Ethereum Blockchain using AES-CMAC

Kunika Mathur[1,*], and *Nandini* K[1]
[1]Dept. of Computer Science and Engineering, Dayananda Sagar University, Bangalore, India

Abstract. Managing and storing data are crucial tasks for any industry, since they require accurate and secure record-keeping. Using blockchain technology has significantly increased recently due to its capability to address some of the challenges that come with traditional data storage and management systems. In this paper, we will explore the use of Ethereum smart contracts powered by the Solidity programming language using AES-CMAC.

1. Introduction

Blockchain is a decentralized, immutable, and transparent ledger that is used to store information securely and in a tamper-proof manner. This is based on a network of nodes which collectively maintain the integrity of the data stored. Using blockchain has many benefits like improving data privacy, reducing data breaches, enhancing data interoperability, and enabling secure data sharing between users.

Ethereum is a blockchain platform which enables creating and deploying smart contracts. Smart contracts are self-executing contracts which can be automatically executed when certain constraints are satisfied. Smart contracts enable creating decentralized applications operating autonomously without intermediaries. Smart contracts over Ethereum are written using Solidity programming language.

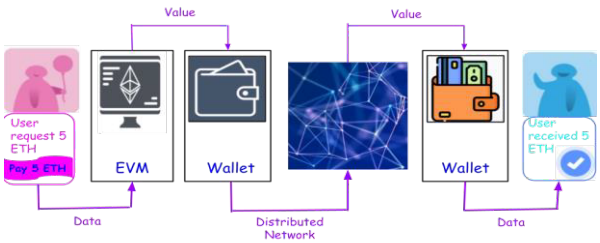

Fig. 1. Ethereum procedure

Figure 1 gives a diagrammatic view of the procedure of Ethereum.

*Kunika Mathur: kannulpal051101@gmail.com

2. Related work

Blockchain is a shared ledger technology which can be used to transfer bitcoins. It also finds use in various other domains such as e-voting system, government, health care etc. The security of transactions is a major concern these days. The blockchain network comes with full-fledged security features and hence are being applied everywhere.[1] One field where blockchain technology has tremendous potential is healthcare, due to the need for a more patient-centric approach to healthcare systems and to connect disparate systems and increase the accuracy of electronic healthcare records (EHRs). In one systematic review, a state-of-the-art blockchain research in the field of healthcare is analysed.The findings show that blockchain technology research in healthcare is increasing and it is mostly used for data sharing, managing health records and access control. Most research is aimed at presenting novel structural designs in the form of frameworks, architectures or models.[2] As blockchain was firstly used for Bitcoin, ongoing research is extended to its non-financial applications.[3] This paper deals with the Ethereum Blockchain and the use of AES-CMAC instead of hashing to check the integrity of the data.

3. Methodology

3.1 Blockchain

Blockchain and distributed ledger technologies are similar in that they both rely on decentralized networks to maintain the integrity of data, while databases are typically centralized or distributed in nature. Blockchain and distributed ledger technologies provide a higher level of security due to their use of cryptographic algorithms, while databases rely on access control mechanisms. Table 1 provides a clearer comparison of blockchain, distributed ledger and database.

Table 1. Feature comparison

Feature	**Blockchain**	**Distributed Ledger**	**Database**
Data Distribution	Decentralized	Decentralized	Centralized or Distributed
Data Immutability	Immutable	Immutable	Mutable
Data Access	Public or Private	Public or Private	Private or Public
Consensus	Proof of Work, Proof of Stake, or other consensus algorithms	Various consensus algorithms, including those used in blockchains	Consensus mechanisms that depend on the type of database

*Kunika Mathur: kannulpal051101@gmail.com

Trust	Trustless system	Trusted network	Requires trust between parties involved
Security	High security due to cryptographic algorithms used	High security due to distributed nature and cryptographic algorithms	Relies on access control mechanisms, vulnerable to cyberattacks
Use Cases	Cryptocurrency transactions, supply chain management, smart contracts	Supply chain management, digital identity, voting systems	Data management, enterprise resource planning (ERP), customer relationship management (CRM)

Fig. 2. Ethereum price chart

The above figure (fig. 2) shows the increase in the price of Ethereum which shows its increasing popularity.

In this paper we implement Ethereum with smart contracts and metamask plugin along with AES-CMAC in place of hashing.Using AES-CMAC in Ethereum blockchain can provide an additional layer of security to the transactions being executed on the blockchain. It can help in preventing unauthorized access and tampering of data on the blockchain. By using smart contracts, the execution of AES-CMAC can be automated, making the process more efficient and secure. Metamask can be used to interact with the smart contract, making it easier for users to execute the AES-CMAC algorithm on the blockchain. We will look into these in detail in the following sections.

3.2 Storing Information using Ethereum Smart Contracts

*Kunika Mathur: kannulpal051101@gmail.com

Storing information on a blockchain-powered by Ethereum smart contracts will provide many advantages like data privacy, security, and transparency. Using smart contracts will enable creating a decentralized system which eliminates the need for intermediaries, hence reducing risk of data breaches and unauthorized access. The storage of information on the Ethereum blockchain involves the following steps:

1. Data Encryption: Before data is stored on the blockchain, we encrypt it using a strong encryption algorithm which will ensure the privacy and security of data. The encryption key is then stored off-chain and will be accessible only to those that are authorized.
2. Smart Contract Creation: A smart contract is created using Solidity. The smart contract will define the rules and conditions that are required to store and access user information.
3. Data Storage: The encrypted user information is then stored on Ethereum blockchain using smart contract. This ensures that only those that are authorized are able to access the information.
4. Data Retrieval: Those that are authorized can then retrieve user information from the blockchain through the appropriate methods of the smart contract. The smart contract will verify the identity of those requesting the information and will ensure that they have the permissions required to access the data.

Figure 3 provides a brief overview of the working of a smart contract.

Fig. 3. Working of a smart contract

3.3 Metamask

Metamask is a cryptocurrency wallet to interact with the Ethereum blockchain. Once installed, Metamask provides users with a unique Ethereum address that they can use to receive and send Ethereum. It is also used to interact with decentralized apps that are built on Ethereum blockchain.

*Kunika Mathur: kannulpal051101@gmail.com

To transfer Ethereum using Metamask, you can click on the "Send" button in Metamask, enter the receiver's Ethereum address, and specify the amount you want to send. Figure 4 shows how Metamask acts as a link between users and Ethereum.

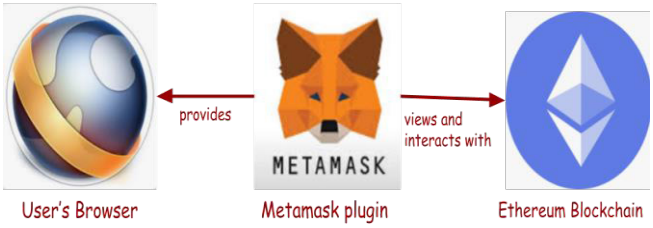

Fig. 4. Metamask block chain

3.4 AES-CMAC

AES-CMAC is a cryptographic algorithm that is used for message authentication. It is based on the Advanced Encryption Standard (AES) block cipher and helps to provide a secure way that verifies the authenticity of a message or data. It can be used instead of hashing to maintain and check data integrity. The AES-CMAC algorithm is as below:

AES-CMAC Algorithm:

Input: Message to be authenticated (M), Secret key (K), Block size of the cipher (n)

Output: Authentication tag (T)

---

1. If the length of the message is less than n bytes, append a single "1" bit and pad the message with "0" bits until it reaches n bytes. If the length of the message is equal to n bytes, append a single "1" bit and pad the message with a block of "0" bits.
2. Divide the padded message into n-byte blocks, where the last block may need to be padded with "0" bits to reach n bytes.
3. Generate two subkeys: K1 and K2, by encrypting a block of all zeros with the secret key using the AES cipher.
4. Initialize a block of all zeros as the initial MAC value, M0.
5. For each block of the message, XOR it with the current MAC value, Mi-1, and encrypt the result using the secret key and the AES cipher to generate a new MAC value, Mi.
6. If the last block of the message is complete (i.e., n bytes), XOR it with K1; otherwise, XOR it with the last n bytes of K2, which is generated by encrypting K1 with the AES cipher.

*Kunika Mathur: kannulpal051101@gmail.com

7. XOR the final MAC value, Mn, with K1 to obtain the final authentication tag, T.

The authentication tag generated by the above algorithm is verified using the MAC verification algorithm.

4. Result

The MAC verification algorithm is used to ensure the authenticity of a message by comparing the computed MAC of the message with the given authentication tag.

MAC verification Algorithm:
Input: Message to be authenticated (M), Authentication tag (T), Secret key (K), MAC generation algorithm (MAC_gen)
Output: Authenticity of the message

1) Compute the MAC of the message using the secret key and the MAC generation algorithm, and compare it to the given authentication tag.
2) If the computed MAC matches the given authentication tag, then the message is authentic; otherwise, the message has been tampered with and should be rejected.

The MAC generation algorithm used for verification must be the same as the one we use for generating the authentication tag. It is also important to make sure that the secret key is kept secure and is not disclosed to any unauthorized parties.

Table 2 below discusses the differences between AES-CMAC and hashing in terms of their purpose, key requirement, security, input and output sizes, performance, and applications.

Table 2. AES-CMAC vs hashing

Criteria	**AES-CMAC**	**Hashing**
Purpose	Message authentication code (MAC)	One-way data integrity check
Key requirement	Requires a secret key for generation and verification of MAC	No key is required
Security	Provides strong security against message forgery and tampering	Vulnerable to collision attacks
Input size	Can handle input of any length	Typically limited to a fixed input size
Output size	Fixed output size (typically 128 bits)	Variable output size

*Kunika Mathur: kannulpal051101@gmail.com

Performance	Slower than hashing due to key generation and encryption	Faster than AES-CMAC, but slower than simple hash functions
Applications	Secure messaging, file transfer, and authentication protocols	Password storage, digital signatures, and data validation

Next we compare AES-Hash and AES-CMAC algorithms in detail and compares their performance in terms of execution time using Python.

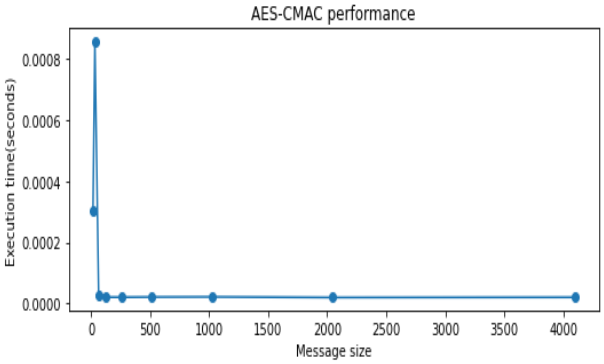

Fig. 5. AES-CMAC performance

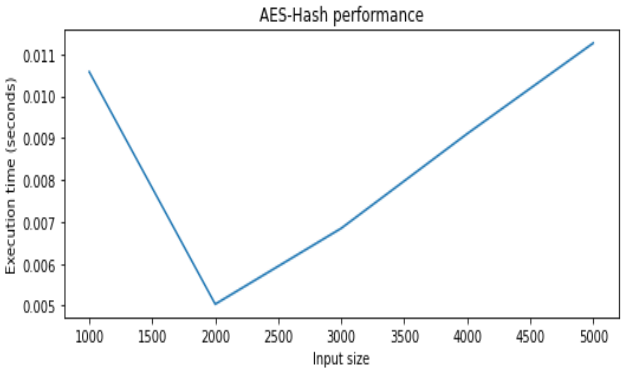

Fig. 6. AES-Hash performance

*Kunika Mathur: kannulpal051101@gmail.com

The plot in fig. 5 shows how the time taken by the aes-cmac algorithm and the plot in fig. 6 shows how the time taken by the aes-hash algorithm as the input message size grows. The aes-hash algorithm is as below:

AES-Hash Algorithm:
Inputs: message (a byte string containing the mssage to be hashed), key (a byte string containing the secret key for the hash function)

Outputs: hash value (a byte string containing the hash of the message)

---

1. Pad the message to a multiple of 128 bits with the bit string 100...0, followed by the minimum number of zero bits such that the length of the padded message is congruent to 112 modulo 128.
2. Split the padded message into 128-bit blocks.
3. Initialize an empty byte string hash_value.
4. For each block in the padded message, encrypt the block using the AES block cipher in ECB mode with the secret key.
5. XOR the resulting ciphertext with the previous hash value, if one exists. If there is no previous hash value, set the hash value to the ciphertext.
6. Return the final hash value as the output.

On implementing these functions in python with input size in bytes, we see that the execution time for the AES-CMAC is more consistent and lesser than AES-hash. Hence, we can infer that the AES-CMAC is more efficient. Furthermore, we also compare the performance of AES-CMAC with Ethereum and Hyperledger blockchain.

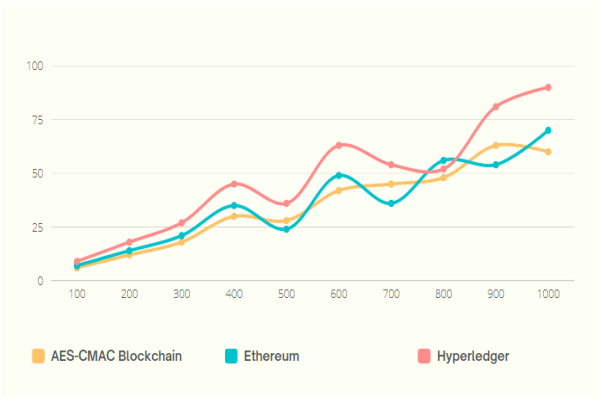

Fig. 7. Comparison of AES-CMAC, Ethereum and Hyperledger

From fig. 7, we see that expected performance for Ethereum blockchain using AES-CMAC is similar compared to usual Ethereum blockchain. But, AES-CMAC is advantageous as it provides more security compared to the regular hashing. The main use for AES-CMAC is to provide the integrity and authenticity for data that is transmitted. It can be used to make sure

*Kunika Mathur: kannulpal051101@gmail.com

that a message has not been modified during its transmission. Therefore, AES-CMAC can be reliable along with being an efficient way to make sure that the data's integrity as well as authenticity is maintained, which is essential in blockchain and hence AES-CMAC can be used in a manner similar to hashing.

5. Conclusion

Using blockchain technology powered by Ethereum smart contracts provides several advantages to store information. Being decentralized and having a secure nature, blockchain provides an efficient and reliable system to store and manage information. Using smart contracts eliminates the need for intermediaries, hence reducing the risk of data breaches and unauthorized access. Also, the use of AES-CMAC provides extra security. Hence, implementing such a system will provide a secure and efficient system to store and access information.

References

1. Singh, Shweta, Anjali Sharma, and Prateek Jain. "A Detailed Study of Blockchain: Changing the World." *International Journal of Applied Engineering Research* 13.14 (2018): 11532-11539.
2. Hölbl, Marko, et al. "A systematic review of the use of blockchain in healthcare." *Symmetry* 10.10 (2018): 470.
3. Agbo, Cornelius C., Qusay H. Mahmoud, and J. Mikael Eklund. "Blockchain technology in healthcare: a systematic review." *Healthcare*. Vol. 7. No. 2. MDPI, 2019.
4. Khan, Umair, Zhang Yong An, and Azhar Imran. "A blockchain ethereum technology-enabled digital content: development of trading and sharing economy data." *IEEE access* 8 (2020): 217045-217056.
5. Agrawal, Kanika, et al. "An Extensive Blockchain based Applications Survey: Tools, Frameworks, Opportunities, Challenges and Solutions." *IEEE Access* (2022).
6. Kushwaha, Satpal Singh, et al. "Systematic review of security vulnerabilities in ethereum blockchain smart contract." *IEEE Access* 10 (2022): 6605-6621.
7. Rajasekaran, Arun Sekar, Maria Azees, and Fadi Al-Turjman. "A comprehensive survey on blockchain technology." *Sustainable Energy Technologies and Assessments* 52 (2022): 102039.
8. Zheng, Zibin, et al. "An overview of blockchain technology: Architecture, consensus, and future trends." *2017 IEEE international congress on big data (BigData congress)*. Ieee, 2017.
9. Tülemez, Şafak, and Deniz Turgay Altilar. "Implementation of Code Integrity Check Module for RISC-V Architecture with AES." *2022 International Conference on Theoretical and Applied Computer Science and Engineering (ICTASCE)*. IEEE, 2022.

*Kunika Mathur: kannulpal051101@gmail.com

10. Monrat, Ahmed Afif, Olov Schelén, and Karl Andersson. "A survey of blockchain from the perspectives of applications, challenges, and opportunities." *IEEE Access* 7 (2019): 117134-117151.
11. Sasaki, Yu. "Meet-in-the-middle preimage attacks on AES hashing modes and an application to whirlpool." *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 96.1 (2013): 121-130.
12. Wohrer, Maximilian, and Uwe Zdun. "Smart contracts: security patterns in the ethereum ecosystem and solidity." *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 2018.
13. Jiao, Jiao, et al. "Semantic understanding of smart contracts: Executable operational semantics of solidity." *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020.
14. Song, Junhyuk, et al. *The aes-cmac algorithm*. No. rfc4493. 2006.
15. Daemen, Joan, and Vincent Rijmen. "AES proposal: Rijndael." (1999)

*Kunika Mathur: kannulpal051101@gmail.com