Blockchain Mining: Understanding its Difficulty in Terms of Hashing Algorithm Efficiency

CARLOS ROBERTO MARTINEZ MARTINEZ (carlos.martinez@catolica.edu.sv)
Catholic University of El Salvador

Research Article

Keywords: MD5, SHA, proof of work, nonce value, performance.

Posted Date: December 20th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2383470/v1

Abstract

Blockchain is an emerging technology that offers great advantages such as global distribution of information and data immutability over time, but its mining process is known to be time consuming and highly resource demanding. There are different algorithms to mine blocks, being able to operate at different levels of complexity and also incurring different compute costs. This study used a custom and simple implementation of the MessageDigest Java class to evaluate the performance of MD5 version 2, SHA2, and SHA3 (the last two in modalities of 256 and 512-bits) when automatically creating a chain of a thousand blocks, measuring its average time for generation and proof of work verification, alongside to the count of incurred iterations. The SHA3 methods produced stronger hash values but performed slower than MD5 and SHA2, and the complexity level of 4 characters offered the best security level in terms of iteration count versus calculation time. Complexity levels of 5 and 6 characters offered greater security but with a drastic performance reduction, which is proper for critical security systems that do not process high volumes of information. These results are useful for those small to medium organizations that are planning to implement blockchain technologies and need a parameter for understanding exactly what 'mining cost' really means in terms of compute costs.

Introduction

Blockchain is a technology that can be used to quickly distribute high volumes of information over the Internet, implementing some combined security mechanisms like block *immutability*, data encryption, mining, and *proof work* verification (Nakamoto, 2008). These advancements allow trustable data storage in a worldwide distributed environment (Herlihy, 2019), where it is very difficult for an individual to alter its content due to the strict chronological dependence among data blocks and the consensus mechanism among blockchain distributed copies, in which the majority of registered compute nodes automatically coordinate to discard manipulated versions of blocks that do not match the mining verification process.

To ensure data integrity, the best practice might be to increase the level of complexity of the mining process. However, this also increases the number of encryption calculations involving a lot of compute power to process time consuming tasks (Misra *et al.*, 2020). For example, actual crypto-currency blockchains cannot be mined by personal computers anymore, but require data centers with many powerful and expensive servers. The practical dilemma is about how much security capabilities are to be sacrificed in exchange for performance, especially in the case of small to middle size organizations that could make this technological advancement a great asset, but with the disadvantage of covering the direct and indirect economic costs, or for the lack of understanding for designing a balanced solution about how strong the mining process must be them (Pika *et al.*, 2021; Hassib *et al.*, 2019; Chand *et al.*, 2018; Blake & Mangiameli, 2008).

Justification

Due to its inherent characteristics, blockchain is suited for many kind of business in the real world, for example, clinic records for hospital networks, financial systems, stock investments, smart contracts, property deeds, logistics chains, product traceability, criminal records, customer care history, etc. Despite of the fact that it is an emerging technology, its implementation in certain applications could result in a revolutionary advancement and a strategic gain for improving data distribution processes, increasing availability, signing digital information, and storing data in an auto-recovery environment in case of disaster.

This article explains how blockchain systems can be overloaded by heavy mining processes and how different methodologies for *hash* calculations could help to mitigate part of the effort.

Theoretical framework

A block is a programmed software object that contains information attributes according to any specific business logic, altogether with a timestamp and a unique digital signature which is calculated from its internal to form a *hash summatory* (Klinkmüller *et al*., 2019). When blocks are added to a controller chain, they are stored according to their chronological precedence which is constituted by their timestamp and the respective hash value, without the possibility to be subsequently changed. In general terms, a *hash* is a set of characters that represents the identity and authenticity of any kind of information. For example, when using the traditional *md5* algorithm (Guzmán *et al*., 2018; Wang *et al*., 2010), the hash value for the phrase "transaction of $10.00 USD" is the following *hex string*: "ec52cfa09c888c7e62cd23cf17ff9c7d". However, a slight change on the decimal point can completely alter the meaning of the information as in the example: "transaction of $ 100.00 USD", for which the respective new digital signature is also different: "80623aa32e567d516e643794935b96c1". *Hashes* are usually 32 characters long, and that is wide enough to represent any kind of text, block or file, with acceptable accuracy. Every time a new block is created, a *hash* of its content is calculated and stored within, and when it is transmitted to a different server, the same *hash* is again calculated and compared with the value of the previously stored block, assuming that if the actual block content has not been maliciously changed, then the two *hashes* must be identical. Otherwise, the block is automatically discarded and the entire *chain* is reported as broken. If the calculation of a block is valid and coherent with the information of its predecessor, it has reached the state of "*proof of work*" (Kiayias & Zindros, 2019; Gervais, *et al*., 2016; Vukolić, 2015).

The mining process comes from the necessity of calculating a unique *hash* for each block that must also be coherent with the chain rules. For actual *hash* calculation, each block must contain the hash of its predecessor in the chain, to establish an unchangeable chronological link (Yaga *et al*., 2019). However, security standards indicate that not every hash value can be accepted in a blockchain because it has to accomplish a specific predefined format, which usually consists of a certain amount of zeros in the first digits of its string. For example, a valid mined *hash* should have the following form: "00004f2282f43794f303661243bf0dd7". Note that the first 4 digits are set to zero, meaning that the complexity of this chain can be considered as level 4. The lowest protection scenario is having one zero at the beginning of the String, which is known as complexity level number 1. *"The average work required is exponential in the number of zero-bits required and can be verified by executing a single hash*" (Nakamoto, 2008).

When an algorithm calculates a hash of a string, the result is always predictable. Therefore, a variable "*nonce*" value is added to the string to produce different results. The simplest way to do this is to run an iteration from the *nonce* value of zero, attach it to the internal content of the block, and then calculate its hash. If the pattern is not reached, the nonce is increased in one and then the process is repeated until one of the *hash* meets the expected format. This trial-and-error process is very heavy in terms of compute power (Schinckus, 2021), but this limitation is the very same reason why hackers are discouraged to attack blockchains. Also falsifying a single mined block is not enough to fool a blockchain because all the *hashes* have to be correspondent in terms of chronological signatures and *nonce* values. This time consuming task makes it very hard to complete a successful attack before the other security mechanisms in the server can take preventive actions.

The Computer Security Resource Center (NIST) of the U.S.A. only recommends the use of SHA hashing algorithms (Dang, 2015), specially *sha-2* and *sha-3*. The first version of *sha,* or *sha-1,* is already considered deprecated for critical mission applications due to its vulnerabilities and poor performance. *Sha-2* is based on the Merkle-Damgard function (Coron et al., 2005), similar to sha1 but implementing improvements in security and performance. These algorithms can work with different-bit ratios: 224, 256, 384, and 512-bits (Lee & Shin, 2018). The most recent *sha-3* method is based on the Keccack Sponge function (Bertoni *et al*., 2018), which is supposed to be slower but safer than *sha-2*. Also, *sha-3* can operate with the same-bit values. Depending on each implementation, it is possible that *sha-3* of 256-bits could perform similarly fast to *sha-2* of 512-bits. To understand these performance differences is the mean of this present article.

Methodology

The testing virtual machine had these hardware characteristics: an i7-10510U CPU of 1.8 GHz, and 4 Gb of RAM. The operating system was Ubuntu 22.04 with the Kernel Linux 5.15.0-50. The developing language was Java with JDK version 18. Each block was an instance of a Java class that contained six minimal attributes (Ismailisufı *et al.*, 2020) as described:

```
public class Block implements Serializable

{

    private int id;

    private int nonce;

    private long timeStamp;

    private String hash;

    private String previousHash;

    private ArrayList<Transaction> aTransactions;
}
```

Each block could store a nonlimited list of transactions, which were defined as the following class:

```
public class Transaction implements Serializable
{
    private int id
    private long timeStamp;
    private String sender;
    private String receiver;
    private double amount;
}
```

As standards indicate, the objects inherited serializable capabilities to be transmittable over P2P TCP-Sockets (Sapkota *et al.*, 2019). However, in this case, the mining process was tested without a cloud environment to measure the pure mining algorithm efficiency, without network *inter-processes* intervention. For this reason, the entire blockchain was constituted as an *ArrayList* of Blocks loaded in RAM, allowing to monitor of active memory consumption and also avoiding the appearance of hard drive bottlenecks when *paging* data to the *swap* area.

After developing the blockchain classes, a routine was coded for generating and populating a set of 1 thousand blocks, each containing 10 transactions; this amount was enough to obtain reliable comparative results (Birim *et al.*, 2021). This process was using 5 different hashing methods: MD5, *Sha-2* of 256-bits, *Sha-2* of 512-bits, *Sha-3 of* 256-bits, and *Sha-3* of 512-bits (Bae *et al.*, 2021). Every test was repeated 4 times, using complexity levels from 2 to 6. The elapsed time of every repetition was recorded for this study. Then, for a clean performance check of each method, it was executed a routine for validating the *hashes* of the entire blockchain, sequentially recalculating every block *hash* (Walker *et al.*, 2017) and comparing the resulting *nonce* values and the integrity of the chain of predecessor *hashes* (Ampel *et al.*, 2019). Alongside each test, the CPU load was obtained as a percentage metric by using the command *mpstat*, and the use of RAM was determined in terms of *available memory* (in Gb) and *active memory* (also in Gb), using the Kernel file */proc/meminfo.*

The data from repetitions were tabulated for comparison purposes, then statistically and graphically analyzed to determine which *hash* method performed better across the levels of complexity (Sumagita *et al.*, 2018), to establish recommendations on *hashing* strategies according to possible blockchain uses. The method used for mining consisted in using a plain text transcription of every attribute contained in each block, concatenated with a *nonce* for calculating a hash value, repeating the cycle until finding the desired proof of work. When the mining process was finished, the recently mined block (named 'candidate') is registered in the blockchain by the main controller Class, as shown in the next source code:

```
 public void mineBlock(Block bCandidate)

 {

   String cad= this.blockChain.get(bCandidate.toString());

   String sHash="";

   Sring sNonce="";

   int nonce=0;

   while(true)

   {

     sNonce=Integer.toString(nonce);

     sHash=this.generateHash(cad+sNonce);

     if(sHash.subSequence(0,                complexity).equals(this.proofOfWork))

     {

       bCandidate.include(nonce, sHash);

       this.blockChain.register(bCandidate);

       break;

     }

     nonce++;

   }

 }
```

The function that generated the *hashes* (presented in the next source code) received the *String* of the block and created an array of bytes using a *MessageDigest* instance that calculated the data according to the provided encryption method. The array was then converted in a Hexadecimal format to be finally formatted into a readable String.

```
private String generateHash(String pStr, String EncryptMethod)
```

```
{
  try
  {
    MessageDigest digest =
        MessageDigest.getInstance(EncryptMethod);
    byte[] hash = digest.digest(pStr.getBytes("UTF-8"));
    StringBuffer hexadecimalString = new StringBuffer();
    for (int i = 0; i < hash.length; i++)
    {
      String hexadecimal =
        Integer.toHexString(0xff & hash[i]);
      if (hexadecimal.length()==1)
          hexadecimalString.append('0');
       hexadecimalString.append(hexadecimal);
    }
    return hexadecimalString.toString();
  }
  catch(Exception ee) return null;
}
```

To corroborate that the mining process was correct, a procedure (next code) was executed for every mined block in an iteration of the entire blockchain, concatenating the string of the block with the previously calculated nonce to generate a new *hash,* which is compared with the *hash* of the previous block. If the two *hashes* concorded, then the proof fo work was considered as valid, if not, the procedure returns false to indicate that the entire blockchain was broken (*Shahriar & Mahmoud*, 2020). For this study, the elapsed time of this validation as also recorded.

```
    public boolean getProofOfWork_overBlock(Block pBlk,
    String EncryptMethod)
  {
        String sStr= pBlk.toString();
        String sNonce= Integer.toString(pBlk.getNonce());
        String sHash=this.generateHash(sStr+sNonce);
        if (sHash.equals(pBlk.getHash())) return true;
        else return false;
  }
```

The complete source code, developed with Apache Netbeans 13, can be freely downloaded from https://github.com/carlosm-sa/mining-performance

Results And Discussion

A chain of 1 thousand blocks was built specifically for each test. Table 1 shows how many bytes composed the *hash* of each algorithm. MD5 (version 2) created the smallest hashes, which were also the easiest to calculate. Hashing algorithms working in the modality of 256-bits returned hashes of 64 bytes, while those with the 512-bit mode, returned hashes of 128 bytes. Both SHA2 and SHA3 in 512-bits mode, showed more security potential because they are also more difficult to compute.

Table 1.

Byte length of the resulting hash values

	Hashing method				
	MD5	SHA2-256	SHA2-512	SHA3-256	SHA3-512
Bytes	32	64	128	64	128

Each algorithm was tested with different levels of complexity. For the first run, the proof of work of two characters was enforced for the *hashing* methods of MD5, SHA2-256, SHA2-512, SHA3-256, and SHA3-512. This complexity level was named “C-2”. The other levels of three, four, five, and six characters, were respectively named “C-3”, “C-4”, ”C-5” and “C-6”. Table 2 shows the maximum amount of used RAM memory consumed per each of them while generating the blockchain of 1 thousand blocks. In fact, those values are the average of the real RAM peaks obtained through the test of every hashing method. It is known that the Linux capabilities for caching and sharing memory can lead to misinterpretations of how much “space” is really used by a process, however, the values returned by *meminfo* offer a general idea of how much load is produced. In this case, the trend was that given the greater complexity, also the greater use of RAM. Note that C-5 and C-6, demanded a lot of resources for a chain of 1 thousand blocks, meaning that the compute costs in a real-life application would be very expensive and could only be afforded in cases of extreme security requirements for those businesses of high revenue that impose security over cost (Albayati *et al.*, 2020).

Table 2.

Maximum RAM usage while mining

	Complexity level				
	C-2	C-3	C-4	C-5	C-6
RAM peak	21.9 Mb	52.3 Mb	251.89 Mb	538.73 Mb	1.44 Gb

Note: each value is the average of the maximum peaks obtained when running the test of each hashing method, configured with the indicated complexity level.

Fig. 1 shows the exponential increase in maximum memory usage when testing different levels of complexity; regression analysis was not required since each graphed level was a discrete variable. According to these results, it would be necessary to improve RAM capacity in servers when the proof of work method is hardened for security means.

The validity of all the generated blockchains was verified by a second and separated calculation, but this time recreating the hash by concatenating the block representing *String* with the nonce value (Puthal *et. al.*, 2018). This verification requires only one iteration and therefore is much lighter than the whole mining process executed when creating new blocks, an action that required thousands to billions of iterations. In Fig. 2, it is shown that the average time required to verify the whole blockchain, was very similar for complexity levels of C-2, C-3, C-4, and C-5 (see tables 3 to 7). On the contrary, the value increased greatly when computing with C-6 for any algorithm.

The blockchain-creating process using C-2 was the fastest test, requiring about a quarter million iterations for each hashing method and up to 5 milliseconds to generate each block (Table 3). This high efficiency is at expense of chain security, which is not recommended for applications where information integrity is critical. Level C-3 constituted a more reliable configuration (Table 4), requiring about 4 million iterations to generate the sample blockchain, and the block mining time took from 13 to 52 milliseconds. Although these measurements do not seem to be significant, this means that the fastest method MD5 would probably consume about 3.6 hours to mine one million blocks in a real-life application, and the heaviest method SHA3-512 would require about 14.4 hours. This difference is very significant and clearly indicates that every mining method can have a different impact on both security and efficiency. For C-4, more than 60 million iterations were required to mine each block and elapsed times from 183 to 838 milliseconds (Table 5). This means that, for example, mining a blockchain with one million elements would last from 50 to 233 hours. When using C-5, the security was improved because about a billion iterations were required to perform the test (Mingxiao *et. al.*, 2017), having approximate mining times from 2.5 to 17.5 seconds per block. In the case of the best method for encryption security, SHA3-256, generating the test of a thousand blocks took more than 4 hours and one million blocks could take up to 5 thousand hours or more than 200 days of computer processing. However, it is important to notice that the proof of work of the entire blockchain test did not take much time regardless of the hashing method. For C-6 (Table 7), the tests consumed more time due to the requirement of more than 10 billion iterations and a few minutes of elapsed time per block; if one million blocks were to be generated, the process would last about 8 years and this invalidate this alternative for the use of common organizations (Yusoff *et al.*, 2022).

Table 3.

Mining test results, using complexity level of 2 characters.

	Hashing method				
	MD5	SHA2-256	SHA2-512	SHA3-256	SHA3-512
Iterations	259,060	235,921	254,267	242,535	263,058
Proof of Work	0.017 s	0.013 s	0.013 s	0.014 s	0.020 s
BMT	0.002 s	0.003 s	0.003 s	0.003 s	0.005 s

Notes: The decimal separator is the point and the thousands separator is the comma. BMT stands for 'block mining time'. The same applies for Tables 4 to 7.

Table 4.

Mining test results, using complexity level of 3 characters.

	Hashing method				
	MD5	SHA2-256	SHA2-512	SHA3-256	SHA3-512
Iterations	4,242,809	4,163,061	4,217,957	3,979,857	4,081,028
Proof of Work	0.013 s	0.017 s	0.016 s	0.014 s	0.021 s
BMT	0.013 s	0.030 s	0.026 s	0.032 s	0.052 s

Table 5.

Mining test results, using complexity level of 4 characters.

	Hashing method				
	MD5	SHA2-256	SHA2-512	SHA3-256	SHA3-512
Iterations	67,948,703	65,845,949	67,982,908	65,054,242	66,335,940
Proof of Work	0.012 s	0.016 s	0.012 s	0.014 s	0.021 s
BMT	0.183 s	0.443 s	0.347 s	0.476 s	0.838 s

Table 6.

Mining test results, using complexity level of 5 characters.

	Hashing method				
	MD5	SHA2-256	SHA2-512	SHA3-256	SHA3-512
Iterations	995,404,013	1,019,771,696	1,033,485,474	1,012,483,152	1,055,056,190
Proof of Work	10 s	13 s	13 s	17 s	28 s
BMT	2.456 s	5.673 s	5.931 s	9.711 s	17.480 s

Table 7.

Mining test results, using complexity level of 6 characters.

	Hashing method				
	MD5	SHA2-256	SHA2-512	SHA3-256	SHA3-512
Iterations	12,329,239,600	13,764,806,500	11,346,805,800	15,032,843,700	14,349,920,600
Proof of Work	124 s	175 s	143 s	252 s	381 s
BMT	36.154 s	90.388 s	66.695 s	137.983 s	248.370 s

The results of the processing time of the five methods per each complexity level, obtained after generating the test blockchain, are shown in Fig. 1. The cluster of graphs allows the rapid comparative appreciation of the average performance of every test. C-2 had the lowest elapsed time and C-6 the highest, but the progression between levels was not linear. Every time a new character was added for proof of work, the load for processing increased exponentially by about ten times (Gui *et. al.*, 2017), which is the reason why the axes of graphs is logarithmic. This has a direct implication on real applications design because each character added to the proof of work will delay the process ten times, therefore, important investments in computing capabilities might be required to compensate for the load. MD5 was the simplest algorithm (Ghoshal *et. al.*, 2020) but if it is configured to operate with C-5, it can still perform faster than other algorithms while its inherent weaknesses could be mitigated due to the elevated exigences of the proof of work. Another scenario is that SHA3-256 might be considered the best option because it is a strong algorithm (Alexan *et. al.*, 2021), but its use should be restricted to C-3 because this algorithm is slow. SHA2-256 can operate with C-4 with similar computing costs to the previously mentioned, but with enhanced mining security. Even if SHA2 is theoretically weaker than SHA3 (Sharma & Khanum, 2022), stronger levels of complexity can offer better mining security. The validity of the test blockchain was verified by executing a new proof of work, but this time concatenating their string value with the previously obtained nonce to recalculate the hash in one single step. Results (Fig. 4) indicated that MD5, SHA2-256, SHA2-512, and SHA3-256 performed very similarly and the cost validating is very low for C-2, C-3, and C-4. However, C-5 and mostly C-6 had a very high compute cost and resulted to be too slow. C-4 was the best in terms of its cost-benefit ratio.

The indicator of algorithm efficiency was determined by the average number of iterations required for generating a new block (Table 8). For MD5 and SHA2-256, the most efficient execution was C-5; for SHA-512, SHA3-256, and SHA3-512, it was C-4. However, even if the first two methods performed better with a higher level of complexity, C-5 still requires a lot more iterations than C-4. This last complexity level demonstrated the best relationship between the benefit of security and the cost of time consumption.

According to the results obtained, the best options for fast processing and acceptable security were MD5 and SHA2-512 when operating complexity level of 3 (at least) or 4 (best). SHA2-256 produced shorter hashes and performed slower than SHA2-512. SHA3-256 and SHA3-512 performed slower than the other methods, but are recommended for applications where security is crucial, due to their strong hashing capabilities. In general terms, complexity levels of 5 and 6 characters, are too heavy to be used in common applications and are only recommended for extreme scenarios where security is the top priority and the volume of transactions is low enough to allow waiting a few minutes for each block to be mined.

Table 8.

Hash iterations per second

	Hashing method				
	MD5	SHA2-256	SHA2-512	SHA3-256	SHA3-512
C-2	160,608	91,442	97,720	82,327	54,964
C-3	332,665	139,485	161,936	122,631	78,507
C-4	370,676	148,637	**195,916**	**136,562**	**79,170**
C-5	**405,329**	**179,762**	174,251	104,263	60,359
C-6	341,025	152,285	170,131	108,947	57,776

Note: The highlighted values are those representing the best average performance per method.

Conclusion

Information security is a topic of permanent relevance. It is achieved through virtual mechanisms that usually require considerable computing power. Evaluating the performance of various hashing algorithms and their different configurations can help organizations to make appropriate decisions when sizing the mining load that a blockchain-based computer system may incur. Since each hashing method can operate with different levels of difficulty, it is necessary to choose appropriately the number of characters that will be validated during a proof of work to keep applications efficient in the long term after deployment. The mining methods must be carefully tested and selected according to the nature and scope of the business on which they will be implemented, taking into count realistic criteria of security needs, processing times, and investment budget.

Declarations

Ethical Approval: Not applicable because there was not conducted any research on humans or animals.

Competing interests: There are no particular conflicts of interest with the funding of this article. It was developed as part of the academic duties

Authors' contributions: Carlos Roberto Martínez Martínez, author of the complete article.

Funding: There was no funding received

Availability of data and materials: The data obtained can be downloaded from https://drive.google.com/file/d/1xL_F5b6fDZRJzY_reJaEKbL_Htz-4UZJ/view?usp=share_link. The complete source code, developed with Apache Netbeans 13, can be freely downloaded from https://github.com/carlosm-sa/mining-performance

References

1. Albayati, H., Kim, S. K., & Rho, J. J. (2020). *Accepting financial transactions using blockchain technology and cryptocurrency: A customer perspective approach.* Technology in Society, 62, 101320.
2. Alexan, W., Ashraf, A., Mamdouh, E., Mohamed, S., & Moustafa, M. (2021). *Iomt security: Sha3-512, aes-256, rsa and lsb steganography.* In 2021 8th NAFOSTED Conference on Information and Computer Science (NICS) (pp. 177-181). IEEE.
3. Ampel, B., Patton, M., & Chen, H. (2019). *Performance modeling of hyperledger sawtooth blockchain*. In 2019 IEEE International Conference on Intelligence and Security Informatics (ISI) (pp. 59-61). IEEE.
4. Bae, Y. S., Park, Y., Kim, T., Ko, T., Kim, M. S., Lee, E., ... & Yoon, H. J. (2021). *Development and Pilot-Test of Blockchain-Based MyHealthData Platform.* Applied Sciences, 11(17), 8209.
5. Bertoni, G., Daemen, J., Peeters, M., Assche, G. V., Keer, R. V., & Viguier, B. (2018, July). *K angarooT welve: Fast Hashing Based on Keccak*. In International Conference on Applied Cryptography and Network Security (pp. 400-418). Springer, Cham.
6. Birim, M., Ari, H. E., & Karaarslan, E. (2021). *GoHammer Blockchain Performance Test Tool.* Journal of Emerging Computer Technologies, 1(2), 31-33.
7. Blake, R. H., & Mangiameli, P. (2008, January). *The Effects and Interactions of Data Quality and Problem Complexity on Data Mining.* In ICIQ (pp. 160-175).
8. Chand, P., Thakkar, J. J., & Ghosh, K. K. (2018). *Analysis of supply chain complexity drivers for Indian mining equipment manufacturing companies combining SAP-LAP and AHP.* Resources Policy, 59, 389-410.
9. Coron, J. S., Dodis, Y., Malinaud, C., & Puniya, P. (2005, August). *Merkle-Damgård revisited: How to construct a hash function. In Annual International Cryptology Conference (pp. 430-448).* Springer, Berlin, Heidelberg.
10. Dang, Q. H. (2015). *Secure hash standard*. Computer Security Resource Center, NIST. USA. https://csrc.nist.gov/publications/detail/fips/180/4/final
11. Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., & Capkun, S. (2016). *On the security and performance of proof of work blockchains*. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (pp. 3-16).
12. Ghoshal, S., Bandyopadhyay, P., Roy, S., & Baneree, M. (2020). *A journey from md5 to sha-3.* Trends in Communication, Cloud, and Big Data, 107-112.
13. Gui, J., Liu, T., Sun, Z., Tao, D., & Tan, T. (2017). *Fast supervised discrete hashing*. IEEE transactions on pattern analysis and machine intelligence, 40(2), 490-496.
14. Guzman, L. B., Sison, A. M., & Medina, R. P. (2018, September). *MD5 secured cryptographic hash value*. In Proceedings of the 2018 International Conference on Machine Learning and Machine Intelligence (pp. 54-59).
15. Hassib, E. M., El-Desouky, A. I., El-Kenawy, E. S. M., & El-Ghamrawy, S. M. (2019). *An imbalanced big data mining framework for improving optimization algorithms performance*. IEEE Access, 7, 170774-

170795.

16. Herlihy, M. (2019). *Blockchains from a distributed computing perspective.* Communications of the ACM, 62(2), 78-85.
17. Huck, K. A., & Malony, A. D. (2005, November). *Perfexplorer: A performance data mining framework for large-scale parallel computing.* In SC'05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing (pp. 41-41). IEEE.
18. Ismailisufı, A., Popović, T., Gligorić, N., Radonjic, S., & Šandi, S. (2020, February). *A private blockchain implementation using multichain open source platform.* In 2020 24th International Conference on Information Technology (IT) (pp. 1-4). IEEE.
19. Kiayias, A., & Zindros, D. (2019). *Proof-of-work sidechains*. In International Conference on Financial Cryptography and Data Security (pp. 21-34). Springer, Cham.
20. Klinkmüller, C., Ponomarev, A., Tran, A. B., Weber, I., & Aalst, W. V. D. (2019). *Mining blockchain processes: extracting process mining data from blockchain applications.* In International Conference on Business Process Management (pp. 71-86). Springer, Cham.
21. Lee, S. H., & Shin, K. W. (2018, January). *An efficient implementation of SHA processor including three hash algorithms (SHA-512, SHA-512/224, SHA-512/256)*. In 2018 International Conference on Electronics, Information, and Communication (ICEIC) (pp. 1-4). IEEE.
22. Mingxiao, D., Xiaofeng, M., Zhe, Z., Xiangwei, W., & Qijun, C. (2017). *A review on consensus algorithm of blockchain.* In 2017 IEEE international conference on systems, man, and cybernetics (SMC) (pp. 2567-2572). IEEE.
23. Misra, S., Mukherjee, A., Roy, A., Saurabh, N., Rahulamathavan, Y., & Rajarajan, M. (2020). *Blockchain at the edge: Performance of resource-constrained IoT networks.* IEEE Transactions on Parallel and Distributed Systems, 32(1), 174-183.
24. Nakamoto, S. (2008). *A peer-to-peer electronic cash system.* https://bitcoin.org/bitcoin.pdf.
25. Nugroho, K. A., Hangga, A., & Sudana, I. M. (2016, October). *SHA-2 and SHA-3 based sequence randomization algorithm.* In 2016 2nd International Conference on Science and Technology-Computer (ICST) (pp. 150-154). IEEE.
26. Pika, A., ter Hofstede, A. H., Perrons, R. K., Grossmann, G., Stumptner, M., & Cooley, J. (2021). *Using big data to improve safety performance: an application of process mining to enhance data visualisation*. Big Data Research, 25, 100210.
27. Puthal, D., Malik, N., Mohanty, S. P., Kougianos, E., & Das, G. (2018). *Everything you wanted to know about the blockchain: Its promise, components, processes, and problems.* IEEE Consumer Electronics Magazine, 7(4), 6-14.
28. Sapkota, H., Murukannaiah, P. K., & Wang, Y. (2019). *A network-centric approach for estimating trust between open source software developers*. Plos one, 14(12), e0226281.
29. Schinckus, C. (2021). *Proof-of-work based blockchain technology and Anthropocene: An undermined situation?*. Renewable and Sustainable Energy Reviews, 152, 111682.

30. Shahriar, S., & Mahmoud, Q. H. (2020). *Improving transaction speed and scalability of blockchain systems via parallel proof of work*. Future internet, 12(8), 125.
31. Sharma, S., & Khanum, S. (2022). *Performance analysis of SHA 2 and SHA 3*. Saba, Performance analysis of SHA, 2.
32. Sumagita, M., Riadi, I., Sh, J. P. D. S., & Warungboto, U. (2018). *Analysis of secure hash algorithm (SHA) 512 for encryption process on web based application*. International Journal of Cyber-Security and Digital Forensics (IJCSDF), 7(4), 373-381.
33. Vukolić, M. (2015). *The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication.* In International workshop on open problems in network security (pp. 112-125). Springer, Cham.
34. Walker, M. A., Dubey, A., Laszka, A., & Schmidt, D. C. (2017, December). *Platibart: a platform for transactive iot blockchain applications with repeatable testing.* In Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things (pp. 17-22).
35. Wang, Y., Zhao, Q., Jiang, L., & Shao, Y. (2010). *Ultra high throughput implementations for MD5 hash algorithm on FPGA.* In High Performance Computing and Applications (pp. 433-441). Springer, Berlin, Heidelberg.
36. Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2019). *Blockchain technology overview.* arXiv preprint arXiv:1906.11078.
37. Yusoff, J., Mohamad, Z., & Anuar, M. (2022). *A Review: Consensus Algorithms on Blockchain.* Journal of Computer and Communications, 10(9), 37-50.

Figures

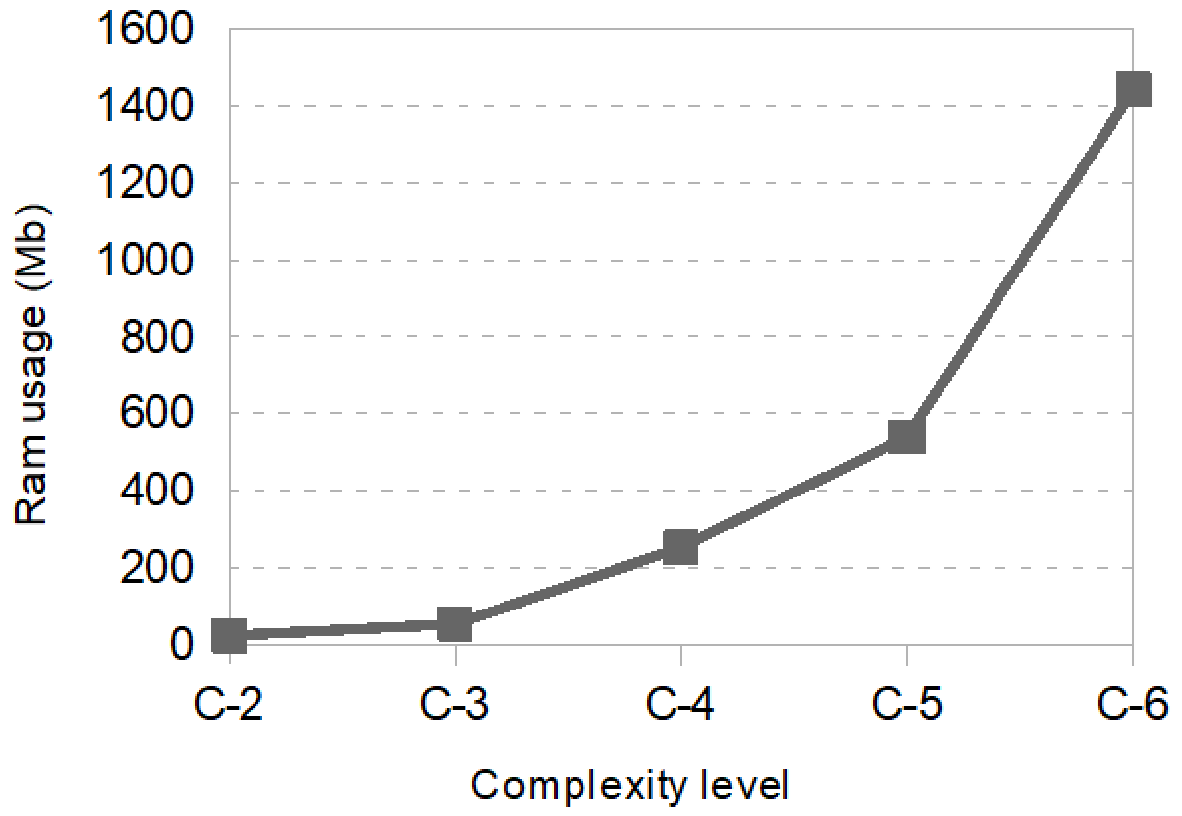

Figure 1

Memory peak while processing each complexity level.

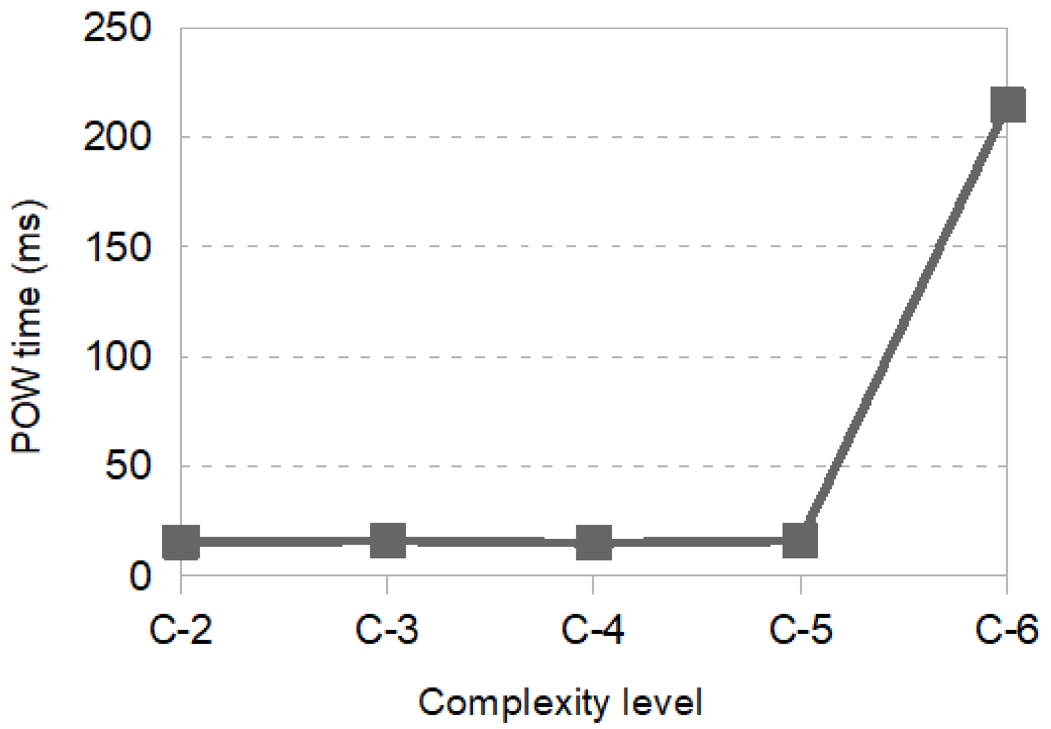

Figure 2

Calculation time proof of work, for each complexity level

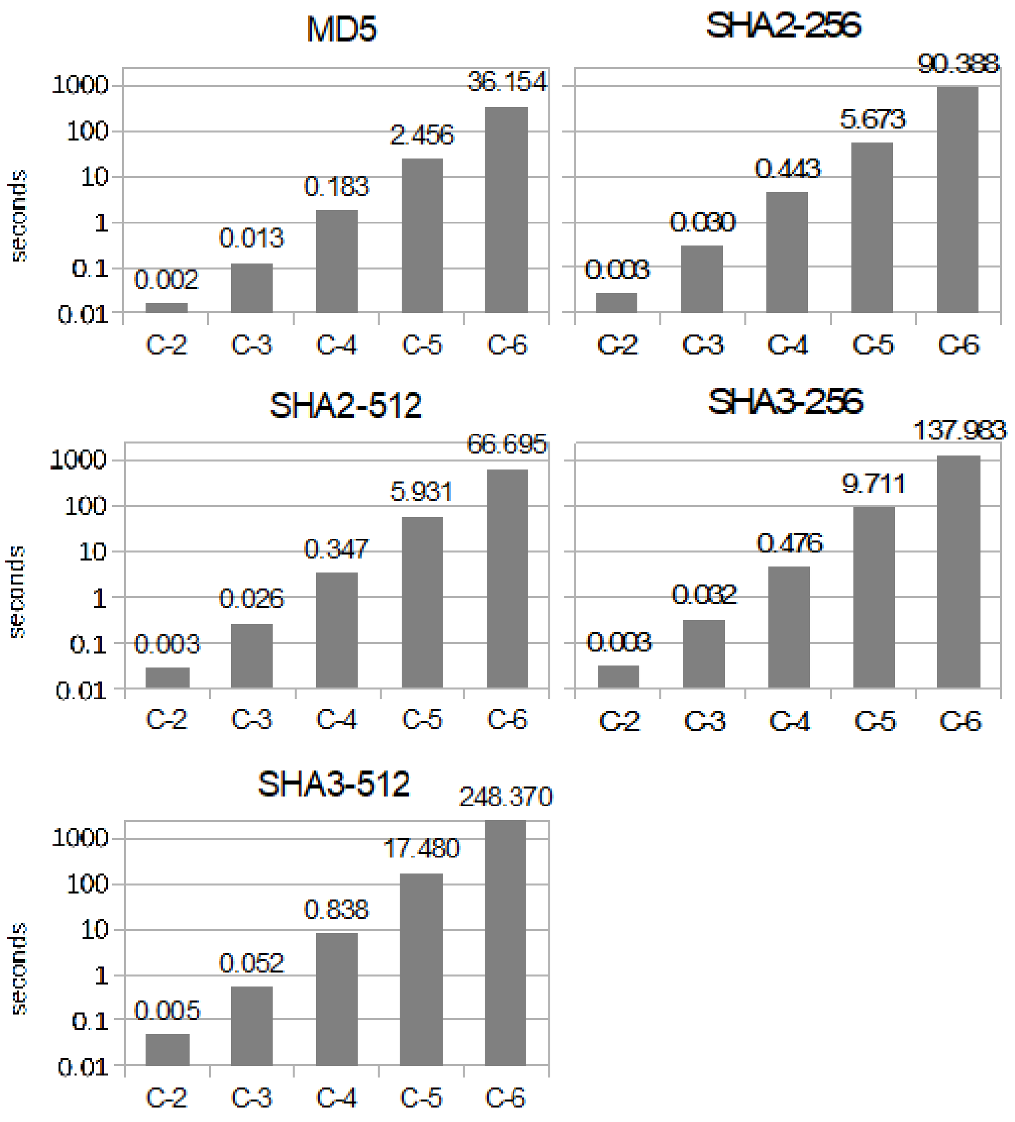

Figure 3

Block mining time vs. level of complexity. The scale is in logarithmic progression.

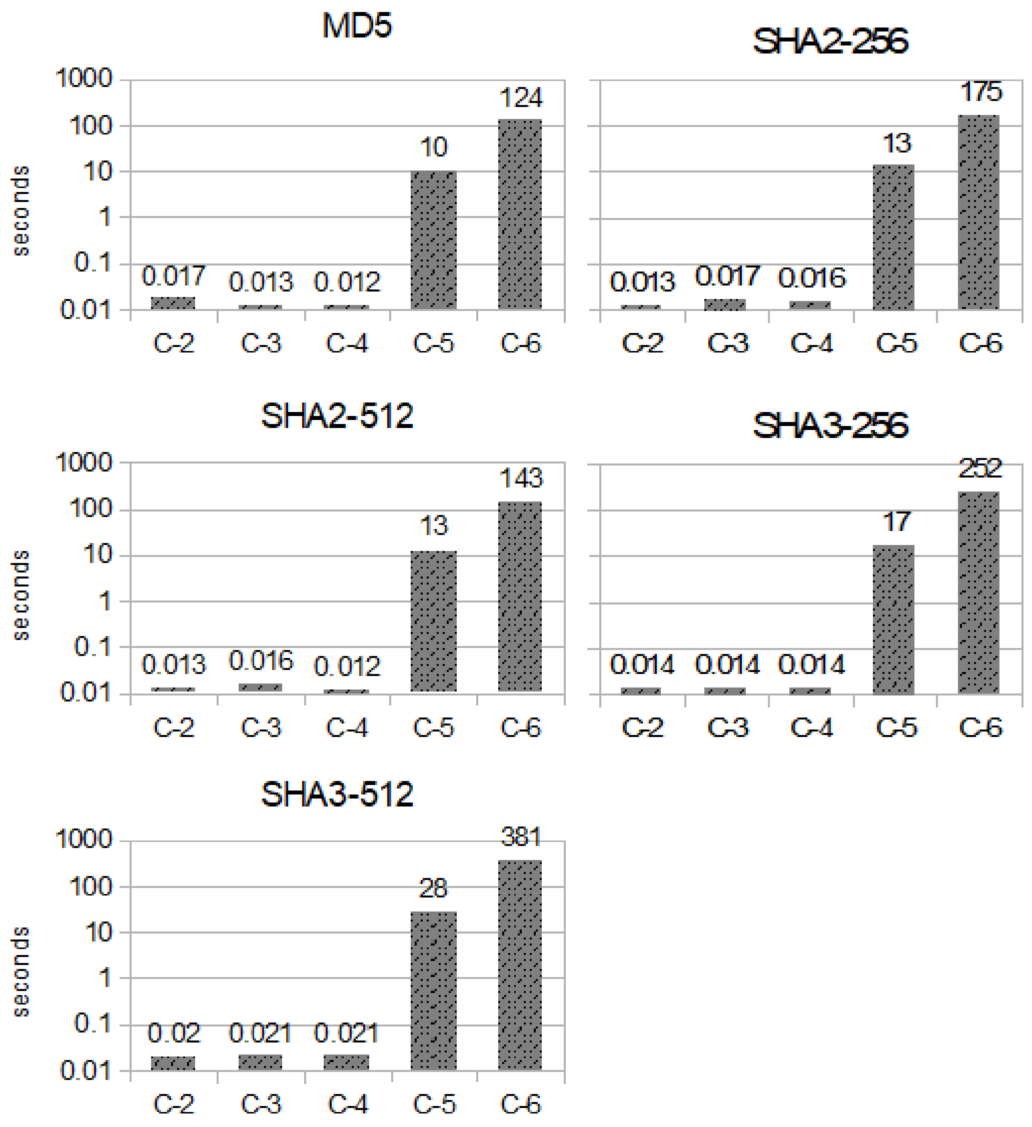

Figure 4

'Proof of work' mining time of the test blockchain vs. level of complexity. The scale is in logarithmic progression.