Improving Authenticated Dynamic Dictionaries, with Applications to Cryptocurrencies*

Shahla Atapoor

University of Tartu, Estonia

shahla.atapoor@ut.ee

November 24, 2017

Abstract

This essay aims to present a short review on the key points of a paper with the same title as this essay, which is published by Reyziny, Alexander, Chepurnoy and Ivanov in 2017 [RMCI16]. The paper presents several improvements on the designing and implementing of two-party and three-party authenticated dynamic dictionaries. The improved schemes are practically evaluated in the proof-of-stake of cryptocurrencies. The result shows that new schemes provide considerable improvements in the proof size, verification time, and proof generation time which are three essential factors in proof-of-stake. More precisely, in comparison with state of the art schemes, their improved schemes reduce the size of proof about 20% and make the verification 1.4 - 2.5 times faster.

1 Introduction

During last few years, cryptocurrencies are one of the most popular and challenging topics in computer science. All cryptocurrencies are based on blockchains which are the entire sequence of all performed transactions. All transactions need to be verified by miners before adding to the blockchain. This verification and validation by miners are categorized in two parts including stateless and state. In the former part (stateless), verifier just checks syntax, signature of prover, fee is positive, and etc. Indeed, verifier needs only information kept in the transaction itself. But in the second part of validation, it is required to use information of the previous transactions. As an example, suppose that Alice wants to prove Bob that she has enough money to buy a product. In this case, verification claims such as Alice has non-zero money or Alice is the right person can be verified in the stateless part of validation. But, in order to prove that Alice has more than a particular value money (e.g. $P_0$) needs to use prior transactions which needs large amount of memory and storage.

Motivation: The main motivation of the paper is that blockchain validation requires storing a large state which needs amount of memory and storage. Author discuss that, in the case that we store all states on a disk, the validation is very slow. In the case that we store them on a RAM, we need large size of RAM which leads to limit the ability of weak devices to validate and extends centralization (only powerful computers can do validation).

More accurately, this paper tries to present an efficient solution to the question that How we can generate an efficient proof (as small as possible) for such a scenario and verify the proof efficiently?

2 Contribution

To answer the stated question in motivation part, author propose to use cryptographically authenticated data structures to make verification of transaction much more efficient. Particularly, they propose to store balance information in a dynamic authenticated dictionary. In that case, provers (miners) hold the entire data structure and update it after each transaction. They also publish proofs that each transaction resulted in the correct modification of the data structure. In the other side, verifiers just hold a short digest of the current data structure and uses them to verify the proof.

Due to importance of smaller blocks in the blockchain systems, authors’ priority is reducing the length of a modification proof, which includes into the block for each transaction. Their protocol does not need to trust any third trusted party or setup phase.

3 Construction

The construction of the presented authenticated data structure by Reyziny, Alexander, Chepurnoy and Ivanov [RMCI16] can be summarized as follows.

![Making a Merkle tree.](image)

First of all, they make a Merkle tree where leaves are the data which need to be authenticated. The label of...
each leaf is the hash of its content. They put the Merkle root into a block header (see Fig. 1).

Then, when Alice (PK_A) wants to send a transaction to David (PK_D), and needs to prove that she has enough money; she sends an authenticating path which is shown with rectangle around in Fig. 1. More precisely, she sends the Merkle path which are the hashes of sibling on the path from the leaf to the root of the Merkle tree. So, the verifier (or anybody who needs to verify) can verify that it is a leaf in the Merkle three. This makes it possible that a light verifier can check the entire block of transactions without storing the huge values (all states).

The verifier just takes the root hash, take the transaction and the proofs. Then for each proof, he does the Merkle verification and output yes or no. Note that verifier can do it very quickly, because that Merkle paths are logarithmic in the size of the stored values (PK_A, PK_B, · · ·).

After a successful transaction, the tree is updated (money of Alice decreases and money of David increases) and the consequently hash values and the root hash are updates which will be put in the next block. Note that, unlike prior schemes, in the new proposed scheme in this paper, in addition to verification, the verifier needs to compute the root hash value for next block and check that it is correct (see Fig. 2). Note that verifier does not have all values of the tree, but he is able to compute the root hash values. It is worth to mention that in prior works, the new root hash is given by a third party, but in this paper verifier computes the new root hash value.

The discussed model, explained the 2-party model which just has prover and verifier. But, in the 3-party schemes there is a third party as well. Based on the most relevant construction which are in the same setting are schemes of works of Papamanthou and Tamassia [PT07] which is based on skip lists and work of Miller et al. [MHKS14] which is based on skip list and read-black trees which their efficiency for length of proof is compared in the following table.

<table>
<thead>
<tr>
<th>Previous methods</th>
<th>The paper work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skip list</td>
<td>AVL +</td>
</tr>
<tr>
<td>Update Existing</td>
<td>1.5 H log N</td>
</tr>
<tr>
<td>Insert New</td>
<td>1.5 H log N</td>
</tr>
<tr>
<td>Required trusted</td>
<td>Required using</td>
</tr>
<tr>
<td>randomness</td>
<td>more nodes</td>
</tr>
</tbody>
</table>

4 Implementation Result

To evaluate the schemes, authors implemented AVL+ trees, treaps and tree-based skip-lists in the Scala using Blake2b hash function with 256-bit outputs. They applied several optimizations such as combining proofs and etc [RMC16]. As an evaluation result, Fig. 3 shows the proof size with and without applying the optimizations.

![Figure 3: A comparison of proof size vs. data size.](image)

As it can be seen, the optimized scheme (blue line) has considerably smaller proof size. As an example, for the case that data size is 2^{20}, optimized case gives a 300 bytes proof which is very less that 600 bytes for gzip with compression and 750 bytes for standard case.

5 Conclusion

In summary, we observed that how the recently proposed two-party authenticated data structures dictionary by Reyzin, Alexander, Chepurnoy and Ivanov [RMC16], can be used to greatly to improve efficiency of proof systems in blockchains. Due to lack of space, we just presented main intuition behind the proposed scheme.

Acknowledgment: The author has received the IT academy Specialization Stipend for the Autumn semester of academic year 2017/2018.

References