Security for Distributed Smart Meter: Blockchain-based Approach, Ensuring Privacy by Functional Encryption

Artem Yurchenko[1], Mahbuba Moni[1], Daniel Peters[1], Jan Nordholz[1,2] and Florian Thiel[1]
[1]*Physikalisch-Technische Bundesanstalt, Berlin and Brunswick, Germany*
[2]*Institute of Software Engineering and Theoretical Computer Science, Technical University of Berlin, Germany*

Keywords: Legal Metrology, Smart Meter, Functional Encryption, Distributed Ledger Technology, Blockchain.

Abstract: Today the trend towards a completely distributed measuring device is progressing and increasing numbers of measuring instruments have already a cloud connection. This development requires new solutions to cover the requirements laid down by legal metrology. These new challenges could be tackled by designing innovative solutions which extend and merge novel technologies. The aim of this publication is to use blockchain technology and functional encryption to develop a model of a secure smart metering system, demonstrating the capabilities and limitations of these technologies in a practical scenario in the framework of legal metrology.

1 INTRODUCTION

The trend towards networking and distribution of system components is increasingly affecting broad sectors of the economy: terms such as IoT, SmartMeter, SmartHome, SmartFabric are often used in this context. The field of legal metrology is affected by these changes as well. An increasing number of measuring instruments are connected to the internet, the measurement data is partially stored in the cloud and the device operation could even be controlled via a browser interface.

All these developments expose legal metrology to new challenges to guarantee a level of metrological security comparable to a concentrated measuring device. The example case of a smart meter shows impressively what efforts have to be made to guarantee the required level of metrological security with conventional means, leading to a rather complex system, e.g. in Germany (BSI, 2019). This publication presents an alternative model of a smart metering system based on the use of blockchain technology and functional encryption. This approach aims at reducing the complexity of the system while achieving the required adequate level of metrological security. Therefore, the goal of this publication is to examine the limits and possibilities of blockchain and functional encryption on a simplified smart meter model. The model presented contains measuring sensors, consumer and service provider entities.While the customer is entitled to entire measurement data, the service provider is only allowed to see the accumulated consumption. Our aim is to guarantee the data authenticity and privacy as well as the integrity of the algorithm by using a combination of blockchain and functional encryption establishing confidence in the correctness of measurements, which is one of the main goals of the legal metrology. But first the context of legal metrology should be described in more detail.

2 CONTEXT OF LEGAL METROLOGY

Legal metrology establishes confidence in the correctness of measurements and protects the users of measuring instruments and their customers (Thiel, 2018). There are more than 130 million of measurement instruments in Germany which are employed for commercial or administrative purposes, such as water meters, gas meters, weighing instruments, taximeters, thermal energy meters and electricity meters (Esche and Thiel, 2015). About 345 million measurement instruments (7 billion in sales) are sold annually in the EU, with most of them being subject to legally regulations (Thiel, 2018).

Yurchenko, A., Moni, M., Peters, D., Nordholz, J. and Thiel, F.
Security for Distributed Smart Meter: Blockchain-based Approach, Ensuring Privacy by Functional Encryption.
DOI: 10.5220/0009377702920301
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 292-301
ISBN: 978-989-758-424-4

2.1 The Framework of Legal Metrology

The International Organization of Legal Metrology (OIML) was founded to harmonize the regulations across national borders and thus to reduce barriers to trade and innovation. For this purpose, the organization publishes various guides and recommendations. From the perspective of metrological software, the document OIML D31 plays the most important role (OIML, 2008). The WELMEC Software Guide 7.2 (WELMEC, 2018) is mostly used on the European level, as it contains practical recommendations for implementing the requirements of the Measuring Instruments Directive (2014/32/EU). This directive serves as the basis of national legislation for measuring instruments in the European Union.

The WELMEC 7.2 Software Guide defines six risk classes, which vary in terms of the necessary software protection, software examination and software conformity. Software examination is usually done by notified bodies prior to market launch. Thereafter, verification is periodically carried out by market surveillance. Often, this verification process is limited to visual inspection of hardware seals, verification of the software identifier, and sample measurements.

In the process of software conformity assessment, a distinction between the legally relevant and non-legally relevant software must be made. According to WELMEC 7.2 all software modules that contribute to or influence measurement results are legally relevant. This not only includes software modules that generate and process, but also those that transfer, store and display the measurement data. This makes a large part of the software legally relevant, which could, potentially, lead to an increased testing effort and corresponding market launch delay.

WELMEC and OIML propose the use of modularization and clear software separation to increase the security and simplify the software assessment. Works such as (Peters et al., 2015) substantiate and extend this approach and seek to achieve the highest level of security using separation kernels and virtualization technologies. Such solutions typically require the presence of a full-fledged operating system on the measurement instruments and require secure hardware.

2.2 Current State in Software Verification and Future Challenges

The current methods to establish confidence in the correctness of measurements are based on system integrity. The hardware is physically sealed, and the software integrity is verified by calculating checksums over all relevant files and modules. This approach provides a temporal security which holds until a software or hardware vulnerability suitable to exploit the security system is found.

For measurement instruments not connected to networks the exploitation of the vulnerability requires the physical presence of the attacker, but with devices connected to the internet the situation is radically changing since the circle of potential attackers gets significantly larger. The network connectivity increases the complexity of the overall system and thus expands the possible attack surface. It also offers a new way of influencing the device so that even the calculation of the checksum might be manipulated without breaking any seal. To prevent this situation measurement instruments need to undergo regular updates. However, each update itself constitutes a (deliberate) manipulation of the software, which usually requires recertification and is inconvenient for all stakeholders. On the other hand, outfitting measuring devices with an internet connection allows completely new usage scenarios and business models which range from connecting to the manufacturer's cloud infrastructure to creating a fully distributed measuring device.

These usage scenarios lead to the emergence of new device architectures whose software integrity cannot be adequately secured by checksum methods. It would therefore be desirable to have a verification method which does not verify the binary image itself, but the functionality contained therein. Since the functionality to be tested so far can be very extensive and complex, it makes sense to subject these to a logical separation. Essentially, a distinction can be made between the core measuring algorithm, which is responsible for the conversion of the raw sensor data into the displayed measured value, and the remaining software. The guarantee of correct execution of the core algorithm thus fulfills the basic requirement for the correctness of a measurement, even if the structural integrity of the software image is not present. The indispensable prerequisite for this is the correctness of the result display as well as the authenticity of the raw sensor data.

Another aspect, which is not yet part of the metrological requirements, but which plays a central role, especially for the customer, is the data privacy. The current encryption methods only guarantee security during transport and storage of the data. Since the data often has to be further processed on its way, the processing system has to decrypt the data, process it and re-encrypt the result. If no separate cryptographic hardware is used the secret keys and data are exposed at least to the system operator. It would therefore be

advantageous to be able to do calculations with encrypted data without first having to decrypt them first. The above considerations result in the following minimum security requirements for a distributed measurement device to establish confidence in the correctness of measurements:

- Privacy and authenticity of measurement data
- Authenticity (implying integrity) of the core measurement algorithm

Based on these requirements, possible building blocks for a solution are presented below. Finally, a simple model of a smart metering system based on selected blocks is presented and evaluated.

3 OVERVIEW OF SECURITY PRIMITIVES

The usual way to ensure privacy and authenticity is the use of cryptography. The purpose of this section is to give the overview of the existing cryptographical primitives which can be utilized in our smart meter example to ensure the authenticity and privacy of measurement data as well as the authenticity and integrity of the core measurement algorithm.

3.1 Re-execution of Core Algorithm by Trusted Third Party

The simplest way to verify the correctness of a calculation result is to re-execute it. At first glance, the re-execution of the core algorithm appears to be an elegant and simple solution, but a closer look reveals some serious issues. The re-executing party must be supplied with a copy of the manufacturer's algorithm as well as the relevant measurement data, thus it has to be trusted by all stakeholders which is not easy to achieve. Assuming the use of encryption and signatures to provide data authenticity and transport privacy this platform also contains all the secret keys. This combination makes the platform a popular target for cyberattacks. If this platform is operated by the device manufacturer, the question arises if the core algorithm on the re-execution platform and the measurement device are the same as that has been certified by the notified body. It can be summed up that this simple approach has some structural and practical vulnerabilities that cannot be easily circumvented. Therefore, this approach has no practical relevance and can be discarded.

3.2 Homomorphic Encryption

Homomorphic encryption allows calculations to be performed on encrypted data without having to decrypt it first which primarily addresses the problem of data privacy mentioned before. Formally homomorphic encryption could be described as a tuple of algorithms $[Gen, Enc, Dec, Eval]$:

- $(pk, sk, evk) \leftarrow Gen(1^{\lambda}, \alpha)$ is the key generation algorithm, λ the unary security parameter and α denotes auxiliary inputs. The result of key generation is a key-triple, where pk is the public key, sk is a secret key and evk is an evaluation key. Usually the evaluation key is considered to be part of the public key.
- $c \leftarrow Enc(m, pk)$ describes the encryption algorithm, where $m \in P$ is the plain text message and $c \in X$ is the ciphertext, where P and X are the corresponding plaintext and ciphertext spaces.
- $c_R \leftarrow Eval(c_1 ... c_n, evk, f)$ is the evaluation algorithm. It takes as inputs a tuple of encrypted values and the evaluation key and produces an encrypted result $C_R \in X$. The input f formally defines the function to be evaluated.
- $m \leftarrow Dec(c, sk)$ defines the decryption function

While some ciphers have homomorphic properties related to a single arithmetic operation, such as (unpadded) RSA, Paillier, and ElGamal cryptosystems, for a long time there was no fully homomorphic encryption that theoretically enabled the execution of arbitrary functions. The situation changed in 2009 as Craig Gentry proposed the first fully homomorphic encryption scheme(Gentry, 2009). Since then, numerous other schemes have been proposed and successfully implemented, for instance(Brakerski et al., 2012) and (Smart and Vercauteren, 2010). From a practical perspective, two classes of homomorphic encryption schemes could be distinguished, the levelled and bootstrapped schemes. In the first case, the depth of the circuit (function) cannot be subsequently changed, so that the number of operations to be performed is limited in advance. In the second case, the circuit depth is theoretically unlimited, but requires an additional computationally expensive operation, the bootstrapping, which renews the encryption on a regular basis. One possible application of a bootstrapped scheme is privacy protection in the area of cloud computing. An example would be a distributed measuring instrument consisting of several sensors, a cloud-based computing unit and secure displays. The entire calculation in the processing unit is encrypted and the secure display has the only secret key to decrypt the result.(Oppermann et al., 2017) While this method

ensures the anonymity of the measurement data, no statement can be made regarding the integrity of the algorithm.

3.3 Proof Systems

The verification of correctness in case of outsourced computations is an important general distributed computing issue. One possible solution is provided by proof systems, whereby the server executing the computation provides proof of its correctness in addition to the result of the computation. Formally, such a proof system can be generally described as a tuple of probabilistic polynomial-time algorithms:

- $(vk, sk, evk) \leftarrow Gen(1^{\lambda}, f)$ is the key generation algorithm, λ the unary security parameter and f denotes the algorithm to be proven. The result of key generation is a key-triple, where *vk* is the verification key, *sk* is a secret key and *evk* is a public evaluation key.
- $\sigma_x \leftarrow InputEncode(x, sk)$ describes the process of encoding a given input x.
- $\sigma_y \leftarrow Prove(\sigma_x, evk)$ is the proof generation algorithm, which generates a proof of correctness for the calculation determined by ek and encoded input data.
- $y or \perp \leftarrow Verify(\sigma_y, pk)$ confirms or refutes the correctness of the calculation.

Roughly, such proof systems can be differentiated into interactive and argument-based systems. While interactive systems assume a super-polynomial prover, the argument-based systems establish a polynomial-bound prover and are thus more practice-oriented. The argument-based approaches can also be realized as non-interactive proof systems, which is important from a practical point of view, since the additional communication between the verifier and the prover can ideally be limited to sending the proof. Especially the efficiency of such processes plays an important role in their practical application. In this context, the literature distinguishes between absolute efficiency and amortized efficiency. In the first case, the total computation time for calculating and evaluating the proof is set in relation to the computational time required by the algorithm to be verified. In the second case, only the time necessary for verification is considered. Many practical implementations therefore distinguish between the setup phase, which is executed once for each algorithm and involves a high computational effort and the proof and verification phase, which is set in relation to the computation time of the algorithm to be tested and serves as an important efficiency measure. Some of the approaches have already gained practical importance and offer implementation evaluations such as Pinocchio(Parno et al., 2016), SNARKS for C(Ben-Sasson et al., 2013) and Geppetto(Costello et al., 2015). The main purpose of such proof systems is to prove the correctness of the (unencrypted) execution of an algorithm, therefore they could serve as a replacement to the software integrity hashes, especially in cloud environments. Even if they solve the problem of algorithm integrity checking, the core algorithm must continue to run with unencrypted data in the distributed system, which will not solve the privacy problem.
It is conceivable to combine such a proof method (input privacy assumed) with homomorphic encryption in order to take advantage of the benefits of both approaches. From a practical point of view, it must be said that proof methods and homomorphic encryption require considerable computational resources and generally have a strong dependency on the depth of the circuit which represents the core algorithm. However, in environments where computing power does not matter much, such a solution may be applicable.

3.4 Functional Encryption

Functional encryption, similar to homomorphic encryption, allows calculations to be performed on encrypted data. The main difference, however, is that the functional encryption provides a plaintext result of the calculation. Formally, a functional encryption can be defined as a tuple of probabilistic polynomial time algorithms:

- $(pk, sk) \leftarrow Setup(1^{\lambda})$ is the key generation algorithm, λ the unary security parameter. The result of key generation is a key tuple, where *pk* is the public key and *sk* is a secret key.
- $ek \leftarrow Gen(sk, f)$ generates an evaluation key with respect to a function f
- $c \leftarrow Enc(x, pk)$ encrypts a value x
- $f(x) or \perp \leftarrow Dec(c, ek)$ computes $f(x)$ and provides the decrypted result.

The functional encryption was proposed in 2005 (Sahai and Waters, 2005) and formalized in 2011 (Boneh et al., 2011). Theoretically, the functional encryption allows execution of arbitrary functions on encrypted data, but in practice there are only a few implementations with a strong limitation on the functions to be performed. In contrast to homomorphic encryption, the calculations cannot be cascaded because the result is decrypted directly after an (albeit complex) operation, which also undermines output privacy.
Nevertheless, such a method offers many advantages. First, it guarantees privacy of the input data, if we as-

sume that the function $f(x)$ is difficult to invert. At the same time, it guarantees the integrity of the algorithm because the evaluation key is tied to the function. Thus, it combines the properties of homomorphic encryption and the proof systems. Another advantage is that the input data can only be decrypted in the context of functions for which the evaluation keys have been generated. If the secret key is securely deleted after a single use, the encrypted data cannot be used for purposes other than the specific function(s).

All these features serve to create trust between the executing party and the data provider. A key weakness of functional encryption for its use in distributed computation lies in the fact that the calculation result is unencrypted. Thus, the executing party can falsify the result. However, there is the possibility of separating the calculation execution and the result decryption, depending on the scheme used. In our construction, such a separation is possible, but not necessary. An essential class of functional encryption is offered by the inner-product schemes (Abdalla et al., 2015), (Agrawal et al., 2016). They allow the calculation of a scalar product of two vectors, where one of the vectors is encrypted and the second represents the function $f(x)$. In the following, the scheme based on Decisional Diffie-Hellman from the Cifer library (Cifer, 2019) is used: due to its relatively simple implementation of the encryption and a relatively small ciphertextsize it is a viable candidate even for the integration on resource-limited devices.

3.5 Blockchain Technology

Recent years have seen the evolution from centralized computational storage to decentralized architectures and systems. Distributed ledger technology innovation is one of the key developments making this move conceivable which includes smart contracts and blockchain technologies. Smart contracts are the self-executing software into the ledger. As a peer-to-peer electronic cash system, the blockchain technology first came up with the Bitcoin cryptocurrency, published by an author under the pseudonym Satoshi Nakamoto (Nakamoto, 2008).

In blockchain technology, blocks are linked in sequential order and having a valid network since each block contains the cryptographic hash of previous block. To authenticate and verify the data, each block holds a permanent timestamp. There can be two forms of blockchain platform named as permissionless and permissioned. In permissionless blockchain, anybody can join and take part in the network consensus while in permissioned blockchain consensus can be achieved by known identifiers. For achieving better transaction latency and throughput, permissioned blockchain consensus protocol needs less computational resources (Melo et al., 2019). This enables blockchain technology to store not only data but also to define inter-participant roles and rules. The immutability feature makes the blockchain more useful once a transaction is written onto the blockchain and cannot be erased or, at least, it would be extremely difficult to change.

Our blockchain implementation setup will take advantages of these features. Taking the specific constraints of legal metrology into consideration we have chosen the permissioned blockchain Hyperledger Fabric (Hyperledger Project) for our proof of concept implementation. Hyperledger Fabric has a modular architecture that enables the configuration of smart contracts (called also chaincode in Hyperledger terminology) which are then executed within the Docker containers. In our experiments we have used version 1.4 of Hyperledger Fabric (Fabric, 2019).

4 SMART METER MODEL

A classic smart metering system consists of several sensors that produce measurement values in given time intervals. The sensors are connected to a gateway, which is used for storing and further processing of the measurement data. The gateway thus represents a central unit for which special security precautions must be taken. Customers as well as the energy provider are able to make requests to the gateway. A special role in this construct is the gateway administrator, who has privileged access to the gateway and can change the configuration.

4.1 Proposed Approach

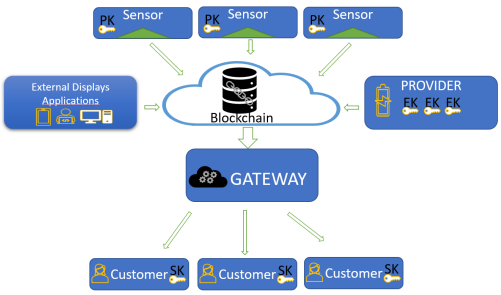

Figure 1: Proposed Smart Metering Model.

The proposed model (see Fig. 1) closely follows the existing smart metering model but has a different distribution of the component roles. The central assumption is that the customer is a proprietor of the gener-

ated measurement data and that the energy provider is only interested in learning the total consumption for billing purposes in a given time period. The sensors are classified as trustworthy and are regularly checked for manipulation by market surveillance. The distribution and storage of measurement data is done on the blockchain. In the following, the entire process is outlined as an example in order to clarify the functioning of the proposed overall system.

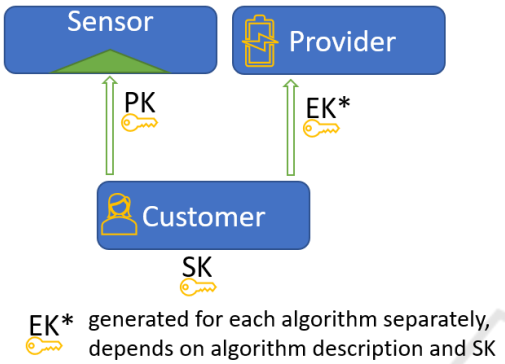

Figure 2: FE-Key Setup Procedure.

In an initial setup step the consumer securely generates a key pair consisting of a secret (SK) and a public functional key (PK). The public functional key is transmitted to the sensor via a secure channel (provided by the gateway). In the second step the consumer and the provider agree on a suitable tariff model, according to which the consumer generates an evaluation functional key (EK). This key is transferred to the provider by means of another secure channel (see Fig. 2). In the last step the sensor is registered on the blockchain and on the gateway.

The blockchain plays the role of a central distributed storage for encrypted measurement data. The sensors periodically encrypt their measurements using their public key (PK) and store the result into the blockchain. After a fixed period of time, these encrypted consumption values are read by the energy supplier from the blockchain and the total sum is calculated with the aid of the corresponding evaluation key (EK). If the function specified by the evaluation key differs from that used by the manufacturer for the calculation and decryption of the sum, the decryption fails. This ensures that both the customer and the provider reach consensus about the function to be performed. The provider has no knowledge of the individual measured values, while the customer can decrypt the individual values and calculate the total consumption on his own. The application of the described method ensures confidence in the algorithm to be executed between the customer and the provider. At the same time, the privacy of the customer is maintained.

In addition to data storage, calculations can be defined to take place within the scope of the blockchain; such programs are generally called smart contracts. These can also be statistical surveys, whereby the customer would have to be asked explicitly each time to generate a corresponding evaluation key. Similar approaches already exist for solutions based on homomorphic encryption (Stan et al., 2018). Also, the update process of existing tariff models can be realized much more easily by using smart contracts and issuing new functional keys, as this does not require any updates of the actual gateway software. The gateway must accommodate less functionality and can therefore be made simpler and possibly with relaxed security requirements while still providing its interfaces to the customer and the energy provider. The administration of the blockchain network can still be done by the original gateway administrator.

The public and redundant nature of the blockchain provides transparency and data authenticity, while the use of functional encryption guarantees data privacy and algorithm authenticity.

4.2 Threat Model

Our approach protects the privacy of customer measurement data and establishes trust between the customer and the service provider regarding the use of the data. In our threat model, service providers are considered to run a trustworthy security infrastructure, including the PKI, but are potentially curious to learn individual usage numbers of their customers, making them attackers on the ”privacy” aspect. Furthermore, we consider the network and third parties as untrusted, so we have to choose a protocol scheme that is resistant to eavesdropping, message manipulation and replay from others. Indeed, our chosen functional encryption scheme is secure under the s-IND-CPA assumption; however, careless generation of evaluation keys may allow third parties to gather information through means of statistical inference. Thus the responsibility of maintaining the security of the measurement data lies with the customer who holds the secret key and is not restricted in any use of his measurement data.

5 IMPLEMENTATION DETAILS

To evaluate the performance, we ran our experiments on an Intel Server S2600CW with 256 GB RAM and two Xeon E5-2650 v4 CPUs running at 2,2 GHz with

24 physical and 48 virtual cores in total. Our software environment is based on Ubuntu Bionic Beaver (18.04.1 LTS). This server hosts the blockchain network along with the client application.
The functional tests were performed on top of Docker containers as Hyperledger Fabric is using docker for containerization and network virtualization. The system was configured as minimal setup with 1 peer, 1 orderer container with solo-orderer service which derives solo consensus mechanism.
Peer (peer0 container) used in our experimental setup was configured both as an endorser and a committer. Therefore, for executing the transaction's required chaincode, an additional chaincode container (dev) is used. The Orderer container is necessary to order transactions and create new blocks establishing consensus. CouchDB has been used to keep the peer ledger state. Additionally, we have created a CLI (command line interface) container to interact with the network.
For both the customer and provider application, we have used one of the available APIs (fabric-java-sdk) to generate a transaction proposal. The SDK sends the transaction proposal to peer0. peer0 plays the special role as the trusted peer which is able to execute smart contracts. It also verifies whether the transaction proposal is well-formed and has not been submitted already in the past (replay-attack protection), and checks the authorization of the customer and provider to perform the proposed operation (Fabric, 2019). The peer0 instance and the peer "dev" (which executes the chaincode) gets assigned one CPU core each. Orderer peers and our client applications are freely distributed over remaining CPU cores.
Such a configuration makes it possible to explore the limits of the smallest possible Hyperledger installation, which in principle can be implemented on a middle-class consumer computer.
In addition to the blockchain installation, there are two applications developed by us (Java) that simulate measuring instruments and the provider application. Both applications are multithreaded, which allows multiple parallel instances to be simulated. Each meter (instance) generates random measurement values, which are then individually encrypted and written to the blockchain. To simulate maximum utilization of the system, the encryption and sending of the data is not done in a 15-minute cycle (BSI, 2019), but directly after receiving the confirmation that the previous data was written to the blockchain. The number of parallel application instances (threads) is varied accordingly, as well as the key length of the functional encryption, which directly affects the size of the ciphertext.

6 RESULTS

When evaluating the results, it should be noted that the generation of each key is a one-time process and therefore not time-critical. The calculation and the decryption of the result takes place monthly and is therefore also less time-critical. The encrypting and storing a single measurement value in the blockchain should not exceed the sensor reading intervall (e.g. 15 minutes for electricity meters) (BSI, 2019), which is a natural limit to the number of possible sensors. The evaluation of the results is done in the order of the processes and is structured according to the respective of an active participant (consumer, measurement sensor and provider client).

6.1 Setup Step

The first initial step concerns the commissioning of the device and is carried out once. The generation of functional key us carried out by customer and takes most of the time in the registration procedure for the sensor(fig. 3). Depending on the size of the key, the time increases significantly. Since this process runs only once, this effort is acceptable, but requires at least a consumer-grade computer (mobile platforms would be less suitable).

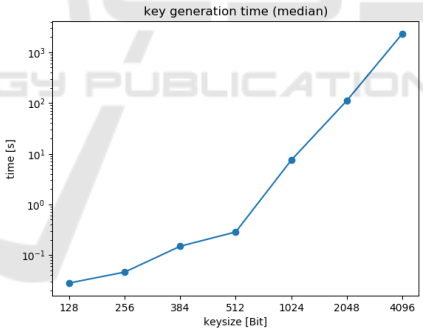

Figure 3: Keygeneration Time.

6.2 Measurement Instrument

The measurement instrument calculates the actual measurement value, encrypts it with functional public key and puts the value on the blockchain using REST protocol. From the figure 4 it can be seen that the encryption time in relation to the time necessary to store the value on the blockchain does not play a significant role.
It should be taken into account that encryption is done on a relatively low-computing device, so the encryption times could be significantly higher. However, the figure 4 shows that the current encryption time could increase by up to the factor 20,000, without exceeding

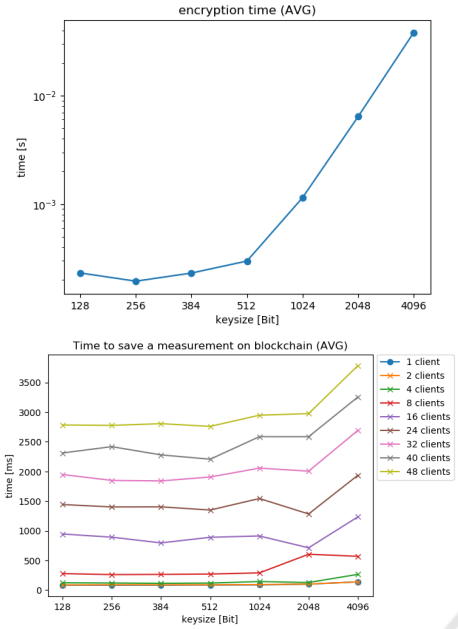

Figure 4: Encryption Time and Time to Store the Value on the Blockchain.

the interval of 15 minutes between successive readouts. The time required for a value to be stored on the blockchain depends heavily on the utilization of the blockchain and does not show any significant dependence on the size of the value to be stored (key size). Considering the performance data of the two containers peer0 and dev (fig. 5), a clear dependency on the number of connected measurement devices (clients) could be seen. While the utilization of the peer increases with increasing number of measuring devices, the utilization of the dev container decreases, which indicates a bottleneck in the peer container.
It can be concluded that an increase in the number of peers would have a positive effect on the overall performance. At the same time there is a dependency on the value size in the dev container, so that a larger key size might require more dev containers. Similar statements can be made when considering the number of transactions per second. The transaction rate seems to break down from at least 8 simultaneous client applications, which can primarily be explained by the high load of the peer container. Nevertheless, in case of optimal system utilization with 8 simultaneous blockchain transactions and assuming the transaction time of 500 ms 1800 measurement sensors could be served at the same time without violating the maximum sensor reading intervall of 15 minutes (cf. fig. 4). Admittedly, it is an ideal case that requires perfect synchronization, in reality the limit would be correspondingly lower.

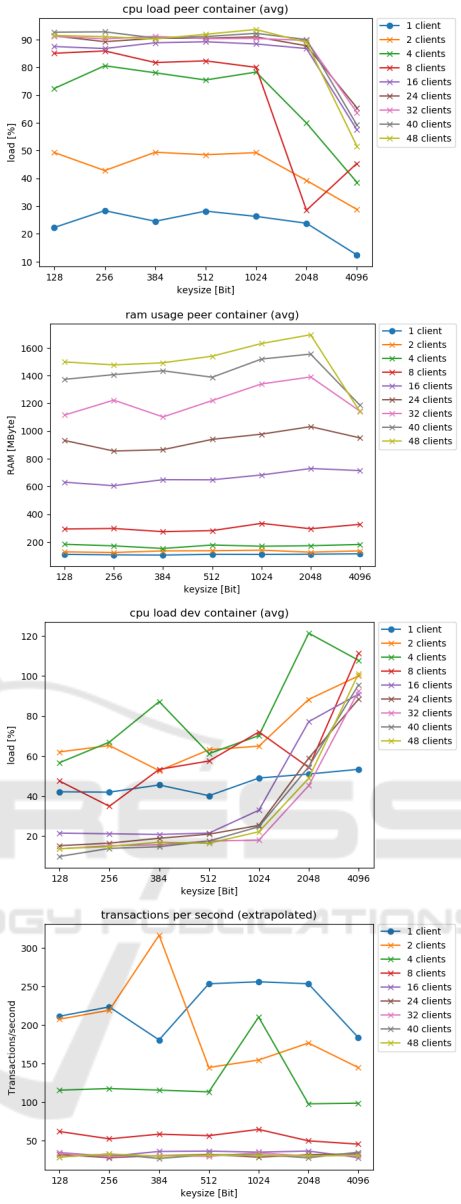

Figure 5: Blockchain Statistics.

6.3 Provider Client

The provider client application is used to collect all the measurement values for each of the measurement instrument at the end of a billing period and to decrypt the overall consumption. Since this operation is performed only at the end of a billing cycle, it is less time-critical and the provider usually has access to higher computing capacity than the customer. At the same time, the blockchain allows parallel access, so billing can be done on multiple parallel instances. Nonetheless, efficiency also plays an important role.

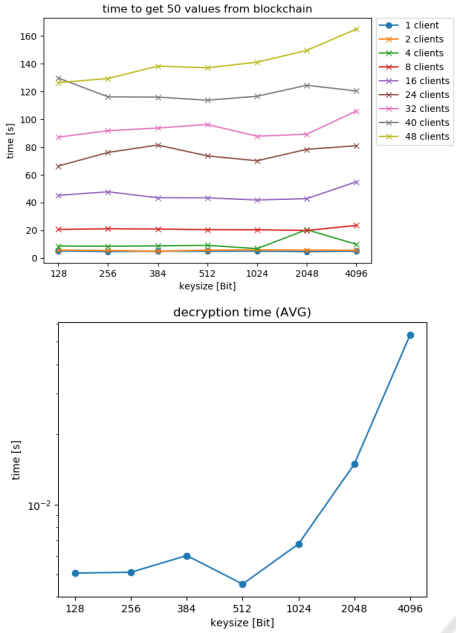

Figure 6: Provider Application (Time to Get All Values and Decryption Time).

The decryption time is short in relation to the time it takes to read a value from the blockchain and the latter has an approximately linear dependence on the number of measured values. We have determined a number of measured values to be 50 to reduce the simulation time (fig. 7). Especially the peer serves as a performance bottleneck as the number of parallel instances increases. Similarly to the cpu load the memory consumption in the peer increases significantly compared to the consumption caused by meter accesses (fig. 7), while the memory consumption of dev container remains below 500 MByte. It is therefore worth increasing the capacity of the peer to ensure optimal throughput and utilization of the dev container.

7 CONCLUSION

It can generally be concluded that the application of functional encryption in combination with blockchain technology opens new perspectives in the field of legal metrology. It allows the customer to determine the use of his data while at the same time giving him the security and confidence in the correctness of measurement results, thereby fulfilling the basic requirements of legal metrology. By using the logical separation presented in this publication, it is possible to define the protected algorithm core, which gets cryptographically secured to provide algorithm authentic-

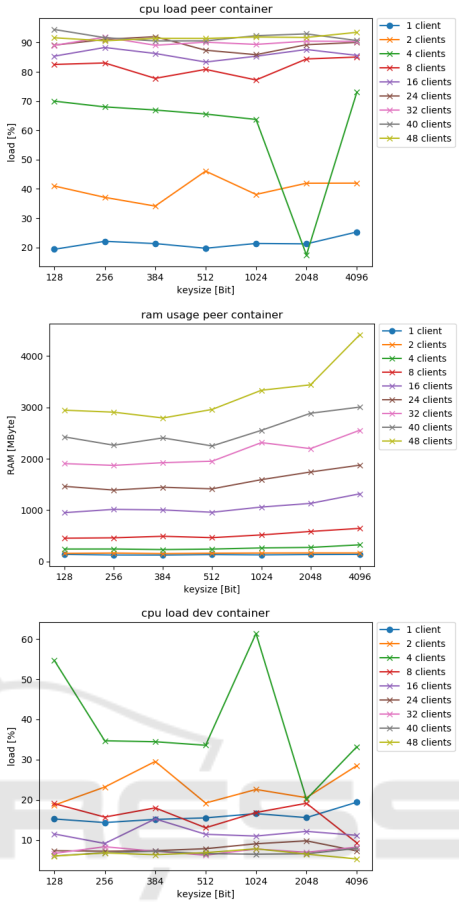

Figure 7: Blockchain Statistics Provider Application.

ity as well as privacy and data authenticity. The concept presented in this publication uses a combination of functional encryption and blockchain technology to achieve these goals. The results of the simulations show the advantages and limitations of both technologies. Furthermore, we have proven that a practical, energy efficient system can be designed even with restricted computing capacity of intelligent sensors and without requiring high computing power from the service provider.

REFERENCES

Abdalla, M., Bourse, F., Caro, A. D., and Pointcheval, D. (2015). Simple Functional Encryption Schemes for Inner Products. In *Lecture Notes in Computer Science*, pages 733–751. Springer Berlin Heidelberg.

Agrawal, S., Libert, B., and Stehlé, D. (2016). Fully Secure Functional Encryption for Inner Products, from Standard Assumptions. In *Advances in Cryptology – CRYPTO 2016*, pages 333–362. Springer Berlin Heidelberg.

Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., and Virza, M. (2013). SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *Advances in Cryptology – CRYPTO 2013*, pages 90–108. Springer Berlin Heidelberg.

Boneh, D., Sahai, A., and Waters, B. (2011). Functional Encryption: Definitions and Challenges. In *Theory of Cryptography*, pages 253–273. Springer Berlin Heidelberg.

Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (2012). (Leveled) Fully Homomorphic Encryptionwithout Bootstrapping. In *ITCS Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*.

BSI (2019). Technische Richtlinie BSI TR-03109-1: Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems.

Cifer (2019). https://github.com/fentec-project/CiFEr retrieved 28. oct. 2019.

Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., and Zahur, S. (2015). Geppetto: Versatile Verifiable Computation. In *2015 IEEE Symposium on Security and Privacy*. IEEE.

Esche, M. and Thiel, F. (2015). Software Risk Assessment for Measuring Instruments in Legal Metrology. *FedCSIS, Vol. 5*.

Fabric, H. (2019). https://hyperledger-fabric.readthedocs.io/en/release-1.4/txflow.html retrieved 9. dec. 2019.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of STOC*.

Melo, W. S., Bessani, A., Neves, N., Santin, A. O., and Carmo, L. F. R. C. (2019). Using Blockchains to Implement Distributed Measuring Systems. *IEEE Transactions on Instrumentation and Measurement*, 68(5):1503–1514.

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. *Bitcoin: A Peer-to-Peer Electronic Cash System*.

OIML (2008). *General requirements for software controlled measuring instruments, OIML, 2008*.

Oppermann, A., Grasso-Toro, F., Yurchenko, A., and Seifert, J. P. (2017). Secure Cloud Computing: Communication Protocol for Multithreaded Fully Homomorphic Encryption for Remote Data Processing. In *IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA 2017)*.

Parno, B., Howell, J., Gentry, C., and Raykova, M. (2016). Pinocchio. *Communications of the ACM*, 59(2):103–112.

Peters, D., Thiel, F., Peter, M., and Seifert, J.-P. (2015). A secure software framework for Measuring Instruments in legal metrology. In *IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*.

Sahai, A. and Waters, B. (2005). Fuzzy Identity-Based Encryption. In *Lecture Notes in Computer Science*, pages 457–473. Springer Berlin Heidelberg.

Smart, N. P. and Vercauteren, F. (2010). Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In *Public Key Cryptography – PKC 2010*, pages 420–443. Springer Berlin Heidelberg.

Stan, O., Zayani, M.-H., Sirdey, R., Hamida, A. B., Leite, A. F., and Mziou-Sallami, M. (2018). A New Crypto-classifier Service for Energy Efficiency in Smart Cities. In *Proceedings of the 7th International Conference on Smart Cities and Green ICT Systems*. SCITEPRESS - Science and Technology Publications.

Thiel, F. (2018). Digital transformation of legal metrology - the European Metrology Cloud. *OIML Bulletin, Vol. LIX, Nr. 1*.

WELMEC, . (2018). *WELMEC 7.2 Software Guide (Measuring Instruments Directive 2014/32/EU1), 2018*.