SNARKs [2, 4] for recursively verifying each block and its state transitions from the previous block. However, their design focuses on building new blockchain systems by integrating a specific consensus protocol with a succinct proof. Furthermore, they are all particularly designed for cryptocurrencies and failed to support smart contracts which involve arbitrary execution logic and state size. In contrast, our proposed DCert considers providing a decentralized certification framework, which is compatible with any existing blockchain systems, including the ones with smart contract functionality. Meanwhile, DCert also enables light clients to support on-demand rich queries with integrity assurance.

8.2 Verifiable Blockchain Queries

Many works studied verifiable query processing for blockchain systems. Hu et al. [13] proposed a searchable encryption scheme on a blockchain system to achieve query verifiability. Xu et al. [33] proposed the vChain framework that supports verifiable boolean range queries over blockchain databases by designing an accumulator-based authenticated data structure. Zhang et al. [35, 36] designed gas-efficient authenticated data structures to support authenticated range and keyword search queries over hybrid-storage blockchains. Wu et al. [31] proposed the VQL system that employs a middleware-based cloud to provide verifiable query services with traceable database fingerprints. Ruan et al. [24] designed a secure and efficient data provenance system, LineageChain, by augmenting the smart contract interface with provenance information and proposing a novel authenticated skip list index for efficient provenance query processing. Different from all these prior works, our proposed DCert does not require modifying the underlying blockchain [24, 33] and is not confined to specific query types [13, 35, 36]. It can dynamically support new query types by constructing corresponding certificates of the indexes in an on-demand manner.

8.3 Blockchain Systems based on Trusted Hardware

The idea of utilizing the trusted hardware for blockchain systems has been adopted by many works [8, 10, 22, 34, 38]. There are mainly two categories of trusted-hardware-backed blockchain models. One category mainly focuses on protecting user privacy across the blockchain network. Ekiden [8] provides a confidential smart contract solution by separating consensus nodes and compute nodes with the trusted hardware. Bite [22] integrates several private information retrieval and side-channel protection techniques with the trusted hardware to protect the clients' addresses and transactions. The other category mainly relies upon the computation integrity feature provided by the trusted hardware. REM [38] proposed a new consensus called proof of useful work, which relies on the faithful random number generator in Intel SGX to replace the hash puzzle. Dang et al. [10] involved the SGX-backed trusted randomness in the running process of the consensus protocol and built a sharded blockchain system with high transaction throughputs. In our DCert, the trusted hardware is used as a trusted computation tool to support the blockchain integrity validation. Meanwhile, our certification framework also relies on the trusted hardware to protect the confidentiality of the initialized secret key used for signing block certificates.

9 CONCLUSION

In this paper, we have proposed DCert, a novel decentralized certification framework that is compatible with existing blockchains and achieves constant-cost chain integrity validation with full security guarantee for light clients. We have developed an efficient certificate construction algorithm to mitigate the performance bottleneck of the trusted hardware. To enable verifiable queries, we have further proposed augmented and hierarchical certificate schemes for proving the integrity of the authenticated indexes, based on which various types of verifiable queries can be supported on superlight clients in an on-demand manner. Extensive experiments show that the proposed DCert can significantly reduce the storage and bootstrapping overhead of light clients and provide efficient verifiable queries over blockchain data.

ACKNOWLEDGEMENT

This work was supported by Hong Kong RGC Grants (No. 12200819, 12201520, and C2004-21GF), CCF-AFSG Research Fund (No. RF20210 014), and Guangdong Science and Technology Special Fund (No. SDZX2020036).

REFERENCES

[1] Colin Bookman Allen Day. 2018. Bitcoin in BigQuery: blockchain analytics on public data. Retrieved October 7, 2022 from https://cloud.google.com/blog/topics/public-datasets/bitcoin-in-bigquery-blockchain-analytics-on-public-data

[2] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference* (Berlin, Heidelberg, August 18-22, 2013). Springer, 90–108.

[3] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *ACM CCS* (London, UK, November 11-15, 2019). 1521–1538.

[4] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2013. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *ACM STOC* (Palo Alto, California, USA, June 1-4, 2013). 111–120.

[5] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. 2020. Coda: Decentralized Cryptocurrency at Scale. Cryptology ePrint Archive, Paper 2020/352.

[6] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. 2020. Flyclient: Super-light clients for cryptocurrencies. In *IEEE S&P* (San Francisco, CA, USA, May 18-21, 2020). 928–946.

[7] Somnath Chakrabarti, Rebekah Leslie-Hurd, Mona Vij, Frank McKeen, Carlos Rozas, Dror Caspi, Ilya Alexandrovich, and Ittai Anati. 2017. Intel® software guard extensions (Intel® SGX) architecture for oversubscription of secure memory in a virtualized environment. In *Proceedings of the Hardware and Architectural Support for Security and Privacy* (Toronto, ON, Canada, 25 June 2017). 1–8.

[8] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *IEEE EuroS&P* (Stockholm, Sweden, June 17-19, 2019). 185–200.

[9] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.

[10] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards scaling blockchain systems via sharding. In *ACM SIGMOD* (Amsterdam, Netherlands, July 5, 2019). 123–140.

[11] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A framework for analyzing private blockchains. In *ACM SIGMOD* (Chicago, Illinois, USA, May 14 - 19, 2017). 1085–1100.

[12] Michael T Goodrich, Charalampos Papamanthou, Duy Nguyen, Roberto Tamassia, Cristina Videira Lopes, Olga Ohrimenko, and Nikos Triandopoulos. 2012. Efficient verification of web-content searching through authenticated web crawlers. *PVLDB* (August 2012), 920–931.

[13] Shengshan Hu, Chengjun Cai, Qian Wang, Cong Wang, Xiangyang Luo, and Kui Ren. 2018. Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization. In *IEEE INFOCOM* (Honolulu, HI, USA, April 16-19, 2018). 792–800.

[14] Gabriel Kaptchuk, Matthew Green, and Ian Miers. 2019. Giving State to the Stateless: Augmenting Trustworthy Computation with Ledgers. In *NDSS* (San Diego, CA, USA, 24-27 February 2019). 1–15.

[15] Assimakis Kattis and Joseph Bonneau. 2020. Proof of Necessary Work: Succinct State Verification with Fairness Guarantees. In *3rd Stanford Blockchain Conference (SBC)* (Stanford, CA, USA, February 19-21, 2020).

[16] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. 2016. Proofs of proofs of work with sublinear complexity. In *FC* (Berlin, Heidelberg, February 26, 2016). Springer, 61–78.

[17] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. 2020. Non-interactive proofs of proof-of-work. In *Financial Cryptography and Data Security* (Cham, February 10–14, 2020). Springer, 505–522.

[18] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE SP* (San Jose, CA, USA, May 22-26, 2016). 839–858.

[19] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. 2006. Dynamic authenticated index structures for outsourced databases. In *ACM SIGMOD* (Chicago, Illinois, USA, June 27-29, 2006). 121–132.

[20] Kai Li, Yuzhe Tang, Qi Zhang, Jianliang Xu, and Ju Chen. 2021. Authenticated key-value stores with hardware enclaves. In *Proceedings of the 22nd International Middleware Conference: Industrial Track* (Virtual Event, Canada, December 6–10, 2021). 1–8.

[21] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. 2019. Teechain: a secure payment network with asynchronous blockchain access. In *ACM SOSP* (Huntsville, ON, Canada, October 27-30, 2019). 63–79.

[22] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. 2019. BITE: Bitcoin lightweight client privacy using trusted execution. In *USENIX Security* (Santa Clara, CA, USA, August 14-16, 2019). 783–800.

[23] Satoshi Nakamoto. 2008. *Bitcoin: A peer-to-peer electronic cash system.* Retrieved October 7, 2022 from https://bitcoin.org/bitcoin.pdf

[24] Pingcheng Ruan, Gang Chen, Tien Tuan Anh Dinh, Qian Lin, Beng Chin Ooi, and Meihui Zhang. 2019. Fine-grained, secure and efficient data provenance on blockchain systems. *PVLDB* (2019), 975–988.

[25] Vasily A Sartakov, Stefan Brenner, Sonia Ben Mokhtar, Sara Bouchenak, Gaël Thomas, and Rüdiger Kapitza. 2018. EActors: Fast and flexible trusted computing using SGX. In *Proceedings of the 19th International Middleware Conference* (Rennes, France, December 10–14, 2018). 187–200.

[26] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *USENIX ATC* (Santa Clara, CA, USA, July 12-14, 2017). 645–658.

[27] Haixin Wang, Cheng Xu, Ce Zhang, Jianliang Xu, Zhe Peng, and Jian Pei. 2022. vChain+: Optimizing Verifiable Blockchain Boolean Range Queries. In *IEEE ICDE* (Kuala Lumpur, Malaysia, May 9-12, 2022). 1928–1941.

[28] Nico Weichbrodt, Pierre-Louis Aublin, and Rüdiger Kapitza. 2018. SGX-perf: A performance analysis tool for intel sgx enclaves. In *Proceedings of the 19th International Middleware Conference* (France, December 10-14, 2018). 201–213.

[29] Ofir Weisse, Valeria Bertacco, and Todd Austin. 2017. Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves. *ACM SIGARCH Computer Architecture News* (June 2017), 81–93.

[30] Gavin Wood. 2014. *Ethereum: A secure decentralised generalised transaction ledger.* Retrieved October 7, 2022 from https://ethereum.github.io/yellowpaper/paper.pdf

[31] Haotian Wu, Zhe Peng, Songtao Guo, Yuanyuan Yang, and Bin Xiao. 2021. VQL: Efficient and Verifiable Cloud Query Services for Blockchain Systems. *IEEE TPDS* 33, 6 (September 2021), 1393–1406.

[32] Cheng Xu, Qian Chen, Haibo Hu, Jianliang Xu, and Xiaojun Hei. 2018. Authenticating Aggregate Queries over Set-Valued Data with Confidentiality. *TKDE* 30, 4 (April 2018), 630–644.

[33] Cheng Xu, Ce Zhang, and Jianliang Xu. 2019. vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases. In *ACM SIGMOD* (Amsterdam, The Netherlands, June 30 - July 5, 2019). 141–158.

[34] Cheng Xu, Ce Zhang, Jianliang Xu, and Jian Pei. 2021. SlimChain: Scaling Blockchain Transactions through Off-Chain Storage and Parallel Processing. *PVLDB* 14, 11 (July 2021), 2314–2326.

[35] Ce Zhang, Cheng Xu, Haixin Wang, Jianliang Xu, and Byron Choi. 2021. Authenticated Keyword Search in Scalable Hybrid-Storage Blockchains. In *IEEE ICDE* (Chania, Greece, April 19-22, 2021). 996–1007.

[36] Ce Zhang, Cheng Xu, Jianliang Xu, Yuzhe Tang, and Byron Choi. 2019. GEM^2-tree: A gas-efficient structure for authenticated range queries in blockchain. In *IEEE ICDE* (Macao, China, April 8-11, 2019). 842–853.

[37] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. In *ACM CCS* (Vienna, Austria, October 24-28, 2016). 270–282.

[38] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. 2017. REM: Resource-efficient mining for blockchains. In *USENIX Security* (Vancouver, BC, August 16–18, 2017). 1427–1444.