EAI Endorsed Transactions
on Security and Safety

Research Article EAI.EU

Efficient Public Blockchain Client for Lightweight Users

Lei Xu[1], Lin Chen[1], Zhimin Gao[1], Shouhuai Xu[2], Weidong Shi[1]

[1]Department of Computer Science, University of Houston
[2]Department of Computer Science, University of Texas at San Antonio

Abstract

Public blockchains provide a decentralized method for storing transaction data and have many applications in different sectors. In order for users to track transactions, a simple method is to let them keep a local copy of the entire public ledger. Since the size of the ledger keeps growing, this method becomes increasingly less practical, especially for lightweight users such as IoT devices and smartphones. In order to cope with the problem, several solutions have been proposed to reduce the storage burden. However, existing solutions either achieve a limited storage reduction (e.g., simple payment verification), or rely on some strong security assumption (e.g., the use of trusted server). In this paper, we propose a new approach to solving the problem. Specifically, we propose an efficient verification protocol for public blockchains, or EPBC for short. EPBC is particularly suitable for lightweight users, who only need to store a small amount of data that is *independent of* the size of the blockchain. We analyze EPBC's performance and security, and discuss its integration with existing public ledger systems. Experimental results confirm that EPBC is practical for lightweight users.

Received on 23 December 2017; accepted on 24 December 2017; published on 4 January 2018

Keywords: blockchain, lightweight client, security

doi:10.4108/eai.4-1-2018.153528

1. Introduction

A public blockchain or ledger consists of a set of blocks that are linked together, where each block contains a set of transactions. A public blockchain is maintained by a group of users, who run a consensus protocol (e.g., proof-of-work with longest-chain) to resolve disagreements regarding the blockchain. In a simple realization of public blockchain, each user keeps a *local* copy of the entire blockchain, meaning that each user has access to all historic activities and can easily test whether a new transaction is consistent with the existing transactions. This explains why a public ledger does not have to rely on any centralized party. This technique is central to many popular applications, such as Bitcoin [1].

Although keeping a local copy of the blockchain in question simplifies many operations (e.g., transaction searching and balance calculation), this imposes a substantial storage overhead because the blockchain keeps growing. For example, the Bitcoin blockchain includes 472,483 blocks in June 2017, or 120 GB in volume. This overhead may not be a problem for modern servers and PCs, but are prohibitive for lightweight users such as mobile devices and IoT devices. In general, this would hinder the development of applications that aim are meant to be built on top of blockchains (e.g., smart contract system [2]). At the same time, smart phones are the major way to get online in some areas, especially in underdeveloped countries, and there is a big need for mobile and lightweight users to use blockchains [3]. Therefore, it is urgent to reduce the storage overhead, especially for those lightweight users.

Indeed, Nakamoto proposes the simplified payment verification (SPV) protocol in the very first Bitcoin paper [1], which requires a client to store *some*, instead of all, blocks while being able to check the validity of transactions recorded in the blockchain. This technique is also widely used in many blockchain-based applications, such as smart contract system [2]. The basic idea underlying the SPV protocol is that each user only needs to keep the headers of blocks,

rather than the blocks themselves. This means that the local storage overhead still increases linearly with the number of blocks, which grows over time and can quickly become prohibitive for lightweight users. An alternate approach is that a lightweight user chooses to trust some nodes in a blockchain system. However, this practice sacrifice the most appealing feature of the blockchains, namely the absence of any trusted third party. Moreover, this approach can be vulnerable to, for example, Sybil attacks [4].

In this paper, we propose an *efficient verification protocol for public blockchain*, dubbed EPBC. The core of EPBC is a succinct blockchain verification protocol that "compresses" the whole chain to a constant-size summary, using a cryptography accumulator [5]. A lightweight user only needs to store the most recent summary, which is sufficient for the user to verify the validity of transactions. EPBC can be incorporated into existing blockchains as a middle layer service, or can be seamlessly incorporated into new blockchain systems.

In summary, our contributions in this work include:

- We design a novel scheme for lightweight users to use public blockchains using cryptographic accumulator.
- We analyze the security and asymptotic performance of the scheme, including its storage cost.
- We report a prototype implementation of the core protocol of EPBC and measure its performance. Experimental results show that the scheme is practical for lightweight users.

The rest of the paper is organized as follows. In Section 2 we briefly review the background of public blockchains and the simplified payment verification protocol. In Section 3 we describe the design of the core component of EPBC, i.e., efficient block verification, and analyze its security. Section 4 describes two common operations for blockchain based applications using the core component of EPBC, and we provide the architecture to integrate EPBC with existing blockchain systems in Section 5. Experimental results are given in Section 6 to demonstrate the practicability of EPBC, and Section 7 discusses the related prior work. We conclude the paper in Section 8.

2. Background of Public Blockchain

A blockchain is a distributed ledger that has been used by Bitcoin and other applications to store their transaction data, where a transaction can be a payment operation, smart contract submission, or smart contract execution result submission. There are different approaches to construct blockchains. In this work, we focus on the class of blockchains that are built on the principle of proof-of-work (PoW) [6]. This class of blockchains have a low throughput and a high latency, but have the desirable properties of fairness and expensive-to-attack. Furthermore, there are many efforts at improving their performance [7, 8] and characterizing their security properties [9].

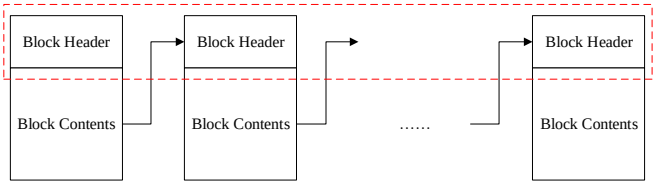

Figure 1. In the SVP scheme, a user stores the headers of the blocks, rather than the blocks themselves. A header contains the relevant meta data (e.g., the root of the Merkle tree whose leaves are the transactions contained in a block). This allows a user to verify whether a given block is valid or not.

Since a blockchain is immutable and append-only, its size keeps growing. There are proposals for coping with this issue. A straightforward approach is to trust some user, who can check the validity of transactions on the user's behalf. This approach assumes that the lightweight user always knows who can be trusted. Another approach is to use the SPV protocol mentioned above [1]. In this scheme, as highlighted in Figure 1, a user only needs to store the block headers, which contain the root of the Merkle tree of the transactions in the corresponding block. When a user needs to verify a transaction, it sends a request to the system asking for the corresponding block, whose validity can be verified by using the root of the Merkle tree.

3. Design and Analysis of EPBC

3.1. Design Objective and Assumption

The objective of EPBC is to allow lightweight users to participate in applications that use public blockchains. By "lightweight users" we mean the users who use devices that have limited computation/storage capacities, such as IoT devices and smartphones. Specifically, EPBC aims to allow lightweight users to achieve the following:

- *Efficient storage*: A user does not have to store or download the entire blockchain. Instead, a user only needs to consume a storage that is ideally independent of the size of the blockchain.
- *Verifiability of transactions*: A user can verify whether a transaction has been accepted by the blockchain or not.

Like any public blockchain constructed according to proof-of-work, we assume that the majority of the users are honest.

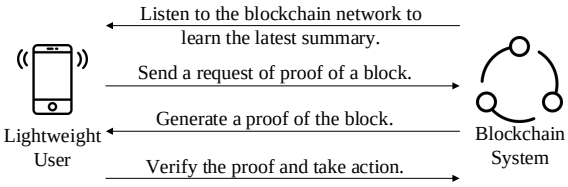

Figure 2. Illustration of the blockchain verification protocol. The nodes in the blockchain system with bigger storage capacities can keep a full copy of the blockchain. These nodes will interact with the lightweight users to help the latter to verify the validity of blocks.

In what follows, we first describe the block verification protocol, which is the core component of EPBC. Then, we describe how to use this protocol to construct the EPBC scheme.

3.2. The Block Verification Protocol

Figure 2 gives an overview of the verification protocol. Basically, a lightweight user can verify the validity of transactions by interacting with the blockchain system.

The blockchain verification protocol of EPBC consists of the following four algorithms:

- *Setup*: This algorithm is executed once by the creator of the blockchain. The algorithm generates the public parameters that are needed by the other algorithms.
- *Block and summary construction*: This algorithm generates blocks and a summary of the current blockchain. Anyone participating in the mining competition to build new blocks is responsible for calculating the summary of the current blockchain. The summary depends on the content of the current blockchain and the public parameters.
- *Proof generation*: This algorithm generates a proof for a given block. The proof may depend on, among other things, the entire blockchain.
- *Proof verification*: Given the summary of a blockchain and a proof for a single block, this algorithm verifies whether the proof is valid or not.

With this protocol, a lightweight user keeps the updated summary of the blockchain. When the user wants to verify a specific block, it can ask the parties that are involved in a transaction for a proof for the block, which is generated by running the *proof generation* algorithm. The user then executes the *proof verification* algorithm to determine whether to accept the block or not. In what follows we describe the details of these algorithms.

Setup. The creator of the blockchain selects two large prime numbers p, q, and calculates $N = pq$ as in the RSA accumulator system. N is embedded into the first block and disclosed to the public; and then p, q are discarded. The creator also selects a random value $g \in \mathbb{Z}_N^*$. Each block will be labelled with an integer, with the "genesis" block (i.e., the first block on the blockchain) has the label "1".

Block and summary construction. Each block contains, in addition to the standard attributes (e.g., transaction information and proof-of-work nonce), a new attribute S, which is the summary of the current blockchain. For the i-th block, which is denoted by blk_i, the attribute S_i is calculated and stored with blk_i as follows:

$$S_i = \begin{cases} g^{\mathsf{hash}(blk_i \| i)} & \mod N, \quad \text{if } i = 1, \\ S_{i-1}^{\mathsf{hash}(blk_i \| i)} & \mod N, \quad \text{if } i > 1. \end{cases}$$

If the current blockchain contains n blocks, S_n is the summary of the current blockchain. The block position information i is used in the computation for the purpose of preventing the attacker from manipulating the position of a block. After the newly generated block is broadcast to the blockchain system, the following two algorithms can be executed.

Proof generation. To generate a proof that shows block blk_i is the i-th block on the blockchain with summary S_n, where $i \le n$, the prover calculates $p_i = (p_i^{(1)}, p_i^{(2)})$ as follows:

$$p_i = \begin{cases} p_i^{(1)} & = \mathsf{hash}(blk_i \| i), \\ p_i^{(2)} & = g^{(\prod_{k=1}^{n} \mathsf{hash}(blk_k \| k))/\mathsf{hash}(blk_i \| i)} \end{cases} \mod N.$$

Note that the proof is generated by a user who keeps the entire blockchain and therefore can compute $p_i^{(2)}$ without knowing $\phi(N)$, where ϕ is the Euler function.

Proof verification. Given a block blk, a claimed proof $p = (p^{(1)}, p^{(2)})$, and a blockchain summary S_n, a user can verify that block blk_i is indeed the i-th block on blockchain with summary S_n, where $i \le n$, as follows:

$$\begin{cases} p^{(1)} \stackrel{?}{=} \mathsf{hash}(blk_i \| i), \\ S_n \stackrel{?}{=} (p^{(2)})^{p^{(1)}} \end{cases} \mod N.$$

If both equations hold, the user accepts that p is a valid proof for blk; otherwise, the verifier rejects the block.

3.3. Parameter Initialization

One of the key steps in the blockchain verification protocol is the parameter initialization, i.e., selecting p and q to generate the modulus N. If p or q is

exposed, the protocol is clearly not secure. This issue can be addressed by generating N using a multi-party protocol. There have been many protocols for this purpose. For example, the protocol proposed by Cocks [10] works as follows. Suppose at the beginning there are ℓ users who work together to generate the first block.

1. Each user i, $1 \le i \le \ell$, selects his/her own prime numbers p_i, q_i.
2. Each user i, $1 \le i \le \ell$, calculates $N = (p_1 + p_2 + \cdots + p_\ell)(q_1 + q_2 + \cdots + q_\ell)$. By leveraging the protocol given in [10], user i can calculate N without knowing the two factors of N.
3. Each user tests whether N is a product of two prime numbers or not. Specifically, the system randomly selects a random number x and each user calculates $x^{p_i+q_i} \mod N$. If $\prod x^{p_i+q_i} \mod N \equiv x^{N+1} \mod N$, N passes the test. Carmichael numbers that can pass this test can be further eliminated by methods given in [11].
4. If the current N passes all tests, users work together to embedded it in the genesis block. Otherwise they repeat the process again, until an appropriate N is found.

Since N only needs to be generated once, the cost of the parameter initialization is not a big concern.

3.4. Security and Performance of the Block Verification Protocol

It is straightforward to verify that the protocol is correct, meaning that any legitimate proof will be accepted as valid. The following theorem shows that for a given summary S of blockchain BC, no attacker can generate a valid proof for a forged block blk' that is not contained in BC under strong RSA assumption.

Theorem 1. Given a summary S_n of blockchain BC, there is no probabilistic polynomial-time attacker $\mathcal{A}$ that can forge a block blk' and an accompanying proof P' that blk' is a valid block on blockchain BC in the random oracle model; otherwise, the Strong RSA assumption is broken.

Proof. Suppose hash() behaves like a random oracle. Let $r_i = \text{hash}(blk_i \| i)$ where blk_i is the i-th block on BC, and $S_n = g^{\prod_{k=1}^{n} r_k} \mod N$. We consider two scenarios of attacks:

- The attacker knows the summary S_n but not the blockchain. Suppose the attacker chooses blk' and position i' for the block. Then, the attacker needs to compute $y \in \mathbb{Z}_n^*$ such that

$$y^{\text{hash}(blk' \| i')} \mod N = S_n.$$

This immediately breaks the Strong RSA assumption.

- The attacker knows both blockchain and the summary S_n. In this case, the attacker knows all valid proofs for blocks in BC, i.e., $(r_i, S_n^{\frac{1}{r_i}} \mod N)$, $i = 1, \ldots, n$. Suppose the attacker can generate a valid proof for a forged block blk' for some position i'. Let $r' = \text{hash}(blk' \| i')$. If $r' | \prod_{i=1}^{n} r_i$, the attacker can successfully make a valid proof for blk' at position i' because the attacker can compute $(r', S_n^{\prod_{i=1}^{n} r_i / r'})$. Because the attacker cannot control the output of hash(), the probability that the attacker can succeed is equivalent to the probability that a random number r' is a factor of another random number $R = \prod_{i=1}^{n} r_i$. According to Erdös-Kac theorem [12] and its extension counting multiplicities [13], the number of prime factors of R counting multiplicity is $\mathcal{O}(\log \log R)$. With Binomial theorem, the total number of divisors of R is $\mathcal{O}(2^{\log \log R}) = \mathcal{O}(\log R)$, and $\lim_{R \to \infty} \frac{\log R}{R} = 0$. Therefore, the probability that an attacker can find r' is negligible when R is large enough. As long as the attacker cannot find such r', a successful attack implies that the the Strong RSA assumption is broken.

In summary, there is no practical attack against the protocol in the random oracle model unless the Strong RSA assumption is broken. □

Performance of the major algorithms is analyzed as follows.

- Block construction. When compared with the straightforward method by which each user keeps the entire blockchain, our method incurs some extra work in the block construction algorithm. The extra work consists of two parts: evaluating the hash value of the new block and calculating the new summary. The computation overhead is constant (i.e., one hash calculation and one modular exponentiation) and the storage overhead is also constant (i.e., an element in $\mathbb{Z}_N$ for the summary). The summary also incurs extra communication cost, which is however small (e.g., 2048 bits for a 2048-bit N).
- Proof generation. The proof generation algorithm does not incur extra storage. The computational cost is proportional to the length of the current blockchain (i.e., the number of blocks in the chain) and the position of the block. Suppose the length of the blockchain is n, and the proof of i-th block needs to be generated, where $i \le n$. The prover needs to conduct one hash evaluation of the ith block, and calculates the

product of hash values of blocks $1, \ldots, i-1, i+1, \ldots, n$. In summary, the prover calculates $n+1-i$ hashes, $n-1$ multiplications, and one modular exponentiation. Since the nodes with sufficient storage capacity (rather than the lightweight users) are supposed to generate proofs, the protocol is practical.

- Proof verification. The computational cost to verify the proof of a block includes one hash evaluation and one modular exponentiation, which is constant. This explains why the protocol is suitable for lightweight users who only keep the summary of the blockchain.

3.5. Reducing Cost of Proof Generation

Although both the cost of updating the summary of a blockchain and the cost of verifying a block are constant, the computational complexity for the prover to generate a proof is $\mathcal{O}(n)$, where n is the number of *current* blocks on the blockchain (i.e., n keeps increasing). In the worst-case scenario, the prover needs to traverse all of the blocks on the blockchain to calculate the second part of the proof, namely

$$g^{(\prod_{k=1}^{n} \mathsf{hash}(blk_k))/\mathsf{hash}(blk_i)} \mod N.$$

In order to reduce the computational complexity incurred by this, we design a scheme that improves the computational efficiency at the price of a slight increase in storage.

Proof generation with a smaller computational complexty. The basic idea underlying the scheme is to let the prover maintain a binary tree $\mathcal{T}$. As illustrated in Figure 3, the binary tree is used to store intermediate results that can be used to generate a proof for a given block. Specifically, each leaf stores the hash value of a corresponding block, and each internal node stores the product of its two direct children nodes. This way, the root node stores the product of the hash values of all of the blocks on the blockchain. The height of $\mathcal{T}$ is predetermined. If a leaf is empty (i.e., currently there is no corresponding block on the blockchain), its value is set to 1 so that it does not contribute to the value stored at the root node.

Suppose the height of tree $\mathcal{T}$ is h and the number of currrent blocks on blockchain is n, where $n < 2^{h-1}$. To calculate a proof for block blk_i, where $1 \le i \le n$, the prover leverages the information stored in $\mathcal{T}$ as follows:

- Find the product of all of the values on the right-hand of blk_i (the blockchain grows from left to right)

$$r \leftarrow \prod_{k=i+1}^{n} \mathsf{hash}(blk_k). \tag{1}$$

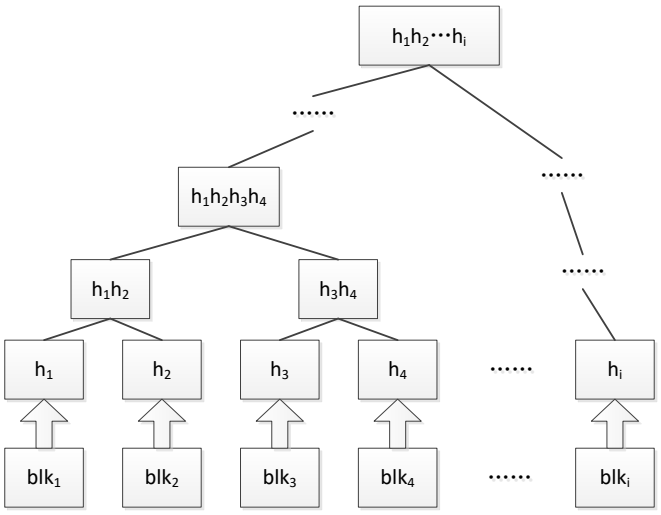

Figure 3. The storage structure that can be used by a prover to reduce its computational complexity when generating proofs. Each leaf h_i stores the hash value of a block, and each internal node stores the product of the values stored at its two children.

Instead of conducting the multiplication operation one-by-one, the prover utilizes different products information stored in $\mathcal{T}$ to accelerate the computation.

- Calculate $LR \leftarrow (S_i)^{r/\mathsf{hash}(blk_i)} \mod N$.
- Set the proof as $P \leftarrow (\mathsf{hash}(blk_i), LR)$.

Note that the height of $\mathcal{T}$ determines the number of blocks it can accommodate, and is therefore a predetermined public parameter. If the height of $\mathcal{T}$ is h, the total number of blocks it can accommodate is 2^{h-1}. This is no significant constraint because a relatively small h can accommodate a large number of blocks. For example, when $h = 32$, the structure can accommodate 4,294,967,296 blocks, which are about 9,000 times larger than the number of blocks on the Bitcoin network as of April 2017.

Analysis of the improved scheme. The improved scheme involves a binary tree $\mathcal{T}$ to store some information that can be used for generating proofs. Let $\mathtt{height}(\mathcal{T}) = h$, meaning that $n = 2^{h-1}$ is the number of leaves. Let $|\mathsf{hash}()| = \ell$. At the leaf level (i.e., the first level), the size of each node is ℓ. Each node at i-th level incurs $i \cdot \ell$ bits of storage, and the size of the root node is $h \cdot \ell$ bits. Therefore, the size of $\mathcal{T}$ is

$$\underbrace{n \cdot \ell}_{\text{first level}} + \cdots + \underbrace{(n/2^i) \cdot (2^i \ell)}_{i\text{-th level}} + \cdots + \underbrace{(n/2^{h-1}) \cdot (2^{h-1} \ell)}_{h\text{-th level, root}}$$

$$= \sum_{i=0}^{h-1} n \cdot \ell = h \cdot n \cdot \ell = (\log_2 n + 1) \cdot n \cdot \ell = \mathcal{O}(n \log n).$$

With intermediate results stored in $\mathcal{T}$, the computation complexity for generating a proof is reduced to h (or $\mathcal{O}(\log n)$) modular exponentiations.

More generally, if each internal node in Figure 3 has m children, the height of $\mathcal{T}$ is reduced to $\log_m n + 1$. A similar analysis shows that the total size of $\mathcal{T}$ is $(\log_m n + 1) \cdot n \cdot \ell$, which is the size of storage a prover keeps locally. In order to calculate r, which is defined in Equation (1), it requires about $\log_m n + m$ multiplication operations in the worst-case scenario, where m is the number of multiplications incurred at an internal node at the second level of $\mathcal{T}$. In order to select the value of m so as to minimize the overall computational complexity, we calculate the derivative as follows:

$$(\log_m n + m)' = (\frac{\ln n}{\ln m} + m)' = 1 - \frac{\ln n}{m \ln^2 m},$$

which monotonically increases with respect to m. Therefore, we get the minimum value when

$$1 = \frac{\ln n}{m \ln^2 m},$$

and $m \approx \ln n$. In practice, we can set the number of branches to a small constant integer so as to reduce the computational complexity of the prover.

4. Using the Block Verification Protocol to Construct EPBC

In this section, we discuss construction of high-layer operations based on the verification protocol described in Section 3. Specifically, we focus on two basic protocols: *blockchain identification* and *transaction verification*.

Blockchain identification. When a lightweight user needs to join a blockchain based application, it needs to obtain the current summary of the blockchain. Protocol 1 is for this purpose.

Note that as long as the attacker does not control majority of the ℓ users, the protocol is secure. The lightweight user can also adopt other strategies to determine the summary, e.g., giving different weights to selected users and include this information when making the decision.

Transaction verification. A transaction is valid if and only if the block it belongs to is accepted by the majority of users, i.e., on the longest branch of the blockchain. Therefore, verification of a transaction is reduced to checking the validity of a block and its position (i.e., block number). A lightweight user can use the block verification protocol to verify that the block in question indeed contains the transaction in question. Then, the lightweight user can check the number of blocks that have been added after the block that is verified. Similar to the Bitcoin system [1], if more than 6 blocks have been added to the blockchain after the block under consideration, the transaction in question can be accepted with high confidentiality.

If the transaction is a smart contract submission or one-time smart contract execution result submission, the above method is also sufficient. However, if the transaction is a payment operation or submission of multiple-time smart contract execution result, freshness becomes a concern. For example, the attacker can provide proof of an old block that contains previous payment of the same value. To prevent such attacks, the lightweight user can maintain a local counter and include the counter in its transactions.

Protocol 1 Blockchain identification.

1: The lightweight user randomly selects a group of ℓ users, denoted by G_u, from the blockchain network;

2: **for all** $u \in G_u$ **do**

3: The lightweight user queries u to get the summary value $S^{(u)}$;

4: The lightweight user interacts with u to verify the validity of $S^{(u)}$ with respect to a random set of blocks chosen by the lightweight user;

5: **end for**

6: The lightweight user calculates

$$S \leftarrow \texttt{SummaryDetermination}(S^{(1)}, \ldots S^{(\ell)}),$$

which returns the summary that is provided by majority of the users, where S is the final summary of the blockchain;

5. Integration with Existing Blockchain Systems

Because a lot of public blockchain applications have been developed, it is useful to enable EPBC for these systems without modifying existing data structures and client. To achieve this goal, EPBC can work as a separate service layer on top of existing blockchain systems. Figure 4 demonstrates the relationship between the existing blockchain system and the newly added EPBC service.

Specifically, a separate EPBC client with embedded parameters can be distributed to users who maintain the blockchain and play the role of a prover. Here parameters are values that used for blockchain summary construction. Summaries of the blockchain are not involved in mining, and users can use existing client to produce new blocks and achieve consensus on the blockchain. After the user decides to accept a new block, the EPBC client produces a new summary based on previous summary value and the new block, and

stores the new summary locally. Note that summaries are determined by the blockchain itself so EPBC client does not need to run any consensus mechanism. If the user wants to reduce the time complexity of generating a proof, EPBC client can maintain the tree structure described in Section 3.5.

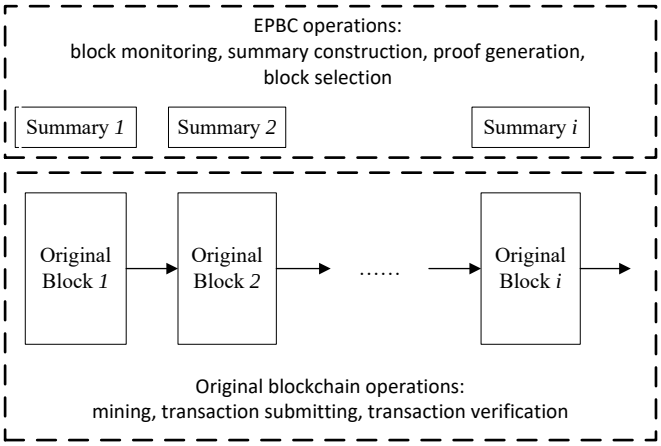

Figure 4. Overview of the integration of EPBC with an existing blockchain systems. A dedicated client is used to support EPBC related operations.

6. Experiments and Evaluation

In this section, we describe the implementation and provide preliminary experimental results of EPBC. We focus on the block verification protocol because it is the core of EPBC.

Implementation and parameters. We implemented a prototype of the block verification protocol based on the MIRACL crypto library [14]. Since security of the protocol depends on the Strong RSA assumption, we chose a 1,024 bits N in the implementation. SHA256 was used for hash(). We also set the height of $\mathcal{T}$ as 32. When a leaf is empty, its value is set to 1 and there is no need to store it.

Experimental results. We conducted the experiments on a desktop with a low-end Intel Celeron 1017U processor, which has a similar Geekbench 4 score of Snapdragon 805 processor [15]. The experimental results are summarized in Figure 5, which shows that although the cost of proof generation depends on the size of the blockchain, the cost of proof verification is independent of the blockchain size.

As discussed in Section 4, some high-level operations like balance checking require the lightweight client to verify more than one blocks. This is not a problem in practice for the user using lightweight client because it only takes about 0.02 second to verify one block.

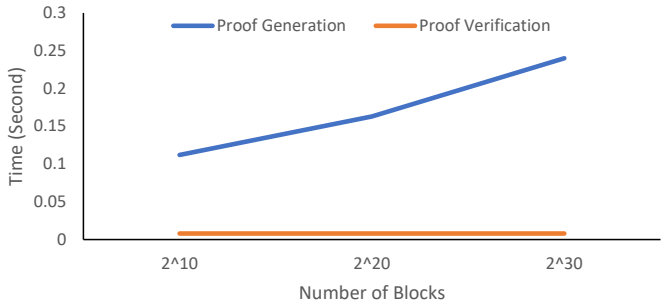

Figure 5. Preliminary experimental results of the block verification protocol using a low end Celeron CPU.

7. Related Works

EPBC only provides the mechanism for checking the validity of a given block and the transactions contained in the block. It does not consider how to determine which block(s) should be checked. It is proposed in BIP 37 to use a bloom filter to select potentially related blocks for verification [16]. The Bitcoin community proposes the UTXO (unspent transaction outputs) technology, which requires the user to store unspent transaction output information instead of transaction information. This reduces the storage cost but does not change the order of storage complexity [17].

Cryptographic accumulator was first developed by Benaloh and De Mare to achieve decentralized digital signature [5]. Barić and Pfitzmann developed a collision-free accumulator and used it for fail-stop signatures without using any tree structure [18]. Cryptographic accumulators are useful (e.g., constructing group signatures [19]). Dynamic cryptographic accumulator can further support adding/removing members [20]. These schemes do not consider features of blockchains, namely that every user has the privilege to construct blocks and generate proofs and lightweight users have very limited computational capability. Recently, e-cash systems such as ZeroCoin also utilizes cryptographic accumulators, but for a different purpose of information hiding [21].

Another line of related research is storage verification in the cloud environment, and several related concepts were proposed, e.g., provable data possession [22] and proof of retrievability [23]. These schemes cannot be applied in our scenario because the lightweight users do not know the blockchain in advance and the blockchain keeps growing as new blocks are created and appended to it.

Both EPBC and SPV assume the records that are embedded into blocks are correct if the corresponding blocks are valid. Some techniques that are applicable to SPV, such as bloom filter [24], are also applicable to EPBC. Nevertheless, EPBC incurs only a constant amount of storage for the lightweight client, assuming

the client cares about most recent transactions. This is significant because storing several block headers might be cheaper than storing the summary value.

8. Conclusion

We have presented EPBC, a scheme for lightweight users to use blockchain-based applications without storing the entire blockchain while still able to verify the validity of blocks and transaction. The basic idea is to "compress" a blockchain to a constant-size summary, which is the only data item a lightweight client needs to keep. We analyzed the security of EPBC and preliminary experiments showed that it is practical. EPBC can be adopted for blockchain-based applications, such as e-cash and smart contract systems.

Acknowledgement. This material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 2015-ST-061-BSH001. This grant is awarded to the Borders, Trade, and Immigration (BTI) Institute: A DHS Center of Excellence led by the University of Houston, and includes support for the project "Secure and Transparent Cargo Supply Chain: Enabling Chain-of-custody with Economical and Privacy Respecting Biometrics, and Blockchain Technology" awarded to University of Houston. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

References

[1] Nakamoto, S. (2008), Bitcoin: A peer-to-peer electronic cash system.

[2] Wood, G. (2014) Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* **151**.

[3] Christidis, K. and Devetsikiotis, M. (2016) Blockchains and smart contracts for the internet of things. *IEEE Access* .

[4] Douceur, J. (2002) The sybil attack. *Peer-to-peer Systems* : 251–260.

[5] Benaloh, J. and De Mare, M. (1993) One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques* (Springer): 274–285.

[6] Vukolić, M. (2015) The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security* (Springer): 112–125.

[7] Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A. *et al.* (2016) On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security* (Springer): 106–125.

[8] Luu, L., Narayanan, V., Baweja, K., Zheng, C., Gilbert, S. and Saxena, P. (2015) *SCP: a computationally-scalable Byzantine consensus protocol for blockchains*. Tech. rep., Cryptology ePrint Archive, Report 2015/1168.

[9] Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H. and Capkun, S. (2016) On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (ACM): 3–16.

[10] Cocks, C. (1997) Split knowledge generation of rsa parameters. In *IMA International Conference on Cryptography and Coding* (Springer): 89–95.

[11] Boneh, D. and Franklin, M. (1997) Efficient generation of shared rsa keys. In *Advances in Cryptology-CRYPTO'97: 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 1997. Proceedings* (Springer): 425.

[12] Erdös, P. and Kac, M. (1940) The gaussian law of errors in the theory of additive number theoretic functions. *American Journal of Mathematics* **62**(1): 738–742.

[13] Billingsley, P. (1969) On the central limit theorem for the prime divisor function. *American Mathematical Monthly* : 132–139.

[14] Certivox Ltd, MIRACL cryptographic library. URL https://libraries.docs.miracl.com/miracl-user-manual/about.

[15] Primate labs, Geekbenchmark 4. URL http://www.geekbench.com/.

[16] Hearn, M. and Corallo, M. (2012), BIP 37: Connection bloom filtering. URL https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki.

[17] Bishop, B. (2015) Review of bitcoin scaling proposals. In *Scaling Bitcoin Workshop Phase*, **1**.

[18] Barić, N. and Pfitzmann, B. (1997) Collision-free accumulators and fail-stop signature schemes without trees. In *International Conference on the Theory and Applications of Cryptographic Techniques* (Springer): 480–494.

[19] Tsudik, G. and Xu, S. (2003) Accumulating composites and improved group signing. In *Asiacrypt* (Springer), **2894**: 269–286.

[20] Goodrich, M.T., Tamassia, R. and Hasić, J. (2002) An efficient dynamic and distributed cryptographic accumulator. In *International Conference on Information Security* (Springer): 372–388.

[21] Miers, I., Garman, C., Green, M. and Rubin, A.D. (2013) Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on* (IEEE): 397–411.

[22] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z. and Song, D. (2007) Provable data possession at untrusted stores. In Ning, P., di Vimercati, S.D.C. and Syverson, P.F. [eds.] *Proceedings of the 2007 ACM Conference on Computer and Communications Security -CCS 2007* (ACM): 598–609.

[23] Zheng, Q. and Xu, S. (2011) Fair and dynamic proofs of retrievability. In *Proceedings of the first ACM conference on Data and application security and privacy* (ACM): 237–248.

[24] Gervais, A., Capkun, S., Karame, G.O. and Gruber, D. (2014) On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference* (ACM): 326–335.