

Efficient patient-centric EMR sharing block tree

Xiaohan Hu^{1,*}, Jyoti Sahni^{1,*}, Colin R. Simpson^{1,2,*}, Normalia Samian³, Winston K.G. Seah^{1,*}

¹School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

²Usher Institute, College of Medicine and Veterinary Medicine, The University of Edinburgh, Edinburgh, United Kingdom

³Department Of Communication Technology And Network, Universiti Putra Malaysia, Selangor Darul Ehsan, Malaysia

xiaohan.hu@vuw.ac.nz, jyoti.sahni@vuw.ac.nz, colin.simpson@vuw.ac.nz, winston.seah@vuw.ac.nz, normalia@upm.edu.my

*corresponding author

Abstract—Flexible sharing of electronic medical records (EMRs) is an urgent need in healthcare, as fragmented storage creates EMR management complexity for both practitioners and patients. Blockchain has emerged as a promising solution to address the limitations of centralized EMR systems regarding interoperability, data ownership, and trust concerns. Whilst its healthcare implementation continues to face scalability challenges, particularly in uploading lag time as EMR volumes increase. In this paper, we describe the design of a novel blockchain-based data structure, MedBlockTree, which aims to solve the scalability issue in blockchain-based EMR systems, particularly low block throughput and patient awareness. MedBlockTree leverages a chameleon hash function to generate collision blocks for existing patients and expand a single chain into a growing block tree with n branches that are capable of processing n new blocks in a single consensus round. We also introduce the EnhancedPro consensus algorithm to manage multiple branches and maintain network consistency. Our comprehensive simulation evaluates performance across four dimensions: branch number, worker number, collision rate, and network latency. Comparative analysis against a traditional blockchain-based EMR system demonstrates outstanding throughput improvements across all dimensions, achieving processing speeds $\nu \cdot n$ times faster than conventional approaches.

Keywords—EMR, Blockchain, Chameleon hash, Throughput, Scalability

1. INTRODUCTION

Electronic medical records (EMRs) have revolutionized healthcare delivery by enhancing productivity, quality, and accessibility of services [26]. However, challenges persist in EMR management, particularly concerning data ownership, privacy, and interoperability. Current healthcare institutions usually maintain EMRs in their centralized databases, where healthcare providers retain control over patient data (although patients may be able to access their EMRs). Hence, the ownership, integrity, and sharing flexibility cannot be guaranteed [11]. Moreover, fragmented storage across institutions further complicates cross-institutional patient care and impedes scientific health data research that requires patient consent. Additionally, EMR uploading latency within healthcare systems consistently triggers public dissatisfaction, especially for patients who need cross-institutional medical care [3].

Numerous policies have been announced to regulate EMR management. For example, the Health Insurance Portability and Accountability Act (HIPAA) [1] and the EU General Data Protection Regulation (GDPR) [9] have established standards for data privacy and patient access. The 21st Century Cures Act [14] further emphasizes interoperability and patient-centric data control. These policies highlight the need for technological solutions that meet regulatory requirements and operational efficiency in healthcare delivery.

Blockchain is considered the preferable option to advance current EMR systems due to its decentralization, traceability, and tamper-proof properties [25]. The fundamental concept is uploading the EMRs' relevant data into the blockchain to facilitate EMR sharing efficiency and usability. Since blockchain ledger is maintained by every system participant, EMRs can be easily shared, whilst protecting the records from unilateral tampering and data loss caused by single-point of failure [2]. However, most blockchain-based solutions target financial industries. Its application in medical data management is still in the research phase, facing several open challenges: (1) scalability limitations (block throughput and decentralization); (2) regulatory compliance (security and ownership); (3) integration with legacy healthcare IT infrastructures (flexible sharing) [4].

In this paper, we propose MedBlockTree, a novel blockchain-based data structure designed to solve the aforementioned challenges in EMR systems. Our main contributions are summarized as follows:

- 1) **MedBlockTree**: A new block tree structure that builds upon the blockchain concept and incorporates the chameleon hash function [15], [17]. MedBlockTree significantly improves the on-chain data processing speed, blocks per second (BPS), by concurrently processing multiple EMR blocks in one consensus round. The reduction in uploading latency considerably improves system efficiency.
- 2) **EnhancedPro consensus**: A new consensus protocol called EnhancedPro is introduced to process multiple winner elections in one consensus round and ensure network consistency. Workers can elect the blocks' proposers autonomously. The recognized randomness of every consensus round improves the fairness and unpredictability of winners' elections.
- 3) **Patients' awareness**: Newly added EMRs are acknowledged by patients since a collision block's generation relies on the patient's secret key. This key is only held by

patients, hence the patients' awareness is strongly ensured. There will not be disputes regarding the generation of EMRs.

The remainder of this paper is structured as follows. Section 2 reviews the current research on EMR sharing and indicates the research gaps. The preliminaries are introduced in Section 3, followed by the MedBlockTree model and EnhancedPro protocol in Section 4. Section 5 reports the performance analysis based on evaluation results. The paper is concluded in Section 6, with a discussion of potential future research.

2. RELATED WORKS

Blockchain adoption in healthcare has been widely studied. This section reviews prior work and highlights the existing gaps.

2.1 Blockchain-based EMR sharing

The classical design of a blockchain-based EMR sharing system is directly uploading EMRs to blockchain, by leveraging blockchain's distributed and tamper-proof characteristics to facilitate secure sharing and protecting the overall records. For instance, Johari et al. [12] proposed a Proof of Work (PoW) based medical data sharing framework to enhance the security and traceability of EMRs. However, the PoW consensus is known for its high resource consumption. Besides, timely on-chain updating with large-sized data is a challenge, since the synchronization of new blocks among participants will be time-consuming, potentially resulting in a long uploading latency [24].

An advanced design combines the existing off-chain database to decrease the EMR uploading latency. SPChain [30] employed a hybrid storage approach where EMR metadata is stored on-chain while original records remain in hospital databases. Proof-of-Reputation (PoR) is utilized in SPChain to replace energy-intensive PoW, ensuring efficient consensus while incentivizing honest participation. Its cross-institutional EMR sharing is achieved through proxy re-encryption, ensuring privacy-preserving data access control. Li et al. [18] uploaded the hashes of the EMRs on a blockchain to create an EMR address ledger, thereby reducing the latency associated with data uploads. The actual EMR data is encrypted and stored in a distributed database, accessible only via smart contracts, which allows patients to retain full control over their data. Similarly, Ma et al. [19] integrated blockchain and InterPlanetary File System (IPFS) [5] to increase blockchain efficiency. EMRs' hash addresses and smart contracts are stored on the index chain to enhance retrieval, while the raw records are maintained in an IPFS constructed by the hospitals.

2.2 Blockchain scalability in healthcare

Blockchain ensures network consistency among the participants by avoiding bifurcations since the forks will easily incur record disputes and reduce system security, particularly in financial areas [23]. In healthcare, data uploading efficiency is critical, especially in emergencies where timely access to

TABLE I: Comparison of blockchain relevant designs

Designs	BPS	Secure	Flexible	Ownership	Emergency
Blockchain	+	✓	✓	✓	✗
Plus off-chain	++	✓	✓	✓	✗
Sharding	+++	✓	✗	-	✗
DAG	++++	✗	-	-	✗
MedBlockTree	++++	✓	✓	✓	✓

+ : Throughput, ✓: Supported, ✗: Unsupported, - : Not applicable

patient information is essential for making rapid, informed decisions.

Examining the previously discussed designs, EMR-relevant data in blockchain-based healthcare systems must first enter a data pool to await processing, hence ensuring the chronological order of EMRs. However, as the institutions expand and patient numbers increase, the waiting queue in this data pool grows substantially. This expansion directly degrades system efficiency, creating processing delays that compromise the performance of time-critical medical applications. The linear processing constraint becomes especially problematic in blockchain applications. Hence, a design that maintains the core characteristics of blockchain while enhancing system efficiency is needed to offer a viable solution to the bottleneck. Sharding and Directed Acyclic Graphs (DAGs) [28] are two popular approaches to solve the low throughput issue in conventional blockchain [22]. Sharding divides the blockchain network into multiple groups, each capable of processing transactions independently. This method improves throughput by enabling intra-shard parallelism. However, sharding introduces complex cross-shard communication protocols, which can lead to latency and consistency issues [13]. Besides, independent ledgers among different shards will also introduce extra risk for global record updates, for instance, record sequential and cross-shard records combination, which makes it not suitable in healthcare. In DAG-based structures [16], [27], transactions are arranged in a graph where multiple blocks can be confirmed concurrently. This design enhances throughput by checking whether the calculated transaction has enough relevant former parent transactions. Nevertheless, DAG is suitable for the domains where transaction confirmation depends on cumulative network activity rather than deterministic consensus [21]. In healthcare, this will compromise the records' sequence and auditability.

Furthermore, both solutions are primarily tailored to financial applications, where eventual consistency and minor delays are tolerable. In contrast, EMR systems demand deterministic finality, strict sequentiality, and the ability to prioritize critical patient data in emergencies. These medical-specific requirements remain largely unaddressed by existing scalability solutions, indicating a critical gap in adapting blockchain for healthcare use cases. Table I exhibits the comparison of the above works regarding the EMR sharing requirements.

3. PRELIMINARIES

This section outlines the underlying algorithmic concepts deployed in MedBlockTree.

3.1 Chameleon hash function

Adi Shamir and Yael Tauman first defined the chameleon hash algorithm [17]. The proposed algorithm is a cryptographic primitive that enables the signers to create an undeniable digital signature for a specific target. With the secret key held by the signers, the signature can also be transformed into a signature for different content [15]. The general outline of the chameleon hash function includes the following algorithms:

- 1) *Key generation* $(p_k, t_k) \leftarrow H_{Gen.}(1^k)$. k is the security parameter. p_k is the public key and t_k is the secret key.
- 2) *Hash value* $(h, \zeta) \leftarrow Hash(h_k, m)$. The standard cryptographic hash function maps the input to a fixed-size hash value and a verification string.
- 3) *Verification* $(True, False) \leftarrow Hash_{Verif.}(h_k, m, (h, \zeta))$. $Hash_{Verif.}$ resembles a verification function in which the recipients use the h_k and m to confirm the legality.
- 4) *Collision* $(\zeta') \leftarrow ColliHash(t_k, m', (m, h, \zeta))$. The t_k holder uses t_k and m' to generate the collision for which: $H_{Verif.}(h_k, m, (h, \zeta)) = H_{Verif.}(h_k, m', (h, (\zeta')))$.

3.2 Verifiable Random Function

A Verifiable Random Function (VRF) is a cryptographic construct that produces a pseudorandom output y and an accompanying proof π derived from a given input x and a secret key SK . π allows any verifier, possessing the corresponding public key PK to confirm the correctness of y without disclosing x and SK [20]. This property of verifiable randomness ensures that VRFs provide a reliable and secure mechanism for randomness verification, making them highly applicable in blockchain consensus protocols [10].

4. MEDBLOCKTREE

In this section, we present an overview of the MedBlockTree design, including its core idea, main components (blocks and branches), and the corresponding consensus protocol (EnhancedPro). To address the inefficiencies associated with uploading large-sized data within a single block, we align with the concepts presented by Ma et al. [19] and Li et al. [18], and only record the main index (metadata) in the system. Hence focusing on improving the performance of on-chain data processing.

Each participant (patients, practitioners, consensus workers) must register with the authority and acquire the key pair denoted by (PK_n, SK_n) generated via the chameleon hash Key pair generation algorithm $H_{Gen.}(1^k)$. To clarify the main targets, we assume:

- 1) SK_n is securely held by the owner.
- 2) Metadata is generated based on the EMR regularities compliance, and is easy to understand.
- 3) Raw medical data is stored off-chain in a secure and encrypted manner.

We provide examples involving patients *Alice*, *Bob*, *Claire*, and *Daisy*, along with representative EMR metadata keywords such as *cold*, *Flu*, *COVID* to further illustrate how MedBlockTree manages EMRs.

4.1 Core idea

MedBlockTree targets the efficient processing of on-chain metadata updates. Its fundamental concept is, instead of eliminating the bifurcations, it generates recognized branches by implementing the chameleon hash function in the blockchain's underlying algorithm. Through expanding the one-chain structure into a growing tree, the throughput, blocks per round, is increased from one to n (n is the branch number) and contributes to improving the data processing efficiency. Since the chameleon hash collision can only be found by combining the secret key, which is only held by the holder, the blocks are generated based on awareness, preserving the system's trust and consistency.

4.2 MedBlockTree blocks

In MedBlockTree, there are two types of blocks: (1) New blocks generated by the winners elected through consensus protocols; (2) Collision blocks created by clinic practitioners.

4.2.1 New blocks

Blockchain structure is a strictly arranged sequence in which the blocks are connected by the previous block's hash value. Since the hash calculation is a one-directional operation, data integrity, credibility, immutability, and traceability can be highly assured. In contrast to most blockchain models in which the default hash function is SHA256, MedBlockTree replaces it with an identity-based chameleon hash function [17] to accommodate its unique design.

Similar to SHA256, the chameleon hash algorithm generates a unique hash value, denoted as *Chamehash*, which is collision-resistant without access to a secret key [15]. In addition, the algorithm produces corresponding chameleon check strings, which are essential for validating hash correctness and enabling controlled collision generation.

To distinguish blocks with identical hash values across branches, block indexes in MedBlockTree are designed as a tuple of the branch identifier $B(n)$ and the block sequence number m . Consequently, the block structure in MedBlockTree differs from traditional SHA256-based designs. Figure 1 illustrates the implemented block format.

4.2.2 Collision blocks

Normally, some patients require ongoing treatment, resulting in the generation of new medical records over time. However, when the metadata pool experiences a long waiting queue, newly created EMRs may suffer from significant upload delays due to the blockchain's sequential consensus process, which is unsuitable for urgent care or emergencies.

MedBlockTree introduces the *collision block* concept to support emergency suitable design. A collision block is a specially crafted block that shares the same hash value as a previous block but contains new metadata. As introduced in Section 2, the chameleon hash algorithm enables controlled collision generation. Specifically, the creation of a collision block involves leveraging the chameleon check strings and content from the patient's prior block, in combination with

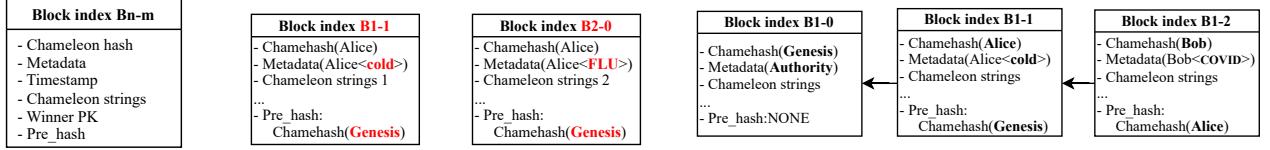


Figure 1: Block structure Figure 2: Alice's former and collision block Figure 3: Example of the default chain $B1$

the patient's private key (SK) and the new metadata. In EMR sharing, one of the primary concerns is data ownership [11]. We assume that if patients are aware of the EMRs awaiting upload, and the corresponding collision blocks are securely and legitimately generated by authorized practitioners, then these records can be considered validated. Consequently, if a returning patient's newly generated metadata is default validated, it can be prioritized and uploaded in the next consensus round. To illustrate this process, we use *Alice* (first patient) and keywords *cold*, *Flu* as an example to show the collision block generation.

Example: Suppose Alice returned to the clinic at a later time. During the treatment, the doctor retrieved Alice's former medical records by $Meta(Alice\langle cold \rangle)$ as a reference, and generated a new EMR with corresponding metadata pack $Meta(Alice\langle Flu \rangle)$. Based on the contents in the former block $Alice\langle cold \rangle$ and her secret key SK_{Alice} , the system can easily find a collision (same hash value different chameleon strings) for the new metadata $Meta(Alice\langle Flu \rangle)$, where:

$$\begin{aligned} Collision(\zeta_2) &\leftarrow \text{ColliHash}(SK_{Alice}, Meta(Alice\langle Flu \rangle), \\ &Meta(Alice\langle cold \rangle), PK_{Alice}, \zeta_1) \end{aligned}$$

All the participants can validate $Collision(\zeta_2)$ by using the verification function $HashVerif$ to confirm:

$$\begin{aligned} \text{HashVerif}(PK_{Alice}, Meta(Alice\langle cold \rangle), (\text{Chamehash}(Alice), \zeta_1)) &= \\ \text{HashVerif}(PK_{Alice}, Meta(Alice\langle Flu \rangle), (\text{Chamehash}(Alice), \zeta_2)) \end{aligned}$$

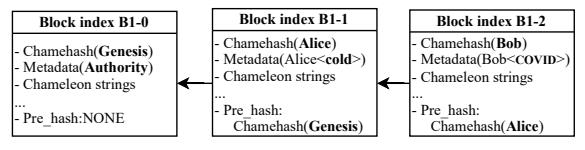
Since the collision block $Alice\langle Flu \rangle$ owns the same hash value as $Alice\langle cold \rangle$ has, and the new block is seen as validated, the pre-hash in block $Alice\langle Flu \rangle$ will also be the hash value in the genesis block (same as block $Alice\langle cold \rangle$). The newly generated block and *Alice*'s former block are shown in Figure 2.

4.3 MedBlockTree branches

MedBlockTree is constructed by the default chain $B1$ and the generated branch $B(n)$, $n > 1$.

4.3.1 Default chain - B1

The default chain generation is similar to the other EMR blockchain sharing systems. In an authorized hospital, patient p_i provides PK_{p_i} to doctors. After treatment, doctors generate medical records Raw_{p_i} of p_i and the corresponding metadata $Meta_{p_i}$. The treatment system packs $Meta_{p_i}$ with PK_{p_i} , doctor PK and the timestamp as a metadata pack, and these assembled packs will be dispatched to the waiting queue, within the metadata pool awaiting to append to the chain. The genesis block is built by the authority as the starting point. And the



following blocks are strictly connected by the previous block's chameleon hash, as shown by the following example.

Example: In the default phase, we assume the sequence in the metadata pool is: $Meta(Alice\langle cold \rangle)$, $Meta(Bob\langle COVID \rangle)$. The participants in MedBlockTree join the consensus competition to win the opportunity to be the block proposer. After two consensus rounds, the default chain $B1$ is extended to three blocks, as shown in Figure 3.

4.3.2 Branch - B(n)

Since patients' collision blocks share the same hash and pre-hash values as their corresponding former blocks, they are positioned at the same level in the chain. This structural overlap forms what is traditionally referred to as a fork in blockchain systems. However, in MedBlockTree, patients are aware of the creation of collision blocks, and those blocks are default as validated. To further enhance the efficiency of metadata processing, those default validated collision blocks are designated as the origin points for new branch growth. Following the given examples of *Alice*'s blocks and default chain, a new branch named $B2$ now has been added to the default chain $B1$, as shown in Figure 4. By analogy, every time an existing patient returns, a new branch will be generated, and a blockchain-based structure will quickly grow into a tree, namely, MedBlockTree.

4.4 EnhancedPro protocol

The consensus protocol is a recognized algorithm that every consensus competition participant needs to comply with to ensure consistency [29]. Various consensus protocols have been proposed for blockchain systems, yet one of the primary objectives is to prevent forks. Therefore, they are not well-suited for the requirements of MedBlockTree, which embraces branching structures.

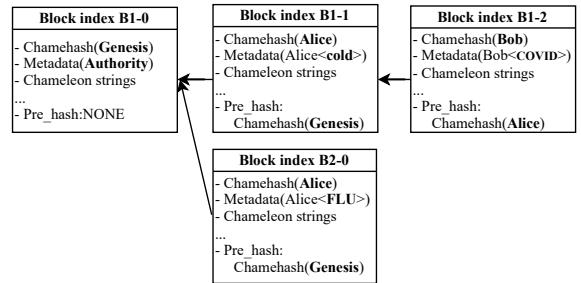


Figure 4: Example of MedBlockTree with two branches

To match the parallel processing of multiple blocks in one consensus round and maintain MedBlockTree network consistency, we build upon an advanced consensus protocol, viz., EnhancedTendermint [6], by refining its underlying algorithm, and designing a multiple winners consensus protocol - EnhancedPro. Similar to EnhancedTendermint, EnhancedPro is also a hybrid of Proof of Stake (PoS) and Byzantine Fault Tolerant (BFT). Consensus workers must undergo a bootstrapping phase (qualification review and deposit tokens) to join the consensus process. EnhancedPro operates in three key stages: (1) PoS-based self-processing, (2) BFT-based network consensus, and (3) Tree update.

4.4.1 Self-processing

Self-processing refers to the period during which each participant autonomously collects round-specific information to conduct winner-election, self-checking, and generate the proposals (only for the winners).

PoS is a consensus mechanism in which workers are selected to propose blocks based on the amount of stake they provide, cryptocurrency or reputation, for instance, promoting energy efficiency without using the computing mining in PoW. However, a critical drawback of PoS is its tendency toward centralization, as nodes with higher token balances have a greater chance of being selected as winners [29].

To improve the unpredictability and fairness of winner elections, we introduced sub-nodes and randomness. It is worth emphasizing that in MedBlockTree, the core mechanism for parallel processing of multiple blocks in one consensus round is that each branch can be seen as a block append point. Hence, a recognized algorithm for estimating winners for each branch also needs to be considered.

Sub-nodes: Given MetaPool $\{meta_0, meta_1, \dots, meta_\iota\}$, workers' public keys $\{PK_{W_0}, PK_{W_1}, \dots, PK_{W_k}\}$, deposit tokens $\{\tau(W_0), \tau(W_1), \dots, \tau(W_K)\}$, and total deposits $\sum(\tau_m)$, the new recognized worker list $\{W_0, W_1, \dots, W_K\}$ is generated through the default sorted configuration in the MedBlockTree system. The sub-nodes and sub-nodes' indexes are defined as follows: sub-nodes $\{\omega_0, \omega_1, \dots, \omega_k\}$, with sub-nodes' indexes $\{0, 1, \dots, k\}$. For example, W_1 deposit two tokens and sub-nodes of W_1 are ω_0 and ω_1 , with indexes 0 and 1.

Randomness: Each worker maintains a former blocks map $\{B1: block_x, B2: block_y, \dots, B(n): block_z\}$ to record the latest blocks for branches. The round randomness is determined by two public parameters: ζ and n . For which ζ is the chameleon check string in the former block, and n is the corresponding branch number.

Winner election: For each branch $B(n)$, workers will perform a module calculation. The result corresponds to the index of the chosen sub-node, which maps to the worker W_k . This W_k will be the block proposer of $B(n)$. As shown in Equation (1).

$$\omega_i = Hash(\zeta, n) \bmod \sum(\tau_m) \quad (1)$$

Furthermore, to prevent malicious workers from broadcasting proposals and compromising network consistency, the VRF is deployed to generate the provable random values, VRF_y and

Label: Pre-vote <True / False, Block Hash, PK(W_i)>	Label: Commit <True / False, Block Hash, PK(W_i)>
---	---

Figure 5: Pre-vote and Commit vote

VRF_π , for winners. The other workers can validate the block by the winners' PKs and the proofs. The overall algorithm of self-processing is shown in Algorithm 1.

Algorithm 1 MedBlockTree Self-processing Algorithm

Input: former blocks map, sub-node indexes, workers' list
Output: winners map= $\{B1: winner_1, B2: winner_2, \dots, B(n): winner_n\}$,
block proposals, VRF_y s, VRF_π s

```

1: New consensus round start:
2: Step 1: Winners election
3: for each branch  $B(n)$  do
4:   Randomness of  $B(n) = \langle (B(n) : block_i : \zeta), n \rangle$ 
5:   winner's sub-node index = Randomness of  $B(n) \bmod \sum(\tau_m)$ 
6: Step 2: Self-checking
7: for each branch  $B(n)$  do
8:   current index (CurInd) = 0
9:   for each worker  $W_i$  do
10:    if winner's sub-node index  $\in [CurInd, CurInd + \tau(W_i))$  then
11:      winner for  $B(n) = W_i$ 
12:      Append map  $\langle B(n) : W_i \rangle$  to winners map
13:      break
14:   current index +=  $\tau(W_i)$ 
15: Step 3: New proposals
16: for each branch  $B(n)$  with meta information  $meta_\iota$  do
17:   if  $n = \iota$  then
18:      $W_i \leftarrow$  winners map[ $B(n)$ ]
19:     block of  $B(n) = \text{createBlock}(winner : W_i, meta_\iota)$ 
20:      $VRF_y \leftarrow \text{VRFProve}(SK(W_i), \text{Randomness of } B(n))$ 
21:      $VRF_\pi \leftarrow \text{VRFProve}(SK(W_i), VRF_y, \text{Randomness of } B(n))$ 
22: return winners map, block proposals,  $VRF_y$ s,  $VRF_\pi$ s

```

4.4.2 Network consensus

We adopt the BFT protocol to enhance efficiency and maintain network consistency. It contains three steps: (1) Winners broadcast the proposals and VRF verification strings they generated through self-processing; (2) Workers validate the proposals through the winner's PK and VRF strings by the verification algorithm in Equation (2), and broadcast pre-vote; (3) Commit voting and the finalization of new blocks. Note that the ultimate *True* of the blocks validation and finalization depends on the voting power of the votes. The worker's voting power equals the tokens deposited, and the requirement is at least 2/3 of $\sum(\tau_m)$ agree with the proposal.

$$\text{Verify}(PK(W_i), VRF_y, \zeta, n, VRF_\pi) \rightarrow \text{True or False} \quad (2)$$

The pre-vote and commit vote structures are shown in Figure 5. New blocks map BlockMap = $\{(B1 : Block_1, B2 : Block_2, \dots, B(n) : Block_n)\}$ will eventually be committed after the network events shown in Figure 6.

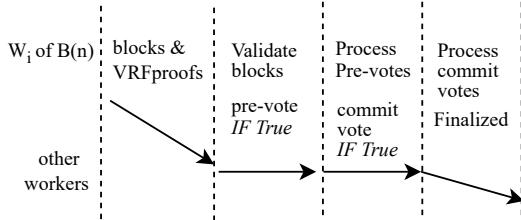


Figure 6: Events in networks

4.4.3 Tree update

Tree updating contains two parts: new blocks appending and collision blocks claiming. The new blocks will be appended to the MedBlockTree according to the BlockMap. The collision blocks created by the practitioners are based on the contents of the patients' previous blocks, as aforementioned. They are initially validated and then uploaded to the data pool to await processing. To support the growth of MedBlockTree, the collisions are updated at the end of each consensus round. Analogously, whenever an existing patient returns, a new branch is generated, enabling the blockchain structure to evolve rapidly into a tree, hence forming the MedBlockTree. This example continues from Example 4, and is provided to illustrate this phase:

Example: Suppose there is a collision block of Bob *Colli(Bob(Flu))*. Daisy and Ella are two new patients with meta packs *Meta(Claire(Flu))* and *Meta(Daisy(COVID))*. In a new consensus round, workers of MedBlockTree propose the blocks of Claire and Daisy based on the Enhanced-Pro consensus protocol and get the final BlockMap = {B1: *Block(Claire(Flu))*, B2: *Block(Daisy(COVID))*}. Following the MedBlockTree we got in Section 4-C2 with two branches: B1 and B2, Block *Claire(Flu)* will be added after Block *Bob(COVID)*, while Block *Daisy(COVID)* will be appended after Block *Alice(Flu)*. As for *Colli(Bob(Flu))*, with the same pre-hash of Block *Bob(COVID)*, it will be added as the starter of a new branch B3. The graphical description of the MedBlockTree is exhibited in Figure 7.

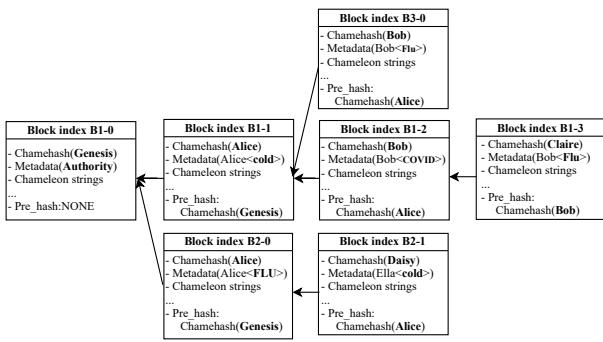


Figure 7: Example of updated MedBlockTree

TABLE II: Simulation environments

Environment	Description
Platform	Catalyst Cloud
Operating System	ubuntu (24.04 64), 2 core, 4 RAM
Network status	100ms / 200ms latency
Data set	2000 generated EMRs
Database	RocksDB, Redis

5. PERFORMANCE ANALYSIS

In this section, we evaluate the scalability of MedBlockTree by comparing its performance with a blockchain system implementing the original Enhanced Tendermint protocol across five key dimensions.

We deploy our nodes on virtual machines hosted by Catalyst Cloud [7], organized into three clusters: patient, doctor, and worker. The worker nodes use sockets to conduct peer-to-peer communication, with network delays configured to simulate real-world conditions. RocksDB [8] is deployed as the worker's database. The doctor and patient clusters are responsible for generating metadata and collision blocks, which are subsequently announced to the worker cluster for processing. All experiments were conducted under identical configurations and network conditions to ensure fair comparison. The simulation configurations are listed in Table II.

5.1 Benchmark test

We first conduct a benchmark test (10000 rounds) to evaluate the time consumption for block formation, generation and upload in the Enhanced Tendermint blockchain using SHA256¹, and in MedBlockTree employing an identity-based chameleon hash [17]. The results are summarized in Table III. SHA256, Chame. and Colli. indicate the time consumption on formatting a SHA256 block, a chameleon hash block, and a collision block respectively. The chameleon hash block generation consumes a bit longer due to its higher computational complexity. Yet we can see with the required information that, collision block generation time indicates a relatively low delay. BC (0.1 s) and BC (0.2 s) depict the average round time for the blockchain-based system with a network latency of 100 ms and 200 ms. DB denotes the average time needed to upload a block into RocksDB.

5.2 D1: Branch numbers

Throughput is a main index to evaluate the system's performance. In this section, we simulate the MedBlockTree with various branch numbers and compare the efficiency, measured in blocks per second (BPS), against that of the

¹ Secure Hash Algorithm 256-bit, which is widely used in blockchain-based systems, such as Bitcoin, Tindermint

TABLE III: Benchmark test (in seconds)

SHA256	Chame.	Colli.	BC (0.1)	BC (0.2)	DB
0.055	0.062	0.027	0.370	0.676	0.011

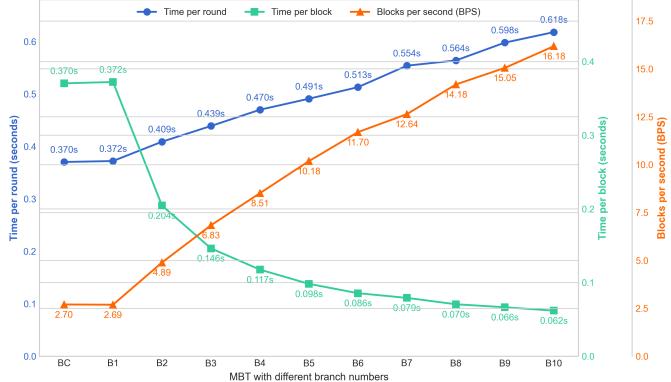


Figure 8: MedBlockTree performance with increasing branch numbers (BC: Enhanced Tendermint blockchain; B1-B10: MedBlockTree with different branch numbers.)

Enhanced Tendermint blockchain. Four workers participate in the consensus competition with a 100 ms network latency.

Figure 8 exhibits the results of a blockchain (BC) and a MedBlockTree with different branch numbers (B1-B10) through three metrics: Time per consensus round (blue), Time per block (green), and Blocks per second (orange). As the number of branches increases, the consensus round time shows a slight increase, reflecting the additional blocks processed in each round in MedBlockTree. However, processing efficiency improves significantly: Time per Block decreases sharply from 0.372 s to 0.062 s, while BC is constantly 0.370 per round. Consequently, the system's throughput exhibits substantial enhancement, with BPS rising from 2.70 in blockchain to 16.18 under the 10-branch configuration (B10) in MedBlockTree. Hence, with the branch number increasing, MedBlockTree can significantly improve the system efficiency, particularly on BPS, with a slight increase in consensus round time.

5.3 D2: Network latency

To clearly compare how network conditions affect performance, we conduct a duration analysis that includes self-processing, network consensus, and database processing times for each design. The results are shown in Figure 9, where the green and blue columns represent network latencies of 100 ms and 200 ms, respectively.

Overall, network latency influences the efficiency of both designs. In MedBlockTree, as the number of branches increases, all three durations exhibit a slight increase in time consumption due to the larger number of tasks processed per round. However, compared to the blockchain design, MedBlockTree leverages the consensus process more effectively by handling multiple blocks per round, thereby achieving significantly higher efficiency.

Additionally, in our implementation, RocksDB appends blocks sequentially, resulting in a linear increase in database update time. If this duration is disregarded (as it falls outside the scope of our research), MedBlockTree demonstrates even greater performance benefits. Notably, the epoch durations of the

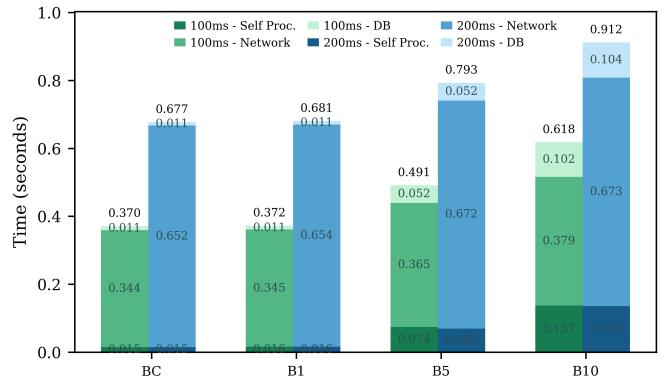


Figure 9: Durations comparison under different network status

blockchain (BC) and MedBlockTree with a single default branch are nearly identical, with only a minor difference attributable to the larger block size in MedBlockTree².

5.4 D3: Number size

In this section, we evaluate the scalability of MedBlockTree across varying network sizes (4, 8, 12, and 16 nodes). During the tests, the doctor and patient clusters generate collision blocks at a fixed rate of one every 60 seconds. The blockchain system with 4 nodes is used as the baseline to provide a clear performance comparison. A total of 2000 metadata entries are processed during each test.

In Figure 10, we exhibit the results by showing the BPS trend under different network sizes. It is obvious that the BPS across all network sizes is increasing regularly with the generation of new branches, which is outstanding compared to a constant processing speed in blockchain. However, the BPS of a larger network size MedBlockTree is smaller than that of one with fewer participants. For instance, the average BPS of MedBlockTree with 4 nodes is 7.77, whereas with 16 nodes it drops to 6.75. This is attributed to the increased communication overhead required for network consensus as more participants are involved, which is also a normal phenomenon in the blockchain systems [22]. Nevertheless, even with 16 nodes, MedBlockTree significantly outperforms the baseline blockchain with only 4 nodes, with 2.5 times higher BPS. We also conclude the detailed comparison results overall processing time (Overall), average BPS (AVGBPS), average time per block (AVGBP), and number of rounds (Rounds) in Table IV. In which with the same network size of 4 nodes (4n), the MedBlockTree (MBT) performs 2.87 times efficiently in processing all metadata, even though MedBlockTree has 16 nodes, it still runs 2.13 times faster. Hence, with more participants in the network, the performance may decrease a bit due to more network communication, but still indicates a great improvement compared to one chain structure.

²The block size is larger than that of SHA256-based blocks primarily due to the verification strings generated by the chameleon hash algorithm, which are essential for collision detection.

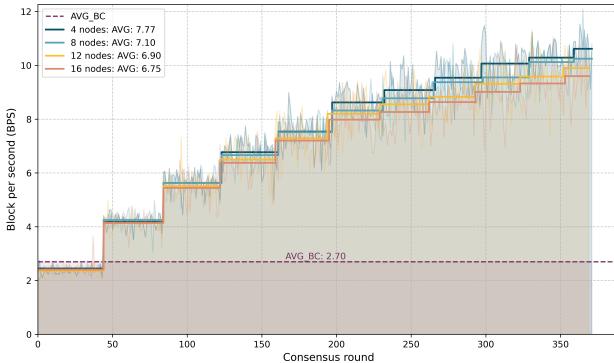


Figure 10: Performance comparison with various network sizes

TABLE IV: Performance details with various network sizes

Sytem	Overall	AVGBPS	AVGPB	Rounds
BC (4n)	738.01s	2.70	0.37s	2000
MBT(4n)	257.43s	7.77	0.13s	371
MBT(8n)	276.63s	7.23	0.14s	371
MBT(12n)	307.22s	6.51	0.15s	371
MBT(16n)	346.62s	5.77	0.17s	371

5.5 D4: Collision rates

The collision rate reflects how frequently collision blocks are generated, which corresponds to how often existing patients return for subsequent treatments in the context of MedBlockTree. This rate directly impacts the expansion speed of the MedBlockTree, as higher collision rates lead to more frequent branch creation, enabling more blocks to be processed in future consensus rounds. In this section, we simulate the system under four different collision rates (30, 60, 90, and 120 seconds) with 4 nodes and 100 ms network latency, to visually examine how this parameter impacts system performance. As shown in Figure 11, the BPS trends for all four rates exhibit a similar stair-like growth pattern. The more frequently the collision is released, the steeper the steps become, leading to

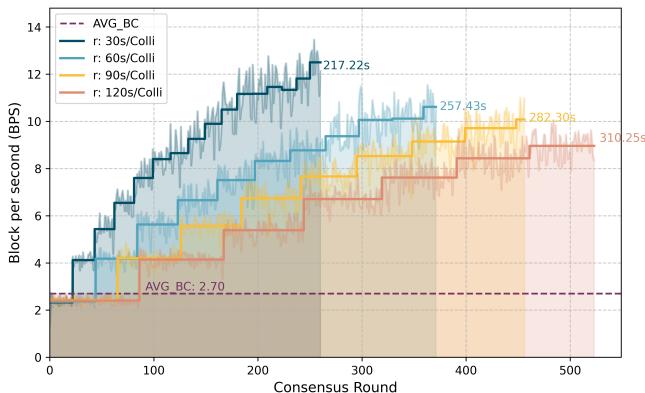


Figure 11: Performance comparison with various collision rates

TABLE V: Performance details with various collision rates

Rate	Overall	AVGBPS	AVGPB	Rounds
BC (4n)	738.01s	2.70	0.37s	2000
MBT(30s)	217.22s	8.39	0.11s	260
MBT(60s)	257.43s	7.77	0.13s	371
MBT(90s)	282.30s	7.08	0.15s	456
MBT(120s)	310.25s	6.45	0.17s	523

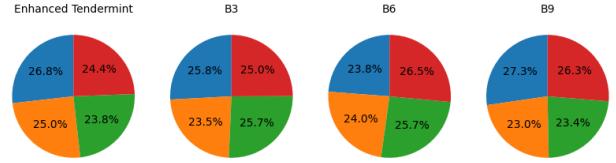


Figure 12: Fairness among workers

a more efficient processing of the overall dataset. When comes to a blockchain system, the consistent processing speed will largely influence the system's efficiency, which needs 738.01 seconds at 2000 rounds to finish. The detailed comparison is summarized in Table V. In a blockchain-based EMR system, the more subsequent treatments a patient requires, the longer the EMR uploading queue tends to be. In contrast, MedBlockTree leverages these frequent updates to improve system efficiency—more collisions lead to faster processing speeds, thereby further reducing lag time.

5.6 D5: Fairness

Fairness and unpredictability among workers are a vital concern of participants and a key contribution of Enhanced Tendermint. Our protocol, EnhancedPro is built upon Enhanced Tendermint. In this section, we measure the fairness among 4 nodes in both systems through the proportion each node is selected as the winner.

In EnhancedPro, the winner election for each branch is based on the chameleon string of the latest block, only till the beginning of the current round can the nodes predict the winners, decreasing the possibility of nodes attacking. Additionally, the chameleon string is generated through the hash function, ensuring the value's randomness. Hence, the EnhancedPro preserves the main characteristic from Enhanced Tendermint with multiple winners being elected in one round. Figure 12 shows the comparison among Enhanced Tendermint and MedBlockTrees using examples of $B(3)$, $B(6)$, $B(9)$ (with fairness remaining consistent across other different branch numbers). Each colour in the pie chart represents one worker.

6. CONCLUSION

In this paper, we propose a novel tree data structure called MedBlockTree and the corresponding consensus protocol, EnhancedPro, to address the scalability limitations in blockchain-based EMR systems, particularly on block throughput. By implementing the identity-based chameleon hash algorithm,

MedBlockTree not only ensures patients' awareness of newly generated blocks but also expands the chain structure into a growing tree, which can help parallelly process multiple blocks in a consensus round. EnhancedPro maintains network consistency by enabling MedBlockTree workers to independently process block proposers' elections using recognized parameters, while implementing a Byzantine Fault Tolerance mechanism to mitigate potential network disruptions. The simulation results show that the block throughput (BPS) / overall processing time of MedBlockTree is significantly enhanced compared with blockchain-based systems throughout branch numbers, network latency, network size, and the collision rates. The findings highlight MedBlockTree's potential as a robust and scalable solution for secure medical data sharing, with the potential to drive the evolution of blockchain applications in healthcare informatics.

However, as the number of branches increases, we observe a pattern of diminishing returns in BPS, suggesting a plateau in scalability benefits with further increases. Future work will focus on developing efficient branch management strategies to address this limitation.

ACKNOWLEDGMENT

This work is supported by the China Scholarships Council and the Victoria University of Wellington Doctoral Scholarship. We extend our gratitude to Catalyst Cloud for their generous support through a student research grant. We also appreciate the valuable assistance and insights from our colleagues in the Wireless Network (WiNe) Research Group at Victoria University of Wellington.

REFERENCES

- [1] Annas, G.J.: Hipaa regulations—a new era of medical record privacy? *New England Journal of Medicine* **348**(15), 1486–1490 (2003)
- [2] Arbabi, M.S., Lal, C., Veeraragavan, N.R., Marijan, D., Nygård, J.F., Vitenberg, R.: A survey on blockchain for healthcare: Challenges, benefits, and future directions. *IEEE Communications Surveys & Tutorials* **25**(1), 386–424 (2023). <https://doi.org/10.1109/COMST.2022.3224644>
- [3] Astle, D.E., et al.: We need timely access to mental health data: implications of the goldacre review. *The Lancet Psychiatry* **10**(4), 242–244 (2023)
- [4] Attaran, M.: Blockchain technology in healthcare: Challenges and opportunities. *International Journal of Healthcare Management* **15**(1), 70–83 (2022)
- [5] Benet, J.: Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561* (2014)
- [6] Bui, N.P., Hoang, M.T., Nguyen, T., Dinh, H.H.Q., Nguyen, B.M.: An enhanced tendermint consensus protocol powered by elliptic curve vrf for beacon chain model. In: *Proceedings of the 12th International Symposium on Information and Communication Technology*. pp. 517–524 (2023)
- [7] Catalyst Cloud: Catalyst cloud – new zealand's leading sovereign cloud provider (2024), <https://catalystcloud.nz/>, accessed: April 12, 2025
- [8] Facebook, Inc.: Rocksdb: A persistent key-value store for fast storage environments (2024), <https://rocksdb.org/>, accessed: April 12, 2025
- [9] Goddard, M.: The eu general data protection regulation (gdpr): European regulation that has a global impact. *International Journal of Market Research* **59**(6), 703–705 (2017)
- [10] Guo, G., Zhu, Y., Chen, E., Zhu, G., Ma, D., Chu, W.C.: Continuous improvement of script-driven verifiable random functions for reducing computing power in blockchain consensus protocols. *Peer-to-Peer Networking and Applications* pp. 1–20 (2022)
- [11] Hermansen, A., Regier, D.A., Pollard, S.: Developing data sharing models for health research with real-world data: a scoping review of patient and public preferences. *Journal of Medical Systems* **46**(12), 86 (2022)
- [12] Johari, R., Kumar, V., Gupta, K., Vidyarthi, D.P.: BLO-SOM: Blockchain technology for Security of Medical records. *ICT Express* **8**(1), 56–60 (2022)
- [13] Kale, D.R., Jadhav, A.N., Salunkhe, S.J., Hirve, S., Goswami, C., et al.: Sharding: A scalability solutions for blockchain networks. In: *2024 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*. pp. 1–8. IEEE (2024)
- [14] Kesselheim, A.S., Hwang, T.J.: Breakthrough medical devices and the 21st century cures act. *Annals of internal medicine* **164**(7), 500–502 (2016)
- [15] Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. *Cryptology ePrint Archive* (1998)
- [16] Li, C., Li, P., Zhou, D., Yang, Z., Wu, M., Yang, G., Xu, W., Long, F., Yao, A.C.C.: A decentralized blockchain with high throughput and fast confirmation. In: *2020 {USENIX} Annual Technical Conference ({USENIX}){ATC} 20*. pp. 515–528 (2020)
- [17] Li, C., Shen, Q., Xie, Z., Dong, J., Fang, Y., Wu, Z.: Efficient identity-based chameleon hash for mobile devices. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. pp. 3039–3043. IEEE (2022)
- [18] Li, J., Pandey, V., et al.: A blockchain-based personal health knowledge graph for secure integrated health data management. In: *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*. pp. 1–7. IEEE (2023)
- [19] Ma, S., Zhang, X.: Integrating blockchain and zk-rollup for efficient healthcare data privacy protection system via ipfs. *Scientific Reports* **14**(1), 11746 (2024)
- [20] Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: *Proceedings of the 40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. pp. 120–130. IEEE (1999)
- [21] Raikwar, M., Polyanskiy, N., Müller, S.: Sok: Dag-based consensus protocols. In: *2024 IEEE International Con-*

- ference on Blockchain and Cryptocurrency (ICBC). pp. 1–18. IEEE (2024)
- [22] Rao, I.S., Kiah, M.M., Hameed, M.M., Memon, Z.A.: Scalability of blockchain: a comprehensive review and future research direction. *Cluster Computing* **27**(5), 5547–5570 (2024)
- [23] Sakurai, A., Shudo, K.: Impact of the hash rate on the theoretical fork rate of blockchain. In: Proceedings of the IEEE International Conference on Consumer Electronics (ICCE). pp. 1–4. IEEE (2023)
- [24] Santiago, C., Lee, C.: Accelerating message propagation in blockchain networks. In: Proceedings of the International Conference on Information and Communication Technology Convergence. pp. 157–160. IEEE (2020)
- [25] Santos, J.A., Inacio, P.R., Silva, B.M.: Towards the use of blockchain in mobile health services and applications. *Journal of Medical Systems* **45**, 1–10 (2021)
- [26] Sheikh, A., Anderson, M., Albala, S., Casadei, B., Franklin, B.D., Richards, M., Taylor, D., Tibble, H., Mossialos, E.: Health information technology and digital innovation for national learning health and care systems. *The Lancet Digital Health* **3**(6), e383–e396 (2021)
- [27] Sompolinsky, Y., Lewenberg, Y., Zohar, A.: Spectre: A fast and scalable cryptocurrency protocol. *Cryptology ePrint Archive* (2016)
- [28] Thulasiraman, K., Swamy, M.: 5.7 acyclic directed graphs. *Graphs: theory and algorithms* **118** (1992)
- [29] Xu, J., Wang, C., Jia, X.: A survey of blockchain consensus protocols. *ACM Computing Surveys* **55**(13s), 1–35 (2023)
- [30] Zou, R., Lv, X., Zhao, J.: Spchain: Blockchain-based medical data sharing and privacy-preserving ehealth system. *Information Processing & Management* **58**(4), 102604 (2021)