# Tamperproof Data Transmission to Offline IoT Devices in a Zero-Trust Environment

1st Richard Vogel
*Faculty Applied*
*Computer Sciences & Biosciences*
*Hochschule Mittweida*
09648 Mittweida, Germany
vogel2@hs-mittweida.de

2nd Robert Manthey
*Faculty Applied*
*Computer Sciences & Biosciences*
*Hochschule Mittweida*
09648 Mittweida, Germany
manthey1@hs-mittweida.de

3rd Matthias Baumgart
*Faculty Applied*
*Computer Sciences & Biosciences*
*Hochschule Mittweida*
09648 Mittweida, Germany
baumgart@hs-mittweida.de

4th Christian Roschke
*Faculty Applied*
*Computer Sciences & Biosciences*
*Hochschule Mittweida*
09648 Mittweida, Germany
roschke@hs-mittweida.de

5th Marc Ritter
*Faculty Applied*
*Computer Sciences & Biosciences*
*Hochschule Mittweida*
09648 Mittweida, Germany
ritter@hs-mittweida.de

6th Matthias Vodel
*Faculty Applied*
*Computer Sciences & Biosciences*
*Hochschule Mittweida*
09648 Mittweida, Germany
vodel@hs-mittweida.de

***Abstract*—This paper presents a holistic approach for transmitting arbitrary data to local Internet-of-Things (IoT) devices using only public services in a zero-trust environment. The approach features a variety of benefits, namely (i) eliminates the need for service providers to deploy their own costly infrastructure, (ii) enables the receiving device to operate without an active network connection, (iii) ensures that the data can be proven untampered on the IoT device itself, (iv) verifies both the validity and recency of the information, (v) requires low administrative and technical efforts, (vi) allows for wireless data updates, including the timeliness of current information, by any participant due to the zero-trust assumption. The described methodology utilizes distinct features of Proof-of-Work (PoW)-based programmable blockchain systems. This work will focus on the usage of Ethereum Classic (ETC) as base layer for trust, the validity and recency of the information.**

**The proposed approach has wide-ranging practical applications, including programmable, rule-based locking systems such as smart locks used in corporate and institutional buildings, cars, and hotels. It also facilitates other forms of access control, such as ticket systems, health certificates, and proof-of-possession for resources or achievements. Due to the zero-trust assumption, data updates as well their recency can be updated by any participant of the system.**

***Index Terms*—Internet of Things, Security, Blockchain, Access Control, Wireless Communication, Distributed Systems**

## I. Introduction

Security-relevant applications such as door locks, car locks and various types of access control systems inherently rely on very high levels of trust, information recency and service availability. However, devices like locks are geographically dispersed, power constrained and may lack network connectivity. This work presents an infrastructure-less method for securely and reliably transmitting arbitrary data to these constrained IoT devices. The method leverages features from public Proof-of-Work-based blockchain systems, offering unique advantages in terms of tamperproof data according to [1, Tab. 1], timeliness of information and availability.

This paper will primarily focus on complex, physically separated locking systems, that require updates whenever access permissions are modified. Typical use cases include car fleet administration, large institutions like universities, and home rental services. For managing access in these contexts, there are generally two approaches. The *local administration*, although simple and well-understood, has its drawbacks. Physical keys can be lost or duplicated, giving unauthorized parties access. Replacing a lock can be expensive[1] [2][3], particularly if multiple locks are keyed alike. The manual distribution of keys also poses logistical challenges, binds massive resources and limits the complexity of access control protocols, i.e. time-based constraints. Other access methods like pin pads share similar limitations; their codes can easily be disseminated, and altering them requires physical intervention.

*Remote administration* solutions, provided by various smart lock systems like the August Door Lock Pro[4] or the Nuki Smart Lock[5], address many drawbacks of local administration. These systems commonly offer features like sharing access rights and Bluetooth connectivity for lock operation. Products, like the *Nuki Smart Lock 3.0 Pro*, support wireless communication for remote administration, whereas others require a specialized bridge for network connectivity. Although these

[1]https://vizpin.com/blog/access-control-pricing/

[2]https://www.forbes.com/home-improvement/home-security/cost-to-hire-a-locksmith/

[3]https://www.verbraucherzentrale.de/wissen/geld-versicherungen/weitere-versicherungen/generalschluessel-verloren-drohende-kosten-bis-zum-preis-eines-kleinwagens-10679

[4]https://august.com/products/august-smart-lock-pro-connect

[5]https://nuki.io/de/

| | | Key System | | |
|---|---|---|---|---|
| | | Mechanical | Electronic | Our Solution |
| Key issuance (min) | Adjustment time | $5-15$ | 1 | 1 |
| | Permission distribution | $30-60^J$ | $1-5^J$ | 1 |
| Permissions change (min) | Change time | $5-15^R$ | 1 | 1 |
| | Change distribution | $30-60^{RJ}$ | $1-5^J$ | 1 |
| Key retraction (min) | Retract key | 1 | 1 | 1 |
| | Retraction distribution | $30-60^{RJ}$ | $1-5^J$ | 1 |
| Costs (€) | Key | $20-90$ | $< 300$ | $< 300$ |
| | Each lock | $50-150^J$ | $1^J$ | $< 1$ |

TABLE I: Comparison of different lock systems characteristics. Efforts are measured in operator journeys $^J$ to each individual lock as well as in replacement tasks $^R$ for specific components.

systems generally offer a high level of security through data encryption [2], they come with their own set of limitations. Specifically, they require a working LAN-infrastructure and vendor-specific services. These dependencies results in well-known security risks, including potential vulnerabilities in the vendor's data storage and transmission models. Additionally, such solutions are vulnerable to DDoS attacks [3], hence impacting availability.

Further prototype approaches, for instance YPTOKEY[6], are using mobile devices as smart keys entities and demonstrate the integration of blockchain features. Nevertheless, all of these latest developments still require continuously network uplinks in order to provide the services. This results in problems for large scale or high security application scenarios.

Thus, we want to combine advantages of traditional, physical locking systems regarding cost-efficiency and compatibility with the flexibility and scalability of modern smart lock systems

The proposed method addresses the shortcomings of existing Smart Lock systems by leveraging public Proof-of-Work blockchains as a trust anchor. The public nodes that constitute this blockchain network are globally distributed, ensuring a highly available infrastructure that is to a considerable extent inherently resistant to trivial *DDoS* attacks. Owing to its consensus mechanism, the network is designed to prevent the propagation of incorrect information, thereby providing a trustless base layer as each participating node acts like a "watchtower", continuously monitoring and validating transactions.

The subsequent section will briefly discuss the essential elements of blockchain technology required to understand the proposed workflow. The following chapter will elaborate on the concept in greater detail.

## II. Related Work & Background

This section will briefly introduce the base building blocks, taxonomy and components used in this work. The concepts mentioned here are vital to understand the method itself. The anchor of security is the blockchain technology which became widely adopted as a decentralized and highly available infrastructure. It provides both algorithmic and game-theoretic measures to create a tamperproof record of stored data, eliminating the need for trust in any specific service provider. Accordingly, blockchain fits perfect to the properties advertised. The technology was publicly introduced by Nakamoto Satoshi [4] by introducing *Bitcoin*. Bitcoin is the most adapted blockchain and supports some of the features needed for this work. However, due to the missing general programmability, Bitcoin is not sufficient to our approach. Hence, this work will focus on Ethereum-based [5], [6] systems, specifically Ethereum Classic (ETC). The relevant concepts are briefly described as follows.

*a) Ethereum Block Structure:* At the core, a blockchain consists of a growing data layer. The data is organized in name-giving `blocks`. Each of the blocks represents an update to the state of the blockchain and consists of several fields [6]. For Ethereum, these include, among others, a `block number`, a `timestamp`, a list of `transactions`, a `mix hash`, a `difficulty`, a `state root`, the previous block hash parent and a random number denoted as `nonce`. The latter is a reference to the previous block, making the blocks an actual blockchain.

*b) Blockchain:* Ethereum Classic as a *Mainline Ethereum 2* compatible public blockchain and serves as our foundational layer for this work. This includes programmability using Smart Contracts, proofing the state of the values at a given moment in time using Merkle Patricia Proofs and being compatible to the Ethereum ecosystem, while featuring an active network of participants. However, due to its consensus mechanism it additionally allows for offline capabilities, which cannot be achieved by most other public blockchains.

*c) Consensus:* Blocks in a blockchain are generated by network nodes. Each participating node can create new blocks by including valid transactions in block data structure. Specifically, in Proof-of-Work (PoW) consensus, the nodes engage in a computationally expensive process called *mining*. Essentially, this involves finding a random value for the block's `nonce` field, so that the hash $h(block)$ satisfies certain criteria enforced by the consensus protocol [6]. This value can only be found by brute force. The PoW model is distinct from other consensus models like Proof of Authority or Proof of Stake [7], which rely on protocols for block generation which lack the computational effort aspect.

*d) Transactions:* Transactions are state changing actions which can be emitted by every user. They are used to transfer tokens, deploy Smart Contracts or call functions thereof. A transaction is propagated to the network and miners include them into their blocks, which they are incentivized to by tips. Transactions are needed in our method to update the information state. Fig. 1 summarizes the process of transaction emission, block generation and block appending.

[6]https://www.yptokey.com/de

*e) Fork:* When two miners simultaneously discover new blocks that both link to the same `parent block`, the blockchain temporarily splits into multiple paths. Each block represents computational work, and the path with the greatest cumulative computational work will eventually be accepted as the truth. Consequently, miners will continue to build upon the longest chain, as measured by this computational effort, thereby resolving the fork. The existence of forks dictates some of the design decision of our method.

*f) Smart Contract:* A Smart Contract is a program code that resides on the blockchain. The described method will make use of Smart Contracts. Once deployed, a Smart Contract has its own storage and can contain functions that alter this internal state (i.e., memory). The functions can be invoked by any network participant. The Smart Contract itself is associated with an unique account itself, enabling it to interact with other contracts and accounts as if it were an individual participant on the network.

*g) Account:* Ethereum is a Account-based system. Each Account is associated with its own address and balance. Further, Smart Contract Accounts additionally contain fields for their Code Hash ($h(SmartContract)$) and a field which holds the root of the *Storage Trie*, which basically is $h(Storage)$. Thus, and account in Ethereum is made of the following fields: *address*, *balance*, *code hash*, and *storage hash*. Understanding that the storage hash is part of the account data structure is crucial for understanding the method.

*h) Merkle Patricia Tree:* Merkle Patricia Trees (MPT) or hash trees are a fundamental data structure used in Ethereums storage system [9], [10]. They serve as a cornerstone for proving the authenticity of data in the method described in this paper. Essentially, an MPT is a tree-like data structure where each leaf node contains a hash of some data stored in the system. Inner nodes, on the other hand, contain hashes of their child nodes. Ethereum employs MPTs in two key instances: (i) for storing each smart contracts data, and (ii) for storing hashes of all known accounts. The root hash of the smart contract's data is saved in its *storage root*, while the hash of all account information is stored in the *state root* field of a block. Thus, proving that a value exists in a given block can be accomplished by providing the appropriate path to the *storage root* of the smart contract and another path to the *state root* of the block header. This mechanism, known as Merkle Patricia Proof, is supported in most Ethereum clients due to EIP-1186 [11].

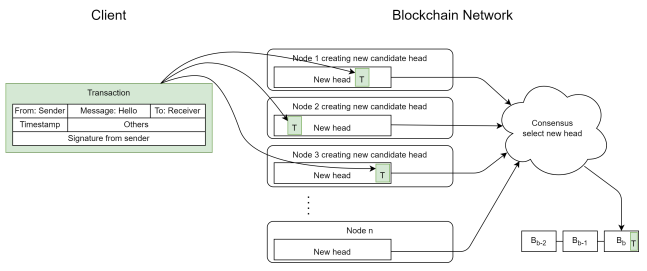

Fig. 1: Process of committing a transaction multiple blockchain nodes, having nodes create blocks including the transaction and eventually appending one valid block to the blockchain data structure using the consensus mechanism. [8]

*i) Ethash:* Ethash is the Proof-of-Work algorithm used by Ethereum, based on the Dagger Hashimoto algorithm [6]. It is a memory-hard algorithm, designed to be resistant to ASIC-based mining, requiring a substantial amount of RAM. A notable feature of Ethash is its usage of a large, read-only data structure. A smaller but crucial part of this structure, known as the *cache* or *light cache* is necessary for verifying the authenticity of a block. This cache evolves over time in a predictable manner, allowing it to be precomputed and stored. Currently, the cache is approximately 55 MB in size increases slowly over time.

## III. Method Description

In the following part, we will provide a detailed description of the method, outlining each step, its implications, design decisions, and prerequisites. The authors of this work are involved in a project that applies this method to the context of configurable door locks. To help clarify the method's utility and application, we will use the example of administering access to commodities in an renting service.

*a) Example Service "Rent a Block":* Our hypothetical rental service, *Rent a Block* (RaB), manages a fluctuating number of properties. These properties are situated in a variety of locations, some in well-connected urban areas and others in more remote regions. Currently, most of these properties use traditional physical keys, which are distributed by staff. Some locations have adopted Smart Locks, but these come with their own set of challenges, including the need for a stable internet connection and being restricted to proprietary software and cloud services.

The requirements for RaB are as follows:

- Door permissions must be updated frequently, at least daily.
- Different parties—such as guests, cleaning services, and housekeepers—may need to access a door at different times.
- The system must allow for updates in locations where Wi-Fi or cellular data may not be readily available.
- Administration should be streamlined and cost-effective, allowing for simultaneous updates on multiple locks.
- In cases of misuse or error or changes in bookings, permissions must be revocable in such a way that even a malicious party cannot gain access after a revocation or permission change has been issued.

Given these constraints, RaB is a suitable candidate for the method proposed in this paper.

*b) General Workflow:* The chronological workflow of the method with the roles **[U]**ser, **[D]**evice and **[A]**dministrator in chronological order is as follows:

A: Initially: Creating and deploying a Smart Contract.
A: As needed: Updating data by invoking the Smart Contract.

U: Continuously: Retrieve blockchain blocks, off-chain data if needed, and Merkle Proofs.
U: Automatically: Transmit required data to the **[D]**evice via Bluetooth when within range.
D: Background: Process and verify transmitted data
U: Active: Send unlock request to device
D: Grant or deny access based on current permissions and validity of transmitted data.

A graphical representation is given in Fig. 2. More details about each step are described in the following part of this chapter.

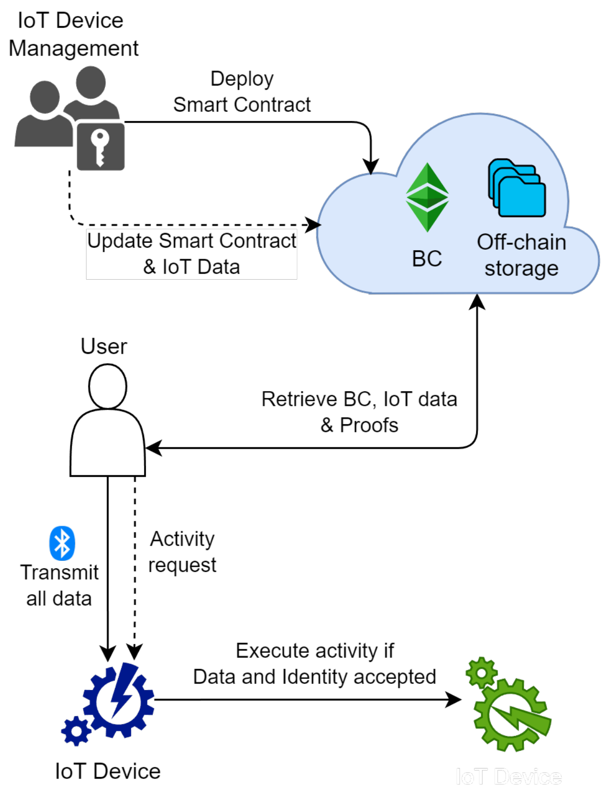

Fig. 2: Exemplary workflow illustrates the device management setting the permission rules or similar as well as the Smart Contracts. These are transmitted to a public accessible storage. The user periodically download the data to present. If a device is within proximity the data are transmitted by Bluetooth. In case of a activity request from the user the lock is checking the received Smart Contract and the permission rules. If rules and identity are accepted, the requested action will be executed.

*c) Step 1: Smart Contract:* As for updating the permissions, the door lock system needs a minimal Smart Contract:

```
contract RaB {
  bytes32 public current_config;
  address public owner;
  constructor() { owner = msg.sender; }

  function update(bytes32 cnfg) public {
    require(msg.sender == owner);
    current_config = cnfg;
  }
}
```

This prototype contract, written in *Solidity*, serves as a foundational layer for permission management. Although very simplistic, it's functional and provides two crucial storage variables: `current_config` and `owner`. Upon deployment, the `constructor` sets the contract's `owner` to the deployer's address. This owner can later invoke the `update` function to modify the stored configuration.

*d) Off-chain storage:* Given RaB's requirements for complex rules, large data structures are inevitable. The company decides for a simplistic, straightforward update scheme:

```
{"lock id 1": [{permission 1}, ...,
               {permission N}],
 "lock id N": [...], ...}
```

Due to the high storage costs on the blockchain, our Smart Contract opts for a small 32-byte (`bytes32`) slot to store just the hash of the lock configuration ($h(configuration)$). Since the hash of the configuration is securely and immutably anchored to the blockchain, the storage location of the actual data structure can be flexible. This flexibility even enables the use of untrusted or public storage solutions. Various storage and sharing options are available, ranging from decentralized systems like IPFS [12] and Torrent[7], to centralized solutions like FTP and HTTP.

For updating the configuration, the administrator has to emit a transaction to the blockchain, to store the configuration hash and transmit the actual data to one or more off-chain storages. The execution model of the blockchain guarantees, that the permission model, as described by the Smart Contract, will be enforced. Note, that off-chain storage is an optional ingredient: if the data are reasonable in size, it can directly be stored on the blockchain.

*e) Transferring data to the device:* Assuming that the device is generally offline, data transmission relies on user intervention. The user's device is responsible for regularly fetching block headers from any node in the ETC network. Public blockchain nodes are available, such as those listed on Chainlist[8]. Further, commercial services like Infura[9] provide access to blockchain nodes. Preferably, the user's device maintains a list of nodes as fallbacks. When within range of the target device (i.e., the lock in our example), the most recently acquired data is transferred using a radio communication protocol such as Bluetooth.

*f) Data verification on the target device:* Despite the data being transferred and stored via untrustworthy mediums, the device can verify its integrity, validity, and timeliness offline, without requiring internet access.

- The device checks the received block headers for validity. This involves hashing the block header and executing

[7]https://www.bittorrent.com/btt/btt-docs/BitTorrent_(BTT)_White_Paper_v0.8.7_Feb_2019.pdf
[8]https://chainlist.org/chain/61
[9]https://www.infura.io/

the *Ethash / Etcash* algorithm, using pre-computed light caches read from a storage medium like an SD card. The cache itself is uniquely determined by the block header. By comparing the algorithm's results with the block's `mix hash` and its `difficulty` field, the device verifies that the block was generated using the correct algorithm, confirms the data's integrity (i.e. data was not tampered with), and ensures the appropriate amount of computational work was invested. After verifying the block, the device should take note of the block's `timestamp` and `number` which are part of the block header and verified in the described process. Due to the potential for blockchain forks and attempts to bypass these security measures using raw computational power, it's strongly advised to verify multiple *consecutive block* headers.

- As block headers themselves do not carry any actual use data on their own, Merkle Proofs are used to verify the existence of a specific value in the blockchain. In our running example, the user needs to prove that the given permission set is indeed anchored in the blockchain at the current time.
  1) Verify the structure of the Merkle proof itself.
  2) Verify the leaf (last node of the proof) contains the expected value (here: $h(configuration)$).
  3) Verify that the proof belongs to the correct Smart Contract (i.e. Account Address).
  4) Verify that the head of the proof matches the `state root` of the block header.

  As the block header has already been proven to be valid and untampered with, the successful verification of the Merkle Proof ensures that the corresponding configuration is accurate and was indeed written into the blockchain at the time indicated by the block's timestamp. This dual-layered verification process provides a robust mechanism for ensuring data integrity, authenticity and timeliness. Importantly, even when the configuration did *not* change, the user can prove this fact. Thus, not only be proven that the data has changed, but also establish that it hasn't changed without requiring additional administrative effort. As a result, a malicious user cannot falsely claim to have permission by withholding blocks, if those permissions have actually been revoked.

*g) Access Authorization:* To gain access, the user submits an unlock request to the device. Since the configuration is already verified, the device only needs to confirm the request's proper format and signature. Following this validation, the device ascertains whether the user holds the requisite permissions under the current rule set. It's crucial to employ standard security protocols, such as Nonces or challenge-response mechanisms, as well as timestamp verification to prevent potential attacks like replay attacks.

## IV. Evaluation

We have undertaken a series of assessments involving both theoretical analyses and first practical performance.

We used a *Teensy 4.1*[10] with an *ARM Cortex-M7* processor connected to a *SDIO* bus equipped with a *San Disk Extreme* 64GB (U3) SD-Card formatted with a *extFat* filesystem to store the blockchain-cache. A commercially available motor operated lock is connected by a custom protocol.

These tests were conducted to obtain a clear indication of the effectiveness of our solution. Given that resources for data transmission or IoT devices face very stringent limitations, the focus of performance testing was directed towards this specific domain.

For this purpose, we evaluated the hardware requirements for JSON-based ETC block headers (as generated by Ethereum client software like *geth*[11], which contain around 1200 characters when encoded as JSON key-value pairs. Encoding as plain binary sequence reduces the data to 330 bytes per block header. Similarly, Merkle Proofs come with an average proof-length of 10 to 12 elements with up to 1088 characters each. In binary representation, this can be reduced to binary sequences of around 620 bytes totally.

Further, we conducted tests for verification times of block headers on the test hardware yielded an average speed of around 21 seconds per block. Notably, the read speed of the SD card emerged as the constraining factor in this context. This comes naturally totally due to the random access factor imposed by the Ethash algorithm.

*a) Verification Speed:* The verification of a block header necessitates approximately $32,768$ random read operations from storage. When relying on an SD Card connected via SDIO interfaces[12] (like the described setup), a single random read operation typically consumes between $0.5$ and $2.0$ milliseconds. This translates to a block header verification time ranging from 16 to 65 seconds. The setup we employed had an average read time of roughly $0.64$ milliseconds per operation when assuming the measured 21 seconds total block verification time. Importantly, the security level of the whole system is determined by the amount of blocks checked on update time, as it becomes increasingly hard to fake valid blocks. To illustrate, opting for a security level of 5 blocks would render the device busy for about 1 minute and 45 seconds for verifying the blocks, which may or may not be acceptable for certain use cases.q

There are, however, quicker alternatives. The microcontroller in use can be augmented with Flash RAM modules. A random read operation from Flash RAM takes only about 100 microseconds, reducing the block header verification time to approximately $3.3$ seconds. However, one must additionally consider the time to copy data from the SD Card to Flash RAM, which, being a sequential operation, takes around $2.5$ seconds. The total time for verifying $n$ blocks would then be $2.5s + n \times 3.3s$ (roughly 19 seconds for 5 blocks). When employing DRAM, with its rapid random access speed of about 10 to 20 nanoseconds, the block verification time could

[10]https://www.pjrc.com/store/teensy41.html
[11]https://geth.ethereum.org
[12]https://www.pjrc.com/store/teensy41.html

shrink to between 0.32 and 0.64 seconds. For 5 blocks, this results in a total time of approximately 2 to 4.1 seconds, including data copy times, as shown in Figure 3.

With the use of RAM, the speed can be improved significantly, since the data can be loaded from the SD card in one read operation and then the operations from the ram can be performed quickly.

## V. Conclusion and Future Work

This study introduces a blockchain-based, universal algorithm for the tamperproof transmission of arbitrary data. Importantly, the entire process leverages highly available and fail-safe public infrastructure. Using a Proof-of-Work-based blockchain like Ethereum Classic enables offline verification of block headers, thanks to the significant computational effort required to forge a block. This unique feature sets our method apart from other energy-intensive approaches that rely on an active internet connection. By utilizing Ethereum-specific features, notably Merkle Patricia Proofs, we demonstrated that the existence of specific data on the blockchain can be verified. Additionally, we showed that it's possible to prove the unaltered state of data without administrative effort.

A prototypical implementation of the proposed method has been developed and successfully tested on a small microcontroller suitable for Internet of Things applications. Initial assessments of computational time and memory constraints have been conducted. Despite these advances, there is still considerable scope for future research. One avenue for further investigation lies in the system's adjustable parameters, such as the number of consecutive blocks required for verification or the acceptable time frame for the last verified block. These factors could be security-relevant and may need adjustment depending on the specific use case. Exploring the applicability of this method across various use cases is a promising path for future work. This could involve the development and testing of frameworks to perform the requisite operations. Additional hardware testing, aimed at reducing resource and time consumption, represents another crucial area of focus. In particular, the development of energy-efficient computation and memory solutions could be significant.

Another design choice worth examining is the selection of Ethereum Classic as the base layer. While it serves the purpose of this work, other platforms may also provide the necessary components for establishing a trustless infrastructure. Investigative work in this area could expand the range of technological options available. Beyond technology-related research, another important direction involves improving public perception of blockchain technology as a whole. This could be achieved by developing useful applications that leverage blockchain to solve real-world problems.

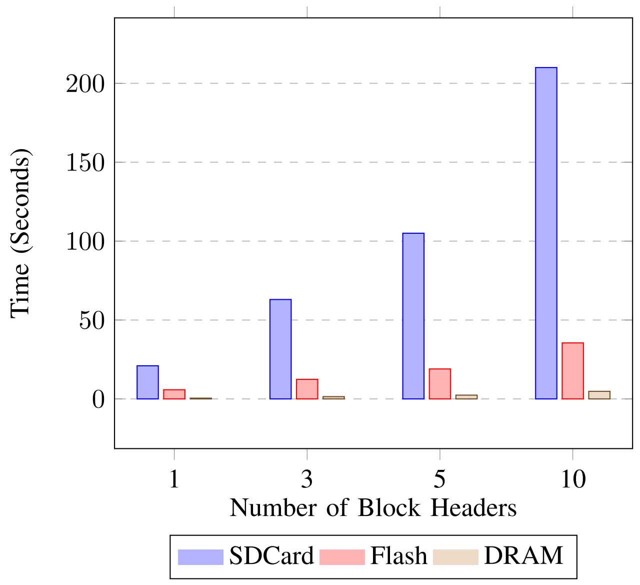

Fig. 3: Verification times in seconds for $n = [1, 3, 5, 10]$ block headers using different memory setups.

## Acknowledgment

This work was partially accomplished within the project *Chainlock - Blockchain-gestützte, smarte Schließanlagen* (funding code 03WIR1321A) funded by the *Federal Ministry of Education and Research* (BMBF, Germany).

## References

[1] R. Chaganti, R. V. Boppana, V. Ravi, K. Munir, M. Almutairi, F. Rustam, E. Lee, and I. Ashraf, "A comprehensive review of denial of service attacks in blockchain ecosystem and open challenges," *IEEE Access*, vol. 10, pp. 96 538–96 555, 2022.

[2] C. Caballero-Gil, R. Álvarez, C. Hernández-Goya, and J. Molina-Gil, "Research on smart-locks cybersecurity and vulnerabilities," *Wireless Networks*, May 2023. [Online]. Available: https://doi.org/10.1007/s11276-023-03376-8

[3] B. Asad and N. Saxena, "On the feasibility of dos attack on smart door lock iot network," in *Security in Computing and Communications*, S. M. Thampi, G. Wang, D. B. Rawat, R. Ko, and C.-I. Fan, Eds. Singapore: Springer Singapore, 2021, pp. 123–138.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 03 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[5] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2014, accessed: 2023-08-29. [Online]. Available: https://ethereum.org/en/whitepaper/

[6] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[7] J. Xu, C. Wang, and X. Jia, "A survey of blockchain consensus protocols," *ACM Computing Surveys*, 2023.

[8] R. Manthey, R. Vogel, F. Schmidsberger, M. Baumgart, C. Roschke, M. Ritter, and M. Vodel, "Blockchain based social commitment -secure & reliable web services," in *International Workshop on Metrology for Living Environment (MetroLivEn)*. New York, NY, USA: IEEE, 07 2022, pp. 101–104.

[9] H. S. de Ocáriz Borde, "An overview of trees in blockchain technology: Merkle trees and merkle patricia tries," 2022.

[10] H. Liu, X. Luo, H. Liu, and X. Xia, "Merkle tree: A fundamental component of blockchains," in *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2021, pp. 556–561.

[11] S. Jentzsch and C. Jentzsch. (2018, June) Eip-1186: Rpc-method to get merkle proofs - eth_getproof [draft]. [Online serial]. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1186

[12] E. Daniel and F. Tschorsch, "Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 31–52, 2022.