Blockchain-Based Reference Architecture for Automated, Transparent, and Notarized Attestation of Compliance Adaptations

Thorsten Weber 1,* and Rüdiger Buchkremer 2

1 Faculty of Legal and Business Sciences, Universidad Católica San Antonio de Murcia, Guadalupe, 30107 Murcia, Spain
2 Institute of IT Management and Digitization Research (IFID), FOM University of Applied Sciences, 40476 Dusseldorf, Germany; ruediger.buchkremer@fom.de
* Correspondence: tweber@alu.ucam.edu; Tel.: +49-241-91998505

Featured Application: This paper presents an approach to using blockchain technology to configure automated, transparent, and cryptographically signed cloud applications.

Abstract: With cloud computing, organizations must comply with applicable laws, policies, and best practices. Companies typically rely on cloud service providers to implement and adopt regulations. This consulting phase is often time-consuming, costly, and not transparent. Organizations must trust the third party’s implementation and associated documentation processes. To resolve this dilemma, we present a blockchain-based reference architecture for the automated, transparent, and notarized attestation of such compliance adaptations. Before proposing a solution, our approach is to understand the underlying research context. We conduct a machine-learning-supported systematic literature review to create a knowledge base. A reference architecture, including a prototype for configuring intrusion-detection systems, is developed using design science research. A mixed-methods-based approach is used for the evaluation of the proposed architecture. A quantitative survey is then used to show that the user experience of the developed prototype can be rated as positive, with an average value of 0.7. Finally, two focus group discussions are used to analyze the presented prototype qualitatively. As a result, we demonstrate how to actively support secure and trustworthy communication between a cloud service provider and an organization applying blockchain configurations.

Keywords: cloud computing; software; configuration; compliance; blockchain; smart contracts; business process

1. Introduction

With the Fourth Industrial Revolution, digital technologies have reformed how companies conduct business [1]. Following the advancement in the digital economy and the emergence of big data, cloud computing has become a popular choice of technology for reducing processing burdens on users. Broadband network access and on-demand services provide many advantages, including scalability, availability, and reduced hardware and maintenance expenses [2,3]. Cloud computing has gained immense significance, and may generate investments, employment opportunities, and tax revenues [4]. Furthermore, the implementation of cloud software enables the creation of novel business models [4]. Cloud computing allows enterprises to pay on demand instead of developing applications on in-house data centers [5]. By purchasing cloud software, enterprises can extend their service portfolio without the need for an in-house software development department.

However, despite the considerable benefits and opportunities associated with cloud services, security, privacy, and trust concerns have hindered cloud technology’s potential utilization and mass deployment [3,5,6]. Research reveals that trust issues in cloud...
computing arise because the data are no longer regulated by the companies themselves, and are instead managed by second or third parties [6]. The trust challenges that arise for companies can be divided into compliance and cost challenges.

Enterprises have to meet the compliance requirements defined by the International Standards Organization (ISO) 27000-series, British Standard (BS) 7799, Payment Card Industry Data Security Standard (PCI-DSS), Information Technology Infrastructure Library (ITIL), or Control Objectives for Information and Related Technology (COBIT) [7]. However, enterprises have limited control over whether the requirements stipulated in their standards are implemented using cloud services. Enterprises must trust cloud providers to ensure that agreements are fulfilled as negotiated and that the provided data will not fall into the wrong hands. This trust includes, for example, the specified data storage location. The enterprise must trust that all of its data will be stored as contractually agreed.

In addition to the challenges experienced in compliance, there are planning challenges—especially in terms of the planning of costs. Hidden costs can also negatively impact the adoption of cloud services [8]. If, for example, enterprises’ compliance requirements require changes (e.g., the data storage location has to be reselected, the backup frequency is changed, or design adaptations must be made), this will incur costs. Changes in cloud applications are typically commissioned via consultants for the application [9]. Enterprises communicate their changes to the cloud service consultants. The consultant consequently passes on these change requests internally, or implements them on behalf of the customer. Adaptations are routinely associated with costs for the consultant service, as well as a time delay [10]. A survey of 250 IT managers in 2015 revealed that over 70% of all companies that relied on additional cloud services were confronted with unforeseen or unexpected costs [11]. Unexpected expenses can thereby easily double the initial budget [12].

Transparency is an important mechanism responsible for removing the trust barrier between enterprises and cloud application providers [13]. In a broader context, this article addresses trust issues in cloud technologies. Within the area of trust, this study focuses on compliance requirements. In particular, it focuses on the transparency, traceability, and automation of recurring cloud compliance adjustments for increased specificity. This study presents a blockchain-based reference architecture and a business process that allows compliance adjustments to be costed transparently, automated without consultants, and made traceable using unique blockchain-based evidence. This approach is generic regarding the cloud application to be configured and the cloud provider involved.

The remainder of this research is structured as follows: Section 2 provides a machine-learning-supported systematic literature review on existing trust-enhancing mechanisms used in cloud computing. Section 3 provides the basic definitions and fundamentals required to understand cloud configurations. Section 4 presents the general approach used for the notarized configuration of cloud applications, including the development of a reference architecture prototype. The evaluation of the proposed architecture is given in Section 5. Therefore, the developed prototype is evaluated using quantitative and qualitative methods. The study’s limitations are discussed in Section 6. Finally, Section 7 provides a conclusion for this research.

2. Related Work

This work takes inspiration from problem-solving-focused engineering disciplines. For this purpose, the design science research (DSR) methodology was followed, which has a problem-oriented approach. According to this methodology, an artifact based on an existing problem is created to solve the issue. Moreover, this study adopted the related information systems research framework for the proposed approach, as described by Hevner et al. [14]. For the literature review, this work implemented the systemic taxonomy for information retrieval from literature (STIRL) approach proposed by Buchkremer et al. [14]. This approach allows for a machine-learning-supported systematic literature review. Thus, it can also be applied to examine large volumes of research articles. Qualitatively highly maintained
online research databases were selected as data sources [15,16]. As shown in Table 1, the literature was selected using online literature databases and search strings.

### Table 1. Knowledge database.

<table>
<thead>
<tr>
<th>Search Term I</th>
<th>Web of Science (WOS)</th>
<th>IEEE</th>
<th>Sage</th>
<th>Science Direct (SD)</th>
<th>MDPI</th>
<th>Wiley</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cloud * AND Blockchain * AND Trust *</td>
<td>Cloud * AND Blockchain * AND Trust *</td>
<td>Cloud AND Blockchain AND Trust</td>
<td>Cloud * AND Blockchain * AND Trust *</td>
<td>Cloud AND Blockchain AND Trust</td>
<td></td>
</tr>
<tr>
<td>Search Term II</td>
<td>Cloud * AND Blockchain * AND Compliance *</td>
<td>Cloud * AND Blockchain * AND Compliance *</td>
<td>Cloud AND Blockchain AND Compliance</td>
<td>Cloud * AND Blockchain * AND Compliance</td>
<td>Cloud AND Blockchain AND Compliance</td>
<td></td>
</tr>
<tr>
<td>Search Field I</td>
<td>Publication Title</td>
<td>Publication Title</td>
<td>Publication Title</td>
<td>Publication Title</td>
<td>Publication Title</td>
<td>Publication Title</td>
</tr>
<tr>
<td>Search Field II</td>
<td>Abstract</td>
<td>Abstract</td>
<td>Abstract</td>
<td>Abstract</td>
<td>Abstract</td>
<td>Abstract</td>
</tr>
<tr>
<td>Additional Requirements</td>
<td>Proceedings Papers, Review Articles</td>
<td>Journals, Conferences</td>
<td>Research Article, Review Article</td>
<td>Review articles, Research articles</td>
<td>Article, Review</td>
<td>Journals</td>
</tr>
<tr>
<td>HITS</td>
<td>757</td>
<td>465</td>
<td>165</td>
<td>65</td>
<td>25</td>
<td>18</td>
</tr>
</tbody>
</table>

Note: Multiple-character wildcard searches with * look for zero, one, or more characters.

When selecting keywords, care was taken to ensure that they were broad enough to identify all relevant papers, but narrow enough to avoid returning too many articles and, thus, possibly corrupting the results. Since compliance is a subtopic of trust, “Trust” and “Compliance” were used as keywords. The other keywords selected were “Cloud” and “Blockchain”. These keywords were then combined to specify the search. To ensure that only papers that focused on the selected keywords were identified, only the papers’ titles and abstracts were searched. In total, 1495 peer-reviewed papers were found. After removing duplicates, 957 papers remained. Our focus was on reviewing high-quality scientific literature, as is generally recommended in research [17]. Therefore, we focused on peer-reviewed articles.

Finally, the text corpus consisted of the titles and abstracts of 957 papers. In the next step, this corpus was prepared for further text analysis. For this purpose, the Natural Language Toolkit (NLTK) developed in Python was used [18]. Using the NLTK, first, the stop words were removed from the text corpus, and then lemmatization and stemming (snowball algorithm) were applied to bring the content of the text corpus into its root forms [14]. The processed text corpus was then analyzed using linear discriminant analysis (LDA) [19]. Nine trending topics (shown in Figure 1) related to the search strings could be identified. Note that LDA is not a strict clustering algorithm; thus, the same words might be used in multiple topics [19].

The statistical word distribution of each identified topic is shown in Figure 2. The number of identified words in each trending topic is nearly normally distributed. Hence, the authors accepted the LDA-created topics. The nine identified trending topic groups were named as follows: Topic 0: trading; Topic 1: vehicular; Topic 2: access management; Topic 3: payment; Topic 4: authentication; Topic 5: Internet of Things (IoT); Topic 6: cloud computing; Topic 7: legal and trust; and Topic 8: healthcare.

Once the trends and their interrelationships were identified, this work was placed in the context of the trends. This work can be classified into Topic 6 and Topic 7. A total of 220 of the 957 identified papers were assigned here. In a final step, a literature review based on the work of vom Brocke et al. [20] was conducted, starting with the top 10 papers whose content most closely matched the keywords from Topics 6 and 7, which were identified and used in the initial literature for this article.
Figure 1. Identified word groupings.

Figure 2. LDA group word counts.
An approach that dates back to 2018 was presented by Koshiba et al. [21]. The authors implemented communication via the blockchain using a programmable interface. The GET and PUT functions could ensure transparent communication with an untrusted domain. This work utilizes the approach of a programmable interface to communicate with an untrusted environment. Rebello et al. [22] presented a blockchain-based system that secures orchestration operations in virtualized networks, ensuring audibility, non-repudiation, and integrity. In this context, the work presented here takes the approach of non-repudiation and data integrity, and extends it with an automated approach to blockchain/cloud communication.

Several studies aiming to increase trust by using blockchain technology have been published. Demi et al. [23] provided a comprehensive overview of the software engineering applications enabled by blockchain technology. The presented overview provided an initial overview of relevant works for this article. In [24], the authors presented a public mutual audit blockchain (PMAB) for outsourced data in cloud storage. Yang et al. [25] proposed a blockchain-based publicly verifiable data migration scheme supporting cloud integrity checks. Zuo et al. [26] introduced a blockchain-based ciphertext-policy attribute-based encryption scheme for cloud data security sharing. Addressing the trust problem between data owners and cloud service providers, Huang et al. [27] proposed a collaborative auditing blockchain framework for cloud data storage, and showed that their framework has the advantage of preserving remote data integrity from various attacks.

Targeting blockchain-based data sharing, MedChain [28] is one of the first methods implemented for sharing a large amount of data via blockchain. Sato et al. proposed a method to manage system operations using a smart contract for blockchain-based systems [29,30]. The smart contract “OpsSC” manages an operational item’s operational rules and configuration parameters in their proposed method. In addition, operational agents execute operations based on the smart contract, and record the results and evidence of the execution on the chain. Within their study, the authors described system backups as a one-use case. However, our presented reference architecture focuses on the generic configuration of cloud applications and their associated business processes. Therefore, this can be seen as a further development of the approach presented by Sato et al. [29]. In addition to individual applications exchanging data and increasing trust in applications, there are also dedicated platforms for this. Hyperledger Fabric (HF) is an open-source system used for deploying and operating permitted blockchains, and one of the Hyperledger projects is hosted by the Linux Foundation [31]. So-called “System chaincodes” are specialized smart contracts used for managing the configuration values of the blockchain network onto the blockchain and smart contracts. Wang et al. [32] developed a blockchain-based data integrity scheme for large-scale IoT data. They provided an experimental result on Hyperledger Fabric that demonstrates that the proposed verification scheme significantly improves the efficiency of integrity verification for large-scale IoT data, with no need for a trusted third party. Even though HF is a recognized permissioned blockchain framework, we focus on describing an independent architecture [33]. The presented architecture can also be implemented using HF.

Overall, the literature review shows two things: First, a large body of work has already addressed the topic of compliance and trust based on blockchain technology. Second, the literature review also showed that there is currently still a research gap in the cloud application configuration. No work to configure cloud applications could be identified. Therefore, in light of the previous literature, this study considers blockchain technology as a suitable measure to increase compliance and reduce hidden costs within cloud computing. Consequently, from one perspective, the blockchain is used to import fully automated configurations into the cloud application; meanwhile, it also enables the transparent and legally secure traceability of a cloud configuration.
3. Background

This work describes how cloud configurations can be implemented transparently, automatically, and signed using smart contracts. For the background, we would therefore like to define configurations. A cloud configuration is understood as adapting software to comply with requirements. The theoretical basis of this broad definition encompasses the mathematical and logical definitions of interactive systems developed by Broy and Stølen [34]. Thereby, systems are described based on their input/output behavior. This method is called FOCUS [34]. FOCUS allows for the formal definition and development of interactive systems. The main idea of FOCUS is that the relevant interfaces of interactive systems can be described by characterizing their message interaction history. FOCUS can determine the internal interaction between various system components and any interaction with the system environment. Information is exchanged through communication lines called channels between system components. Through such channels, information messages are exchanged. A channel transmitting in only one direction is termed a directed channel. The communication history of a controlled channel is represented through streams. A distinction can be asserted between untimed and timed streams. Untimed streams represent a sequence of messages in their order of transmission.

For defining configurations, timed streams were applied in this study. Timed streams are sequences of messages, including associated time ticks. Discrete time periods in which no message is transmitted are represented by √. Timed streams allow the discrete representation of the time a message arrives in the sequence. While only the arrangement of the messages can be represented with untimed streams, timed streams can also have their temporal appearance modeled. With timed streams, the time at which a message appears and the time interval duration between two messages can be shown. Note that timed streams, which represent the complete communication history, are infinite, since time never stops.

3.1. Formal Definition of Streams

We define the set of messages as \( M \) and \( s \) as a timed stream. If all messages in \( s \) are contained in \( M \), we call \( s \) a timed stream over \( M \). We define \( √ \) as a discrete time period in which no messages arrive. Note that \( √ \) is not a message and, thus, \( √ \notin M \). We define the set of all finite time streams over \( M \) as \( M^\times \), and the set of all infinite streams over \( M \) as \( M^\infty \). Furthermore,

\[
M^\infty := M^\times \cup M^\infty
\]

is the set of all timed streams. Thus, timed streams can be seen as mathematical functions mapping natural numbers to messages.

\[
M^\times := \bigcup_{n \in \mathbb{N}} \left( \{1, \ldots, n\} \rightarrow M \cup \{ √ \} \right)
\]

\[
M^\infty := \left\{ s \in \mathbb{N}_+ \rightarrow M \cup \{ √ \} \mid \forall j \in \mathbb{N} : \exists k \in \mathbb{N} : k \geq j \land s(k) = √ \right\}
\]

We assume that we have a time stream starting with a message \( m_1 \), followed by \( m_2 \). Consequently, we do not receive any messages \( √ \) within the next two time slots, and end with two messages: \( m_1 \) and \( m_3 \). We can characterize this stream by the following function:

\[
s \in \{1, 2, 3, 4, 5, 6\} \rightarrow \left\{ √, m_1, m_2, m_3 \right\}
\]

where

\[
s(1) = m_1, \ s(2) = m_2, \ s(3) = √, \ s(4) = √, \ s(5) = m_1, \ s(6) = m_3
\]

Thus, streams map the indices in their domains to their messages.

3.2. Formal Definition of Configurations

We define \( \Sigma^* \) as the set of all finite texts generated using the ASCII alphabet. Furthermore, based on the definitions of [35], we define \( f : I^\infty \rightarrow O^\infty \) as a stream processing
function that maps an input stream to a corresponding output stream. If we now consider all of the preceding definitions, we define a software configuration as follows:
\[ c : \sum^* \rightarrow (I^\omega \rightarrow O^\omega) \]

Thus, a configuration \( c \) is a partial function that maps an ASCII text \( e \in \sum^* \) to a function that maps an input timed stream to a corresponding output timed stream. Note that \( c \) is a partial function, and maps configurations to the input/output stream only if the configuration is valid for the corresponding system. This definition allows us to change a cloud application’s input/output behavior based on textual commands. A software configuration is thus the user-based definition of an input/output behavior of a certain software package at a given point in time.

To render this more concrete, let us assume that we want to set the behavior of a firewall, and further assume that the behavior of the firewall can be set to textual via \( e = \langle \text{key}_1 : \text{value}_1, \ldots, \text{key}_n : \text{value}_n \rangle \ n \in \mathbb{N} \)

Note that the assumption of a textual configuration is w.l.o.g. The function \( f : I^\omega \rightarrow O^\omega \) consequently describes the firewall’s behavior. For a given input stream, the firewall generates a given output stream. Based on the definition used in this study, the function \( f \) thus depends on the set text \( e \). Suppose that \( e = \langle \rangle \), i.e., the empty set. The result of this would be that each input stream from the firewall would reflect the output stream. The configuration \( e' = \langle \text{port} : 1812, \text{status} : \text{closed} \rangle \) would, in turn, lead to an input stream solely routed to the output stream if it does not involve port 1812. Streams affecting port 1812 would thus be blocked. A configuration \( e'' = \langle \text{noconfig} \rangle \) would not map to any input/output behavior, since \( e'' \) is not a valid setting. The example of \( e'' \) also explains why \( c \) is a partial function. Not every setting can lead to a certain input/output behavior.

4. Results

This section is divided into two parts: First, we present the general software architecture of our developed approach. The focus is on the generic formulation of the design. The second part demonstrates how the generic approach was implemented as a prototype. The evaluation of the proposed software architecture follows in the next section. The source code of the presented work is publicly available on GitHub [36].

4.1. Software Architecture

The general software architecture components are shown in Figure 3. It is evident that our proposed software expects the following three participants: The cloud application provider (provider) is a customer at a cloud vendor that develops an application based on the resources provided by the cloud vendor (e.g., Microsoft Azure, Amazon AWS, Google Cloud). The cloud application consumer (consumer) is a user of the cloud application that the cloud provider offers. This cloud application (application) is configured through a smart contract on a blockchain.
4.1.1. Management Smart Contract

A fundamental aspect of the proposed architecture is that each interaction between the three participants is managed via a smart contract. The management smart contract ensures that each communication step is digitally signed and documented on the blockchain. Consequently, our approach aims to ensure that any cloud configuration to be implemented is also stored on a smart contract. If configurations are directly written to the blockchain, any blockchain network participant could read them. This transparency would be a disadvantage when implementing confidential cloud application configurations. Therefore, cloud application configurations must be encrypted when stored. For this to occur, all communicating parties must have the ability to store and retrieve configurations in an encrypted form. Our approach utilizes the Diffie–Hellman key-exchange protocol [37] to ensure the creation of a shared encryption key for cloud configurations. The sequence diagram in Figure 4 and the smart contract’s function description highlight the sequence used for creating a shared key via the management smart contract. As already mentioned, the management smart contract is a deployed smart contract, through which it is possible to (1) exchange public Diffie–Hellman values, (2) automatically and transparently store and retrieve cloud application configurations, and (3) store and track the cryptographic hash.
values of committed cloud configurations. The cloud application provider initially creates the management smart contract on the blockchain. Following this step, the smart contract is then used by the provider, consumer, and application to configure the application. The basic functions of the smart contract are as follows:

1. Constructor(uint g, uint q, address cloudAppProvider, address cloudAppConsumer, address cloudApp): The Constructor of the smart contract. The unsigned Diffie–Hellman $g, q \in [1, q - 1]$ must instantiate a new management contract. Moreover, the Constructor expects the blockchain addresses for all three communicating parties. The values $g$ and $q$ are later used for creating a shared secret key between the communicating parties. The blockchain addresses of the cloud application provider (cloudAppProvider), the cloud application consumer (cloudAppConsumer), and the cloud application (cloudApp) are all employed for authenticating smart contract requests. All management smart contract requests are authenticated based on the provided digital signature of placed transactions.

2. getSymmetricParameters(): Returns the public Diffie–Hellman key-exchange values $g$ and $q$. The values are initially set at the construction of the smart contract.

3. setFirstSymmetricKey(uint value): Past research [38] has already shown that the Diffie–Hellman key-exchange protocol is vulnerable to man-in-the-middle (MITM) attacks; therefore, public keys should only be exchanged in an authenticated manner. The setFirstSymmetricKey function ensures an authenticated first exchange of the public Diffie–Hellman keys.

4. setSecondSymmetricKey(uint value): At the same time as the setFirstSymmetricKey function, the setSecondSymmetricKey function ensures an authenticated first exchange of the public Diffie–Hellman keys.

5. getFirstSymmetricKey(): Using the setFirstSymmetricKey function, the public Diffie–Hellman keys are stored on the blockchain. The getFirstSymmetricKey ensures that the set public keys can be retrieved from the blockchain.

6. getSecondSymmetricKey(): At the same time as the getFirstSymmetricKey function, the getSecondSymmetricKey function ensures that the public keys set with the setSecondSymmetricKey function can be retrieved from the blockchain.

7. setConfiguration(string c, string t): Using the Diffie–Hellman key-exchange protocol, a shared symmetric key $s$ is created among the three communicating parties. Using the secret key $s$, the cloud application provider and cloud application consumer can create an encrypted configuration $c$ and its authentication tag $t$ [39]. $c$ and $t$ represent the inputs for this function.

8. getConfiguration(): Returns $(c, t)$—the latest set configuration.

9. setStatus(string h): This function should only be executed by the cloud application. It stores a hash value at the smart contract.

10. getStatus(): Returns $h$—the last hash value of a cloud instance successfully implemented the configuration set by setConfiguration.

Consequently, the cloud application provider first creates the initial smart contract on the blockchain using the Constructor. The cloud application provider provides the public Diffie–Hellman key-exchange protocol values $g$ and $q$, along with their blockchain address, the cloud application consumer, and the cloud application. Using the getSymmetricParameters function, all communication participants retrieve $g$ and $q$ from the blockchain. If the participants retrieve values $g$ and $q$, the three-party Diffie–Hellmann protocol is automatically executed using the cloud and consumer management script.
4.1.2. Cloud Management Script

Our proposed software approach relies on a cloud management script to provide a solution independent of the selected cloud service model and vendor. The cloud management script must run on a cloud instance (i.e., a virtual machine in which the cloud application is deployed), and must have a data connection to (1) the blockchain, (2) the provided cloud application, and (3) the cloud management system.

The cloud management script synchronizes with the management smart contract through the blockchain connection. The cloud management script uses a pull-or-push mechanism to monitor the \texttt{getFirstSymmetricKey}, \texttt{getSecondSymmetricKey}, and \texttt{getConfiguration} functions for changes. Such a connection can usually be ensured using remote procedure calls (RPCs).

The cloud management script requires a connection to the cloud application to process a detected application configuration. Consequently, through this connection, new configurations can be implemented. This connection can be ensured via a representational state transfer (REST) interface or an exchange directory.

Moreover, the cloud management script requires connecting to the cloud management console. The cloud provider provides cloud management, and is used to configure existing cloud instances or deploy new cloud services. The cloud management script serves as the underlying configuration layer for cloud instances. For example, the cloud management script can change an existing cloud instance’s memory or CPU capacity, or can launch another virtual cloud instance. The script can store and retrieve snapshots (the current state of a cloud instance). Generally, cloud providers offer a cloud management connection via an application programming interface (API) in a specific programming language, such as Python, (Power) Shell, or Ruby. Brokered by the cloud management connection, the script can store and retrieve snapshots (i.e., the current state of a cloud instance).
Once the cloud management script obtains all three described connections, it needs to monitor the blockchain (through the blockchain connection) for changes in the smart contract. The monitoring may result in two main scenarios:

The cloud management script detects a change triggered by the `getFirstSymmetricKey` function within the first scenario. Consequently, a participant wants to change the secret key $s$. Subsequently, the cloud management script performs the steps shown in Figure 4.

The cloud management script detects modifications when triggered by the `getConfiguration` function within the second scenario. Consequently, a participant wants to change a cloud application configuration. This scenario is shown in Figure 5. If the cloud management script detects such a change, it executes the `getConfiguration` function of the monitored smart contract, and receives the ciphertext $c$ and the corresponding authentication tag $t$. Since the cloud management script has access to the secret symmetric key $s$, it verifies the message tag $t$ and decrypts the configuration to be implemented from $c$. Using the blockchain–cloud application connection, the cloud management injects the new configuration into the cloud application. Concurrently, the cloud application can also use this blockchain–cloud connection to provide feedback on the implementation status (or, alternatively, to monitor the log files). If the implementation fails, the cloud management script writes the implementation error to the blockchain via `setStatus`, terminates the cloud configuration, and continues monitoring the smart contract.

Conversely, if the cloud application reports a positive status (i.e., the implementation of the configuration succeeded), the cloud management script uses the cloud management connection. The cloud management script triggers the execution of a snapshot from the current cloud instance through the cloud management system. This snapshot is a full backup of the current cloud instance that the cloud application is running. Consequently, this snapshot includes the successfully configured cloud application and all of the data required to restore the current cloud instance. Modern cloud vendors store this snapshot in a separate external data storage location. However, the snapshot can be retrieved by utilizing the cloud management connection, and loaded onto the local cloud instance on which the cloud application is running. The cloud management connection downloads the created snapshot from the external data storage onto the local instance using this proposed approach. Subsequently, the cloud management script creates the snapshot’s hash value and deletes the snapshot again from the local instance. The calculated snapshot’s hash value is subsequently written to the blockchain by the cloud management script using the `setStatus` function. The generated hash value is a full backup of the cloud instance under which the cloud application is running. If an entity restores the snapshot associated with

**Figure 5.** Sequence diagram of configuring a cloud application.
the stored hash value, it will have a cloud instance that reflects the immediate status after
the cloud application has been successfully configured. Subsequently, the execution of the
configuration ends in this case. The management script continues monitoring the smart
contract under close observation, as it does in the error scenario.

4.1.3. Consumer Management Script

The consumer management script sets cloud application configurations in a client-
sided manner. Therefore, decentralized applications (dApps) are used [40]. dApps enable
participants to utilize the script described in Section 4.1.2 automatically. The configuration
itself must be defined in a standardized manner. The Extensible Markup Language (XML)
and JavaScript Object Notation (JSON) are two possible formats [41]. The initial starting
point for this communication is that all communicating participants have already executed
the protocol described in Section 4.1.2, and have access to the shared secret key $s$. Conse-
quently, the cloud application receives this key via the cloud management script. The other
two communication participants receive this through the consumer management script.
In summary, the consumer management script allows consumers to (1) participate in the
Diffie–Hellman key-generation protocol, (2) automatically store configuration changes in
the blockchain, and (3) retrieve snapshot hash values from the blockchain.

4.2. Software Implementation

Having introduced the theoretical concept of our software architecture, we describe
how we implemented the software as a prototype. The presented architecture was im-
plemented using rapid prototyping [42]. The source code of the implementation and the
running prototype can be found online [36]. Note that our software architecture presented
in Section 4.1 was formulated generally. Consequently, the approach we present can be
used for configuring a variety of cloud applications. Our prototype is designed for the
configuration of an intrusion-detection system (IDS). The concept of the implementation is
illustrated in Figure 6. The developed prototype should automatically and transparently
configure an IDS and notarize its configuration implementation. The prototype used the
Microsoft Azure Cloud as a cloud instance and deployed two virtual machines (VMs)
of the “Standard B4ms” type, with 5 vCPUs, 16 GiB RAM, and 30 GiB disk storage on
the MS Azure Cloud. For identification, the deployed VMs were termed VM1 and VM2.
Ubuntu Server 18.04 Long-Term Support (LTS) was installed on both VMs as the operating
system. Regarding the notarized configuration of the IDS, the Ethereum blockchain [43]
was subsequently employed to implement smart contracts. On VM1, Ganache Command
Line Interface (CLI) v7.0.4 was installed [44].

Moreover, Ganache CLI was employed to run an Ethereum blockchain for devel-
opment. Since the Diffie–Hellman protocol is vulnerable to MITM attacks, the created
management smart contract must ensure the authenticated provisioning of the public
Diffie–Hellman keys [38]. The blockchain provided by the Ganache CLI could be accessed
through a standardized interface. This standardized interface enabled the script-based
use of the blockchain provided in VM1. Solidity [45] was used as the programming lan-
guage, allowing the management smart contract (Web3.py) to interact with the Ethereum
blockchain via a script-based client [46] and the digital signing scheme that the Ethereum
blockchain provides.

Within VM2, this prototype ran two applications and the proposed cloud management
script. The Microsoft Azure Software Development Kit [47] was employed for the cloud
management connection. The initial application was an Nginx version 1.21.0 web server,
which was used for providing a template website containing static demo web content. The
second application was the IDS, which had to be configured using this proposed approach.
Consequently, the IDS Snort in version 2.9.8 [48]—a signature-based intrusion-detection
system upstream of the webserver—was used. We used a cryptographic hash function to
genenerate the snapshot hash value described in the architecture in Section 4.1.2. The use
of cryptographic hash functions ensures that the hash value generated by the VM can be
used in litigation to prove that a configuration has been changed. It is computationally infeasible to preserve the original input value of a cryptographic hash function [49]. This thus ensures that a generated hash value cannot be implemented with a manipulated VM and, thus, with a forged configuration. The prototype we developed uses the SHA-512 hash function for this purpose. Even though SHA-256 is often used in blockchains, we decided to use the SHA-512 hash function. In the architecture we presented, the operating system performs the hashing. Today’s operating systems mostly run with 64-bit memory management. Research has shown that the SHA-512 hashing algorithm might run faster on larger data sets and 64-bit systems than SHA-256, due to parallelization [49].

Based on our mathematical definition of the configuration $c : \Sigma^* \rightarrow (I^\omega \rightarrow O^\omega)$, $\Sigma^*$ is the English alphabet, which is mapped to an input–output behavior $(I^\omega \rightarrow O^\omega)$ only if the described text represents a valid snort rule.

![Figure 6. IDS prototype architecture.](image)

5. Evaluation

This work aimed to develop an architecture to allow enterprises to implement compliance requirements and monitor costs in an automated, transparent, and tamperproof manner. The evaluation of the proposed architecture must show whether this aim was reached. We developed our proposed architecture based on the design science research described by Hevner et al. [50]. The goal of design science is the development of an artifact (e.g., our developed software architecture). Tremblay et al. [51] describe focus groups as an ideal means of evaluating developed software artifacts. The authors describe focus groups to evaluate an artifact as so-called confirmatory focus groups (CFGs): “The CFGs are used to demonstrate the utility of the artifact design in the application field” and “This allows for the comparison of the results across CFGs to demonstrate and corroborate proof of utility of the artifact.” [51]. Based on the number of focus groups, the authors state that at least two CFGs should be run. The evaluation of our architecture is based on a mixed-methods approach. Qualitative focus group discussions are used to evaluate whether the goal of this work has been achieved. Therefore, the prototype of our proposed architecture will be evaluated. However, before the prototype is presented to the focus groups, its user experience (UX) is examined by utilizing quantitative methods. In other words, before presenting the prototype in the focus group discussion, whether the developed software provides the intended customer journey is evaluated. This quantitative evaluation is needed, as the UX evaluation ensures a distortion-free discussion regarding the UX in the focus groups.
5.1. Quantitative User Experience Survey

The UX was measured before the developed prototype was submitted to the qualitative evaluation. First, this allowed us to ensure that the prototype would be tested again by third parties. Second, measuring the prototype’s UX also reduced or even prevented possible biases due to user interaction in the qualitative analysis. Managers and experts were invited to evaluate the prototype. To ensure that the developed prototype also represented the expected customer journey, and that the experts were not biased by any problems with the user experience of the software, the UX was measured using quantitative methods. The user experience questionnaire (UEQ) developed by Schrepp et al. [52] was used. This approach was based on a questionnaire with 26 questions and 6 scales. Note that questionnaires are a common and effective approach used for evaluating a product and its features [53].

Twenty people took part in evaluating the application presented in this article. The authors of the UEQ explicitly state that it makes no sense to determine a single value representing the 26 medians (e.g., a median of the medians), since this value could not be interpreted in a meaningful manner. Instead, each question corresponded to one of the six scales. Thus, a mean of medians was computed for each scale. According to the authors of the UEQ evaluation tool [52], the individual means can be evaluated in the following manner: The value scale ranges from −3 (extremely bad) to +3 (extremely good) but, realistically, the value of each mean will be somewhere between −2 and +2. This difference in values is because the medians from which the means are computed are based on various responses from various people with diverse opinions and answer tendencies.

Consequently, good results such as +1.5 on a scale of −3 to +3 appear worse than they are. The results of the evaluation are shown in Figure 7.

![Figure 7. User experience evaluation results.](image)

The mean value for attractiveness was 0.86, meaning that the evaluation was positive. However, there was still some room for improvement. With a value of 0.4, perspicuity was perceived as neutral. On the other hand, the results for efficiency paint a completely different picture. Here, the mean was −0.28. Even if the evaluation here did not fall below −0.8 and, therefore, did not fail, improvements needed to be made. One reason for this might be the hash value creation. Using our prototype VM setup, the hash creation took about two minutes. The creation of hash values needed to be improved by having a faster VM setup during the focus group discussion. With a result of 1.24, the application’s dependability was good—there was no need for improvement here for the focus group discussion. The result for stimulation was 0.7; thus, it was neutral and fine for demonstration purposes. Novelty achieved a result of 1.27, which was also extremely good. This high novelty value is probably since users did not yet know of any other application that would have enabled them to configure cloud applications via dApps. All in all, quantitative methods were used to show that the UX of the developed prototype was within the expected customer journey, and that the prototype could be presented to the focus group without any significant UX bias.
5.2. Qualitative Focus Group Discussions

As part of the research methodology, focus group discussions were conducted. To be more precise, two focus group discussions were conducted to obtain higher reliability of statements [51]. The focus groups were conducted as described by Krueger et al. [54], and analyzed using the MAXQDA software [55]. A total of two focus group discussions were conducted, with seven and six people, respectively. The focus groups included two CEOs of software companies developing cloud applications, two software company team leaders, three scientific researchers, and six employees in cloud-based planning, developing, or testing services.

Both focus groups emphasized that compliance is the central issue for practically adopting cloud applications. The focus groups confirmed that compliance issues primarily arise due to the adaptation of internal cloud applications or the sale of cloud applications to customers. The discussions reveal that technical solutions such as a configurable password policy, single sign-on, or multifactor authentication increase security and trust in the cloud application. However, when it comes to implementing service-level agreements by cloud providers (such as backup policy, geolocation of data storage, the configuration of a firewall, and intrusion-detection systems), it becomes essential to trust them. Due to their market position, bilateral agreements with service providers are often possible only to a limited extent. All participants confirmed that existing business processes for adapting compliance requirements have weak points. If disputes arise during the implementation of compliance requirements, the traceability of adjustments is often limited. Adjustments are typically discussed in emails or telephone meetings, and only recorded in notes. It is only possible to trace back exactly who made which decision and when to a limited extent. Consequently, changes can often only be implemented with delays, and are associated with consulting costs. Thus, both focus groups agreed on the need for additional transparency, traceability, and automation requirements in existing business processes.

After discussing the current business state, the developed architecture prototype was presented as a demo. Subsequently, all focus group participants were provided with access to the prototype. The participants were thus able to test the prototype. In summary, the following can be noted from the focus group evaluations of the prototype. First, the participants agreed that they had not seen the presented approach before. Based on the literature review and the quantitative analysis, the approach presented is a new development that does not yet exist in this form. Overall, the experts also confirmed that the proposed prototype and its underlying business process and architecture could support the adoption of cloud services. However, the experts also agreed that the solution presented could mainly be used by small and medium-sized enterprises (SMEs), as introducing the proposed architecture might be very complex and resource-intensive in a large-enterprise environment.

Furthermore, the experts see the presented approach primarily in the software-as-a-service area. They considered it as unlikely that the architecture would be used in the private sector. In particular, the architecture would mainly be used to implement and realize compliance requirements. Experts considered this use case to be unlikely in the private environment. The experts also stated that the prototype would still require some development before being used in the production environment. In particular, the participants considered implementation on the Hyperledger Fabric [31] platform to be the next important step.

In summary, qualitative methods were used to show that the presented software solution solves the problem described. The experts evaluated the presented approach as being successful for the described problem. Based on the work of Hevner et al. [50], the artifact can be evaluated as having been successful, and the development can be considered to have been completed.

6. Discussion

It is widely accepted that trust and compliance in cloud applications is a wide and extensively researched area due to its significance. Throughout this research effort, the focus
primarily remains on compliance issues. Specifically, the focus was on the automation and decentralized provability of configuration changes, which could benefit a dispute. While blockchain is a promising technology, it faces numerous challenges, including scalability, privacy leakage, governance, and compliance with regulations [2,56]. Data sharing is a challenge in cloud applications, since users lack a mechanism to regulate data. This article introduces a novel architecture to configure cloud applications. The proposed architecture is based on data encryption and digital signatures to guarantee data confidentiality and integrity. This study presents a blockchain-based protocol that can automatically negotiate a key between multiple parties. The negotiated multiparty key is used to store configurations via a smart contract. The cloud application is notified of changes in the configuration via a monitoring mechanism.

Consequently, the configuration changes are uploaded onto the cloud application and implemented using a cloud-side script. Once the configuration has been successfully implemented, the cloud-side script triggers a snapshot of the cloud instance loaded from external storage onto the cloud instance. On the cloud instance end, the cryptographic hash value for the snapshot is computed and, subsequently, written onto the blockchain. These steps occur automatically, and require the user to provide an encrypted configuration with the negotiated key.

The main objective of this study was to present a decentralized reference architecture for the automated, transparent, and notarized configuration of cloud applications. The adopted approach can be used to configure cloud applications in a non-reputable way. Once the cloud management script has detected a successful implementation, it triggers a backup of the cloud instance, and the computed hash value forms a unique fingerprint of the cloud instance. If a dispute arises between two parties, the stored and created snapshot can be consulted, and its hash value can be compared with the hash value stored in the blockchain. A legal dispute can be conducted based on this snapshot. Neither of the parties can deny that the snapshot is not the actual configuration of the cloud application.

However, the qualitative evaluation of the proposed architecture also has certain limitations. Creating a snapshot can be a bottleneck for the application. The prototype presented here needs almost four minutes to set the configuration through a smart contract to confirm the configuration by saving the snapshot hash value. The main reason for this was the hash value generation of a 30 GiB snapshot. Therefore, operating the cloud instance with the smallest possible hard-disk space is recommended. In addition, experts primarily see the architecture in the environment of SMEs and their cloud applications. The implementation of large cloud applications can be very complex. However, this was not generally ruled out by the experts. In the final analysis, experts considered the approach to be conceivable and feasible in the software-as-a-service area, as they saw the greatest need for a compliance solution there.

In addition to technical benefits, such as the tamperproof properties of blockchain entries, the application for configuring applications also offers economic advantages. Cloud configurations are set using smart contracts. By linking transaction costs to the smart contract function, it is possible to transparently indicate upcoming costs for configuration changes and prevent unexpected costs based on compliance adjustments. Furthermore, the payment function of smart contracts can also be used to compensate for other services, such as the cost of storing snapshots, or to pay for additional cloud resources. Based on the proposed reference architecture, the possibilities for adjusting cloud applications are broad. Further research in this context can reveal which acceptance and business models this might enable.

7. Conclusions

This study illustrates that compliance, transparency, and unexpected costs are significant elements for companies considering migration to cloud services. Based on previous research and focus group discussion, this study presents a reference architecture to remove transparency issues and enable legally binding proof of a cloud configuration. Through
the DSR approach, this study provided a blockchain-based architecture for configuring cloud applications. The results of the study highlight that transparency—and the resulting uncertainty—are significant factors for trust issues within cloud adoption. Using the proposed reference architecture, it is possible to configure cloud applications:

- Automatically: No user is required to implement the configuration. Furthermore, configurations are executed automatically, without delays or human error.
- Transparently: Every participant can see what the last successfully implemented configuration was at any time. Moreover, the previously stored configuration can also be seen due to blockchain technology. Configuration changes are not solely stored transparently. The costs of a change are also published transparently using a smart contract. Thus, configuration changes can be automated, and their costs are predictable.
- Notarized: If a cloud application succeeds, a snapshot is generated. The hash value of this snapshot is stored in a smart contract. If a dispute or uncertainty related to the configuration occurs, the hash value of the last successfully created snapshot can be retrieved from the blockchain. Simultaneously, the snapshot associated with the stored hash value can be retrieved from the data storage of the cloud provider. The integrity of the selected snapshot can be verified using the retrieved hash value. However, due to the properties of cryptographic hash values, a snapshot matching a hash value cannot be changed afterward. Consequently, the cloud configuration is documented in a tamperproof manner.

The evaluation presented in this article shows that the generic architecture can be implemented in practice. The configuration of an IDS is one possibility for the proposed generic approach. Nevertheless, the evaluation demonstrates that the general approach could be more user-friendly. Moreover, future research may succeed in performing cloud configuration through a dApp. Further research is required to investigate the rapid provision of the hash value of a snapshot. Overall, a reference architecture was presented with which cloud configurations can be implemented transparently and verifiably, directly, and without a middleman. It was demonstrated at the same time that the proposed reference architecture could be applied to a wide variety of cloud application scenarios. The possibility of importing attested configurations into cloud applications, the predictability of the configuration costs, and, thus, the presented reference architecture can increase the acceptance of cloud solutions.

**Author Contributions:** Conceptualization, T.W. and R.B.; methodology, T.W. and R.B.; software, T.W.; validation, T.W. and R.B.; formal analysis, T.W.; investigation, T.W.; resources, T.W.; data curation, T.W.; writing—original draft preparation, T.W.; writing—review and editing, R.B.; visualization, T.W.; supervision, R.B.; project administration, T.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are available upon request from the corresponding author. The data are not publicly available due to privacy restrictions.

**Conflicts of Interest:** The authors declare no conflict of interest.

**References and Notes**


32. Wang, H.; Zhang, J. Blockchain Based Data Integrity Verification for Large-Scale IoT Data. IEEE Access 2019, 7, 164996–165006. [CrossRef]


