Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 52-10 (2019) 125–130

# Securing the Data Flow for Blockchain Technology in a Production Environment [⋆]

Tobias Korb [*] David Michel [*] Oliver Riedel [*] Armin Lechler [*]

[*] *Institute for Control Engineering of Machine Tools and Manufacturing Units, University of Stuttgart, 70174 Stuttgart, Germany (e-mail: tobias.korb@isw.uni-stuttgart.de)*

**Abstract:** For several years, blockchain technology have been used and tested in various prototypes for production environments. The main focus of these approaches is the feasibility of different applications. It is usually simply assumed that data in a blockchain is stored immutably. However, the path from data generation to entry into a blockchain is usually neglected. Exactly this topic is discussed in this paper and a solution proposal for a secure data flow from the machine to the blockchain is presented. For this, the hardware and software architecture of the solution is shown, followed by measurements of the operational capability of the solution.

*Keywords:* Blockchain, Manufacturing, Immutability, Trust, Microcontroller, Digitization

## 1. INTRODUCTION

Digital data plays an ever-growing role in today's industry. Data are collected during different types of manufacturing processes. The more data there is available, the better an optimization of single steps during production can be performed. A lot of this data are used company-internal and never leaves its isolated environment. In such a case, no proof of data integrity is necessary. Easy and fast data acquisition and processing increases flexibility in production processes. Nonetheless, the manufacturing industry gets increasingly intertwined. Dependencies on suppliers and cooperation with other companies demand for cross-company data exchange. As soon as digital data is shared with other parties, a need for trust and verification of data arises. The problem with this is how to proof the correctness of data to other parties. At the same time that this question becomes evermore important, the blockchain technology is on the rise. Originating from the financial industry, many of its concepts can be transferred to other application fields as well. One area of application is the manufacturing industry. Blockchains can potentially be used to facilitate the cooperation of different companies working on a shared project by ensuring data integrity. There is industrial potential lying within blockchain technology, yet we are just at the beginning of blockchain adoption and several uncertainties exist until today. One task needing to be solved is the question how industrial machines can be connected to blockchain implementations while keeping the security properties a blockchain offers.

## 2. RELATED WORK

During the last years, multiple prototypes using blockchain technology for industrial purposes were presented. For the most part, the papers show a feasibility study for use cases. The different scenarios shown in these papers, be it industrial energy trading by Li et al. (2017) or orchestration of robotic swarm systems by Castelló Ferrer (2019), are well thought through and reflect the current state of distributed ledger technology in production. However, to move from a theoretical prototype or feasibility study to a working product state, there are requirements which are not met yet.

The requirement this work focuses on, is the immutability of stored data. As shown by Michel et al. (2018) stored data is almost unchangeable in a blockchain after only a few blocks got added. Equation (1) originally released by Nakamoto (2008) shows the numeric values of the rate of success when trying to change data in an existing bitcoin blockchain. Factors that have to be taken into account are the number of blocks that have to be changed ($z$), the probability that the next block is added by a non attacker ($p$), the probability that the next block is added by the attacker ($q$) and their interaction $\lambda = \frac{(z \cdot q)}{p}$. The main meaning of this equation is that the rate of success decays down to zero if $p > q$ with many blocks that need to be changed ($z$). This is the case in almost every blockchain economy. This equation is only true for bitcoin systems using the proof of work consensus mechanism but is adoptable for other implementations too, as shown by Michel et al. (2018).

$$P(X = success) = 1 - \sum_{k=0}^{z} \frac{\lambda^k \cdot e^-\lambda}{k!}(1 - (q/p)^{z-k}) \quad (1)$$

Those facts lead to the result, that once data is stored in a blockchain, it can be viewed as unchangeable. Every presented use case from related work uses this fact to create business models that rely on a blockchain. In reality, things are more complicated. Immutable data in a blockchain based database only contains reliable information if the process from the point of creation

[⋆] This research was funded by the German Federal Ministry of Education and Research (BMBF) under the funding code 02P17D020. The responsibility for the content of this publication lies with the authors.

until the point of storing the data was not disturbed. In others words, wrong data that is written into a blockchain remains wrong. In recent papers, a few researchers, for example Imeri and Khadraoui (2018), have tackled this topic without having a solid solution for it.

In most cases the common approach is to implement a blockchain client that runs on simple systems instead of using a general purpose computer. Florea (2018) created an IOTA-client that runs on a Raspberry Pi and is directly connected to sensors using the GPIO-Interface of the Raspberry Pi module. Received data is used to fill a JSON based data model and encode it to trytes, which is a unit of measurement used in IOTA systems. The signing of the transaction is still processed on another, more potent computer in a later step. Özyılmaz and Yurdakul (2017) are also using a Raspberry Pi to create a similar scenario using Ethereum. Whether or not Özyılmaz and Yurdakul (2017) are signing their transaction on the Raspberry Pi or not, is not presented in their paper. Nevertheless, a Raspberry Pi is a microcontroller that has the power and capabilities of a small general purpose computer. Running a Linux operating system on a Raspberry Pi lets the system behave similar than a weak general purpose computer. The security and data immutability benefit therefore is negligible, because the system is vulnerable to all frauds a general purpose computer has to face.

More practice based approaches also make its way into the market. Companies currently recognized the need to identify the data origin by using different marker technologies. One advanced competitor is Riddle & Code. They produce small hardware identity tokens to uniquely identify physical systems as released by M. Sallaba et al. (2018). This is an important step towards a stable link between physical components and their digital twins, but does not solve the immutability of collected production data. Summarized, what current state of the art lacks, is a system that ensures an immutable transport of data from the sensor into a blockchain. A concept and its realization to achieve this goal will be presented in the following pages.

## 3. INDUSTRIAL BLOCKCHAIN CLIENTS AND THEIR FLAWS

Before a proposal for a secure blockchain client is presented, current approaches to the technical implementation of a blockchain client in a production environment are presented below. For the meaningful use of a blockchain client, it can be assumed that the basic features of digitization are present. This means a digital data acquisition of production data and their transfer into internal as well as external databases. Actual data processing as well as a feedback of generated information is not assumed or necessary, because the IT infrastructure of the production remains approximately the same in both scenarios. The prerequisite for this infrastructure is that these technical possibilities are required for the use of blockchain technology in production. Looking at the data flow of a production in the basic outlines of digitization, the following chain of technological components results as shown in Fig. 1.

A physical system, in the following referred to as a machine (1), but actually regarded as individual components such as drives, working pieces or even gears, performs work

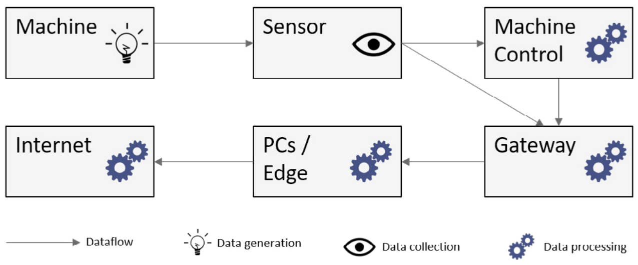


Fig. 1. Usual data flow within a digitized production environment

and thus influences surrounding physical values. These values are recorded with the aid of different sensors (2). In most cases, this information is fed into a machine controller (3), which is then able to intervene and manage the currently running process. Irrespective of whether the values are fed into a machine controller or not, the next step in the data flow is a gateway (4). A gateway connects the production network with external networks. In some cases, the machine controller acts as a gateway, but usually a separate physical component is provided for the gateway functionality. From the gateway, the data can be transferred to the Internet (6), where it can be used for simple data storage, further data processing or in the case of this work, storage in a blockchain. Additional PC's or edge devices (5) can be used for data processing between the gateway and the Internet. This procedure is usually used for larger productions with local data processing.

Why is this technical infrastructure interesting for the implementation of a blockchain client? Since a blockchain is basically a database, this data flow also applies to a production blockchain solution. In contrast to a conventional database, a blockchain offers an unchangeable storage of information. In order to be able to use this property, it must be ensured that the data to be stored reaches the blockchain correctly and without manipulation from the start. Considering this task, a problem results from the previous realization of the data acquisition. As soon as the sensor data are passed on, they reach either a machine control or a gateway. Both systems are able to change data before they find their way into a blockchain. The further the data flows in Fig. 1, the more possibilities there are to change the recorded sensor values. As blockchain technology is used to bring trust in a multi company environment where trust usually is absent, this leads to the fact that one advantage of the blockchain, the unchangeable data storage, becomes null and void. The solution to this problem sounds very simple. If the data is written into a blockchain directly after being captured by the sensors, it is unchanged and guarantees the full value of a blockchain solution. However, this naive but promising approach encounters some challenges on closer inspection:

*(A) Unchanged data* The information recorded by any sensor must be processed and therefore another device is required. This device is just another component in the data flow chain. In order to ensure that the additional device cannot change the collected data, such as controller, gateway and other devices in the chain, a solution with unchangeable data processing logic and a certificate mechanism must be installed. In order to meet both requirements, no general purpose operating system may be used,

as this allows too much freedom in changing processing logic. However, a hardware oriented approach without a general purpose operating system can meet both requirements. For example a microcontroller that is parameterized once, then sealed and certified and communicates with the sensors via hardwired GPIO connections can be a solution. Based on this approach, however, further challenges arise.

*(B) Blockchain requirements* Due to the potentially massive memory requirements and, depending on the consensus method used, the computing power required by a blockchain client, a rudimentary microcontroller quickly reaches its limits. Using a powerful microcontroller contradicts the solution found in problem 1. A valid solution to this problem is not to run a full blockchain node on the client. The use of a proxy node should generate necessary information and forward it to an externally operated full node. The memory problem can thus be solved, since the history of the blockchain does not have to be stored locally. Computing power is also not required, since the proxy does not actively participate in the consensus procedure of the blockchain. In a theoretical view, it is therefore possible to integrate a hardware-based system into the data flow, which can package data correctly and then send it to a blockchain. In practice, however, there is still a challenge to be overcome.

*(C) Production network* To be able to send data from the production network to a full node, the data flow described in Fig. 1 must nevertheless be adhered to. A way must be found to send the data unchangeably through the individual components. An enrichment of the collected data with meta information for the used blockchain implementation, an encryption of the actual data as well as a subsequent signature of the entire data package, provides for an unchangeable in blockchain terms called transaction. If this signed transaction is changed in any way, the signature is no longer valid and the full node rejects the transaction. Accordingly, it is only possible to write correct information into the blockchain, but there is no guarantee that a transaction will eventually end up in the blockchain. Any changes to the transaction will cause the data to be discarded.

In summary, there is a potential approach to solve the described challenges. This approach could ensure valid data storage of information sources in the production process. The previously derived and superficially described solution was implemented in practice and will be presented in detail in the following chapter.

## 4. PROPOSED SOLUTION FOR THE HARDWARE ARCHITECTURE

The hardware structure of the implemented solution should be different from the current gateway solutions. Therefore, value is placed on minimum performance with almost no operating system functionality. This design philosophy ensures that as little manipulation potential as possible exists on the hardware. In the test setup, a NodeMCU with an ESP8266 module is used to meet these criteria. Communication with connected sensors takes place via general purpose inputs and outputs (GPIO).

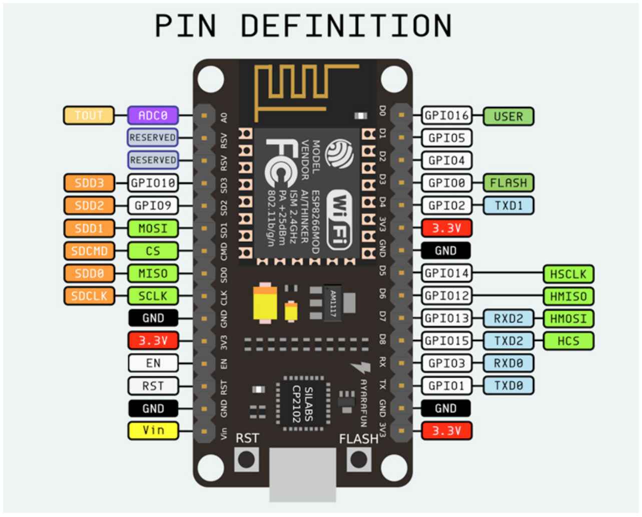


Fig. 2. Pin layout of the NodeMCU ESP8266 used in this paper [1]

Standardized interfaces such as a Serial Peripheral Interface (SPI) are also available. The complete layout of the microcontroller can be seen in Fig. 2. Captured information is processed on the microcontroller. The processor core L106 by Espressif runs at 80MHz and has access to 224 kb RAM which is public accessible information [2]. Finished transactions are sent via the wifi module integrated in the ESP8266. The processing logic is written in the Arduino C/C++ programming language and deployed on the microcontroller via USB, visible at the bottom of Fig. 2. As soon as the USB port is dismounted or physically enclosed and sealed, the processing logic on the MCU node can no longer be changed. It is therefore very difficult to manipulate the device after its initial setup.

## 5. PROPOSED SOLUTION FOR THE SOFTWARE ARCHITECTURE

The software architecture is modular in principle and can support different blockchain implementations. However, each blockchain implementation has different transactions and signing mechanisms. The software described below refers to the Ethereum blockchain and its transaction protocol. First a sequential overview on the actions of the system will be presented. Afterwards a closer look on the software architecture itself describes how the functionality is achieved.

### *5.1 Program flow*

The core of the software is a component that controls the entire process of the system. How exactly this process behaves is shown in Fig. 3. The UML flowchart shows the individual steps, which are explained in the following:

(1) **Setup Wifi:** Retrieval of sensor data is hardwired and therefore has no need to be configured on program start. The wifi connection on the other hand, must be established initially in order for the full node to be reached later. This step is always carried out initially when the MCU node is started.

[1] Figure by circuits4you
[2] `http://esp8266.net/`

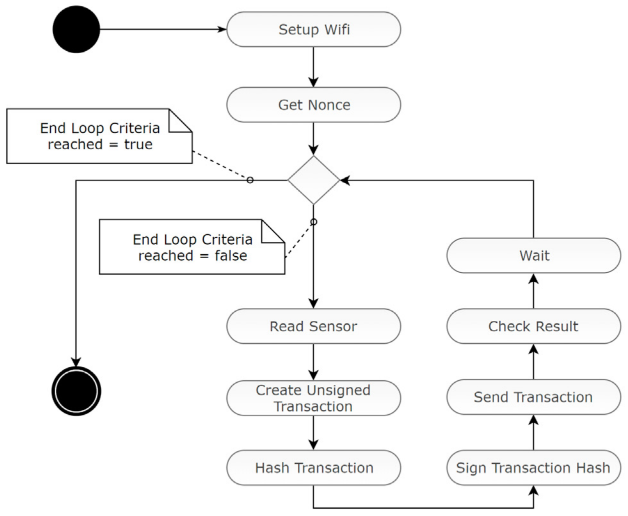


Fig. 3. UML chart that displays the sequence of actions the presented prototype executes

(2) **Get Nonce:** The request of the initial nonce at the used full node is the second part of the system setup. The nonce is needed to create valid transactions as described by Michel et al. (2018). To obtain the nonce, an HTTP-GET request is sent to the provided interface of a full node. Both `Setup Wifi` and `Get Nonce` are performed once. After successful execution of both steps, a loop starts that stops by self-definable termination criteria.

(3) **Read Sensor:** The first step after the start of the loop is the acquisition of the sensor values by querying all connected pins. If data is to be deliberately pre-processed or concatenated, the pin assignment must be known in advance. If only the raw data of all pins is to be stored, a generic solution is possible.

(4) **Create Unsigned Transaction:** Starting with this step, the three-step process of transaction creation begins. A transaction object, as provided by the Ethereum protocol, is created and filled with the stored user information and received sensor data.

(5) **Hash Transaction:** The unsigned transaction is hashed using a Keccak-256 algorithm. The choice of the hash method is determined by the Ethereum protocol.

(6) **Sign Transaction Hash:** The signature of the created transaction is based on the ECDSA procedure which was published by Johnson et al. (2001). Therefore a secp-256k1 curve is used to generate the signature. A private key and the previously created keccak-256 hash are used as input parameters. The signature obtained is then added to the transaction. The thorough explanation follows in Section 5.3.

(7) **Send Transaction:** The sending of the signed transaction is based on an HTTPS request to the interface of the full node used. Depending on which interface a full node provides, there might be differences on how to actually send the transaction.

(8) **Check Result:** If a valid transaction was sent, the response of the full node contains a corresponding message. If the microcontroller communicates directly with the full node, it is possible to validate whether a transaction arrived unchanged at the blockchain and therefore is accepted. This procedure was used in the prototype implemented in this work. However, if a gateway and other systems are located between the microcontroller and the blockchain, the response message of the full node should be forwarded to the microcontroller. This is not only an advantage for checking the validity of the data sent, but also for incrementing the nonce. If the nonce is not incremented correctly, following transactions are rejected by the full node.

(9) **Wait:** The last logical function of the loop has two tasks. The obvious task is to manage the termination condition. This can be freely chosen and implemented. In the case of the prototype, two termination conditions have been defined. Multiple failures when sending a transaction as well as a fixed time period, programmatically defined, terminate the loop. The second task of the function is to define the sampling interval of the data to collect. A defined pause can be specified in milliseconds, e.g. to query the sensor values only every 10 seconds.

After describing the logical steps of the workflow, in the next section the actual software realization is presented.

## 5.2 Program structure

As pictured at the beginning of this chapter, a central component controls the function flow of the while system. Six additional software components offer functionalities that are used by the `Main` component. A UML component diagram in Fig. 4 describes the structure and interaction of the components. The `Transaction` component is a purely structural component that contains the composition of an Ethereum transaction. In order to make the software adaptable for further implementations, the transaction model was nevertheless outsourced. The `Config` component is also a purely descriptive component that contains parameters to be defined. For example, private and public key for the Ethereum blockchain as well as wifi access data are included in the `Config` component. The `WifiClient` component is responsible for establishing the wifi connection and contains functionality for sending and receiving HTTP(S) messages. This component is

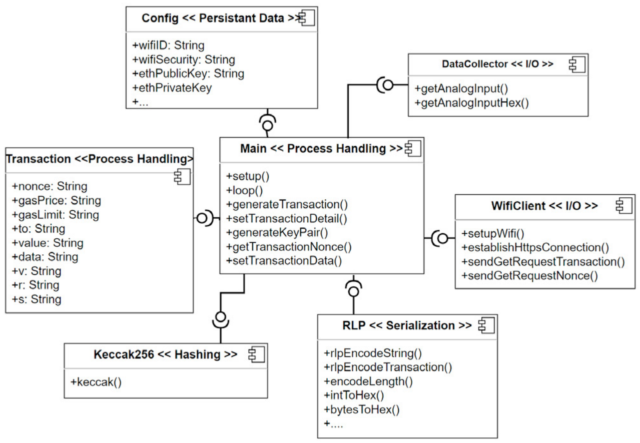


Fig. 4. UML chart that displays the software components developed for the presented prototype

based on the freely available libraries ESP8266WiFi and ArduinoJson. The DataCollector component abstracts the query of the connected sensors via the pins of the NodeMCU. This built-in functionality is provided by the Arduino environment. The Keccak256 component of the signature process was already superficially introduced in the previous section. It is based on a freely available C++ library [3]. During the hash process byte representations of the data are used and not, as optionally possible, the string representation. This is also due to the requirements of the Ethereum protocol. The last modular separated component RLP, the abbreviation for recursive length prefix, is responsible for the correct serialization of the transaction. The implementation of the method was outsourced into a separate module, since other blockchain technologies do not use this serialization method and therefore it might be exchanged for future usage.

With the description of the program structure presented so far, the system sequence previously shown in Fig. 3 cannot yet be completely retraced. The component responsible for signing the transaction is missing. The external C library uECC was used for this. This is the only component, whose handling is not strictly modular, but takes place directly in the Main component. An exact usage of uECC is explained in the next section.

### *5.3 Signature process*

uECC contains the logic for various encryption functionalities. For the presented system an implementation of the signing process based on secp-256k1 curve called ECDSA is used. The procedure generates a 64 byte signature. However, this signature cannot be used directly for Ethereum transactions. The resulting signature is manually split into two parts and then, converted to hexadecimal numbers, forms the r and s values of a signature required for Ethereum. However, due to an unrecorded change from 2016 in the Ethereum protocol, there are restrictions which s values of the signature are valid. The maximum value of s is restricted to $(secp256k1n_{max}/2) - 1$. In other words, the randomly generated s value must be in the lower half of all possible results. On average, this procedure must therefore be performed twice before a valid value for s is produced. Furthermore, Ethereum needs a third bit value v, which is necessary for the unique generation of the public key. This value is not included in the result of a secp-256k1 based signature by default, but it is necessary to be able to uniquely generate the public key based on s and r. This process has to be executed by every full node on each incoming transaction and therefore is necessary. In summary, the reason for this is an ambiguous solution of the equation system for generating the public key. An addition of the v value results in a unique solution of the equation system. The uECC library used was adapted accordingly in the course of this work.

### *5.4 Experimental setup*

After the software and hardware structure of the developed microcontroller system has been completely described, the structure for testing the system is presented in the following.

[3] http://create.stephan-brumme.com/hash-library/

A self made brightness sensor was attached to the NodeMCU to validate the overall system and to perform measurements. The brightness sensor consists of a static resistor and a photosensitive resistor, which moves between a few ohm and several megaohm depending on the brightness level. A voltage divider is generated at pin A0 of the NodeMCU board, which can measure values between 0 and 3.3 volts voltage depending on the incidence of light on the dynamic resistor. A resolution of 10 bit leads to values between 0 and 1023 when reading the pin. This simple brightness sensor is used as a data collection source for the evaluation described below.

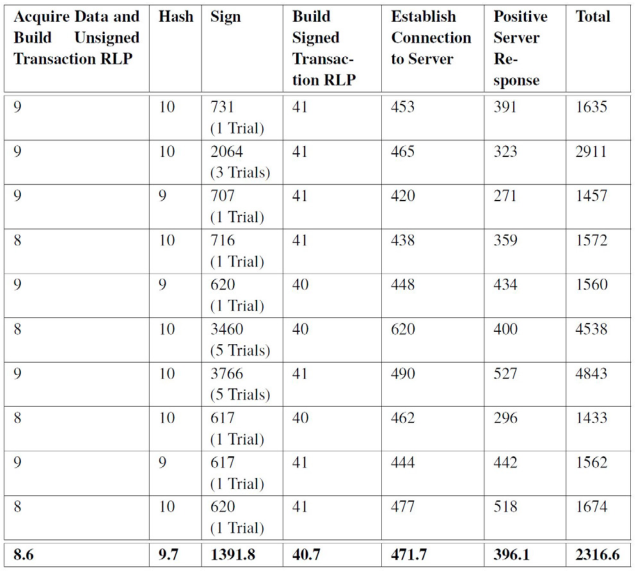
## 6. MEASUREMENTS AND EVALUATION

Different measurements were carried out using the setup explained in the previous section. For this purpose a Smart Contract was created in the public Ethereum test network Ropsten. The Smart Contract receives numerical values between 0 and 1023 and stores them in a list. The current value and any number of old values from the list can be viewed on request. The transactions that led to the following values can also be viewed publicly [4] and are therefore completely traceable.

The entire measurement system was set up as follows: The microcontroller was connected to the presented sensor. This sensor was continuously illuminated in different ways. This was not important for the actual measured metrics but for the values written in the Smart Contract. The initial setup was carried out by establishing the wifi connection on the one hand and requesting the nonce from the used full node, in this case Etherscan, on the other hand. After this initial setup, the loop was executed permanently, storing times for all important events. Writing these log files could potentially affect performance, but should have no significant effect. A representative run of the test is shown in Fig. 5. The duration of the data acquisition and the creation of the unsigned transaction, the hash process, the signing process, the creation of the signed transaction, the establishment of the connection to Etherscan as well as the duration until the response of Etherscan arrived were measured. All values are added up in the last column as total time. The average of the recorded values is displayed in the last line. It is obvious that the durations of the individual process steps are extremely different. On average, capturing the data and creating the unsigned transaction together takes less than 0.5 percent of the total time. The generation of the keccak-256 hashes also lies within this time span. Creating the signed transaction is much slower than creating the unsigned transaction. Nevertheless, in relation to the total duration, this process is on average less than 2 percent of the duration. The increased duration relative to the unsigned transaction is probably due to the increased complexity of serialization. All previous serialization steps have to be repeated and additional parameters were added on top.

The connection setup to Etherscan and waiting for the appropriate response to the transaction combined takes on average just under 900ms. This value depends strongly on the used blockchain implementation, the addressed

[4] https://ropsten.etherscan.io/address/0x5e2a561439602495e2dddf0ed59f247a6ce01e23



| Acquire Data and Build Unsigned Transaction RLP | Hash | Sign | Build Signed Transaction RLP | Establish Connection to Server | Positive Server Response | Total |
|---|---|---|---|---|---|---|
| 9 | 10 | 731 (1 Trial) | 41 | 453 | 391 | 1635 |
| 9 | 10 | 2064 (3 Trials) | 41 | 465 | 323 | 2911 |
| 9 | 9 | 707 (1 Trial) | 41 | 420 | 271 | 1457 |
| 8 | 10 | 716 (1 Trial) | 41 | 438 | 359 | 1572 |
| 9 | 9 | 620 (1 Trial) | 40 | 448 | 434 | 1560 |
| 8 | 10 | 3460 (5 Trials) | 40 | 620 | 400 | 4538 |
| 9 | 10 | 3766 (5 Trials) | 41 | 490 | 527 | 4843 |
| 8 | 10 | 617 (1 Trial) | 40 | 462 | 296 | 1433 |
| 9 | 9 | 617 (1 Trial) | 41 | 444 | 442 | 1562 |
| 8 | 10 | 620 (1 Trial) | 41 | 477 | 518 | 1674 |
| **8.6** | **9.7** | **1391.8** | **40.7** | **471.7** | **396.1** | **2316.6** |

Fig. 5. Results of measurements made with the developed prototype. All values in ms.

service of a full node and finally on the local network. Optimizations at these times directly on the controller are only possible by inserting an asynchronous behavior. However, this is only possible to a limited extent, since the changes to the nonce must be known before generating the next transaction.

By far the most time intensive part of the entire process is the signing of the transaction. The aforementioned change in the Ethereum protocol means that the signing process must potentially be executed multiple times as soon as the resulting value for `s` is in the greater half of the possible value range. A single run of the ECDSA algorithm already takes quite a long time with about 700ms. With a 50 percent probability of getting a correct value, the average result of 1391.8 ms is very close to the purely theoretical average. The signing process is specially designed for Ethereum. Other technologies with a different signing process would probably have completely different values for this task. The other operations will most likely take similar time using other technologies than Ethereum, as the functionality requires little computing power and mainly involves basic read and write operations that are independent from the used blockchain technology.

Ultimately, a transaction with the described system architecture takes slightly more than 2.3 seconds on average. These values are completely sufficient for use in a production environment, since the blockchain is usually used as a journal for aggregated data. Aggregation of data consists of reading from pins and afterwards create and fill data structures. As can be seen in the first column of the table shown in Fig. 5, these types of actions are to neglect.

## 7. SUMMARY AND FUTURE WORK

In this work, an alternative concept for the technical implementation of a blockchain client for industrial use was shown. The focus of the concept is to capture production data directly from sensors installed in a machine and to transfer it to a blockchain without any potential changes. That said, there are no unhackable and non-manipulable systems and the approach presented also cannot call these characteristics its own. However, in comparison to current applications, the approach presented provides significantly better protection for the collected data against unwanted changes before they are stored in the, in this case, Ethereum blockchain. In consideration of the reference architecture for Industrie 4.0 (RAMI 4.0)[5], this approach improves both bottom layers, Integration and Asset.

The authors see the potential to further develop this approach. In currently planned work, the developed Software will be extended to other blockchain technologies such as Hyperledger, Nextchain and Orbiterchain. This work shows the basic architecture and the implementation approach. In future work, the focus will be on the temporal measurement of the actual use of said implemented technologies.

## REFERENCES

Castelló Ferrer, E. (2019). The blockchain: A new framework for robotic swarm systems. In Arai, Kohei and Bhatia, Rahul and Kapoor, Supriya (ed.), *Proceedings of the Future Technologies Conference (FTC) 2018*, 1037–1058. Springer International Publishing, Cham.

Florea, B.C. (2018). Blockchain and internet of things data provider for smart applications. In R. Stojanovic, L. Jóźwiak, D. Jurišić, and B. Lutovac (eds.), *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, 1–4. IEEE, Piscataway, NJ. doi:10.1109/MECO.2018.8406041.

Imeri, A. and Khadraoui, D. (2018). The security and traceability of shared information in the process of transportation of dangerous goods. 1–5. doi:10.1109/NTMS.2018.8328751.

Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1), 36–63. doi:10.1007/s102070100002.

Li, Z., Kang, J., Yu, R., Ye, D., Deng, Q., and Zhang, Y. (2017). Consortium blockchain for secure energy trading in industrial internet of things. *IEEE Transactions on Industrial Informatics*, 1. doi:10.1109/TII.2017.2786307.

M. Sallaba, D. Siegel, and S. Becker (2018). Iot powered by blockchain: How blockchains facilitate the application of digital twins in iot. URL `https://static1.squarespace.com/static/58f7bc39bebafb94498d25bf/t/5af42d3b0e2e728fc7dc086e/1528125064397/RIDDLE%26CODE_Deloitte_Report_IoT_powered_by_Blockchain`.

Michel, T. Korb, G. Falazi, and F. Leymann (2018). Conception of a blockchain client for secure transmission of production data. URL `https://elib.uni-stuttgart.de/handle/11682/9999`.

Nakamoto (2008). Bitcoin: A peer-to-peer electronic cash system. URL `http://bitcoin.org/bitcoin.pdf`.

Özyılmaz, K.R. and Yurdakul, A. (2017). Integrating low-power iot devices to a blockchain-based infrastructure. 1–2. doi:10.1145/3125503.3125628.

[5] `https://industrie40.vdma.org/en/viewer/-/v2article/render/15557415`