Scaling Blockchain-Based Tokens with Joint Cryptographic Accumulators

Trevor Miller
trevormil@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

ABSTRACT

As digital tokens on blockchains such as non-fungible tokens (NFTs) increase in popularity and scale, the existing interfaces (ERC-721, ERC-20, and many more) are being exposed for being expensive and not scalable. As a result, tokens are being forced to be implemented on alternative blockchains where it is cheaper but less secure. To offer a solution without making security tradeoffs, we propose using joint cryptographic accumulators (e.g. joint Merkle trees). We propose a method of creating such joint accumulators in a decentralized fashion which is secured by the same set of validating nodes as existing blockchains. Such accumulators allow the tokens for certain applications to be implemented using up to 99.99% less of the blockchain's resources by outsourcing most of the storage and computational requirements to the users creating the tokens. This is done without sacrificing permanence and verifiability of these tokens. This system achieves optimizations mainly by allowing certain storage of a blockchain to be used in a cross-application manner, instead of a per-application manner. Additionally, we show how it can be beneficial in other areas like privacy-preserving timestamps or shortening file hashes.

KEYWORDS

blockchain, commitments, privacy, merkle, tokens, nfts, scaling, accumulators

1 INTRODUCTION

Many digital and real-world applications have been touted to be revolutionized by blockchains [10] [19]. This is because blockchains offer these applications a unique set of characteristics which has never been seen before in the digital world. Specifically, blockchains are public, decentralized, distributed, and append-only ledgers whose current state is continuously agreed upon through the consensus of network participants and not by any centralized party [1]. So for example, the finance sector can transition to using cryptocurrencies like Bitcoin which use blockchains for an immutable, public, decentralized, tamper-proof ledger of digital payment transactions [14]. This offers many advantages over existing currencies such as not being controlled by a single, centralized entity [14].

Cryptocurrencies are a specific type of digital blockchain token which has monetary value, but digital tokens can be used for many different purposes on blockchains [24]. For example, a currently popular use case of blockchain tokens is digital identity as seen with the Ethereum Name Service (ENS). In ENS, individual tokens representing a unique username can be created, so instead of interacting with others via a complex pseudonymous public key or address consisting of random letters and numbers, blockchain interactions can be done via verifiably owned human-readable username tokens [20].

Different use cases of digital tokens will have different requirements. For ENS, each token should have its own unique properties (i.e. representing a unique username) [20]. Alternatively, some use cases may require all tokens being fungible. Some may require different capabilities like being able to revoke a token. As a result of these various needs, multiple different token interfaces have been developed [24]. For example, Ethereum (the most popular blockchain currently for digital tokens) has the ERC-721 interface for non-fungible tokens (NFTs), ERC-20 interface for fungible tokens, and ERC-1155 interface for semi-fungible tokens [6]. Verifiable credentials (VCs) is another token standard that is used for more credential-based applications like diplomas or driver's licenses [23]. Or, soulbound tokens (SBTs) have recently been proposed by Vitalik Buterin which are tokens that permanently live in a user's account and can never be transferred [25].

These interfaces are versatile enough to handle a broad set of use cases, but the problem is that they are all expensive, not scalable, and take up a fair amount of storage which is especially scarce for append-only, permanent, distributed blockchains. For example, Seaport from OpenSea is recognized as the most gas efficient implementation for an ERC-721 non-fungible token marketplace, but transaction fees for buying a token can exceed $15 per purchase at the time of writing this (136,736 gas * 50 gwei gas price * $1300 per Ethereum) [21] [3] [13] [7].

While a solution is to use alternative blockchains which are cheaper and have more storage available, this typically comes with tradeoffs such as decreased security and less decentralization. For example, many globally recognized brands like Instagram, Starbucks, and Reddit have chosen to implement their NFT projects on the Polygon blockchain, a blockchain which is cheaper than Ethereum but is less secure [18]. Given this dilemma, we ask the following question: can digital tokens be more efficiently and cheaply implemented on the most secure and expensive blockchain which decreases the need to use alternative blockchains and make such security tradeoffs as often?

2 OVERVIEW

2.1 Hybrid Proof-of-Commitment Applications

This paper will focus mainly on the efficiency of implementation for a specific subset of applications whose tokens have unique properties. We will call such applications hybrid proof-of-record (HPoR) applications. HPoR applications are applications which only use the blockchain as a way to record some information in a tamper-proof and permanent way, and everything else is done off-chain. For example, a teacher in an in-person classroom setting may want

to use the blockchain to issue a tamper-proof, verifiable, permanent attendance token to all students who attend class, and this token is not used for anything else on-chain besides verifiable, permanent record-keeping.

Furthermore, we will specifically focus on a subset of HPoR applications: hybrid proof-of-commitment (HPoC) applications. In these applications, the information to be recorded is stored off-chain, but it is committed to using a binding cryptographic commitment scheme on-chain, such as a hash, for verifiability and permanence. For verification of the information at a later time, one can prove the information committed to in the commitment matches the information that is to be verified.

To demonstrate, let's reuse our attendance token example from above. A teacher could digitally sign the following message with their key pair: "Student X attended class today on 1/1/2020". Then, they could hash the signature (or use another commitment scheme), post the hash to the blockchain, and send the signature-hash pair to Student X off-chain. If the student needs to verify their attendance in the future, they can just prove the preimage of the on-chain, tamper-proof hash is the signature. If needed, the timestamp of the on-chain hash is also public. Note how the blockchain is only used for verification and permanent record-keeping and nothing else in HPoC applications. Also, note how to other users on the network, the attendance message is not actually visible, only the hash is.

2.1.1 Categories. We further categorize the commitments for HPoC applications into two categories: explicitly known and not explicitly known. For applications that need the binding commitments to be explicitly known, they have to be stored on-chain in some queryable form where a user can not change their commitment. This is typically needed in applications where volume needs to be enforceable. For example, let's say commitments are used by students to prove knowledge of answers to a multiple choice quiz question by the submission deadline. It doesn't suffice for a student to just reveal any previous arbitrary on-chain commitment. This is because the student could potentially commit to all possible answers before the deadline and when they later learn the right answer, they can just use their commitment which had the right answer. The solution is for each student's commitment(s) to be explicitly known by being queryable to others which enforces that they can not change it.

For applications where commitments do not need to be explicitly known, there is no such requirement. These applications typically only require proving some unique knowledge by a specific time. Our previous attendance token example is one such application. If all that is required is for a student to prove their attendance in the case of disputes, then as long as the unique attendance message signed by the teacher is verifiable on-chain, that is satisfactory.

2.1.2 Optimizations. Because HPoC applications are hybrid and only use the blockchain for permanent record keeping, there are some optimizations that can be implemented to take advantage of this. Notably, since the only requirement is verifiability, we can efficiently combine multiple commitments for each application into a cryptographic accumulator such as a Merkle tree and store only the fixed-length digest on-chain [17]. This still maintains the verifiability if additionally membership of the commitment in the accumulator can be proven using the fixed length digest (i.e. a commitment's Merkle path to the on-chain Merkle root).

By using cryptographic accumulators, this achieves optimizations for both commitments which do and do not need to be explicitly known. For commitments that do not need to be explicitly known, this is straightforward as N commitments for an application can now be stored using a fixed-length digest instead of being stored directly. This decreases the on-chain storage requirement from $O(32N)$ to $O(32)$ (assuming 32 bytes for the digest and commitments).

For commitments which do need to be explicitly known, this is not as straightforward because volume is not verifiable from a fixed-length digest which is required to be known. For example, if presented only a Merkle root, the number of valid Merkle paths and leaf node inputs is infinite.

However, what is possible is to store the commitments in a cryptographic accumulator and then additionally make these commitments explicitly known via a lightweight on-chain mapping. An example of such a lightweight mapping could be claiming that your commitment lies at location 4 in the Merkle root with ID #1000. In this example, the on-chain storage requirements decreases from $O(32N)$ (assuming 32 bytes for each of the N commitments) to $O(32 + 8N)$ (assuming 32 bytes for the accumulator's digest and 8 bytes for storing two unsigned 128-bit integers used as identifiers). Future research can involve optimizing these mapping schemes.

2.1.3 Problem Overview. While joining multiple commitments into accumulators is trivial for a single user and can be done off-chain with no trust assumptions, multiple users must either mutually trust each other or introduce a blockchain to join all commitments into a single accumulator. Mutual trust is dangerous because if the trust is broken, this can result in severe consequences such as manipulating an application's outcome. On the other hand, when a blockchain is introduced, all contents that are joined into the accumulator must also be stored in a permanent manner on the blockchain. Thus, this defeats the purpose of our idea of joining commitments into accumulators off-chain first to save storage. While this can be outsourced to a less secure blockchain with more storage available, this results in sacrificing security, users needing to trust an alternate set of validating nodes, and still uses up storage on this alternative blockchain.

As a result, our idea can not realize its potential because it achieves the best results as more commitments are combined within every accumulator. In the next section, we will outline our approach to solving this joint accumulator dilemma.

By enabling this, we allow A applications to all use the same accumulator and on-chain fixed-length digest. This can drastically decrease the on-chain storage requirements to $O(32)$ for commitments which do not need to be explicitly known and $O(32 + AX)$ for commitments which do need to be explicitly known. As a result of this increased scalability, this eliminates the need to use less secure blockchains for these applications.

2.2 Other Applications

In addition to optimizing HPoC applications, having the ability to construct joint accumulators is beneficial to many other on-chain applications. In this section, we dive further into a couple examples.

2.2.1 File Hashes. Many blockchain-based tokens use images as a visual representation to display their tokens. Images files are large and storage-intensive, so storing them on the blockchain is not a realistic option in practice. Currently, the most popular method is to store the image file using a permanent file storage solution like IPFS or Arweave, and then, the permanent hash of the file is stored directly along with the token on-chain [11]. Because such a hash is collision-resistant, only the image can map to this hash.

However, this image is typically never used on-chain in any computation and only used when querying the token. Thus, storing a 46-byte hash (as used by IPFS) for every digital token is incredibly redundant and unnecessary. By combining all the hashes into an accumulator and using a similar lightweight mapping such as explained with the HPoC applications, we can greatly optimize the storage of these hashes from 46-bytes per token down to a few bytes on average per token.

2.2.2 Privacy-Preserving Timestamps. For applications which require a commitment to some information without ever revealing its timestamp, joint accumulators can be incredibly useful. An example application is not revealing when a student submitted a test because that may reveal if late penalties were applied. Or, one can verifiably timestamp an important legal document while keeping the timestamp and document contents private.

If commitments can be added to the joint accumulator in an anonymous and confidential manner (as we will show in our proposed system), this enables privacy-preserving timestamps for such commitments (which is not currently possible on existing blockchains). When the timestamp needs to be proven, one simply needs to prove that the target commitment's preimage maps to a valid on-chain (digest, timestamp) pair. The timestamp can be privately but verifiably used in verifiable computations (either on or off chain) using cryptographic techniques like zero-knowledge proofs. For example, a student's exact exam submission can be first mapped to an on-chain (digest, timestamp) pair and then, the timestamp can be used in calculation of their grade without ever revealing the exact timestamp or if late penalties were applied. This is possible due to the blockchain's tamper-proofness and verifiability.

3 DESIGN

In this section, we propose our approach to enables users to combine multiple inputs into a joint accumulator in a decentralized fashion and post the fixed-length digest on-chain while outsourcing most of the storage overhead of the accumulators to the users. While this can potentially be done using an alternative blockchain (similar to a Layer-2 rollup), this requires applications to trust a second decentralized network with an alternative set of validating nodes. Additionally, this requires the accumulator's contents to be permanently stored on this alternative blockchain, using up a large amount of this chain's resources. Our system can be built on top of any existing blockchain and use the same set of validating nodes which eliminates the need to split such trust. This can also be done in a way that only uses up a trivial amount of any network's resources and does not add any additional trust assumptions that are not already common practice today. We design our system to only temporarily store the accumulators while users locally store what they need. Because most of the storage of the accumulators is outsourced to the users, this does not permanently burden the network, as would be the case if it was stored directly on the blockchain. To our knowledge, there has been no existing research which has demonstrated such a system.

3.1 Overview

Our system is to be used in parallel with some "main" blockchain (the one which the accumulator is to be posted). All validating nodes of the main blockchain will also run our system which joins users' commitments into a joint accumulator. A generated accumulator is to be posted on the main blockchain once per block via being stored directly in each block's header, exactly how the Merkle root of all transactions is stored in a blockchain's block header.

For the implementation, we have chosen Merkle trees as our choice of cryptographic accumulator for their efficiency, augmentability, and simplicity. But, this can be replaced with any cryptographic accumulator [17].

Our approach does not have any financial-based incentive mechanisms to prevent spam like typical blockchain transactions do, but we will show how this will not be a problem in practice and is similar to existing systems put in place already. While we believe our proposed solution is the most applicable in practice, we note it is not the only solution. In a later section, we will show some potential alternative approaches which can be used and show the pros / cons of each approach.

3.2 Algorithm

To demonstrate our approach, we will explain chronologically through the execution flow starting with a user generating their commitment until it is stored on-chain. For reference, see Figure 1. For explanation purposes, we split our algorithm into six stages. We will call the user Bob, the information he wants to commit to I_B, and his cryptographic commitment to that information CM_B (e.g. $Hash(I_B) = CM_B$). I_B is to be kept confidential by Bob at the minimum until our algorithm is complete and potentially forever depending on what it is being used for.

3.2.1 Stage 1: Commitment Generation. First, Bob will generate his commitment, CM_B, locally. The commitment CM_B is to be publicly known, but I_B should only be known by Bob. If Bob doesn't even want CM_B to be public, he can optionally doubly nest his commitment, (e.g. $Hash(CM_B) = CM'_B$).

3.2.2 Stage 2: Node Selection. Next, Bob will either run his own validating node in our system, N_B, or he can select a validating blockchain node N_A from the set of all blockchain nodes {N_1, N_2, ..., N_N} which he minimally trusts. The trust assumptions with a minimally trusted node are explained later in this paper. There is no trust assumption if Bob runs his own node.

For convenience, we will say Bob selected a node which we will denote N_A. Bob will send CM_B to N_A. No other information besides CM_B is sent like a pseudonym, digital signature, or fees. Bob should also use privacy-preserving tools like Tor to maintain anonymity, so this node can learn nothing even from the network level.

3.2.3 Stage 3: Local Merkle Tree Generation. Each node will maintain a local Merkle tree constructed using all commitments

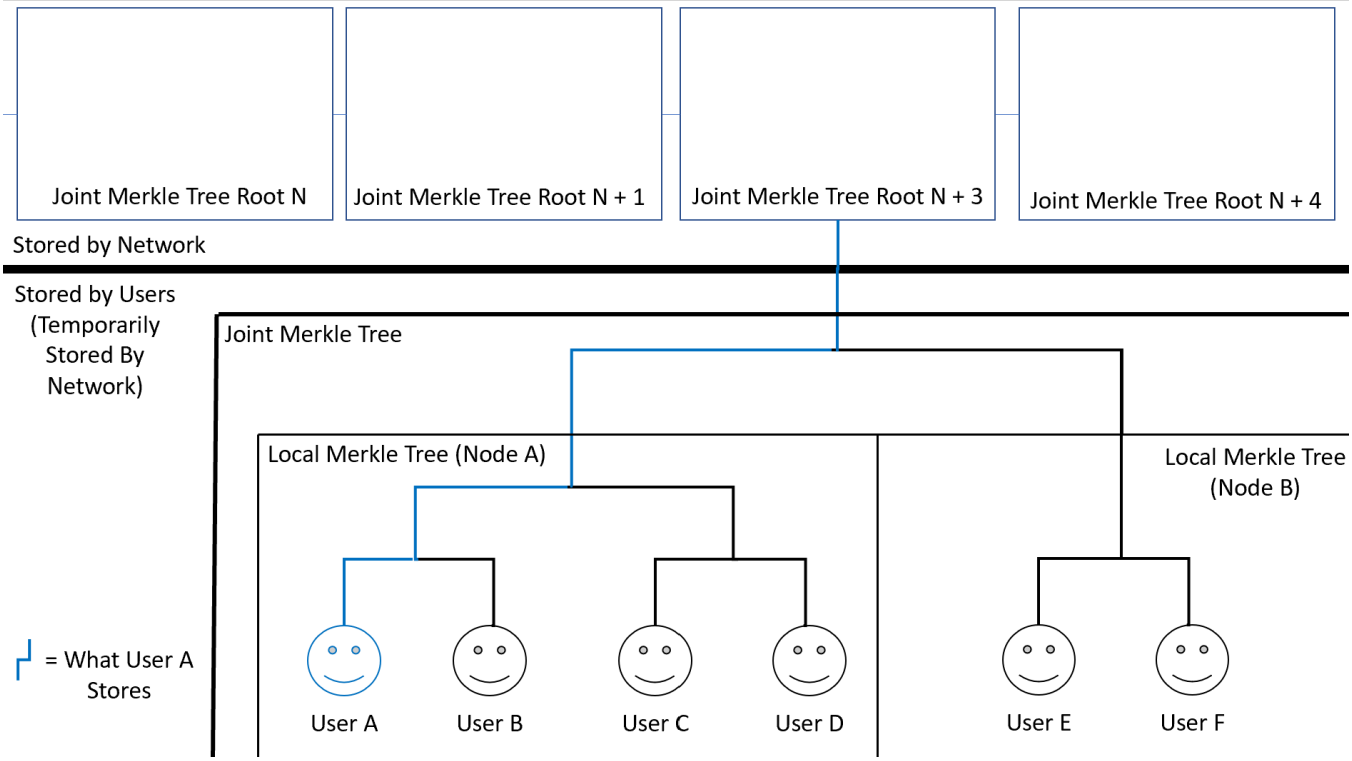

Figure 1: System architecture that demonstrates our approach, specifically the difference between local Merkle trees / roots (LMTs / LMRs) and joint Merkle trees / roots (JMTs/ JMRs).

submitted to them. So, N_A will have a local Merkle tree which includes CM_B and any other commitments sent to them. We will call these local Merkle trees maintained by each node, LMTs, and the local Merkle roots of these trees, LMRs. This can be seen in Figure 1.

3.2.4 Stage 4: Broadcasting the Root of the Local Merkle Tree. Periodically, each node will broadcast their LMR to the network along with a signature and nonce that is unique to that node, $(LocalMerkleRoot_X,\ Sig_X(LocalMerkleRoot_X, Nonce_X))$. At this point, the LMR and LMT is finalized and cannot be edited. At this point, in addition to broadcasting the LMR to the network, the broadcasting node will also relay the LMT contents back to the users (such as Bob) whose commitments are stored somewhere in the LMT.

Bob will now have a valid Merkle path from CM_B to LMR. Once all such users have received the LMT contents, the LMT can be safely discarded by the node to save storage. Although, we recommend nodes to store this temporarily for some additional length of time, such as a day.

The broadcasting of the LMR to the network is done in the same way that a typical blockchain transaction is broadcasted in a peer-to-peer manner, but these broadcasted LMRs will be treated differently than normal transactions. They should have a separate mempool (which we will call LMR mempool) from "normal" transactions. Whenever a node receives a new (LMR, signature, nonce) pair from another node, they will add it to their LMR mempool.

3.2.5 Stage 5: Block Producing. Whenever a validating node is chosen to produce a block, they will join all the LMRs currently stored in their local mempool into a single joint Merkle tree (JMT), as seen in Figure 1. The joint Merkle root (JMR) of this JMT will then be stored within the 32-byte slot in the block header, thus making it permanently stored on-chain.

The JMT and all its contents will then be propagated back to all other nodes. When each node receives the JMT contents, they can then compare it to their local LMR mempool and remove all the LMRs included in the JMT. At this point, they will also verify the JMT is well-formed and only consists of valid LMRs signed by other nodes with valid nonces. We also propose to set an upper limit on the JMT size and check this to prevent the attack where the block producing node makes the JMT super large to attempt to clog the network. If these checks fail, the block is to not be accepted.

Users such as Bob can now query these JMT contents via any node to find the JMT which includes the LMR which holds their commitment. Once found, Bob can now construct and locally store the valid Merkle path all the way from CM_B to the JMR posted on-chain. Thus, Bob's commitment is now stored in a verifiable and tamper-proof manner on the blockchain. This Merkle path is to be stored by Bob for as long as needed. For reference, see the blue path for User A in Figure 1.

3.2.6 Stage 6: Discardable Storage. Up until now, our algorithm has practically the same timeline as a typical transaction being

submitted and broadcasted to the network. However, our system achieves great benefits in this stage.

The JMT contents are recommended to be stored by nodes for a long enough predefined time period (such as a day) to allow for all users to locally store their augmented Merkle path. But after such a time period, nodes will permanently discard all JMT contents (except the JMR which is already on-chain) to continue operating in a lightweight manner.

Note that this is permanently discarding and not just pruning (i.e. where it is discarded by some nodes but is still required to be stored by archive nodes in order to reconstruct the blockchain verifiably from genesis). This is okay because if the JMR is not discarded, none of the users' verifiability is sacrificed. Additionally, since the contents of the JMT are never used in any on-chain computation and users who need their Merkle paths for their commitments already have them stored, there will be no future need for nodes to store such contents.

As a result of these discards, our system only adds a constant of storage overhead to the network because over time, almost all the storage is outsourced to the users storing it locally, and all that needs to be stored by the network is the temporary storage.

3.3 Threats

In this section, we outline the potential threats and security of such a system and why we designed it in the way it is.

3.3.1 Doubly Storing. To start, let's explain some concepts that are unique to our approach. We take the unique approach of not caring if commitments are doubly stored in the rare case in order to achieve a much higher throughput and efficiency. All we care about is that a user ends up with a valid verifiable Merkle path to the on-chain JMR for their respective commitments. If not being doubly stored is an important requirement for an application, an alternative system must be used, potentially one in Section 3.5.

We do not care about double storing because there is no extra overhead added if it does happen. Only the 32-byte JMT will be stored in the long run, and there is no financial penalty like a gas fee in our approach for the user if it is added multiple times.

3.3.2 Learning Information from Pending Commitments. First, there is nothing to be learned by any party who knows some binding commitment from a user before it is added to the chain (i.e. pending). We assume that the preimage to the binding commitment is kept confidential by the user, at the minimum until our algorithm is complete. This also includes potentially applying randomness to the commitment so that the preimage can not be brute forced. Because the commitments are binding, their preimages kept confidential, and having no identity or pseudonyms attached. there is nothing to be learned from knowing some commitment value before it is posted on-chain. While network traffic can be potentially analyzed to learn information about who submitted a commitment, our algorithm uses privacy-preserving tools like Tor to keep this anonymous and confidential.

3.3.3 Minimally Trusting a Node. As explained in the previous section, users have the option to run their own node or minimally trust a node. Running one's own node has no additional trust assumptions when using our system. Minimally trusting a node requires trusting the node for availability and not denying service. We define two ways to deny service in our context: a) simply ignoring some received information or b) editing the received information.

If a minimally trusted node denies service to a user, the user can simply submit to another minimally trusted node or run their own node. Nothing malicious can ultimately happen in this case. If the node refused to broadcast the user's commitment, we have already shown how there is no information that can be learned. If the node edits the user's commitment, there is no financial penalty for the user, and they can simply resubmit. This is practically the same as refusing to broadcast. While a minor inconvenience in the rare case, this trust relationship is common practice in existing blockchain settings today, and running your own node with no trust assumptions is always a possibility.

Other trust assumptions for minimally trusting a node includes that they actually construct the LMT correctly. If they do not, this is again practically the same as refusing to broadcast. Or, if the node broadcasts the LMR but refuses to relay the LMT contents back to the users, this is again practically the same. This will be caught early in the process, and users can simply resubmit.

3.3.4 Trusting the Block Producing Node. Now, let's say the user's commitment was properly broadcasted to the network as a LMR. All nodes will eventually receive this LMR as it is propagates in a peer-to-peer fashion.

The next trust assumption we will look at is the block producing node. If the block producing node chooses to exclude or edit any LMRs it has received when constructing the JMT, this ultimately again does not matter. Other nodes will still have such an LMR in their local mempool, and it can be trivially added in the next block by the next block producing node. This again is common practice in existing blockchain settings. Block producers choose which transactions to include and which to exclude, and the excluded ones will eventually be added in subsequent blocks.

3.3.5 Discarding Storage Too Early. Lastly, the JMT and LMT contents are to be discarded after a predefined time period by nodes in our system. If the contents are discarded too early before a user gets the chance to store locally, the user's augmented Merkle path is broken and cannot be verified. We believe one day is long enough to allow the users enough time to store their Merkle paths locally. If users still have not stored their Merkle paths locally after a day, we believe this is the user's fault and not our system's. The user will then have to repeat the process.

3.4 Spam Prevention

Our system has no inbuilt spam prevention mechanism like typical blockchains do with transaction fees. With typical blockchain transactions, each transaction has a fee associated with it, and nodes can individually set a threshold of a minimum fee to recognize and relay to other nodes. This prevents the attack of spamming $0 no-op transactions being propagated throughout the network. However, this does not prevent such transactions from being spammed to a node in the first place; it only prevents the propagation.

Our system is similarly prone to spam submitting commitments to a single node launching a DoS (denial-of-service) attack. However, to bring down the whole decentralized blockchain network, one would need to attack every node individually which is very costly and very computationally infeasible in practice. They would need to bring down every node individually because in our solution, only the LMR of each node is broadcasted to the network. So for example, regardless if an attacker spammed 10,000 commitments or only 1 commitment to a node, only a single LMR (a 32-byte hash) will be propagated.

Next, let's look at the attack where a node spams the network with lots of new LMRs in an attempt to flood the network. In our system, we accompany each LMR with the node's signature and nonce. What we propose is a threshold of only five (can be customized) LMRs from any single node that can be in their local mempool at any given time. If they receive any more, they are to be ignored and not propagated, similar to how transactions with $0 gas fee may not be propagated. This is because in practice, Merkle trees can be joined, and we do not care about doubly storing information. So, any honest node who has multiple LMTs can simply join them and submit it as a single LMR, instead of submitting multiple. This serves the same purpose as not propagating $0 fee transactions across the network.

The final attack we will explore is an attacker running a large number of unique nodes to get around the above upper limit requirement. First, it is very costly to run a large enough number of nodes to be able to clog the network due to the time and space efficiency of Merkle trees and creating a JMT every block. Second, this can be protected against using other methods. While out of scope for this paper, an initial idea for a proof-of-stake consensus model is to weight each node's LMR by their amount staked in the network. When the queue overflows, the LMRs from the nodes with the least amount staked are ignored first.

3.5 Alternative Approaches

While we believe our explained approach above is most applicable in practice, there are definitely alternative approaches. In this section, we will explore two other approaches and compare them to our approach.

The first approach is the smart contract approach. In the smart contract approach, instead of storing the joint Merkle root in the block header, anyone can append a Merkle root at any time by interacting with a smart contract. The joining process is to be done in an off-chain manner, but such a process involves additional trust assumptions in other parties (not validating nodes). This approach also may run into problems with regards to who pays the transaction fee and how the Merkle tree contents are stored.

The second approach is to use a Layer-2 blockchain. With a Layer-2 blockchain, users will submit their commitments to the Layer-2 blockchain, and periodically, the roots will be posted to the main blockchain. With such a system, this is more verifiable as this blockchain can permanently store all users' commitments and their Merkle paths to the on-chain posted one. This increases the verifiability of the whole joining process instead of relying on performing this in an off-chain manner. Additionally, a native incentive and spam prevention mechanism can be worked in such as gas fees for every submitted commitment. Although, this approach will not scale as well for high volume and high storage needs. Also, it involves trusting an alternative decentralized blockchain system which was a main problem we identified for this paper.

While these solutions are acceptable alternatives, our goal was to achieve construction of a joint accumulator secured by the same set of blockchain validating nodes and only using a constant amount of the network's storage resources. We believe our approach is the best approach in practice to achieve this functionality.

4 IMPLEMENTATION

We built a simple blockchain framework with JavaScript to demonstrate and evaluate our approach which can be found on GitHub under trevormil/demo-chain. In all, everything was about 450 lines of code.

We tested our system assuming the blockchain has 7,187 nodes and a block time of 12 seconds, which are the current numbers for Ethereum mainnet [15]. For evaluation purposes, we assume that users will submit their commitments to nodes in a perfectly load balanced way. We assume that each node broadcasts one LMR to the network per block on average (although, we envision this will probably be much, much lower on average in practice).

We use SHA-256 for our Merkle tree hashing algorithm, ECDSA signatures (65 bytes), and a uint256 (32 bytes) for the nonce. So in all, each LMR broadcast will be 129 bytes. The LMT and JMT will only consist of the SHA-256 hashes, but the signature and nonce are needed to accompany the Merkle leaves in the JMT to make sure everything is well-formed.

We evaluated this on a laptop with a 4-core Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 16 GB RAM, and a 512 GB hard drive.

5 EVALUATION

Our solution's optimization performance is variable based on the number of applications who use such a joint cryptographic accumulator. Thus, we will not be able to provide exact evaluation metrics if our solution is not used in practice yet, but we can show how it will perform based on some estimations.

5.1 Storage Saved

Our solution adds a constant 32-bytes of on-chain storage per block. For convenience, we will assume ~12 second block time (average for Ethereum), 32-byte commitments, and 32-byte Merkle roots. As explained in Section 2.1.1, there are two categories of commitments: a) commitments that need to be explicitly known and b) those that do not. As a refresher, commitments that need to be explicitly known are for applications where the actual value of a users' commitment must be queryable so that they can not change it (i.e. not changing your input commitment in a game of rock-paper-scissors). For these applications, we achieve optimizations from $O(32N)$ to $O(32 + 8N)$ where it is assumed 8-bytes for a lightweight mapping and N is the total number of commitments per block (see Section 2.1.2). This means that we achieve positive scaling results if $> 4/3$ commitments are made on average per block. In other words, as the number of commitments per block grows to infinity, we are only adding 8-bytes per commitment instead of 32-bytes. This is shown in Figure 1.

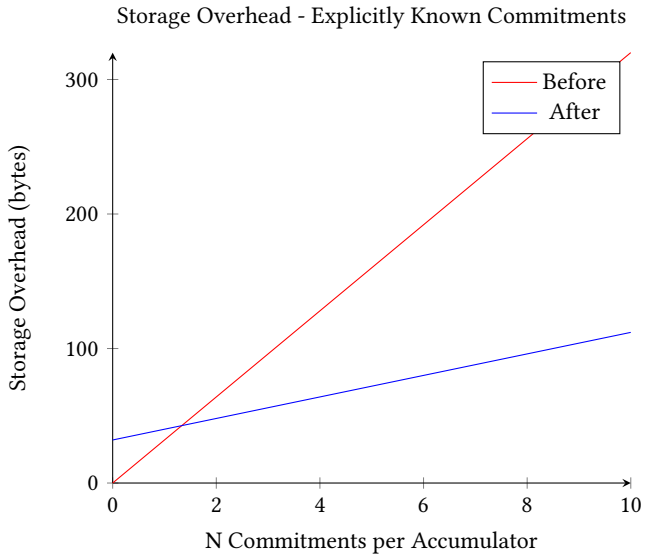

Figure 2: Storage optimizations for explicitly known commitments on the main blockchain.

For applications with commitments that do not need to be explicitly known, we achieve optimizations from $O(32N)$ to $O(32)$ where N is the total number of commitments per block (see Section 2.1.2). This means that we achieve positive results if > 1 commitments are made per block. And as the number of commitments per block grows, we achieve almost a 32-byte per commitment optimization. In other words, as the number of commitments per block grows, we are not adding anything per commitment instead of 32-bytes each time. This is shown in Figure 2.

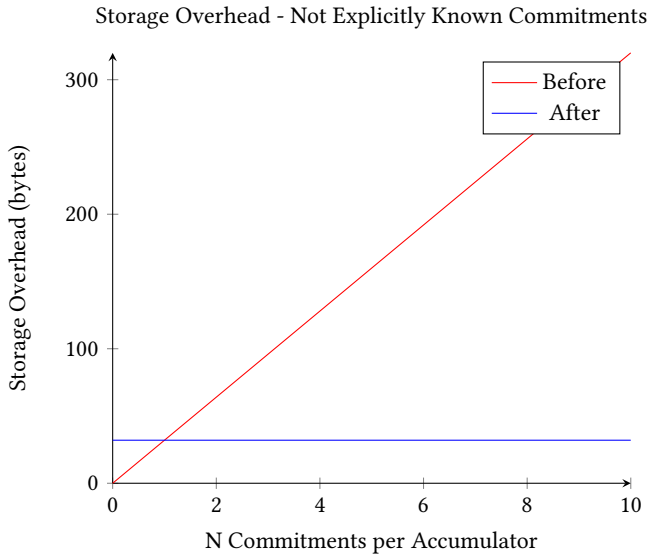

Figure 3: Storage optimizations for explicitly known commitments on the main blockchain.

With a block time of 12 seconds and one 32 byte Merkle root added per block, our approach would take 11.88 years to add 1 GB of storage to the "main" blockchain. For comparison, Ethereum grew 1.65 gigabytes in a day from June 14, 2022 to June 15, 2022 [26].

Table 1 shows the storage saved when X explicitly known commitments are spread across Y accumulators (or Y blocks). Table 2 shows the storage saved when X non-explicitly known commitments are spread across Y accumulators (or Y blocks).

5.2 NFT File Hashes

For an evaluation of its performance in practice, let's see how much it optimizes if all current digital token file hashes used our approach. As of June 2022, there were 12,359,778 NFTs on the Ethereum blockchain, and about 6,124,261 or 49.55% of them used IPFS 46-byte file hashes stored on-chain which point to large files stored off-chain on IPFS like images [11]. If these file hashes were efficiently represented in accumulators and stored via a lightweight mapping (which we will assume 8-bytes), this save about 38-bytes per file hash (46-bytes minus 8-bytes). This could save an additional ~232.721918 megabytes of on-chain storage for digital tokens, if everything was represented in a single accumulator. To break even and start using additional storage, these file hashes must be spread out over ~7,272,560 accumulators. On Ethereum with a ~12 second block time, this means that all NFT files hashes would need to be spread out over a period of 2.7655 years to break even and start gaining extra storage with our approach. And, note these metrics are only if the joint accumulators are explicitly used for NFT IPFS file hashes within these 2.7655 years. Our joint accumulators can be used in a cross-application manner and will achieve further scaling and efficiency as additional applications use them.

5.3 Overhead of Our System

The metrics explained so far only focus on the overhead and storage saved of the "main" blockchain. In this section, we focus on the time and space overhead of our additional proposed system which runs in parallel to the main blockchain. This can be found in Table 3.

As seen in Table 3, the time overhead from tree generation is very efficient, and if implemented in practice, this will have minimal effect on the main blockchain's current operations and is quick enough to be performed and verified during every 12 second block interval. Network propagation may add slight end-to-end delay of a transaction being confirmed. According to [12], block propagation in Bitcoin ranges from ~0.5-2.5 seconds for a 1 MB block size. We will need to propagate ~1-2MB worth of data per block, as seen in Table 2. However, this should not be an issue in practice as existing blockchains have comparable block sizes and block times (e.g. ZCash has a 2MB block limit with 75 second block times [8]).

If we assume that the LMTs and JMTs are stored temporarily for a day and discarded afterwards, the total storage added is in the range of ~8-14 GB. While not trivial, this is a constant storage cost and does not accumulate over time because it acts as a first in first out queue; it can be thought of as a one-time cost addition. Because this cost is one-time, we can potentially save a large amount of overhead as shown in Tables 1 and 2 with only adding a one-time

Storage Overhead Saved - Explicitly Known Commitments									
	Number of Accumulators (or Blocks)								
Commitments	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
10^2	-0.8KB								
10^3	20.8KB	-8KB							
10^4	236.8KB	208KB	-80KB						
10^5	2.396 MB	2.368MB	2.08MB	-800KB					
10^6	~24 MB	23.96 MB	23.68MB	20.8MB	-8MB				
10^7	~240 MB	~240 MB	239.6 MB	236.8MB	208MB	-80MB			
10^8	~2.4 GB	~2.4 GB	~2.4 GB	2.396 GB	2.368GB	2.08GB	-800MB		
10^9	~24 GB	~24 GB	~24 GB	~24 GB	23.96 GB	23.68GB	20.8GB	-8GB	
10^{10}	~240 GB	~240 GB	~240 GB	~240 GB	~240 GB	239.6 GB	236.8GB	208GB	-80GB

Table 1: Storage saved for explicitly known commitments assuming a 32 byte accumulator and 8 byte lightweight mapping.

Storage Overhead Saved - Not Explicitly Known Commitments									
	Number of Accumulators (or Blocks)								
Commitments	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
10^2	0 KB								
10^3	28.8 KB	0 KB							
10^4	316.8 KB	288 KB	0 KB						
10^5	~3.2 MB	3.168 MB	2.88 MB	0 KB					
10^6	~32 MB	~32 MB	31.68 MB	28.8 MB	0 KB				
10^7	~320 MB	~320 MB	~320 MB	316.8 MB	288 MB	0 KB			
10^8	~3.2 GB	~3.2 GB	~3.2 GB	~3.2 GB	3.16 GB	2.88 GB	0 KB		
10^9	~32 GB	~32 GB	~32 GB	~32 GB	~32 GB	31.68 GB	28.8 GB	0 KB	
10^{10}	~320 GB	~320 GB	~320 GB	~320 GB	~320 GB	~320 GB	316.8 GB	288 GB	0 KB

Table 2: Storage saved for non-explicitly known commitments assuming a 32 byte accumulator.

storage cost overhead and minimal time overhead seen in Table 3, especially as the number of commitments grows larger.

Also, note that we believe these numbers, especially the JMT numbers, to be on the very high-end. In practice, we do not believe that all nodes will have enough volume of submitted commitments to submit a LMR each block on average. And if this is true, our system will add even less overhead.

6 RELATED WORK

6.1 Digital Tokens

Digital tokens on blockchains have been touted to revolutionize many different applications, but every application will have their own implementation requirements. As a result, many different token interfaces have been developed to support such requirements for each application [24]. The goal for such interfaces is to be implemented in a way which uses as little of a blockchain's time and space resources as possible. The most popular interfaces currently are the ERC-721 and ERC-20 for non-fungible and fungible tokens on Ethereum, respectively [6]. Due to their popularity and reach, most existing research has gone into optimizing these interfaces in terms of space and time complexity. For example, Seaport and TinyERC721 have proposed many micro-optimizations to improve implementations and transaction fees for these interfaces [13] [4]. Or, LiftChain attempts to optimize through batching NFT transactions off-chain without using an alternative blockhain and benefitting from the security of the mainnet [2]. Although, these works are trying to optimize within the limits of existing interfaces which as a whole are still very expensive and not scalable (i.e. still costs >$15 per token transfer as shown in the introduction).

Alternative interfaces and scaling techniques with various properties, attributes, and functionality are being proposed which are more suited to specific applications (i.e. Verifiable Credentials, Soulbound Tokens, ERC-1155, etc.) [25] [23] [6]. But from our observations, all these interfaces share one common property: they all rely on a per-application implementation basis. In other words, every application must still individually implement such a token interface, and as a result, this uses up more of the blockchain's time and space resources for every additional application.

What our solution focuses on is optimizing on a cross-application basis. Our optimizations come from focusing on reusing the same blockchain resources like storage for multiple applications without sacrificing key properties like verifiability. This enables us to achieve great scaling results over any existing research because less resources (if any) are additionally used when more applications are implemented. Although, this is only applicable to a specific subset

Time and Space Overhead of Our Parallel System on Average Per Node - 7187 Total Nodes					
	LMR / LMT Generation		JMR / JMT Generation		Total - 1 Day (7200 Blocks)
Commitments per Block	Storage (KB)	TreeGen (ms)	Storage (KB)	TreeGen (ms)	Cumulative Storage (GB)
10^2	0.0004	0.068	1157.33	146.40	8.33
10^3	0.004	0.073	1157.33	146.40	8.33
10^4	0.057	0.752	1157.33	146.40	8.33
10^5	0.89	2.164	1157.33	146.40	8.34
10^6	9.03	7.622	1157.33	146.40	8.40
10^7	89.13	35.956	1157.33	146.40	8.97
10^8	890.66	231.14	1157.33	146.40	14.75

Table 3: Time and storage overhead of our proposed parallel system.

of applications which are able to be use their resources in such a cross-application manner.

6.2 Layer-2 Blockchains and Decentralized Storage Solutions

Our solution utilizes a decentralized blockchain system in parallel with the "main" blockchain in order to save storage for NFTs and scale without sacrificing verifiability and security. Many such multi-chain compatible systems do exist, but, they often operate as a whole separate blockchain or decentralized system with their own set of validating nodes, such as Polygon, Optimism, IPFS, or Arweave [18] [16] [5] [22]. Our system is proposed in parallel using the same set of validating nodes as the "main" blockchain, and thus, users do not have to rely on two separate decentralized networks with different sets of validating nodes.

Works such as StateSnap have identified the same problem and proposed a solution without such trust assumptions for quick retrievable storage for tokens [9]. But, this still adds significant storage overhead to existing networks, where storage is already a scarce resource. Additionally, StateSnap focuses on storage that is retrievable on-chain which is not necessary for many applications, as explained with HPoC applications. To our knowledge, there is no such system which is custom tailored to the creation of joint accumulators in a discardable, temporary fashion like ours does.

7 CONCLUSION

In this paper, we presented a decentralized system to be run in parallel with existing blockchains. This system allows for great storage savings for digital tokens and other applications by focusing on discardable rather than permanent storage for the network and combining multiple commitments into an efficient fixed-length digest cryptographic accumulator. These accumulators are to be stored off-chain by the users who need them instead of permanently burdening the network if stored on-chain. We do this while not sacrificing verifiability and only ever needing to trust the same set of validating nodes as existing blockchains, thus not sacrificing security. As a result of these storage optimizations, this lessens the need for applications to have to settle for implementation on a blockchain which is less secure but has more storage available. Our system also is beneficial in many other ways such as enabling privacy-preserving timestamps. In the future, we believe it is worthwhile for all blockchains to implement a system similar to ours.

REFERENCES

[1] Fran Casino, Thomas K. Dasaklis, and Constantinos Patsakis. 2019. A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and Informatics* 36 (2019), 55–81. https://doi.org/10.1016/j.tele.2018.11.006

[2] Hari Kishore Chaparala, Sai Vineeth Doddala, Ahmad Showail, Abhishek Singh, Samaa Gazzaz, and Faisal Nawab. 2022. LiftChain: A scalable multi-stage NFT transaction protocol. *2022 IEEE International Conference on Blockchain (Blockchain)* (2022). https://doi.org/10.1109/blockchain55522.2022.00057

[3] CoinGecko. 2022. Ethereum Price in USD: ETH Live Price Chart and News. (2022). https://www.coingecko.com/en/coins/ethereum

[4] tajijen.eth (ETH: 0x2dFD38D75f090A40DcFC1dce816B75378ffAaBcB). 2022. TINYERC721: A new standard in NFT gas optimization. (2022). https://mirror.xyz/tajigen.eth/fraoPkDEYf1U5yOmke3SdFny5EnA6fZwiupaWMX3Yeg

[5] Ethereum. 2022. Scaling. (2022). https://ethereum.org/en/developers/docs/scaling/

[6] Ethereum. 2022. Token standards. (2022). https://ethereum.org/en/developers/docs/standards/tokens/

[7] Etherscan. 2022. (2022). https://etherscan.io/gastracker

[8] ZCash FAQ. 2022. Frequently asked questions. (Aug 2022). https://z.cash/support/faq/

[9] Siqi Feng, Wenquan Li, Lanju Kong, Lijin Liu, Fuqi Jin, and Xinping Min. 2022. STATESNAP: A state-aware P2P storage network for blockchain NFT content data. *Algorithms and Architectures for Parallel Processing* (2022), 3–18. https://doi.org/10.1007/978-3-030-95391-1_1

[10] Tharaka Mawanane Hewa, Yining Hu, Madhusanka Liyanage, Salil S. Kanhare, and Mika Ylianttila. 2021. Survey on Blockchain-based smart contracts: Technical Aspects and Future Research. *IEEE Access* 9 (Mar 2021), 87643–87662. https://doi.org/10.1109/access.2021.3068178

[11] Nick Hladek. 2022. How many nfts are actually on the blockchain? (2022). https://www.rightclicksave.com/article/how-many-nfts-are-actually-on-the-blockchain

[12] DSN Kastel. 2022. DSN Bitcoin Monitoring. (2022). https://www.dsn.kastel.kit.edu/bitcoin/

[13] Allie Mack. 2022. Launching Seaport and Saving the community millions in fees. (Jun 2022). https://opensea.io/blog/announcements/launching-seaport-saving-the-community-millions-in-fees/

[14] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). https://bitcoin.org/bitcoin.pdf

[15] Ether Nodes. 2022. Ethereum Mainnet statistics. (2022). https://ethernodes.org/

[16] Optimism. 2022. (2022). https://www.optimism.io/

[17] Ilker Ozcelik, Sai Medury, Justin Broaddus, and Anthony Skjellum. 2021. An Overview of Cryptographic Accumulators. In *Proceedings of the 7th International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science and Technology Publications. https://doi.org/10.5220/0010337806610669

[18] Polygon. 2022. Bring the world to ethereum. (2022). https://polygon.technology/

[19] Ashutosh Ranade and Zaheed Shaikh. 2020. A survey on blockchain technology with Use-cases in governance. *SSRN Electronic Journal* (Apr 2020). https://doi.org/10.2139/ssrn.3568629

[20] Ethereum Name Service. 2022. Ethereum Name Service Domains. (2022). https://ens.domains/

[21] OpenSea Support. 2022. Seaport Gas Costs. (Jun 2022). https://twitter.com/opensea_support/status/1537815479061868545

[22] ThorChain. 2022. ThorChain. (2022). https://thorchain.com/
[23] W3C. 2022. Verifiable credentials data model V1.1. (2022). https://www.w3.org/TR/vc-data-model/
[24] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. 2021. Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges. (2021). https://doi.org/10.48550/ARXIV.2105.07447
[25] Eric Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. 2022. Decentralized society: Finding web3's soul. *SSRN Electronic Journal* (2022). https://doi.org/10.2139/ssrn.4105763
[26] YCharts. 2022. Ethereum Chain Size. (2022). https://ycharts.com/indicators/ethereum_chain_full_sync_data_size