Decentralized Runtime Monitoring Approach Relying on the Ethereum Blockchain Infrastructure

Ahmed Taha*, Ahmed Zakaria*
STRD
Mannheim, Germany
ataha@strd-eg.de
azakaria@strd-eg.de

Dongseong Kim
Faculty of Engineering
University of Queensland
Brisbane, Australia
dan.kim@uq.edu.au

Neeraj Suri
Department of CS
Lancaster University
Lancaster, England
neeraj.suri@lancaster.ac.uk

***Abstract*—Cloud computing offers a model where resources (storage, applications, etc.) are abstracted and provided "as-a-service" in a remotely accessible manner. Although there are numerous claimed benefits of the Cloud to ensure confidentiality, integrity, and availability of the stored data, the number of security breaches is still on the rise. The lack of security assurance and transparency prevented customers/enterprises from trusting the Cloud Service Providers (CSPs). Unless the customer's security requirements are identified and documented by the CSPs, customers can not be assured that the CSPs will satisfy their requirements. Furthermore, the customer's compensation upon a violation is a manual time intensive process.**

In this paper we address the aforementioned challenges by proposing a decentralized customer-based monitoring approach running over Ethereum blockchain. The proposed approach allows the customer(s) to validate the compliance of CSP(s) to the contracted services in the Service Level Agreements (SLAs) and "autonomsly" compensate customers in case of security breaches. At the same time, the proposed approach prevents customers from misreporting for financial gain. The approach builds upon the Ethereum blockchain infrastructure in order to securely store monitoring logs and incorporate SLAs as smart contracts. The compliance validation framework is implemented and its functionality is evaluated on Amazon EC2 and Ethereum Blockchain.

***Index Terms*—Cloud, Security, SLA, Ethereum Blockchain**

I. Introduction

Cloud computing drives the vast spectrum of both current and emerging applications, products, and services, and is also a key technology enabler for the future Internet. In such a service-based environment, service provisioning relies on a Service Level Agreement (SLA) which represents a formal contract established between the Cloud customer and the Cloud Service Provider (CSP). The SLA specifies how provisioning takes place as well as the respective rights and duties of the customer as well as the CSP. Furthermore, the SLA includes the list of Service Level Objectives (SLOs) which are the measurable elements of an SLA that specify the Cloud services' levels requested by the customers, and required to be achieved by the CSP.

Although Cloud computing direct economic value is unambiguously substantial, taking full advantage of Cloud computing requires considerable acceptance of off-the-shelf services. Specifically, both security assurance and transparency remain as two of the main requirements to enable customer's trust in CSPs. The lack of assurance and transparency, along with the current paucity of techniques to quantify security, often results in Cloud customers being unable to assess the security of the CSP(s) they are paying for. In this context, a number of Cloud community stakeholders (e.g., ISO 2700x [1], the European Union Agency for Network and Information Security (ENISA) [2], and Cloud Security Alliance (CSA) [3]) are pushing towards the inclusion of security parameters and CSP's security implementation in Service Level Agreements (named security SLAs or secSLAs [4]). Examples of security-related information are ciphers used to encrypt data, vulnerability management/assessment procedures, minimum/average incident response times, security controls', and configuration elements such as metrics for measuring cybersecurity performance, etc.

A. Problem Statement

Despite the benefits of disclosing the security related information in secSLAs, Cloud customers still face major problems that need to be addressed. In theory, by signing the secSLA, a CSP commits to providing the specified security level throughout the full life-cycle of the service. However, in practice, the service may not comply with the security level contracted in the secSLA during its full operation time. Accordingly, the compensations to be paid to the customer upon any detected violation are specified in the secSLA. Thus, monitoring the service levels described in secSLAs throughout the Cloud full life-cycle is a critical task. The responsibility of this task is usually taken over by either Cloud providers or third-parties.

- **Cloud provider side monitoring:** The CSPs develop their own monitoring tools and market them as part of the Cloud service package (e.g., Amazon CloudWatch [5])
- **Third-party system:** A trusted organization provides a monitoring tool and takes over the responsibility of monitoring the system's quality levels [6]. For example, Nagios [7], and Zabbix [8]

Thus, most of the existing monitoring and security compliance approaches have been proposed to help the CSP to manage security compliance of their services, whereas none

*Ahmed Taha and Ahmed Zakaria are currently working at SAP SE

of the approaches addresses compliance management from the customer side. Hence, how can customers:

P1. Verify if the contracted security level is actually delivered.

P2. Detect and prove any violations to the contracted service levels in the secSLA at any time during the service life-cycle.

P3. Get compensated "autonomously" in case of security breaches and at the same time, how to prevent the customers from misreporting for financial gain.

In [9], we proposed an approach to continuously monitor and validate the compliance of the provided Cloud service to the contracted security level in a secSLA. However, on the one hand, how can customers prove a detected violation and on the other hand, how to prevent customers from misreporting violations for financial gain were not presented. Furthermore, one of the problems facing the Cloud customers is the "manual time intensive" compensation process. The current compensation process involves: (a) the customer opening a case with the support team, (b) waiting for someone to analyze the validity of the case, and finally (c) a payment or a refund is initiated in terms of credit for future usage.

B. Contributions

To solve the aforementioned challenges, we propose an approach to continuously monitor and validate the compliance of the provided Cloud service to the contracted security level in a secSLA and autonomously compensates the customer if a secSLA violation is detected. The approach runs over Ethereum blockchain to automate the validation and compensation process, eliminating the need for human intervention during the measurement, monitoring, and validation processes. The aggregated monitoring logs are stored on the blockchain in a trustworthy manner. This allows the CSPs to validate the claimed violation in a transparent and trustworthy way. To summarize, we make the following contributions:

1) Defining a measurement procedures to determine the measurable SLO values. The measurements are used by the proposed framework to validate the SLOs compliance to the contracted values in the secSLA. Furthermore, we establish a monitoring scheme for validating the compliance of the service to the secSLA.
2) Enforcing the secSLA using smart contract deployed over the Ethereum blockchain. The smart contract receives and aggregates the monitoring logs corresponding to the measured SLOs. The aggregation scheme is fully decentralized.
3) Implementing and evaluating the functionality and performance of the monitoring scheme on Amazon Elastic Compute Cloud (Amazon EC2) instances [10] and Ethereum blockchain [11].

To the best of our knowledge, our approach is the first attempt to provide customers with a decentralized customer side monitoring scheme with the enforcement of the secSLA using a smart contract deployed over the Ethereum blockchain. In addition to, implementing an autonomous compensation process for customers if a violation is detected.

C. Outline

The rest of the paper is organized as follows. Section II develops the background and the basic terminologies related to Cloud secSLAs and blockchain. The architecture of the proposed framework is elaborated in Section III. Section IV presents an evaluation of the proposed approach using Amazon EC2 and Ethereum blockchain. Section V describes the related work.

II. Background

In this section we provide background information, which is necessary for better comprehension of the other sections. Practically, we define the concepts of security Service Level Agreement (secSLA) which is used throughout the paper. Furthermore, we give a brief explanation of blockchain technology.

A. Security Service Level Agreements

A Cloud secSLA describes the provided security services, and represents the binding commitment between a CSP and a customer. Basically, this outlines the desired security services, each of which contains a list of SLOs. Each SLO is composed of one or more metric values that help in the measurement of the Cloud SLOs. Based on the analysis of the state of practice presented in [12], Cloud secSLAs are graphically modeled using a hierarchical structure, as shown in Figure 1. The root of the structure defines the main container for the secSLA. The intermediate levels (second and third levels in Figure 1) are the services which form the main link to the security framework used by the CSP. The lowest level (SLO level) represents the actual SLOs committed by the CSP which are consequently offered to the Cloud customer. These SLOs are the threshold values which are specified in terms of security metrics.

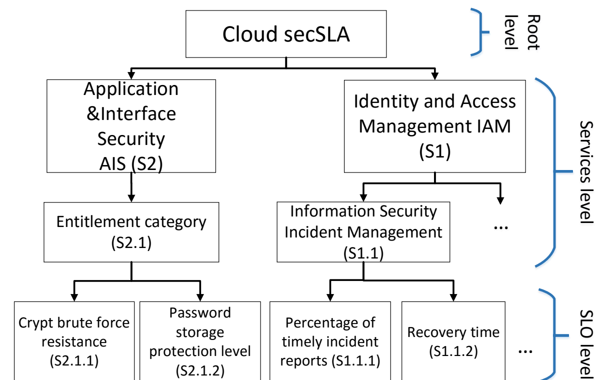

Fig. 1. Cloud secSLA hierarchy based on security posture provided by the STAR repository and compliant with the relevant ISO/IEC 19086 standard.

B. Consensus Algorithms

Blockchain is a distributed public ledger storing all cryptocurrency transactions whether it's Bitcoin transactions or any other digital currency. The transactions are stored in blocks and these blocks are cryptographically chained together forming the blockchain. Unlike traditional systems which require users to trust third parties for its operation, the blockchain enables a trustless environment, where no trusted third-party exists.

In order for the blockchain technology to enable a trustless environment, a consensus mechanism is needed where all participating nodes in the blockchain network adhere to. The most prominent consensus mechanism widely adopted is the proof-of-work algorithm PoW which is introduced by Bitcoin [13]. The PoW is a puzzle competition which allows the first node to find a random number, called nonce, the right to propose the next block in the blockchain. Using this nonce, the hash of the entire block becomes lower than the current difficulty target. The blockchain difficulty target is used to adjust the average time spent by miner nodes to provide the proof of work solution.

C. Ethereum Blockchain

Unlike Bitcoin which is utilized only as digital decentralized currency, Ethereum is a distributed computing platform using the computing platform turing-complete programming language [11]. Ethereum enables complex computations to be executed in the blockchain. Ethereum has its own cryptocurrency called ether "ETH". One ether can be divided into smaller units of currency, where one ether corresponds 10^3 finney, 10^6 szabo, and 10^{18} wei. Ethereum allows developers to develop smart contracts and decentralized applications "Dapps". The following sections present the most important aspects of the Ethereum blockchain.

1) Ethereum Accounts: There are two main types of accounts in Ethereum:

- Externally Owned Account (EOA): This account is held by external actors, in other words, it's controlled by private keys of the account's creator and hence, transactions must be signed with the corresponding private key before being stored in the blockchain.
- Contract Account: Is an account controlled by the source code of the smart contract which defines its behaviour. When this account receives a transaction, it starts to execute the corresponding snippet of code and changes its state after a successful execution. This type of accounts can also interact with another contract account or an EOA.

2) Ethereum Virtual Machine: Ethereum Virtual Machine "EVM" is the runtime environment for Ethereum smart contracts. The EVM executes the smart contracts in an isolated fashion, in other words, the execution has no access to network, file-system or other processes [14]. This makes the execution results deterministic and any malicious node trying to alter the execution results can be easily detected. Computations in the EVM are done via a stack-based bytecode language similar to traditional assembly languages, where the programs are composed of operational codes (opcodes). Each opocode has a predefined execution cost in a unit called "Gas", where gas is the internal pricing for running a transaction or contract in Ethereum.

3) Smart Contracts: Smart contracts are basically the programs that reside in the Ethereum blockchain network and are executed by miner nodes. Any smart contract has set of functions that define its behaviour and data that defines its state [15]. For every smart contract created and deployed over the Ethereum blockchain, there exists a contract account. Each contract account has an address which allows users to interact with the contract by invoking its functions.

Fortunately, smart contracts are not directly written in the EVM language but are written in high level language such as Solidity. The code is then compiled into bytecode to be deployed over the blockchain and gets executed by the miner nodes EVM.

4) How to Interact with Smart Contracts: The first step towards interacting with a smart contract is running an Ethereum node connected to the public Ethereum blockchain. Geth is the official Ethereum node implementation that can be installed in order to connect to the public Ethereum blockchain [16]. There are two main components that are crucial to interact with the smart contracts deployed over the blockchain, namely the Web3 API and oracles.

- Web3 API: The Web3 API is the most widely used API by decentralized applications, it offers the Ethereum nodes to interact smoothly with the blockchain.
- Oracles: By design the smart contracts are passive when it comes to the interaction with the outside world (outside the blockchain). This means smart contracts are not allowed to fetch data from external sources and need trustworthy data feeders to feed them with the required data. An oracle is basically a data feeder that collects real-world data and send it to the smart contract. Hence, the oracle plays the role of an intermediary between the outside world and the blockchain. Oracles can utilize the web3 API in order to be able to interact with the smart contracts deployed on the blockchain [17].

III. SecSLA Compliance Monitoring Framework Architecture

After the customer finds the best matching CSP which satisfies his/her requirements, the CSP commits to deliver the agreed-on SLOs in the secSLA. In order for the customer to validate the agreed-on SLOs compliance, a monitoring approach is proposed as shown in Figure 2. The proposed approach is composed of the following three progressive stages:

Stage (1) The agreed-on SLOs' values are extracted from the secSLA and the measurement technique for each SLO is defined. The defined SLOs' measurement techniques are formulated as "tests" to be executed during the monitoring scheme. These tests are used to determine the real-time values of the SLOs. Both

the contracted SLOs and the tests are developed in the smart contract. The smart contract is deployed and executed in the blockchain.

Stage (2) The CSP's SLOs are monitored over a certain period of time using customer data oracle as shown in Figure 2.

Stage (3) The monitoring logs measured in Stage (2) along with the measurement techniques defined in Stage (1) are used to validate the contracted SLOs; whereas the contracted values from the secSLA are used as the validation reference.

Before detailing each of the three monitoring stages, we explain the main components of our model as depicted in Figure 2:

1) Ethereum private blockchain is the underlying infrastructure of our system. It serves as a decentralized trustless computing platform, where our smart contract is executed in a decentralized manner. The execution results are verified and stored by every blockchain node.
2) The smart contract is the key component of our proposed approach. The smart contract includes the contacted secSLA SLOs and the measurement techniques of each SLO. Furthermore, the smart contract is responsible of:
   - Interacting with different oracles (e.g., receiving the monitoring logs from data oracles).
   - Aggregating the monitoring logs using the defined measurement techniques (where each SLO value is validated against its contracted SLO value).
   - Autonomous compensation in case of secSLA violation.
   - Offering both time-based and on demand secSLA cancellation.
3) Oracles which represent the implemented off-chain software, which depends heavily on the Ethereum web3 API and certainly can only interact with the blockchain through a valid Ethereum EOA account. In the proposed approach, there are three types of oracles (a) customer data oracles, (b) log-retrieval data oracle, and (c) cancellation oracles. The customer data oracle (shown in Figure 2) is responsible for:
   - Executing the monitoring tests according to a predefined monitoring frequency to report the values of the SLOs.
   - Preprocessing and batching the monitored logs to be understandable by the deployed smart contract.
   - Sending the preprocessed batched logs along with the secSLA compliance session parameters to the smart contract (e.g., number of batches, oracleID).

   The log-retrieval data oracle (shown in Figure 2) is responsible of retrieving back the batched logs based on transaction ID from the blockchain.

   The cancellation oracles are used in the default smart contract cancellation mechanism. Note that, oracles can only interact with the blockchain through a valid Ethereum EOA account.

We do mention that, both the CSP and the Cloud customer are running nodes in the same Ethereum blockchain and can interact with the deployed smart contract via their corresponding oracles as depicted in Figure 2. No other entity has access to the smart contract's functions as our developed smart contract only accepts transactions originating from either the CSP address or the customer's address. Both the CSP and the customer must agree on all the used oracles and their responsibilities beforehand. Each monitoring stage is detailed in the following sections.

A. Stage (1): Measurement Definitions

In this stage, the process of defining SLO measurement techniques is specified. The description as well as the metric of each SLO are studied to define an appropriate measurement of the SLO's value. The measurement definition describes the process used to determine the value of each SLO from the customer side. In [9], 142 SLOs were examined (provided by NIST [18], Center of Internet and Security (CIS) [19], EC CUMULUS [20], EC A4Cloud [21] and EC SPECS [22]). Out of all the examined SLOs, 9.4% can be measured from the customer side [9]. The measurement can yield exact values which can be directly compared to the values provided by the CSP. The small number of the SLOs which can be measured is due to the limited visibility and control of the customer on the Cloud model. The majority of the proposed SLOs requires access to the Cloud platform. We use three different SLOs in our evaluation and validation experiments to illustrate our proposed approach, namely percentage of uptime, percentage of processed requests, and secure cookies forced. We explain each of these SLOs and their measurement techniques as follows:

1) Percentage of Uptime: It is the percentage of time slots in which the service is considered available. Accordingly, the measurement period is divided into timeslots of fixed length slotSize. A slot is considered available if the percentage of failed requests within a timeslot is less than a defined percentage.

Measurement. To monitor the availability of the service throughout the full life cycle, requests are sent periodically to the service with a predefined frequency and the response of the CSP is verified. The status of the requests are used to calculate the percentage of uptime using Equation 1.

$$\frac{\sum AvailableSlots}{\sum slots} \tag{1}$$

2) Percentage of Processed Requests: A request is considered successful, if the service was delivered without an error and within a predefined time frame.

Measurement. To measure this SLO, service requests are generated at a predefined frequency and the percentage of successful requests is calculated over the measurement period.

$$\frac{\sum SuccessfulRequests}{\sum Requests} \tag{2}$$

3) Secure Cookies Forced: Secure cookies forced SLO [22] reports whether the service enforces the usage of secure cookies or not. This SLO is used to serve sensitive data

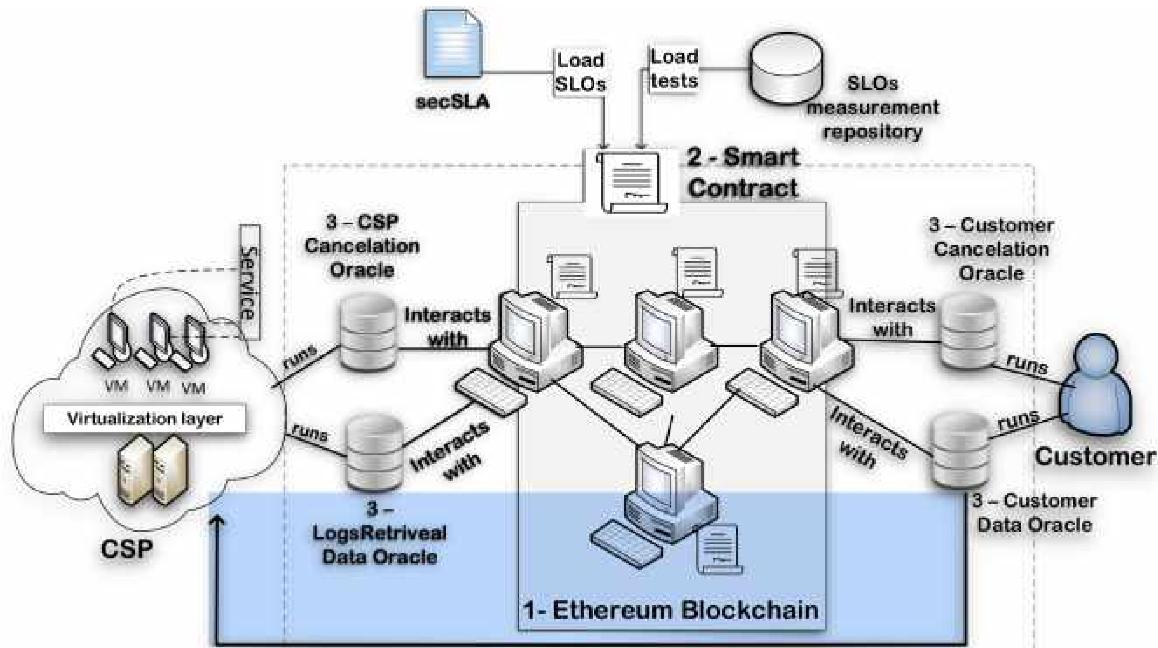

Fig. 2. Monitoring system architecture

protection, for protecting data in transmission. The secure attribute is set for a cookie to restrict sending it over secure channels only (HTTPS connections) [23].

Measurement. By sending HTTP GET request to the service and examining the Set-Cookie header in the responses, we can check if the secure attribute is set to true in order to validate the usage of secure cookies.

As previously mentioned, the measurement for each SLO is formulated as tests and developed in the smart contract.

B. Stage (2): Monitoring Approach

In this stage we introduce the Cloud monitoring approach that enables customers to validate the compliance of a running Cloud service to its secSLA as depicted in Figure 3. The monitoring and validation processes comprise the following phases:

1) Phase 0:

- Private Ethereum blockchain is created and both the CSP and the customer exchange their Ethereum addresses.
- The CSP deploys the smart contract with an ether deposit value equals to the customer subscription's value and includes the customer's address.
- The smart contract's constructor function is autonomously called on successful deployment to set its balance to the received ether and to mark the start time of the secSLA.
- After a successful deployment, the CSP shares the smart contract's address with the corresponding Cloud customer.
- The smart contract contains the contracted SLO values as well as each SLO measurement technique defined in Section III-A.

2) Phase 1:

- The customer initiates a validation session by sending a validation request to his/her data oracle. The request includes some information about the CSP (e.g., IP address or hostname) and the SLO(s) required to be validated.
- The customer's data oracle is responsible of: (a) executing the tests according to a predefined monitoring frequency to report the values of the measured SLO(s), and (b) batching the monitoring logs and then preprocessing the batched logs to match data formats that can be manipulated by the smart contract.
- The customer's data oracle interacts with the corresponding function of the smart contract by sending the total number of batches and its oracle ID.
- The smart contract grants the customer's data oracle access to the corresponding receiving logs function only if there is no other oracle already interacting with the same function. This feature acts like a mutex which synchronizes the access of the distributed customer data oracles to the different smart contract's receiving logs functions.
- The smart contract receives the batches from the customer's oracle and once all the agreed on batches (e.g., 30 batches for one month monitoring logs) have been received, the contract starts the validation phase (Phase 2).

3) Phase 2:

- The smart contract starts "aggregating" the received batches that constitute the whole monitoring logs session (e.g., one month). Smart contract aggregates the received monitoring logs according to the measurement technique of each SLO in order to obtain the SLO measurement value. This value is validated against the corresponding contracted SLO value developed in the smart contract in the initial phase.
- The smart contract performs an SLO compliance by validat-

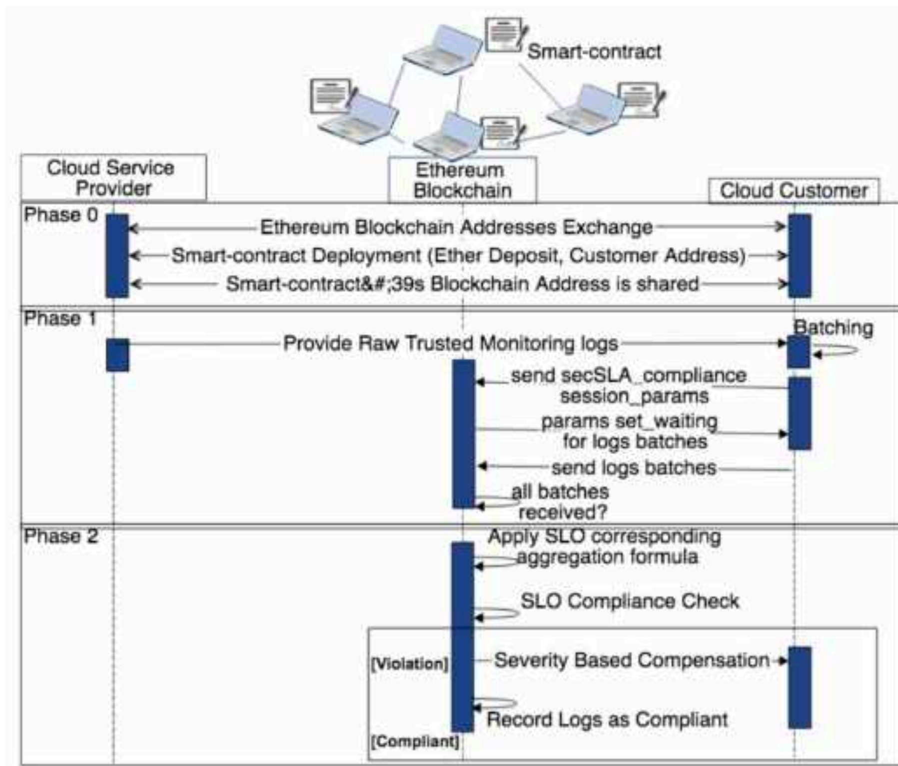

Fig. 3. System workflow phases - sequence diagram

ing the aggregated values against the SLO values from the secSLA.

- In case of a detected violation, the smart contract compensates the customer based on the severity of the violation. The equivalent compensation value in ether is automatically sent to the customer address.
- If the aggregated values are in adherence to the agreed on SLOs, the smart contract records the incidence as compliant logs.
- Phase 1 and Phase 2 are repeated until, the contract's end date arrives or the smart contract receives an early contract cancellation from both parties (the CSP and the customer).

The proposed monitoring framework enables the customer to adjust the monitoring configuration (e.g., monitoring frequency, monitoring duration, validation period) according to his/her required monitoring coverage. All the processes performed by the framework are automated.

C. Stage (3): Monitoring System Processes

The smart contract executes the aggregation and compensation process such that:

1) The smart contract starts a secSLA enforcement session only when the authorized customer[1] sends to the smart contract the session parameters.
2) To verify the logs freshness, the smart contract examines the session ID, which is a unique value which represents the secSLA enforcement session and relates to the monitoring logs. If the session ID has not been recogonized (not stored in the sessionIDs contract's storage), this emphasizes that the logs are fresh and eligible for a secSLA enforcement session.
3) The smart contract keeps on receiving the logs' batches and once the number of batches received are equal to the contracted number of batches (e.g., thirty batches for one month logs), it starts aggregating the received batches using the corresponding measurement formula and stores the aggregated result.
4) The aggregated result is validated by the smart contract against its corresponding SLO. If the aggregated result is not compliant to its contracted SLO, the customer is compensated via Ethereum blockchain transaction originating from the smart contract (deducted from the CSP's deposit paid on deployment).

Cancellation Process: Cancellation function is used in case both the CSP and the customer decided to cancel the secSLA smart contract before its intended termination date. Cancellation function entails two main steps: (a) receives from the CSP/customer their request to cancel the contract, and (b) sends the deposit back to the CSP (or the remainder) and then deactivate permanently the smart contract. The cancellation function only enforces the cancellation process if it receives from both the CSP and the customer a cancellation request.

Furthermore, we utilize a time-based cancellation mechanism to adhere to the traditional agreements that define

[1]The customer's Ethereum address matches the Cloud customer's address stored in the smart contract

TABLE I
INSTANCES CONFIGURATIONS

	US_EC2	**FRA_EC2**	**TKY_EC2**
Region	US East(N. Virginia)	EU(Frankfurt)	Asia Pacific(Tokyo)
Instance type	t2.nano		
Amazon M/C Image	Amazon Linux AMI 2016.09.0 (HVM) [10]		
Network	Default		
Availability zone	us-east-1d	eu-central-1b	ap-northeast-1a
Tenancy	Shared		

start/end date. We utilize the block timestamp, in which the smart contract was mined at, in order to mark the start date of the contract. Beside that, we rely on the date data-type in the solidity programming language to set beforehand the termination date of the smart contract.

D. Security Vulnerabilities of Ethereum Smart Contracts

Software vulnerabilities are very destructive in the context of smart contracts, as they can result in losing control of customers' digital wallets containing ether. This section discusses a duplicate compensation scenario. Duplicate compensation is the process of compensating a customer more than once based on the same monitoring session. Thus, the deployed smart contract has to keep track of the successfully aggregated monitoring sessions to prevent duplicate compensations. To avoid this, the proposed system includes a unique session ID attribute which is basically used to uniquely identify the different monitoring sessions by the customer's data oracle. The session ID is sent to the smart contract along with the monitoring logs. The session ID is used by the smart contract for the aggregation, compliance tests and compensation processes. There are three session types in the proposed approach which are related to the three used SLOs. Each session has its own session ID variable declared in the smart contract. The smart contract marks the session ID as a successful aggregated session only when the whole batches are received and the aggregation process along with the compliance and the compensation processes are successfully executed.

IV. IMPLEMENTATION AND EVALUATION

The proposed approach is evaluated by conducting two experiments. In the first experiment, we evaluate the functionality of the approach on Amazon EC2. The second experiment measures and compares the overall gas consumed by the oracle (with different batch sizes).

In these experiments, a machine with the following specifications is used: 4 GB RAM, Intel Core i5-3337U CPU 1.80 GHz x4, Ubuntu 16.04 LTS, OS-type 64 bit, disk 60 GB.

A. Setting-up Ethereum Blockchain

Due to the need of high storage requirements to run a Geth Ethereum node and connect to a public test-network[2] as well the intensive computational power needed for mining, we execute our evaluation and analysis using the TestRPC Ethereum blockchain network simulation. TestRPC is a node.js implementation of the Ethereum protocol which offers web3 API for the interaction with the blockchain. However, it only simulates the mining process which requires high computational resources. TestRPC provides ten external Ethereum accounts each one with a hundred ethers.

Firstly, we implement three different aggregation mechanisms according to the SLOs specified in Section III-A (i.e., uptime-availability, percentage of processed requests, and secure cookies forced). Further, we develop a synchronization mechanism for the interleaving oracles. The on-demand and the time-based cancellation is also assured through the smart contract. Finally, we develop a compensation mechanism. The developed smart contract functions as well as the smart contract deployment process are presented in the following subsections.

B. Smart Contract Functions

The deployed smart contract is composed of the following functions:

1) SecSLA enforcement smart contract constructor. The smart contract constructor is called automatically when the contract is successfully deployed.
2) SLO related functions, which are implemented to receive the SLOs' logs from the customer data oracle. These functions are also used to synchronize the access to these functions in case of interacting with multiple oracles.

C. Customer Data Oracle

The Cloud customer data oracle is implemented in python programming language. To monitor the service provider, firstly the information about the service to be monitored must be provided, including the IP address of the service and the hostname of the web based management interface. Secondly, the customer specifies (a) the duration of the monitoring session[3], and (b) the frequency at which the measurements of the SLO are conducted during the monitoring session. The higher the chosen frequency the more fine grained is the measurement.

Using these specifications, as well as the SLOs' values, the monitoring session is initiated. Using the frequency specified by the customer, the monitoring framework measures the value of the SLOs' by continuously monitoring the provided service.

[2]Examples of public Ethereum test-networks are ropsten [24] (so far more than 2.7 million blocks with almost 16 million transactions) and Rinkeby [25] (so far almost 2 million blocks with more than 4 million transactions)

[3]The duration of the monitoring session is the duration over which the service is validated

Then, the monitoring logs (of the three SLOs) are batched according to both the slot and batch sizes. These batches' feed the smart contract using a smart contract handler class. The constructor function of this class is responsible for: (a) creating a web3 API object that connects to the customer's Ethereum node, (b) instantiate the deployed contract via its application binary interface (ABI JSON file) and the contract's Ethereum address, and (c) allows the customer's address to run the oracle. Using the monitoring logs and the measurement mechanisms specified in Section III-A, the smart contract aggregates and validates the values of the SLOs with the built in contracted SLOs. Finally, a batch retrieval script is developed (to be used by either the CSP or the Cloud customer) to extract the batches stored on the blockchain by the smart contract's aggregation functions. It can be used, for instance, by the CSP to make sure that the batches sent by the customer to the blockchain are exactly the same as the logs the CSP provided.

Experiment 1: Evaluating the Functionality of the Approach on Amazon EC2

Three EC2 instances each in a different region, along with their web based management interfaces (Amazon management console) are monitored. The instances configuration is shown in Table I.

The SLA defined by Amazon for EC2 only includes service availability. For Amazon EC2, unavailable means that all of your running instances have no external connectivity. The value committed in the SLA for this SLO is a lower bound of 99.95%. The oracle frequently sends Internet Control Message Protocol (ICMP) echo requests to check if the service is available. With the aim to detect the shortest possible change and thereby achieve maximal coverage of changes in the monitored SLO values. For availability, the shortest possible change was assumed to be the reboot time of an instance during which the instance is unavailable. The reboot time of an instance has been experimentally assessed to be 5 seconds. Hence, the monitoring frequency is set to one test every 5 seconds. For the other security properties of the management subsystem, a lower monitoring frequency is used, since changes in the configuration of the web server hosting the console are expected to occur less frequently. Moreover, automated access to the console at high rates might be considered malicious and thus, the requests might get blocked by the CSP. Accordingly, the monitoring frequency for the consoles was configured as 6 times per hour.

TABLE II
AMAZON EC2 INSTANCES RESULTS

	US_EC2	FRA_EC2	TKY_EC2
% of Uptime	99.0915%	100%	99.9303%
% of Processed Requests	99.8088%	99.9930%	99.7571%

The experiment was run for a month with slot size of one hour for both the *uptime-availability* and the *percentage of processed requests* service level indicators, and a slot size of 10 minutes for the secure cookies forced. Table II shows the measured values of the SLOs for all three instances. According to the results, the instance deployed in the US East (New Virginia) offered the least percentage of uptime with a percentage still greater than 99%. The lowest percentage of processed requests is reported for the instance located in Asia Pacific (Tokyo).

We analyzed the collected data to investigate the frequencies and duration of the outages during the measurement period. The shortest observed outage is the failure of a single request, i.e., an outage duration of less than 10 seconds. The longest outage duration extracted from the results of all instances is 255 seconds experienced in the US_EC2 instance. Finally, all the consoles enabled secure cookies.

TABLE III
TOTAL GAS CONSUMED BY THE VALIDATION AND COMPENSATION PROCESSES BASED ON ONE MONTH LOGS WITH DIFFERENT BATCH SIZES

Batch size	Total Gas Consumed	Equivalent in Ether	USD Equivalent
1 day	58848167	0.0588	11.64$
3 days	57360993	0.0573	11.34$
5 days	58283227	0.05828	approx. 11.5$

Experiment 2: Consumed Gas (Cost) Evaluation

In this experiment, we compare the overall gas consumed by our approach according to different batch sizes. As already mentioned, the SLOs' monitoring logs are batched[4] and used as an input to the smart contract. We evaluate our approach based on sending (a) daily monitoring logs (one day batch), (b) three days batch, and (c) five days batch, to the smart contract.

A python script is developed to calculate the total consumed gas by the secSLA compliance validation and compensation for one month raw logs as depicted in Table III. The total gas consumed by the approach is depicted in Table III. These results are calculated based on the current estimated gas price which is (10^{-9}) Ether at the Ether current market value[5].

Furthermore, we calculate the amount of gas consumed by every transaction sent from the customer's oracle to the smart contract. Table IV shows the average cost per transaction for each SLO's batch sizes. The web3 estimate transaction gas function is used to estimate the cost of one transaction holding 5 days of raw secure cookies logs.

It is worth noting that, the cost of the secure cookies enforcement session cost for the five days batch scenario cannot be measured, due to the huge transaction size which exceeds the block limit (6721975 gas). This huge size comes from the fact that the secure cookies logs were collected at a higher frequency (10 minutes slots). The number of slots comprising a one day of secure cookies logs is 6 times larger than the uptime-availability and the processed requests.

[4]The monitoring logs batching is performed according to the slot and batch sizes

[5]https://ethereumprice.org/; Currently 1 Ether corresponds to 132.56$

TABLE IV
THE CONSUMED GAS OF THE VALIDATION AND COMPENSATION PROCESSES PER SLO

Average cost per transaction			
Service Level Indicator	1 day of logs per Tx	3 day of logs per Tx	5 day of logs per Tx
Uptime-Availability	337705	951055	1661498
Percentage of processed requests	348743	989013	1725654
Secure-cookies forced	1275156	3796030	> 6721975

V. RELATED WORK

Multiple approaches have been proposed to validate the compliance of the service to the SLA, by verifying the enforcement of the contracted properties. Haq et al. [26] proposed a framework to manage SLA validation by defining rules to map high-level SLOs to low-level metrics that the CSP can monitor throughout the service life-cycle. Furthermore, Rak et al. [27] and Liu et al. [28] proposed Cloud application monitoring tools that detect SLA violations. Although these approaches provide effective techniques to detect SLA violations, they are only focused on (as well as most other existing monitoring techniques, such as Rana et al. [29] and Zhang et al. [30]) monitoring performance properties rather than security properties.

Few approaches that are concerned with monitoring security properties of Cloud services have been proposed in the literature. Ullah et al. [31] proposed an SLA management solution that installs monitoring agents on the Cloud service to measure service security properties. Ullah et al. [32] and Majumdar et al. [33] proposed tools to be used by the CSP to audit the compliance of their services to some security properties by depending on log analysis. Nevertheless, all of the previously mentioned approaches address CSP side validation by proposing solutions managed/deployed by/at the CSP or require cooperation from the CSP. Hence, customers cannot validate the effectiveness of these approaches or guarantee transparency of the reported results. In our previous study [9], we proposed a customer side monitoring framework validation which enables cloud customers to validate the compliance of the provided cloud service. However, how can customers detect/prove violations and get compensated autonomously at the same time how to prevent customers from misreporting for financial gain were not presented in the paper.

Few approaches have been proposed to utilize the blockchain technology in Cloud computing. Margheri et al. [34] proposed a design and implementation of a governance approach for the Federated Cloud as-a-Service (FaaS) European union project (sunfish) [35]. Their proposed approach utilizes the blockchain technology as an infrastructure to ensure a distributed and democratic governance. Gaetani et al. [36] proposed an approach for tackling the database integrity challenges in the Cloud environment through utilizing the blockchain technology. Shafagh et al. [37] proposed a new paradigm for managing IoT data in the Cloud. Liu et al. [38] proposed a blockchain based approach to replace the data integrity services provided by the Cloud service providers on the IoT data. Ferdous et al. [39] proposed a decentralized approach for runtime access control systems in Cloud federations. Lee et al. [40] proposed a new approach for identity and authentication management as-a-service (IDaaS) offered by Cloud providers to their customers.

VI. CONCLUSION

Although many approaches help CSPs to monitor their compliance to the secSLA to take corrective actions in case of violation, existing approaches do not allow customers to manage the compliance validation process by themselves. Consequently, limiting their ability to assess whether contracted security levels are actually provided. In this paper, a decentralized customer side monitoring approach to monitor and detect secSLA violations and autonomously compensate customers is proposed. The proposed approach monitors the compliance of Cloud services to the contracted properties in secSLAs. The approach relies on Ethereum blockchain as a decentralized platform to securely store monitoring logs and incorporate secSLAs as smart contracts. The deployed smart contract integrate and aggregate measurable SLOs and check continuously the compliance of Cloud services to contracted SLOs. Furthermore, autonomously compensate the customer upon violation. The proposed monitoring approach has been evaluated on commercial IaaS Cloud service. The results have proven our approach suitable for measuring the values of the SLOs and identifying violations of contracted SLO values.

VII. ACKNOWLEDGEMENT

Research supported, in part, by EC H2020 CONCORDIA GA# 830927 and by the Lancaster Security Institute.

REFERENCES

[1] International Organization for Standardization, "Information technology, security techniques, code of practice for information security management," Tech. Rep. ISO/IEC 27002, 2013.
[2] National Institute of Standards and Technology (NIST), "Security and Privacy Controls for Federal Information Systems and Organizations, NIST 800-53v4," 2014.
[3] Cloud Security Alliance, "Cloud Controls Matrix v3," *https://cloudsecurityalliance.org/research/ccm/*.
[4] J. Luna, R. Langenberg, and N. Suri, "Benchmarking Cloud Security Level Agreements Using Quantitative Policy Trees," *Proc. of Cloud Computing Security Workshop*, pp. 103–112, 2012.
[5] Amazon Web Services, Inc., "Amazon CloudWatch," https://aws.amazon.com/cloudwatch/.
[6] C. Molina-Jimenez, S. Shrivastava, J. Crowcroft, and P. Gevros, "On the monitoring of contractual service level agreements," in *Proc. of Electronic Contracting*, 2004, pp. 1–8.
[7] "Nagios Monitoring Tool," https://www.nagios.org/.
[8] A. Dalle and S. Lee, *Zabbix network monitoring essentials*. Packt Publishing Ltd, 2015.
[9] S. Alboghdady, S. Winter, A. Taha, H. Zhang, and N. Suri, "C'mon: Monitoring the compliance of cloud services to contracted properties," in *Proc. of ARES*, 2017.

[10] Amazon Web Services, Inc., "Amazon EC2 - Virtual Server Hosting," https://aws.amazon.com/ec2/.
[11] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, pp. 1–32, 2014.
[12] J. Luna, A. Taha, R. Trapero, and N. Suri, "Quantitative reasoning about cloud security using service level agreements," *In Transaction on Cloud Computing*, no. 99, 2017.
[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[14] Y. Hirai, "Defining the ethereum virtual machine for interactive theorem provers," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 520–535.
[15] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[16] "GO Implementation of Ethereum Node," url: https://github.com/ethereum/go-ethereum/wiki/geth.
[17] "Oracles in Blockchain Contex," url: https://blockchainhub.net/blockchain-oracles/.
[18] National Institute of Standards and Technology, "Performance and Measurements Guide for Information Technology," Tech. Rep. NIST 800-55 Revision 1, 2008. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-55r1.pdf
[19] "The CIS Security Metrics v1.1.0," Tech. Rep., 2010. [Online]. Available: https://benchmarks.cisecurity.org/tools2/metrics/CIS_Security_Metrics_v1.1.0.pdf
[20] A. Pannetrat, G. Hogben, S. Katopodis, G. Spanoudakis, and C. Cazorla, "D2. 1: Security-aware sla specification language and cloud security dependency model," *CUMULUS project deliverable*, 2013.
[21] D. Nunez and C. Fernandez-Gago, "D:C-5.2 Validation of the Accountability Metrics," Accountability For Cloud and Other Future Internet Services (A4Cloud), Tech. Rep., October 2014. [Online]. Available: www.a4cloud.eu/sites/default/files/D35.2\%20Validation\%20of\%20the\%20accountability\%20metrics.pdf
[22] SPECS Project, "Report on requirements for Cloud SLA negotiation - Final," Tech. Rep. Deliverable 2.1.2, oct 2014. [Online]. Available: http://www.specs-project.eu/publications/public-deliverables/d2-1-2/
[23] A. Barth, "RFC 6265 HTTP State Management Mechanism," https://tools.ietf.org/html/rfc6265#section-4.1.2.5, April 2011.
[24] "Ropsten - Ethereum Testnet," url: https://ropsten.etherscan.io/.
[25] "Rinkeby - Ethereum Testnet," url: https://rinkeby.etherscan.io/.
[26] I. Haq, I. Brandic, and E. Schikuta, "Sla validation in layered cloud infrastructures," in *Proc. of GECON*, 2010, pp. 153–164.
[27] M. Rak, S. Venticinque, G. Echevarria, G. Esnal *et al.*, "Cloud application monitoring: The mosaic approach," in *Proc. of CloudCom*, 2011, pp. 758–763.
[28] D. Liu, U. Kanabar, and C.-H. Lung, "A light weight SLA management infrastructure for cloud computing," in *Proc. of CCECE*, 2013, pp. 1–4.
[29] O. Rana, M. Warnier, T. Quillinan, F. Brazier, and D. Cojocarasu, "Managing violations in service level agreements," in *Grid Middleware and Services*. Springer, 2008, pp. 349–358.
[30] Z. Zhang, L. Liao, H. Liu, and G. Li, "Policy-based adaptive service level agreement management for cloud services," in *Proc. of ICSESS*, 2014, pp. 496–499.
[31] K. Ullah and A. Ahmed, "Demo paper: Automatic provisioning, deploy and monitoring of virtual machines based on security service level agreement in the cloud," in *Proc. of CCGrid*, 2014, pp. 536–537.
[32] K. Ullah, A. Ahmed, and J. Ylitalo, "Towards building an automated security compliance tool for the cloud," in *Proc. of TrustCom*, 2013, pp. 1587–1593.
[33] S. Majumdar, T. Madi *et al.*, "Security Compliance Auditing of Identity and Access Management in the Cloud: Application to OpenStack," in *Proc. of CloudCom*, 2015, pp. 58–65.
[34] A. Margheri, M. S. Ferdous, M. Yang, and V. Sassone, "A distributed infrastructure for democratic cloud federations," in *Proc. of Cloud Computing (CLOUD)*, 2017, pp. 688–691.
[35] F. P. Schiavo, V. Sassone, L. Nicoletti, A. Reiter, and B. Suzic, "Faas: Federation-as-a-service: The sunfish cloud federation solution," in *FaaS: Federation-as-a-Service: The SUNFISH Cloud Federation Solution*, 2016.
[36] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Blockchain-based database to ensure data integrity in cloud computing environments," *ITA Security*, 2017.
[37] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of iot data," in *Proc. of Cloud Computing Security Workshop*, 2017, pp. 45–50.
[38] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain based data integrity service framework for iot data," in *Proc. of International Conference on Web Services (ICWS)*, 2017, pp. 468–475.
[39] M. S. Ferdous, A. Margheri, F. Paci, M. Yang, and V. Sassone, "Decentralised runtime monitoring for access control systems in cloud federations," in *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2632–2633.
[40] J.-H. Lee, "Bidaas: Blockchain based id as a service," *IEEE Access*, vol. 6, pp. 2274–2278, 2017.