2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) | 978-1-6654-1658-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/TRUSTCOM53373.2021.00149

Energy Efficient Merkle Trees for Blockchains

Cesar Castellon
School of Engineering
University of North Florida
Jacksonville - USA
n01453427@unf.edu

Swapnoneel Roy
School of Computing
University of North Florida
Jacksonville - USA
s.roy@unf.edu

Patrick Kreidl
School of Engineering
University of North Florida
Jacksonville - USA
patrick.kreidl@unf.edu

Ayan Dutta
School of Computing
University of North Florida
Jacksonville - USA
a.dutta@unf.edu

Ladislau Bölöni
Department of Computer Science
University of Central Florida
Orlando - USA
ladislau.boloni@ucf.edu

***Abstract*—Blockchain-powered smart systems deployed in different industrial applications promise operational efficiencies and improved yields, while mitigating significant cybersecurity risks pertaining to the main application. Associated tradeoffs between availability and security arise at implementation, however, triggered by the additional resources (e.g., memory, computation) required by each blockchain-enabled host. This paper applies an energy-reducing algorithmic engineering technique for Merkle Tree root calculations, a principal element of blockchain computations, as a means to preserve the promised security benefits but with less compromise to system availability. Using pyRAPL, a python library to measure computational energy, we experiment with both the standard and energy-reduced implementations of the Merkle Tree for different input sizes (in bytes). Our results show up to 98% reduction in energy consumption is possible within the blockchain's Merkle Tree construction module, such reductions typically increasing with larger input sizes. The proposed energy-reducing technique is similarly applicable to other key elements of blockchain computations, potentially affording even "greener" blockchain-powered systems than implied by only the Merkle Tree results obtained thus far.**

***Index Terms*—Merkle Tree, Blockchain, Energy.**

I. Introduction

Blockchain technology, popularized by different cryptocurrency systems, is seeing extensive use in different fields. Advocates for such uses cite the blockchain's inherent properties of a decentralized structure alongside enhanced security with mechanisms for privacy and non-repudiation [1], [2], [3], [4]. One particularly promoted use case is the *Internet of Things (IoT)* [5], [6], [7], which embodies the vision by which different computing devices may communicate with each other to map a physically connected world onto its digital mirror. The IoT vision also motivates prospects of so-called *smart* systems [8] e.g., smart cities, smart homes, smart grid, smart health, smart agriculture. The potential uses and benefits of smart systems recognizably also raises critical security and privacy challenges to be addressed, which motivates the vision of blockchain-powered smart systems.

This research study is supported in part by NSF CPS #1932300, and Cyber Florida #220408 grants.

Smart and secure systems implemented upon IoT technology require device inter-connectivity for extended time frames, delivering continuous data. Such operations demand constant power supply [8]—within a world that demands more environmentally-friendly (or "green") solutions, in general, IoT realizations also face the challenge of energy efficiency i.e., minimizing their energy footprint. Thakore et al. [9] acknowledge the additional energy optimization requirements that blockchains require when implemented together with IoT. Depending on the specific type of blockchain-IoT combination, precise analysis of performance and energy requirements becomes critical [10]. As an example of these challenging tradeoffs, consider a particular blockchain-IoT implementation with a fixed power budget. To be viable for an application that values autonomy for greater lengths of time, the system must be configured to make more efficient use of energy. Disabling the blockchain will certainly save energy, but also weaken security: it is in such contexts that the exploration of ways to reduce energy consumption of blockchain functionality alone can be of tremendous practical significance.

A. Related Work

Energy efficiency in computation is a widely studied topic, with numerous points-of-view: hardware-specific platforms, operating systems, hypervisors and containers [11]; software development and security [12]; and algorithms [13], [14]. Energy measurements are sometimes obtained by uniquely instrumented equipment [15], while other times can leverage hardware providers' Application Programmer Interfaces (APIs) in which firmware counters are recalled to provide near real-time information e.g., Running Average Power Limit (RAPL) technology [16]. Blockchain implementations are actively under study as providing a decentralized ledger (i.e. record of transactions) by which to optimize energy management in a variety of scenarios (e.g., generation & distribution [17], [18], micro-grid networks [19], [20], [21] and smart contracts [22]). In contrast to our motivation, however, these studies define

2324-9013/21/$31.00 ©2021 IEEE
DOI 10.1109/TrustCom53373.2021.00149

the optimized management objectives such that the energy footprint of the blockchain itself is out of scope.

There are past studies who also recognize that the blockchain itself will draw energy away from any symbiotic system it is integrated with. Examples include Sankaran et al. [10] and Sanju et al. [23], who perform power measurements and evaluate real experiments on the energy consumption of two different blockchain implementations, namely Ethereum and Hyperledger. A similar analysis of energy consumption is presented in [15] for XRP validation, which is a key element of decentralized consensus processes within many Internet services. A particularly novel theoretical approach is reported by Fu et al. [24], first modeling a blockchain-IoT caching infrastructure and posing its energy optimization within a geometric programming formulation whose solutions allocate resources accordingly. A recent performance evaluation survey, also conducted by Fu et al. [1], illustrates how diverse and sophisticated current implementations of blockchain ledgers are. Despite this diversity, however, all existing implementations at their core remain faithful to Nakamoto's original blockchain concept [25], within which the Merkle Tree construction module is essential.

B. Our Scope and Contributions

We study the extent to which Merkle Tree construction, a principal element of blockchain computations, can be made more energy efficient. Our approach employs an energy-reducing algorithmic engineering technique, based upon an Energy Complexity Model (ECM) proposed by Roy et al. [13], [14], on the SHA256 encryption algorithm, which is central to the Merkle Tree. Using pyRAPL, a python library to measure an executable's Runtime Average Power Limit, we experiment with both the standard and energy-reduced implementations of the Merkle Tree for input sizes (in bytes) that are commonly seen within blockchain implementations. Our results show significant reductions in energy consumption, up to 98% but on average 50% across the tested input sizes. At present, it is only a conjecture that reduced energy consumption in the Merkle Tree construction module itself extrapolates to comparable reduction of a blockchain on the whole. In any case, to the best of our knowledge our work is the first to address energy optimization of blockchains by re-engineering the implementation of one of its component algorithms. Moreover, the proposed energy-reducing technique is similarly applicable to other key elements of blockchain computations, potentially affording even "greener" blockchain-IoT systems than implied by only the Merkle Tree results obtained thus far.

II. Methodology

This section describes our application of the Energy Complexity Model (ECM) [13], [14] to the Merle Tree (MT) root construction module of the blockchain. Described first is the process by which a block of the blockchain is computed based on the MT root, in which so-called hash calculations play a central role, followed by a summary of how the ECM works, in general. This section ends with a detailed description of how the central hash calculations of the MT are re-engineered based on the ECM.

A. Merkle Tree Based Block Generation

A graphic representation of a simple block generation in a blockchain is shown in Fig. 1. The bottom layer shows the stored transactions (e.g., $T001$) for the block, which later are converted to their SHA256 Hash signatures (e.g., $H001$) and represent the Merkle Tree leaves. Merkle Tree root calculations involves the recursive hash computation starting from these leaves until a final hash determines the Merkle Tree root (labeled TX_ROOT in Fig. 1).

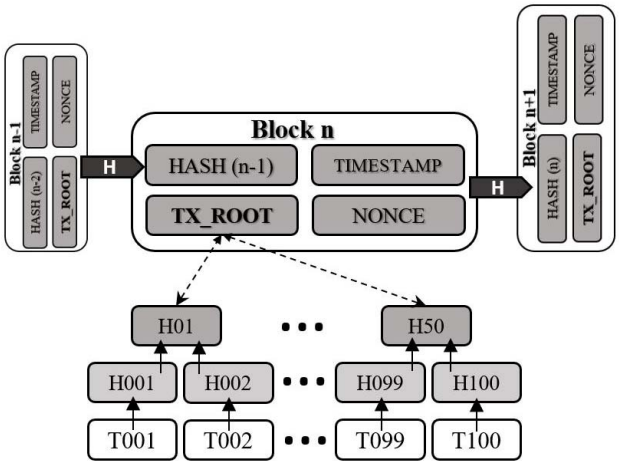


Figure 1: Basic Block Generation in Blockchain.

Conceptually, the process of Merkle Tree calculation through hashing can be viewed as a state transition in which an investment of computational resources is required e.g.,

$$\delta_t \xrightarrow{f(T)} \delta_{t+1} \quad (1)$$

$$T = cost[energy, time] \quad (2)$$

That is, the block generation is represented by the state transition in (1), which depends upon a function $f(T)$ with parameter T denoting the *cost* as represented by (2). This cost has two main components: one is the energy consumed by the hardware devices to compute the hash of the input vectors, while the other is the execution time of those computations. This paper strives to reduce the overall transition cost T by reducing the energy consumption of the hardware devices, employing a technique based upon the ECM next described.

B. The Energy Complexity Model (ECM)

The ECM developed in [13] is built upon an abstraction of the Double Data Rate Synchronous Dynamic Random Access Memory (DDR SDRAM) architecture [26], which is illustrated in Fig. 2. Main memory in DDR is divided into *banks*, each of which contains a certain number of *chunks*[1]. Data is allocated

[1]The term "block" is used in DDR specifications, but we use the term "chunk" to avoid confusion wiithin our blockchain context.

over chunks in each bank, and each bank also contains a special chunk called the *sense amplifier*. When data needs to be accessed, the chunk containing the data is fetched into the sense amplifier of the corresponding bank. The sense amplifier can only hold one chunk at a time, so the current chunk has to be put back to its bank before the next one can be fetched for access. While only one chunk of a particular bank can be accessed at a time, chunks of different banks (each with their own sense amplifier) can be accessed in parallel. Therefore, if the DDR memory is organized into P banks (where $P = 4$ in Fig. 2), then P chunks can be accessed at a given time. In the popular DDR3 architecture, the DDR1 notion of the per-bank sense amplifier is referred to as the per-bank cache, albeit still only capable of accessing one chunk at a given time.

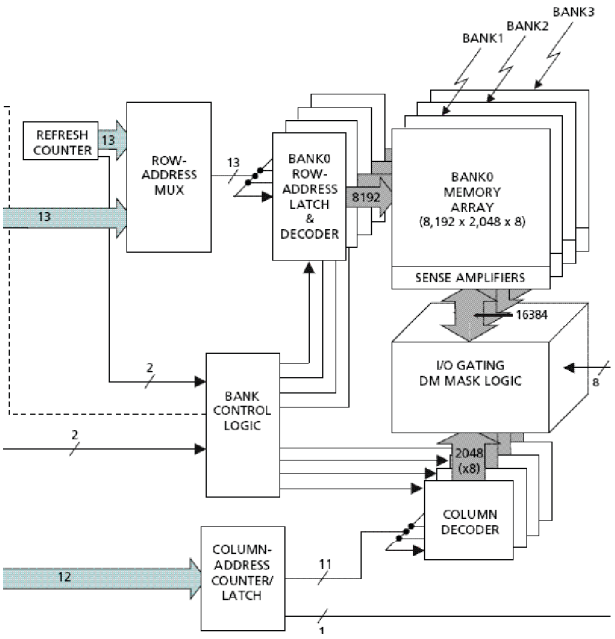


Figure 2: Internal DDR SDRAM memory chip block diagram.

The ECM denotes the P banks of a given DDR3 SDRAM resource by $M_1, M_2, \ldots, M_P$, each such bank M_i comprised of multiple chunks of size-B (in bytes) and its own cache C_i. The illustrative example of Fig. 3 assumes $P = 4$ banks, as was the case in Fig. 2, with just four chunks per bank, assigning numerical labels $1, 2, \ldots, 16$ to the memory's collection of data chunks. Heeding the DDR constraint that each cache C_i may access exactly one chunk at a time, the access patterns $(1, 2, 3, 4)$ or $(5, 6, 7, 8)$ imply a completely serial execution, while the access patterns $(1, 5, 9, 13)$ or $(3, 8, 10, 13)$ are each completely parallel. The authors of [13] discovered two key properties of DDR memory: firstly, the difference in power consumption between the same number of chunks accessed sequentially or in parallel is not significant; however, the execution time of an algorithm when chunks are accessed in parallel is significantly lower than when chunks are accessed sequentially. Because the associated energy consumption depends upon both power and time, it follows that parallelizing chunk accesses offers the potential for energy reduction of any algorithm! More formally, as derived by Roy *et al.* [13], the energy consumption (in Joules) of an algorithm $\mathcal{A}$ with execution time τ, assuming a P-bank DDR3 architecture with B bytes per chunk, is given by

$$E(\mathcal{A}) = \tau + (P \times B)/I \tag{3}$$

where I denotes the so-called *parallelization index*, essentially the number of parallel block accesses across memory banks per P block accesses made by $\mathcal{A}$ on the whole. That is, under the ECM, an algorithm's potential for energy reduction is inversely proportional to the degree to which it can be re-engineered for parallelization of its memory accesses.

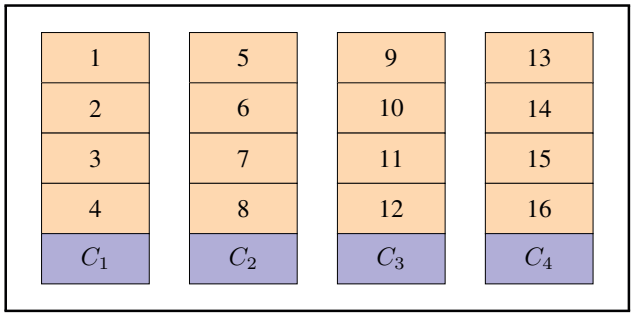


Figure 3: ECM for DDR3 Resource with $P = 4$ Banks

C. Re-engineering Hash Calculations Using ECM

In this work, we engineer the hash algorithm of Merkle Tree (MT) construction based on ECM to reduce energy consumption. First, we briefly describe how any algorithm $\mathcal{A}$ can be parallelized based on ECM. We then illustrate how MT's hash calculation, specifically the SHA encryption algorithm, is re-engineered for parallelization.

1) Parallelizing any algorithm: Given an algorithm $\mathcal{A}$, the input to $\mathcal{A}$ is considered to identify the most common access sequence in $\mathcal{A}$. The required level of parallelism for the vector formed by the desired access sequence is then engineered using a logical mapping over chunks of memory that store data accessed by $\mathcal{A}$. As mentioned above, the order of chunk accesses is different for different levels of parallelization. But the physical location (chunks) of the input in the memory is static, and is handled by the memory controller of DDR. Therefore, to implement parallelization of access over physical chunks, a different page table vector $\mathbf{V}$ is generated for each level of parallelization, which defines the ordering among the chunks to be accessed (see Fig. 4).

For 1-way access, the page table vector $\mathbf{V}$ has the pattern $(1, 2, 3, 4, \ldots)$ and for 4-way access it has the pattern $(1, 5, 9, 13, \ldots)$. A function is then created to map the pattern of the page table vector $\mathbf{V}$ to the original physical locations of the input. Algorithm 1 shows the function to create an ordering among the chunks. The ordering is based on the way we want to access the chunks (P-way would mean full parallel access). The page table is populated by picking chunks with *jumps*.

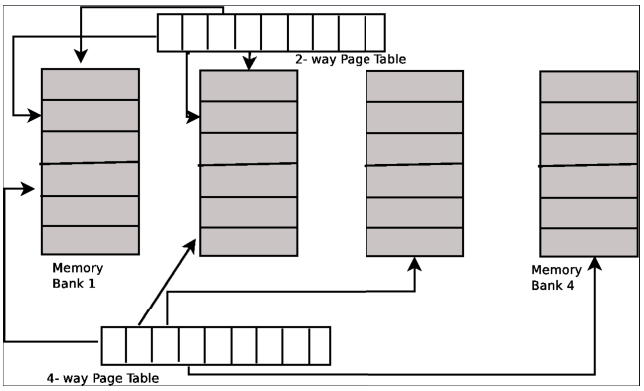


Figure 4: Memory Layout ($P = 4$) and Role of Page Tables

For P-way access, jumps of P are selected that ensure the consecutive chunk accesses lie in P different banks. Going by the above example, for $P = 1$, jumps of 1 ensure that 4 consecutive chunk accesses lie in the same bank (bank 1 of Fig. 3). On the other hand, for $P = 4$, jumps of 4 ensures that 4 consecutive chunk access lie in 4 different banks (banks 1 through 4 of Fig. 3).

Input: Page table vector $\mathbf{V}$, jump amount *jump*.
factor $= 0$;
for $i = 0$ *to* $\frac{N}{B} - 1$ **do**
  if $i > 1$ *and* $(i \times jump) \mod \frac{N}{B} = 0$ **then**
    factor $=$ *factor* $+1$;
  end
  $\mathbf{V}_i = (i \times jump + factor) \mod \frac{N}{B}$;
end

Algorithm 1: Create a Page Table for N Chunks

2) Parallelizing SHA Encryption: As described earlier, Merkle Tree construction performs its hash calculations via repeated use of the SHA256 encryption algorithm. Specifically, as shown in Fig. 5, the input is partitioned into fixed size message blocks, presented in sequence to separate compression functions. This block sequence is identified in correspondence with the access pattern of the SHA256 algorithm, which we subject to re-engineering based on the ECM. The input vector, in a Merkle Tree being the concatenation of three strings (see Fig. 5), is pre-processed into another vector by applying Algorithm 1. The mapping is then stored in a page table to be used in subsequent hash calculations. An example of this operation for 16 blocks and a parallelization index (jump) of 4 is shown in Fig. 6.

Fig. 7 shows the outcome of re-engineering the SHA256 algorithm based on ECM. In our experimentation, an 8-bank DDR3 SDRAM is used and the parallelization index is set to $I = 8$. This essentially means that for any set of eight consecutive block access in SHA256, we created a virtual mapping using techniques described in [14] to ensure that each size-8 access occurs across all eight banks.

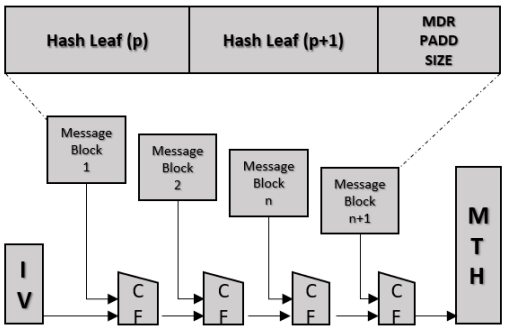


Figure 5: SHA256 for Merkle Tree Calculation

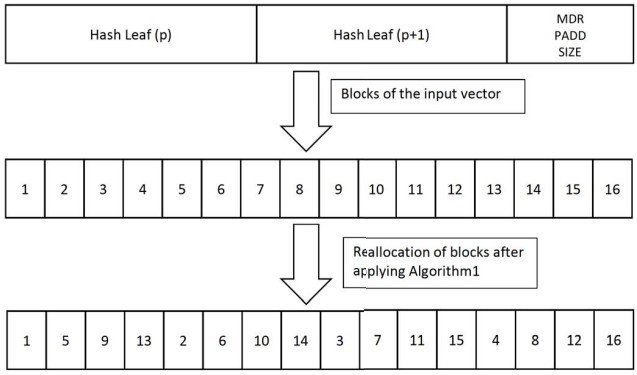


Figure 6: Mapping of SHA Input Blocks based on ECM.

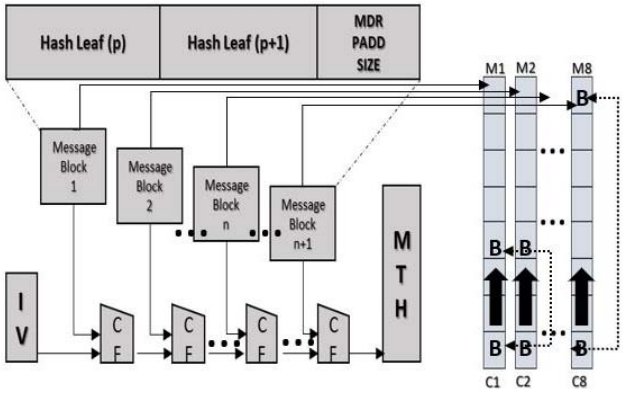


Figure 7: ECM-Enhanced Merkle Tree Calculation

III. Experiments

We now proceed to describe computer experiments designed to quantify the energy savings of the methodology detailed in the previous section. By virtue of the ECM's formulation, the enhanced implementation requires computer hardware using a DDR RAM architecture. Maximum energy reduction is promised by a parallelization index taken to equal the number of memory banks, which depends upon the DDR version: 4 for DDR2, 8 for DDR3 and 16 for DDR4 and higher. The machine used for our experiments features a 64-bit dual-core

processor (Intel i5-2410M @ 2900MHz with cache size L2 256KB and L3 3072KB), running Linux Mint version 19.3 with a 8GB DD3 RAM and 500GB SSD storage. We use pyRAPL, a software toolkit to measure a host machine's energy footprint along the execution of a piece of Python code, to compare energy consumptions between the standard and ECM-enhanced implementations. pyRAPL is built upon Intel's Running Average Power Limit (RAPL) technology that estimates a CPU's power consumption; depending on the hardware and operating system configurations, pyRAPL can measure energy consumption of the following CPU domains: CPU socket, GPU, and DRAM [16].

A. Implementation Details and Setup

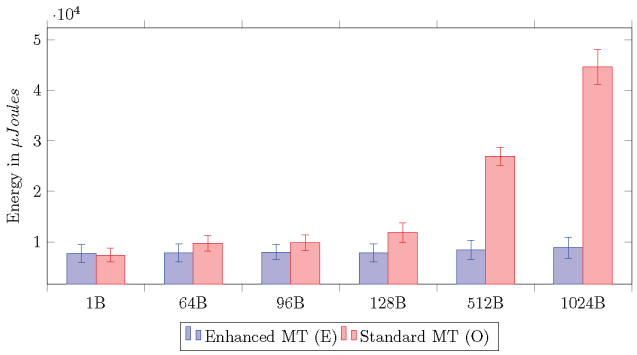
Our experimental objectives could not be met by using the SHA256 function in the Hash Python library. This is because memory management in Python involves a private heap, containing all objects and data structures. The control of this private heap is ensured internally by the Python memory manager, with different components dealing with sharing, segmentation, pre-allocation or caching. Our ECM-enhanced implementation of SHA256 requires greater control over memory allocation than Python's memory manager permits. Such low-level control on memory management is possible in the standard C programming language. We thus implement the standard and ECM-enhanced versions of the SHA256 algorithm within separate C programs, which are called from a Python script (upon importing the `ctypes` module) as an external routine. This permits the use of pyRAPL for the needed energy measurements without denying low-level memory control to implement the ECM-enhanced SHA256 functionality during Merkle Tree calculations.

Our experiments simulated the Merkle Tree calculation with Python code that runs 103 consecutive two-leaves-input hashes with pyRAPL invoked. Each execution of the code yields an energy measurement, but because the instrumentation is subject to noise we invoke 5000 repetitions and report the average energy (mean and deviation). Our experiments also vary the input size (i.e., the compounded-leaf size) to the Merkle Tree calculations, choosing 1, 64, 96, 128, 512, 1024, 16384 and 262144 bytes motivated as follows:

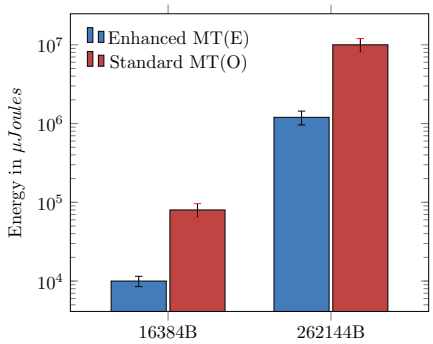
1) the 1B input is the bare minimum that the ECM permits for any algorithm [13];
2) the 64B, 96B and 128 inputs are common in blockchain applications [6];
3) the 512B and 1024B inputs are common in file hashing applications [27]; while
4) the 16384B and 262144B inputs are common in the Interplanetary File System (IPFS) [28], [29].

B. Results and Discussion

Recall that our experimental setup features two implementations of Merkle Tree (MT) calculations, the standard one (which we label by "O" as it uses the original SHA256) and the re-engineered one using ECM (which we label by "E" as it uses the enhanced SHA256), as well as eight different input sizes. Per implementation and per input size, our experimental Python script leverages the pyRAPL toolkit to measure the average energy (mean and deviation over 5000 trials) of simulated Merkle Tree calculations. Fig. ener summarizes the sixteen average energy measurements in two bar charts, per input size comparing the Standard MT (O) and the Enhanced MT (E) average energy (in μJoules). Fig. 8(a) renders the comparison over the six smallest input sizes (using a linearly-scaled vertical axis), while Fig. 8(b) is over the two largest input sizes (using a log-scaled vertical axis). It is seen that the ECM-enhanced implementation consistently requires less energy that the standard implementation, the difference being increasingly significant with the larger input sizes that befit file hashing applications (i.e., 512B and above) (but still meaningful for input sizes 64B, 96B and 128B that befit blockchain applications). This observed dependence on input size may be a consequence of CPU memory caching. DRAM memory often allows the memory controller to optimise accesses by L1/L2/L3 caching of data. With smaller inputs, such caching enables parallelization of bank accesses even in the standard implementation. The comparison for the 1B input size corroborates this point, where we observe the enhanced implementation consume more energy than the standard implementation!



(a) Versus Small Input Sizes (in Bytes)



(b) Versus Large Input Sizes (in Bytes)

Figure 8: Comparison of Average Energy Consumption

Fig. 9 presents the average energy comparison on more relative terms, namely as a percent reduction achieved by

the enhanced implementation over the standard implementation versus all eight input sizes. The energy savings for the blockchain-motivated input sizes range between 19% and 34%, while the energy savings for the file-system-motivated input sizes range between 69% and 98%, the case of 16384B exhibiting that maximum 98% savings. As noted in Fig. 8, the 1B input renders a savings of -4%, meaning the standard implementation is more energy-efficient by virtue of the parallelism invoked within the CPU's L1/L2/L3 cache in this case.

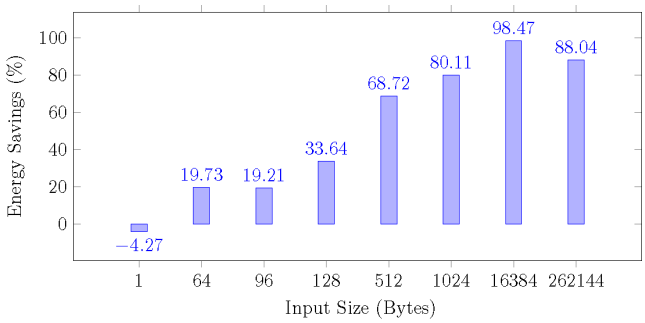


Figure 9: Comparison of Energy Savings

IV. Conclusion

This work considers reducing the energy consumption of Merkle Tree (MT) calculations within blockchains by re-engineering the core hashing algorithm, namely SHA256 encryption, via the Energy Complexity Model (ECM) [13]. The ECM-enhanced implementation was compared to the standard implementation via experimental energy measurements with various input sizes of practical significance. The results show up to 34% energy savings are possible for input sizes typically used by blockchains, while up to 98% is possible for input sizes used in other applications (e.g., file systems). It remains conjecture that reduced energy consumption in the Merkle Tree construction module itself extrapolates to comparable reduction to the blockchain on the whole. Future work can also assess the energy saving opportunities in other applications of Merkle Trees e.g. authentication schemes [29], healthcare systems [30], embedded systems [31], network protocols [32], [33]. Also of interest is the exploration of parallelized hash calculations to reduce energy consumption of the blockchain Proof-of-Work (PoW) algorithm. Similarly, because the proposed energy-reducing technique and energy measurement instrumentation may be applicable to other key computations, similarly "greener" solutions may be possible within Internet-of-Things (IoT) technology and envisioned smart systems.

References

[1] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126 927–126 950, 2020.

[2] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[3] N. Gaur, L. Desrosiers, V. Ramakrishna, P. Novotny, S. A. Baset, and A. O'Dowd, *Hands-on blockchain with hyperledger: building decentralized applications with hyperledger fabric and composer*. Packt Publishing Ltd, 2018.

[4] C. Alcaraz, J. E. Rubio, and J. Lopez, "Blockchain-assisted access for federated smart grid domains: Coupling and features," *Journal of Parallel and Distributed Computing*, 2020.

[5] B. Yu, J. Wright, S. Nepal, L. Zhu, J. Liu, and R. Ranjan, "Trust chain: Establishing trust in the iot-based applications ecosystem using blockchain," *IEEE Cloud computing*, vol. 5, no. 4, pp. 12–23, 2018.

[6] Y. Sun, L. Zhang, G. Feng, B. Yang, B. Cao, and M. A. Imran, "Blockchain-enabled wireless internet of things: Performance analysis and optimal communication node deployment," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5791–5802, 2019.

[7] A. Banafa, "Iot and blockchain convergence: benefits and challenges," *IEEE Internet of Things*, 2017.

[8] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with iot. challenges and opportunities," *Future generation computer systems*, vol. 88, pp. 173–190, 2018.

[9] R. Thakore, R. Vaghashiya, C. Patel, and N. Doshi, "Blockchain-based iot: A survey," *Procedia Computer Science*, vol. 155, pp. 704–709, 2019.

[10] S. Sankaran, S. Sanju, and K. Achuthan, "Towards realistic energy profiling of blockchains for securing internet of things," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1454–1459.

[11] J. Westin, "Evaluation of energy consumption in virtualization environments: proof of concept using containers," 2017.

[12] P. D. Harish, "Towards designing energy-efficient secure hashes," Ph.D. dissertation, UNF Digital Commons, 2015.

[13] S. Roy, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, 2013, pp. 283–304.

[14] ——, "Energy aware algorithmic engineering," in *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*. IEEE, 2014, pp. 321–330.

[15] C. A. Roma and M. A. Hasan, "Energy consumption analysis of xrp validator," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–3.

[16] M. Santos, J. Saraiva, Z. Porkoláb, and D. Krupp, "Energy consumption measurement of c/c++ programs using clang tooling." in *SQAMIA*, 2017.

[17] F. Imbault, M. Swiatek, R. De Beaufort, and R. Plana, "The green blockchain: Managing decentralized energy production and consumption," in *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*. IEEE, 2017, pp. 1–5.

[18] T. Yang, Q. Guo, X. Tai, H. Sun, B. Zhang, W. Zhao, and C. Lin, "Applying blockchain technology to decentralized operation in future energy internet," in *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*. IEEE, 2017, pp. 1–5.

[19] E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer, and C. Weinhardt, "A blockchain-based smart grid: towards sustainable local energy markets," *Computer Science-Research and Development*, vol. 33, no. 1-2, pp. 207–214, 2018.

[20] T. Li, W. Zhang, N. Chen, M. Qian, and Y. Xu, "Blockchain technology based decentralized energy trading for multiple-microgrid systems," in *2019 IEEE 3rd Conference on Energy Internet and Energy System Integration (EI2)*. IEEE, 2019, pp. 631–636.

[21] S. Noor, W. Yang, M. Guo, K. H. van Dam, and X. Wang, "Energy demand side management within micro-grid networks enhanced by blockchain," *Applied energy*, vol. 228, pp. 1385–1398, 2018.

[22] M. Mylrea and S. N. G. Gourisetti, "Blockchain for smart grid resilience: Exchanging distributed energy at speed, scale and security," in *2017 Resilience Week (RWS)*. IEEE, 2017, pp. 18–23.

[23] S. Sanju, S. Sankaran, and K. Achuthan, "Energy comparison of blockchain platforms for internet of things," in *2018 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS)*. IEEE, 2018, pp. 235–238.

[24] S. Fu, L. Zhao, X. Ling, and H. Zhang, "Maximizing the system energy efficiency in the blockchain based internet of things," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[25] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system. whitepaper bitcoin," 2008.

[26] T. Farrell, "Core architecture doubles mem data rate," *Electronic Engineering Times Asia*, vol. 16, 2005.

[27] B. Dowling, F. Günther, U. Herath, and D. Stebila, "Bsecure logging schemes and certificate transparency," in *European Symposium on Research in Computer Security*, vol. 452. IACR, 2016.

[28] R. Kumar and R. Tripathi, "Implementation of distributed file storage and access framework using ipfs and blockchain," *Fifth International Conference on Image Information Processing*, pp. 246–251, 2019.

[29] Y.-C. Chen, Y.-P. Chou, and Y.-C. Chou, "An image authentication scheme using merkle tree mechanisms," *Future Internet*, vol. 11, no. 7, 2019.

[30] U. Chelladurai and S. Pandian, "Hare: A new hash-based authenticated reliable and efficient modified merkle tree data structure to ensure integrity of data in the healthcare systems," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–15, 2021.

[31] Y. Zou and M. Lin, "Fast: A frequency-aware skewed merkle tree for fpga-secured embedded systems," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 326–331.

[32] S. You, F. Xue, Z. Qi, and H. Liu, "Wireless sensor network protocol based on hierarchical merkle tree," in *International Conference On Signal And Information Processing, Networking And Computers*. Springer, 2018, pp. 9–15.

[33] S. Dhumwad, M. Sukhadeve, C. Naik, K. Manjunath, and S. Prabhu, "A peer to peer money transfer using sha256 and merkle tree," in *2017 23RD Annual International Conference in Advanced Computing and Communications (ADCOM)*. IEEE, 2017, pp. 40–43.