QRES: Quantitative Reasoning on Encrypted Security SLAs

Ahmed Taha[1], Spyros Boukoros[1], Jesus Luna[2], Stefan Katzenbeisser[1], and Neeraj Suri[1]

[1]Technical University of Darmstadt, Germany
[2]Bosch Germany
{*ataha,suri*}*@deeds.informatik.tu-darmstadt.de,*
{*boukoros,katzenbeisser*}*@seceng.informatik.tu-darmstadt.de*
Jesus.LunaGarcia@de.bosch.com

Abstract

While regulators advocate for higher cloud transparency, many Cloud Service Providers (CSPs) often do not provide detailed information regarding their security implementations in their Service Level Agreements (SLAs). In practice, CSPs are hesitant to release detailed information regarding their security posture for security and proprietary reasons. This lack of transparency hinders the adoption of cloud computing by enterprises and individuals. Unless CSPs share information regarding the technical details of their security proceedings and standards, customers cannot verify which cloud provider matched their needs in terms of security and privacy guarantees. To address this problem, we propose QRES, the first system that enables (a) CSPs to disclose detailed information about their offered security services in an encrypted form to ensure data confidentiality, and (b) customers to assess the CSPs' offered security services and find those satisfying their security requirements. Our system preserves each party's privacy by leveraging a novel evaluation method based on Secure Two Party Computation (2PC) and Searchable Encryption techniques. We implement QRES and highlight its usefulness by applying it to existing standardized SLAs. The real world tests illustrate that the system runs in acceptable time for practical application even when used with a multitude of CSPs. We formally prove the security requirements of the proposed system against a strong realistic adversarial model, using an automated cryptographic protocol verifier.

1 Introduction

Cloud computing allows customers to develop, manage, and access a spectrum of resources (storage, software, applications, etc.) which are typically offered as-a-service in a remotely accessible fashion. In such a service-based environment, the cloud provisioning relies on stipulated Service Level Agreements (SLAs). Such an agreement is basically a contract between the Cloud Service Provider (CSP) and the customer regarding the offered service. These SLAs specify the cloud service levels requested by the customers, and required to be achieved by the CSPs.

A variety of parameters for different aspects of a service can be included in the SLA, such as but not limited to: availability, performance, downtime and location of the data.

Albeit the 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 [1,6]. The lack of security assurance and transparency has prevented customers and enterprises from trusting the CSPs, and hence not using their services. Unless the customers security requirements are identified, documented, and communicated by the CSPs, customers can not be assured that the CSPs will satisfy their requirements.

In this context, a number of cloud community stakeholders (e.g., ISO 27002 [25] and the European Union Agency for Network and Information Security (ENISA) [40]) are pushing towards the inclusion of security parameters and CSP's security implementation in security SLAs (named secSLAs [37]). Basically, secSLA has the same SLA structure however, it discloses detailed security-related information[1] about each CSP security offers. The customers can use this information to assess and compare different service offerings provided by various CSPs and then select the best CSP that satisfies their requirements.

Despite the benefits of these kind of information, still CSPs do not disclose security related information in their secSLAs for security and/or commercial reasons [30]. The dangers of including security related information in the secSLA where pointed out by ENISA [30] as:

- Publicly disclosing security parameters may assist attackers to penetrate the system using a hole in the publicized data. Accordingly, the rate of malicious security breaches increases, which can be much harder to detect. This can also lead to a significant financial loss as a result of the customers compensation.

- Publicly detailing commercial sensitive information (i.e., financial terms, service levels, cost information, vulnerability descriptions which may include proprietary information, etc.) can be used by other competitors to improve their services.

To that end, we tackle the aforementioned problem by designing and implementing a system called *QRES* (Quantitative Reasoning on Encrypted SLAs). Our system simultaneously allows:

1. CSPs to specify their services along with the key measurable parameters in secSLAs, without revealing information about the offered security parameters or commercial sensitive information.

2. Customers to assess and evaluate the CSP's offered security services and choose the best CSP matching their needs.

In our system (Figure 1), CSPs' encrypted secSLAs are certified and digitally signed by an trusted certification authority (i.e., auditor). The trust assumption relies on the fact that the CSPs' certificates are valid and trusted and thus, their encrypted secSLAs are verified and digitally signed by the auditor. After a successful authorization, every provider possesses a digital signature on their encrypted secSLA. The CSPs send their signed, encrypted secSLAs to an intermediate broker afterwards. After the broker

[1]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.

verifies the auditor's signature, it stores each encrypted secSLA in a database.

Furthermore, customers send their requirements to the broker[2] which tries to match the customer's requirements against the stored encrypted data in order to find the best matching CSP's secSLA. During this phase, neither the broker can learn the CSP's encryption key nor the CSP's can learn the customer's requirements. Finally, the broker sends the customer the CSPs ranking according to the customer requirements.

To verify the *QRES* system's correctness[3], we start by formally defining the required security properties; the presented system must ensure the following privacy requirements: data confidentiality and data fairness. Then, we conduct a formal security analysis of *QRES* using ProVerif [11], an automated cryptographic protocol verifier, establishing the defined properties against a strong adversarial model (i.e., Dolev-Yao adversary model [24]). To validate our system model, we implement *QRES* using Amazon AWS DynamoDB [4]. We utilize real-world CSPs' secSLAs found on the public STAR (Security, Trust and Assurance Registry) [21] repository, which are complaint with the relevant ISO/IEC 19086 standard [32].

The ProVerif scripts and proofs used in this paper as well as the system implementation are publicly available at [46].

Contributions. The contributions of our system are summarized below.

1. We propose the first system (*QRES*) which enables CSPs to publicly disclose detailed information about their offered services in encrypted secSLAs and customers to assess the CSPs' offered security services and find those satisfying their requirements.

2. *QRES* is built around a novel two-party privacy preserving query over encrypted data scheme (named *QeSe*).

3. We formally prove the system's correctness by defining the needed security properties and proving that the system holds these properties against a strong adversarial model.

4. We implement, evaluate and benchmark *QRES* using real-world CSPs' secSLAs. We

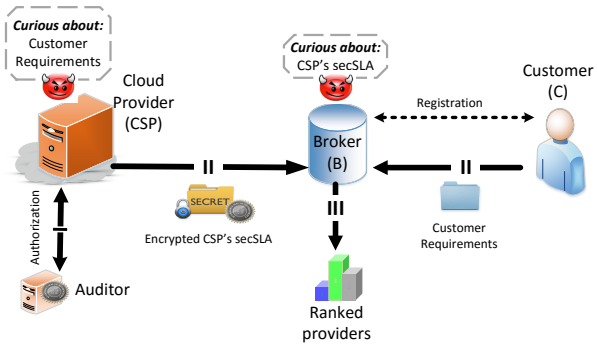

Figure 1: *QRES* System Model. The system involves a customer (C) who wants to find the best CSP according to her/his needs. CSPs are certified by a trusted certification authority. A broker (B) receives the CSPs' encrypted secSLAs and the customer's requirements. Then, it ranks the CSPs according to the customer requirements and returns the result to the customer

[2] In this paper, we assume the case of a novice or basic customer who can not search for her/his over encrypted data

[3] *QRES* must ensure that the entities should learn nothing except their output and each entity should receive its correct output.

show that the system's performance is practical for the presented use case.

Outline

The rest of the paper is organized as follows: The basic concepts of the cryptographic tools, notations, and security definitions used are developed in Section 2. Section 3 defines the system and threat models, as well as the system objectives and requirements. Section 4 elaborates the architecture of the proposed system. A security analysis of the *QRES* system is presented in Section 5. The implementation of the presented system is detailed in Section 6. We present the related work in Section 7 and the conclusion in Section 8.

2 Basic Concepts

This section briefly explains the basic concepts, cryptographic tools, notations and security definitions used in this paper. We adapt and implement well known cryptographic mechanisms, and therefore we do not include proofs as they already exist in the referenced papers.

2.1 Notations and Preliminaries

We summarize most of the terminology used in Table 4 (in the Appendix). Furthermore, we define the searchable encryption notations used in the paper in the Appendix.

2.2 Security Service Level Agreements

A security service level agreement describes the CSP's offered security services, and represents the binding commitment between a CSP and a customer. Basically, each CSP's secSLA consists of a number of offered security services which contain a list of Service Level Objectives (SLOs). The SLOs are the single measurable elements of an SLA/secSLA that specify the cloud service levels required by the customers and to be achieved by the CSP. Each SLO is assessed using one or more key measurable parameters. These parameters help in the measurement of the cloud service objectives by defining measurement rules that facilitate the assessment and decision making.

For example, how a CSP recovers from incidents is typically defined in terms of severity and time to recovery. As specified by ENISA [30], a severity classification scheme detailing levels from 1 to 5 could be defined in an SLA, where a "N/A" level could be included in the SLA for incidents which have no security impact. The criteria of each level is based on various parameters.

Based on the analysis of the state of practice presented in [37], security SLAs are modeled using a hierarchical structure, as shown in Figure 2. The root of the structure defines the main container for the secSLA. The intermediate levels are the services which form the main link to the CSP's offered services. The lowest level (SLO level) represents the actual SLOs committed by the CSP and consequently offered to the cloud customer. To formalize the concept of an SLA/secSLA, we use the definition presented in [51].

Definition 1 *An SLA consists of a set of services S. Each service consists of a finite positive number n of SLOs o_i; where $i = 1 \dots n$. Each SLO consists of l different values v; such that $o_i = v_1, v_2, \dots, v_l$. Each of these values implies a specific service level offered by the CSP and*

required by the customer.

We illustrate the secSLA's structure and functionality better with an example. We consider a customer processing financial transactions using Software-as-a-Service (SaaS). The customer looks for a secSLA which specifies a recovery time objective of less than 1 minute and a monthly report offered by the selected CSP specifying the mean recovery times [30]. Using this example we specify two SLOs ("Percentage of timely incident reports $S_{1.1.1}$" and "Recovery time $S_{1.1.2}$") as shown in Figure 2. The "Percentage of timely incident reports" SLO is composed of $\{yearly, half-yearly, monthly, weekly\}$ values which are defined using service levels as $level_1, level_2, \ldots, level_4$ respectively. If a CSP is committing a "Percentage of timely incident reports" of *monthly*, then $v_{S_{1.1.1}} = level_3$. Similarly, a CSP commits other SLOs so that the overall CSP's secSLA contains a list of SLOs with different values that is committed to fulfil.

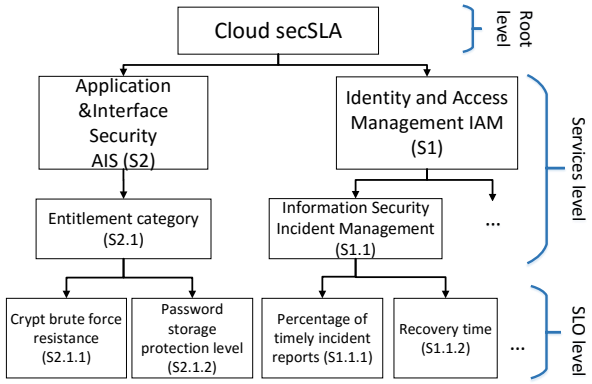

Figure 2: Cloud secSLA hierarchy based on security posture provided by the STAR repository and compliant with the relevant ISO/IEC 19086 standard.

```
< SLA slaid="sla1">
<service id="S1" name ="Identity and Access Management
    " category="IAM" pre="1">
<control id="S1.1" name ="Information Security Incident
    Management" category="IAM-09" pre="2">
<slo id="S1.1.1" name ="Percentage of timely incident
    reports" value="level3" pre="3"></slo>
<slo id="S1.1.2" name ="Recovery time" value="level2"
    pre="4"></slo>
</control></service></ SLA>
```

Listing 1: Excerpt of the SLA depicted in Figure 2

Format of secSLAs. The secSLA shown in Figure 2 can be specified using a machine readable format such as an XML structure as depicted in Listing 1. The XML data values can be captured using the pre-fields shown in Listing 1 (named "pre"), which are sequence numbers that count the open tags in an XML. An example of how the pre-fields are computed is depicted in Table 1. Each pre-field is used as a service identifier as each service/SLO has a unique pre-field in the secSLA.

XML structure	pre-field
$< S_1 >$	1
$< S_{1.1} >$	2
$< S_{1.1.1}$ value=$level_3 >$	3
$< S_{1.1.2}$ value=$level_2 >$	4
$< /S_{1.1} >$	

Table 1: Excerpt of an SLA XML structure with the calculation of pre-fields

Note that, customers can only assess the CSPs and choose the best one satisfying their requirements, only if the CSPs' shared information are relevant to customers' concerns (i.e., stemmed from customers' requirements) [38]. To achieve this, the customer C has to create her/his set of requirements using the same secSLA-XML structure used by the CSP.

2.3 Searching Over Encrypted Data

We define the problem of searching over encrypted data using the following example. Assume a customer who encrypts her/his documents and stores those at the CSP's storage server. However, by encrypting these documents, the customer can not search for certain keywords anymore and thus, content retrieving is very inefficient. The basic approach for retrieving the required data related to a certain keyword would require the customer to download *all* stored encrypted documents, and then decrypt them to perform the keyword search. However, this solution is time consuming and impractical. In addition, retrieving all files incurs unnecessary network traffic, which is undesirable in the pay-as-you-use cloud paradigm used today.

From the above problems, the need arises for an efficient data retrieval scheme which enables the customer to search directly over encrypted data. A solution to this problems is what is widely known as searchable encryption (SE). SE allows a customer to encrypt data in such a way that she/he can later generate search tokens to send queries to the CSP. Given these tokens, the CSP can search over the encrypted data and retrieve the required encrypted files. As specified in [33], a SE scheme is secure if: (i) the ciphertext alone reveal no information about the encrypted data, (ii) the encrypted data together with a search token (i.e., queries) reveals at most the result of the search, and (iii) search tokens can only be generated using the same encryption key used to encrypt the data.

There exists a large number of SE schemes [10, 16, 33, 50] which are either deterministic or randomized. Deterministic schemes [10] encrypt the same message to the same ciphertext. However, it does not protect against frequency analysis attacks. On the other hand, randomized schemes [33, 50] prevent frequency analysis by salting ciphertexts and thus providing stronger security guarantees. However, the usage of salt in these schemes requires combining each token with each salt, resulting in a processing time linear in the number of salts for each token. In [49] Sherry et al., introduced an encryption scheme which achieves both the detection speed of deterministic encryption and the security of randomized encryption. Our system uses a deterministic encryption scheme for searchable encryption.

2.4 Privacy Preserving Computations

Secure two party computation. The aim of secure two-party computation is to enable both parties to carry out computing tasks without revealing information of any kind about private data to the participants. Assume two parties, A and B have some private information. They want to learn the result of some function using both of their inputs, while each party would learn nothing about the other party input. To achieve this, Yao's protocol based on garbled circuit (named Yao garbled circuit) [36, 53] is used. Yao's protocol based on garbled circuits allow two parties to exchange a garbled circuit and garbled inputs for a function, which can be used to compute an output without leaking information about their inputs.

A garbled circuit is a circuit that consists of garbled gates and their decryption tables. In a garbled gate, two random bits have been selected for every input wire to the gate, representing 0 and 1. Those bits garble the gate, making it impossible to compute the output unless someone has access to the garbled computation table. The garbled computation table maps essentially the random inputs to the output of the gate,

which is also random.

We illustrate the basic idea of how Yao's garbled circuit can be used, assume two parties A and B that have secret inputs, x and y respectively. Both of them want to compute $F(x, y)$ without revealing their private inputs to each other (x to B and y to A). To achieve this, one party (for instance A) prepares a garbled version of the computing function F (named $\mathsf{Gar}F$). Basically, $\mathsf{Gar}F$ produces the same output as $F(x, y)$, if given the corresponding encoding of each bit of both inputs x and y.

The inputs garbling is done by producing a pair of labels for each input bit of F (G^0, G^1, that is one label corresponds to bit 0 and the other to 1). Next, A sends $\mathsf{Gar}F$ along with the encoding of x to B, which only needs the encoding of y from A to compute the $\mathsf{Gar}F$ without learning any intermediate values. For this task, both parties use oblivious transfer [7, 39, 47].

Oblivious transfer (OT). OT is a crucial component of the garbled circuit approach, as it enables party B to obtain the encoding of the b bit from A, without (i) A knowing b and (ii) B learning the encoding scheme. In this way, party B can request from A the keys that he can use for his input encoding without i) A learning B' input and ii) B exploiting the protocol by having access to the encoding scheme and computing much more than allowed.

3 Requirements Analysis

In this section, we describe the system model, present the system requirements, and define our threat and trust models.

3.1 System Overview

Finding the best matching CSP (according to the customer's security requirements) is the objective of the proposed system. Our system model (depicted in Figure 1) involves m CSPs[4], a customer C, and a broker B. The customer C is a company or an individual who is searching for the best provider that satisfies her/his requirements. The CSPs are cloud providers that disclose information about the offered security posture in their secSLAs. CSPs are encrypting their secSLAs before sending them to the broker. The broker B is an entity that performs the searching of the customer requirements over the CSPs' encrypted secSLAs on behalf of the customer. Hence, B ranks and manages the selection of the best matching CSP.

3.2 Threat Model

Security literature distinguishes between two adversarial model for secure computation; participants can be either semi-honest or malicious. In this work we consider a semi-honest (also known as honest-but-curious) threat model. This is a commonly used security model for secure computation (we refer the reader to Goldreich [29] for details) where the semi-honest participants correctly follow the introduced protocol but attempt to obtain additional information about the other participants. By considering the semi-honest model, a dishonest participant observing the system's network should not be able to alter or recover stored data.

The semi-honest setting is relevant in this study, as all entities, the CSP, B, and C, would

[4]Throughout the paper, we explain our model and queries searching scheme using only one CSP. Nevertheless, the same model applies for all CSPs.

like to continue the protocol; acquire the best matching CSP according to C's requirements. However, each entity can attempt to obtain additional information about the other entity's input as depicted in Table 2.

Customer C	Tries to learn the CSP's private key k in order to learn the CSP's secSLA
Cloud Provider CSP	Tries to learn C's requirements or sends a faulty secSLA[5]
Broker B	Tries to (a) alter C's requirements in order to match a colluded CSP, or (b) collude with C to learn the CSPs' secSLAs and identify their identities

Table 2: The semi-honest threat model of our system. Every entity tries to enhance their knowledge about the others while following the protocol.

3.3 Trust Model

In this paper we consider a trusted auditor. Before detailing the proposed system model, we note that customers can only trust the result of an assessment if the information taken as an input is reliable. In other words, in order to guarantee the validity of the proposed system, the encrypted secSLAs provided by the participating CSPs are required to be certified from the trusted certification authority (the auditor). For example, an auditor certifying the CSP's security posture as reflected by its secSLA (e.g., based on an ISO 27001 certificate [20]). The trust assumption relies on the fact that the CSPs' certificates are valid and trusted and thus, their encrypted secSLAs are verified and digitally signed. Such signing scheme should provide a proving statement without revealing the secSLA input. For simplicity, we assume that the participating CSP's encrypted secSLAs are to be verified and digitally signed. We summarize the initial knowledge of each entity in Table 3.

Parameter	Each entity's knowledge			
	C	CSP	B	Auditor
CSP's secSLA		×		
CSP's Encryption Key		×		
CSP's Certificate/ID		×		×
C's Requirements	×		×	

Table 3: In the table we summarize every entity's initial knowledge. Every participating entity has minimal knowledge regarding the others.

3.4 System Requirements

In order to provide the privacy and correctness guarantees, the presented system must ensure:

1) *Input Validation:* The CSPs' encrypted secSLAs provided by the participating CSPs are digitally signed by an auditor.

2) *Fairness:* The system must ensure that if one entity (CSP or B) quits the computation, it can not learn more information than the other entity. In other words, none of the entities can learn the result first and then abort.

3) *Data Confidentiality:* The encrypted CSP's secSLA can only be decrypted using its CSP's private key k. Further, the broker B should *only* learn the output of the searching queries

(i.e., the search queries represent the customer requirements). Moreover, the CSP should not have access to C's requirements to avoid changing its secSLA specifications according to those.

Note that, B or C can ensure the secSLA compliance by monitoring and verifying the offered service levels by (i) using appropriate log samples provided by the CSP, and/or (ii) adding alerts and triggers based on the service key measurable parameters [30]. However, in this paper we only focus on the CSPs evaluation and services assessment and refer the readers to [30, 52] for further monitoring process details.

4 QRES Architecture

In this section, we detail each of *QRES* phases using the progressive stages depicted in Figure 3. As stated earlier, as an initial phase the customers register and are verified by the broker B before *paying* the broker for the offered assessment service. Furthermore, we assume that the broker's certificate ($cert_{\mathsf{B}}$) is validated by the auditor. Then, send by the auditor to the CSPs.

Stage 1: Providers Authorization.

Every CSP is first registered and verified by a trusted auditor. The auditor verifies the CSPs certificate and then signs the encrypted secSLA. The broker B can verify the validity of the encrypted secSLAs by checking for the auditor's signature.

Stage 2: Tokenization.

After C creates her/his set of requirements using the same secSLA-XML structure as used by the CSP (similar to the template shown in Listing 1)[6], both the CSP and C tokenize their secSLA-XML data specifications such that:

- Each CSP splits its specified services (in the secSLA) into substrings. For every substring, it creates a fixed length token (8 bytes per token as depicted in Table 4). The generated tokens are denoted as "t_{CSP}", such that $T_{\mathsf{CSP}} = t^1_{\mathsf{CSP}}, \ldots, t^n_{\mathsf{CSP}}$; where T_{CSP} specifies the set of services offered by the CSP in its secSLA and n is the number of tokens.
- Similarly, C splits her/his requirements into substrings and then generates a fixed 8 bytes token for every substring. The customer's generated tokens are named "keywords" and denoted as "w_{C}" so that $W_{\mathsf{C}} = w^1_{\mathsf{C}}, \ldots, w^n_{\mathsf{C}}$; where W_{C} is the set of customer requirements.

Both the customer and the CSP use the same secSLA template with the same services/SLOs where each of them specify different SLO values. Tokens are generated for each SLO, by searching for the SLO id and extracting only its value and pre-field of the specified SLO (i.e., each SLO in an SLA XML has a unique pre-field as specified earlier and depicted in Table 1). For example, the tokens generated from Listing 1 are: "$level3||3$" and "$level2||4$". Therefore, for each secSLA no

[6]Cloud Security Alliance (CSA) has created a consensus assessments initiative questionnaire (CAIQ) [19], to define the controls contained in an SLA. CSPs have used it as an SLA template by detailing their offered controls and publishing them on the STAR repository [21].

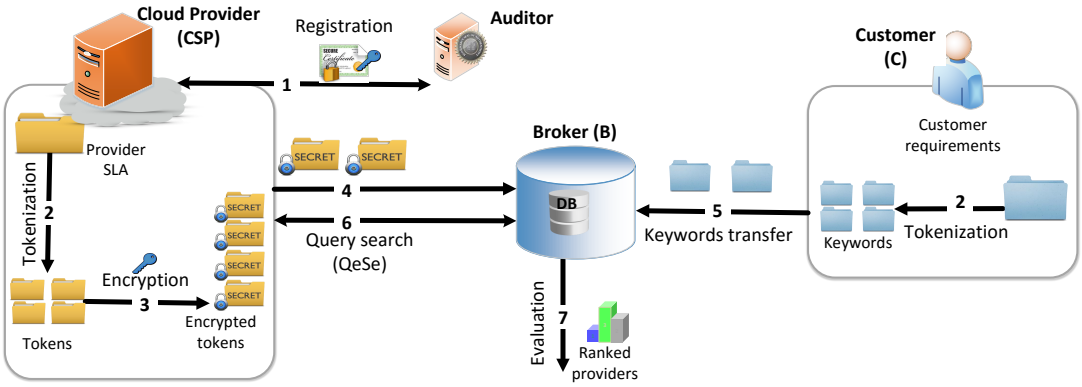

Figure 3: *QRES* System Architecture. Every step is ordered with an increasing number. The procedure in our system begins with an auditor verifying the cloud providers. Then continues with the tokenization of the secSLAs and the customers' requirements and ends with the *QESE* protocol. This protocol enable the matching between the requirements and the secSLAs in the encrypted domain.

similar tokens can be generated as no equal pre-fields exist. Even if the CSP is offering two values for a specific SLO, both tokens would be different because of the different values.

Stage 3: Tokens Encryption.

QRES utilizes a deterministic encryption scheme $\mathsf{Enc}(\mathsf{k}, x)$, such that the encryption of the CSP token (t_{CSP}) is denoted by $\mathsf{Enc}(\mathsf{k}, t_{\mathsf{CSP}})$. For instance, in order to check if the CSP's token (t_{CSP}) is matching the customer keyword (w_{C}), we can simply check if $\mathsf{Enc}(\mathsf{k}, t_{\mathsf{CSP}})$ is equal to $\mathsf{Enc}(\mathsf{k}, w_{\mathsf{C}})$. Unfortunately, deterministic encryption schemes, which are rather fast, cannot be used in every case, as every occurrence of t_{CSP} will result to the same ciphertext. However, this is not the case in our system as every generated token t_{CSP} is unique by design in the CSP's secSLA, as every token contains different pre-field. Therefore, we utilize the $AES-CBC$ encryption scheme to encrypt each CSP's generated token $AES_{\mathsf{k}}(t_{\mathsf{CSP}})$. We use a typical instantiation of a hash function, which is SHA256, to compute the initialization vector (refer to Table 4 for the hash function definition).

Stage 4: Tokens Transfer.

Both the CSP and the customer, send their tokens to the broker. B verifies the auditor's signature and then saves the CSP's encrypted tokens in a database. For m CSPs, B saves m different lists with encrypted tokens.

Stage 5: Keywords Transfer.

In this stage, B receives C's requirements (i.e., keywords) securely via an encrypted channel

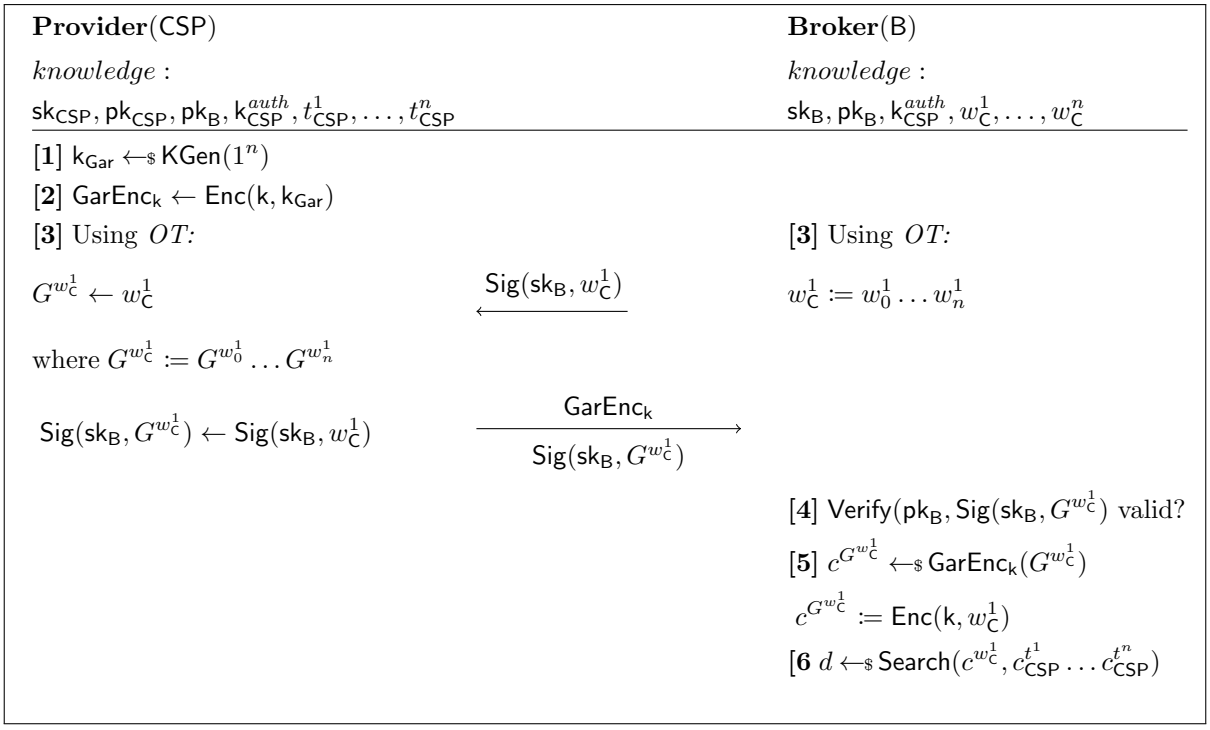

Figure 4: Secure computation protocol between a CSP and the broker (QeSe). The initial knowledge and the generation of the garbled circuit are depicted in lines 1 and 2. The oblivious transfer start in line 3 and the secure two party computation protocol ends at line 5. Then, the broker has acquired the $\mathsf{Enc}(\mathsf{k}, w_{\mathsf{C}}^1)$ and can match the requirements to the secSLAs.

(e.g., SSL).

Once B receives C's messages, it saves the messages and starts a secure two-party computation with each CSP in order to find the best matching CSP according to C's security requirements in the next stage.

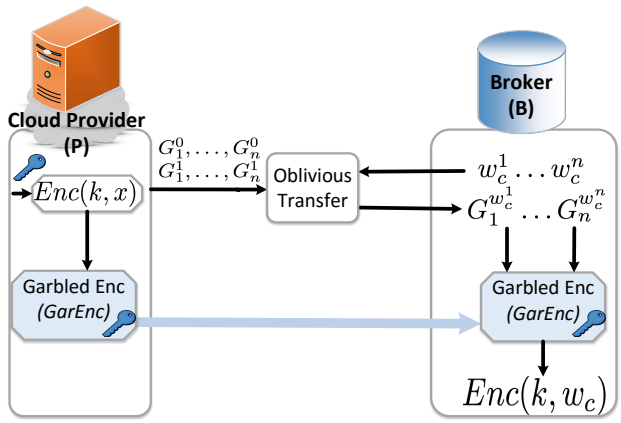

Figure 5: Illustration of the Secure Computation Protocol between the CSP and the broker (*QeSe*).

Stage 6: Query Search (QeSe).

In order to design and build such a privacy preserving system, several challenges need to be addressed:

- In order for customers to search over the encrypted secSLAs, they must have access to the same secret keys as the CSPs. Consequently, the system should uphold the privacy of all parties by allowing customers to search/match their requirements blindly without neither the CSPs learning the customers' queries, nor the broker (or customer) obtaining the CSP's secret key.
- In addition, the system should prevent malicious CSPs from deliberately providing false information trying to match the customer requirements.
- Lastly, the system should prevent malicious customers from just gathering information about each CSP offered services and then quitting (i.e., by performing several searching iterations, each iteration with different requirements or by performing frequency analysis).

To tackle these problems, *QRES* utilizes a novel two-party privacy preserving query over an encrypted data scheme (named *QeSe*). *QeSe* is based on two-party privacy preserving computation (i.e., Yao garbled circuit [36, 53]) to allow CSPs to encrypt the customers' search queries without learning them. Accordingly, customers can find the best matching CSP's secSLA without knowing the CSP's encryption key. Furthermore, the protocol supports a new security property by enhancing the security setting of the traditional two-party privacy preserving computation (Yao garbled circuit evaluation). This is achieved by validating the participants' inputs (i.e., CSP's secSLA and broker's signature on the customer's keywords) before computation.

QeSe. For simplicity, we explain this stage using one CSP and B. The broker B has already stored each CSP encrypted tokens ($\mathsf{Enc}(\mathsf{k}, t_{\mathsf{CSP}})$) (from the previous steps) in a database. Furthermore, B has already received the customer keywords. In order to find the encrypted tokens matching the customer keywords, they have to be encrypted

by the same encryption key used by CSP to encrypt the tokens. In other words, B has to compute $\mathsf{Enc}(\mathsf{k}, w_\mathsf{C})$ for every w_C using the same encryption key used by the CSP. The main challenge here is for B to obtain the encrypted w_C without knowing the CSP's secret key k and without allowing the CSP to learn w_C.

QeSe allows all parties to jointly achieve their purpose (exchange k so w_C can be encrypted) by running a secure two party computation between the CSP and the broker. We enhance the security setting of the traditional two-party privacy preserving computation by allowing the garbled circuit to also check the validity of both the encrypted tokens and the broker's digital signature of the customer's requirements. In case there is a problem with the validation our protocol quits.

The CSP provides B with a "garble" of the encryption function with the key k hardcoded in it, as depicted in Figure 4 (line 2) and Figure 5 (we denote this garbled function by $\mathsf{GarEnc}_\mathsf{k}$). This garbling hides the CSP's secret key k. $\mathsf{GarEnc}_\mathsf{k}(x)$ produces the same output of $\mathsf{Enc}(\mathsf{k}, x)$ if given the corresponding encoding of each bit of input x (i.e., G^x). Thus, B can run this garbled function on each keyword w_C in order to obtain the $\mathsf{Enc}(\mathsf{k}, w_\mathsf{C})$ only if he is able to acquire from the CSP an encoding for the w_C (G^{w_C}) as depicted in line 3 in Figure 4. For this task, both parties (CSP and B) use an oblivious transfer protocol, where B receives an encoding of w_C from the CSP without revealing w_C. The encoding of keywords $w_\mathsf{C}^1, \ldots, w_\mathsf{C}^n$ is denoted as $G_1^{w_\mathsf{C}^1}, \ldots, G_n^{w_\mathsf{C}^n}$. Similarly, the broker receives the encoding of $\mathsf{Sig}(\mathsf{sk}_\mathsf{V}, G_1^{w_\mathsf{C}^1})$, that is the encoding of the keyword's signature (line 3 in Figure 4). Note that, on input $\mathsf{Sig}(\mathsf{sk}_\mathsf{B}, w_\mathsf{C})$, $\mathsf{GarEnc}_\mathsf{k}$ first checks if $\mathsf{Sig}(\mathsf{sk}_\mathsf{B}, w_\mathsf{C})$ is a valid signature of w_C using B's public key (pk_B). If it is valid, w_C is encrypted as $\mathsf{Enc}(\mathsf{k}, w_\mathsf{C})$. It is important for the security of the protocol to mention that, a garbled circuit is no longer secure if B receives more than one encoding for the same circuit. Thus, B obtains a fresh, re-encrypted garbled circuit $\mathsf{GarEnc}_\mathsf{k}$ for every w_C. At the end of this process, B gets the $\mathsf{Enc}(\mathsf{k}, w_\mathsf{C})$ for every w_C. Afterwards, B searches for an exact matching of $\mathsf{Enc}(\mathsf{k}, w_\mathsf{C})$ over all CSPs encrypted tokens (as depicted in line 6 in Figure 4). Each successful matching result is saved in a list at B using an index d. The process is repeated for all CSPs. Hence, for m CSPs, B creates m lists. Each of these lists contains the number of encrypted tokens matching C's keywords.

Stage 7: CSPs Ranking.

As a last step, the CSPs are evaluated and ranked according to the number of tokens matching the customer requirements. The CSP with the highest number of encrypted tokens matching the customer keywords, is given the highest score and selected as the best matching provider.

5 Security Analysis

In this section we formally analyse and verify the security considerations of the system architecture described in Section 4, with respect to the security requirements defined in Section 3.4.

5.1 Formal Analysis

Formal methods can be used to model a cryptographic protocol and its security properties against an adversarial model, together with an efficient procedure to determine whether a model satisfies those properties.

In this section, we analyse the security of the proposed system protocols with respect to the security objectives outlined in Section 3.4 using ProVerif [11]. ProVerif is an automated verification tool that can handle an unbounded number of protocol sessions. It is used to model cryptographic protocols and their security properties against a strong adversarial model. In contrast to other state-of-the-art tools (e.g., the Avispa tools [5] and Scyther [22]), ProVerif provides a larger feature set [23]. Furthermore, it allows the modelling of cryptographic primitives using equational theory which is used to model the Yao's garbled circuit protocol.

In order to verify a system's security, first we define a list of security properties. Then, the context in which the system functions are created. The system's context consists of assumptions about the environment and the adversarial model. We model the formal specification of the *QeSe* protocol, the security objectives and the adversary model using applied pi-calculus [11]. The applied pi-calculus modelling is used as an input to the ProVerif tool. ProVerif then proves if the claimed security properties are fulfilled. We refer the readers to Appendix 8 for more details about the applied pi-calculus semantics and for more details about protocol modelling and the property verification. The formal specification includes the following components:

1. Agent model. Agents represent the protocol parties which execute the roles of the protocol. For instance, we have the sender and the receiver roles in the protocol. Therefore, in each protocol, each agent performs one or more roles. The agent model is based on a "closed world assumption", which means that honest agents show no behaviour other than the one described in the protocol specification (this model corresponds to honest-but-curious threat model in Section 3.2).

2. Communication channel. The communication model describes how the messages are exchanged between the agents. The channel can be private or public according to the threat model. In our scenario the channel is public.

3. Threat model. Adversaries are modelled as agents that aim to violate the security objectives. ProVerif uses the standard Dolev-Yao adversary model [24]. In this model the adversary has complete control over the public communication network.

4. Protocol specification. The protocol specification describes the behaviour of each of the roles in the protocol. The system does not execute the actual protocol but it executes the protocol roles performed by the agents.

To model *QeSe* protocol, we model Yao's garbled circuit protocol to verify *Data confidentiality and Fairness*. Two agents are used to model our protocol in the applied

pi-calculus (i.e., one CSP and one broker B). Based on Yao's circuit two algorithms are defined (namely garble Gar and evaluate Eval). Gar takes an input function with n bits and outputs a garbled function and a pair of labels for each input bit of the function. Eval takes the garbled function and garbled inputs and returns the required function output.

We model the CSP's encryption function as a public parameter represented by a free variables z_f (i.e., free variable is similar to global scope in programming languages; that is, free names are globally known). The private parameters of the CSP and B are the protocol inputs (the encryption key and the keyword, respectively). The CSP's and B's private parameters are represented as free variables x_{CSP} and y_{B}, respectively.

The goal of B is to obtain the encryption of its keyword using the CSP's key. To achieve this we use the protocol specification shown in Figures 4. The garbling function is modelled using a generated random key, named as the garbling key $\mathsf{Gar}(., \mathsf{k}_{\mathsf{Gar}})$. The garbling key is used to securely garble the circuit at each protocol run (as stated earlier, to ensure the garbled circuit security, B receives a fresh, re-encrypted garbled circuit for each input $\mathsf{Gar}(y_{\mathsf{B}}, \mathsf{k}_{\mathsf{Gar}})$).

To model the oblivious transfer, the Broker B generates a commitment of each keyword with a fresh generated nonce. The commitment of each keyword cannot be modified and is hidden using the nonce. The commitment is used to request a garbling of the keyword from the CSP without disclosing the keyword to the CSP. We do model the encryption output as $\mathsf{Eval}(z_f, x_{\mathsf{CSP}}, y_{\mathsf{B}})$, where both CSP and B want to find the keyword encryption output without disclosing x_{CSP} to B and y_{B} to CSP.

5. Security properties. These properties specify the security requirements of the protocol defined in Section 3.4.

(a) ***Data Confidentiality:*** Modelling strong secrecy to verify the encrypted SLA's confidentiality is straightforward and modelled easily in ProVerif (adversary learns nothing about the SLA). However, to prove the confidentiality property in *QeSe*, we have to prove that the only leakage about the input of the two parties (CSP and B) should come from the result of the evaluated function. In other words, if a party obtained the output of the evaluated function, no leakage should occur about the two parties' inputs.

This is proved using the indistinguishability property (the notion of indistinguishability is generally named observational equivalence in the formal model [11]). Intuitively, two processes (P_1 and P_2 are observationally equivalent (i.e., written $P_1 \approx P_2$), when an attacker cannot distinguish between the two. Formal definitions of the indistinguishability property can be found in [2, 14]. For example, the privacy property of an electronic voting protocol is expressed as [11]:

$$P(\mathsf{sk}_A, v_1)|P(\mathsf{sk}_B, v_2) \approx P(\mathsf{sk}_A, v_2)|P(\mathsf{sk}_B, v_1)$$

P is the voting process, and an attacker cannot distinguish between the two situations; i) in which A votes for v_1 and

B votes for v_2 from ii) that A votes for v_2 and B for v_1, where v_1 and v_2 are the candidates for whom A and B vote.

To prove the data confidentiality, we model two processes that are replicated an unbounded number of time and executed in parallel. In the first process P_1, CSP and B sends two inputs, represented as free variables x^0_{CSP} and y^0_{B}, respectively. While in the second process P_2, the CSP and B sends free variables x^1_{CSP} and y^1_{B}, respectively. If the two defined processes are observationally equivalent ($P_1 \approx P_2$), then we say that the attacker cannot distinguish between the two process input.

Theorem 1: Data confidentiality is preserved in the system if P_1 and P_2 are observationally equivalent ($P_1 \approx P_2$). In our protocol this is proven using ProVerif.

(b) ***Fairness:*** In *QeSe*, fairness is preserved if the final result obtained by both parties CSP (sender) and B (receiver) after a secured computation is consistent. None of the parties can learn the result first and then abort. This can be proved using the ProVerif's correspondence property [12]. An example of the correspondence property is, $e(M_1, \ldots, M_j) \implies e'(N_1, \ldots, N_k)$, where for any trace of the protocol for each occurrence of event $e(M_1, \ldots, M_j)$, there is a previous occurrence of $e'(N_1, \ldots, N_k)$.

Therefore, we define the fairness property as follows:

$$\mathsf{CSP}term(z_f, x_{\mathsf{CSP}}, y_{\mathsf{B}}, r) \implies$$
$$y_{\mathsf{B}} = \mathsf{com}(y_1, n) \wedge r = \mathsf{Eval}(z_f, x_{\mathsf{CSP}}, y_1) \text{ and}$$
$$\mathsf{B}term(z_f, x_{\mathsf{CSP}}, y_{\mathsf{B}}, r') \implies$$
$$x_{\mathsf{CSP}} = \mathsf{Gar}(x_1, \mathsf{k}_{\mathsf{Gar}}) \wedge r' = \mathsf{Eval}(z_f, x_1, y_{\mathsf{B}}) \quad (1)$$

Note that, the CSP's and B's private parameters are represented using free variables x_{CSP} and y_{B} as stated earlier. The property in equation 1 states that the CSP with arguments $z_f, x_{\mathsf{CSP}}, y_{\mathsf{B}}, r$ terminates a protocol run with B after (i) CSP receives a commitment of B's input with a random value n ($y_{\mathsf{B}} = \mathsf{com}(y_1, n)$) and (ii) finding the result r of the of the function z_f on inputs x_{CSP} and y_1. Similarly, B terminates the protocol run with the CSP after receiving the CSP's garbled input x_{CSP} and finding the result r' of the garbling function z_f on inputs x_1, y_{B}.
Using the correspondence property defined in equation 1, the process $!(\mathsf{CSP}(z_f, x_{\mathsf{CSP}}))|!(\mathsf{B}(z_f, y_{\mathsf{B}}))$, which models the two agents processes and can be executed any number of sessions.

Theorem 2: Fairness of all possible executions of sessions of honest parties in process P is preserved if the correspondence property defined in equation 1 is true. For our protocol this is proved using Proverif.

5.2 Further Security Considerations

Our system and its protocol are designed and verified against honest but curious entities.

However, with minor protocol modifications our system mitigate against malicious parties.

5.2.1 Malicious entities

- **Malicious CSP:** In the case of a malicious CSP, it might send (i) an incorrect garbling function to broker B, and/or (ii) a faulty encrypted secSLA. To prevent the first form of attack (i), B can prove that the garbling is correct by using the "cut-and-choose" technique [31, 35]. Under this technique, a CSP constructs n versions of the circuit, each structured identically but garbled differently so that the keys for each gate in each circuit is unique. Additionally, the CSP generates a commitment for each of its garbled inputs. The CSP then sends each of the garbled circuits with its garbled inputs to B. Further, B selects $n-1$ versions of the circuit to verify. The CSP degarbles each of the $n-1$ selected circuits, so that B can verify that each of the revealed circuits are constructed correctly and as expected.

  B checks whether the CSP's garbled inputs match their corresponding (previously sent commitments). If everything is correct, B evaluates the rest of the circuits and derives the output from them. Thus, if a malicious CSP constructs the circuits incorrectly, B will detect this with high probability. Regarding faulty SLA's, as explained earlier, an auditor (whose reputation depends on its trustworthiness) first ensures the validity of the CSP's input and thus prevents any CSP from providing faulty encrypted SLAs.

- **Malicious customer:** A single customer or a group of colluded customers can try to use the *QRES* system to find each CSPs security posture. Each of the colluded entities can specify different requirements and at the end, each customer gets the best matching CSP's secSLA according to her/his requirements. To better illustrate this kind of attack, we show an empirical validation of the proposed system through the secSLA information used in the implementation section; *a secSLA with 50 SLOs, where each SLO is composed of four service levels*. The number of possible SLA combinations is "4^{50}" (i.e., generally x^y, where x is the number of service levels and y is the number of SLOs). Each customer keyword takes on average "0.52" seconds to search over one CSP's encrypted secSLA with 50 SLOs with 1 level each (i.e., we demonstrate the system evaluation and performance in Section 6; as depicted in Figure 6). Thus to find all the possible combinations, it takes $2*10^{22}$ years to learn a CSP encrypted secSLA. Hence, this type of attack is not feasible in our system. In addition, countermeasures with specific time between sequential queries could be implemented in the either the broker or the CSP.

- **Malicious broker:** A theoretical attack that a broker could perform would be to attempt and alter C's keywords. However, such an attack would be immediately detected by the customer as at the end they get to know the security details they 'negotiated' with the cloud provider.

5.2.2 Providing CSP Anonymity

System Changes The *QRES* system should prevent malicious customers to ascertain information about each CSP offered services (i.e., by

performing several searching iterations, each iteration with different requirements or by performing frequency analysis). To prevent malicious customers from learning information about CSPs SLAs, *QRES* allows the customers to search for their requirements over "anonymous" CSPs encrypted secSLAs. The CSPs anonymity mitigates the malicious customers attacks as customers learn the result of the searching queries and nothing else.

In order for *QRES* to provide anonymity for the CSPs, the auditor should first generate a unique authentication secret for each CSP during their registration phase. This secret then is send to the broker. Every communication between the broker and the CSP runs over onion routing [48] anonymity network to ensure anonymity of CSPs. In addition, the CSP generates an authentication challenge using the hash of a nonce and their unique authentication secrets. The challenge together with the nonce are send to the broker. The broker stores the nonces and the challenges alongside with every CSP's secSLA, but cannot tell the real identities of the CSPs. After finding the best matching CSP, the selected CSP's authentication secret is sent to the auditor by the broker in order to identify the CSP's identity. Next, the auditor sends the selected CSP's identity to B to manage the agreement between both the CSP and the customer.

Verifying Anonymity In order to verify the anonymity property of this modification of the *QRES* system, we use the ProVerif tool. We model two processes that are replicated an unbounded number of time and executed in parallel. In each process two CSPs participate by sending their tokens. First, each CSP constructs an OR circuit and sends the onion data ($\mathsf{CSP}_1 \leftrightarrow N_1 \leftrightarrow N_2 \leftrightarrow N_3$) and ($\mathsf{CSP}_2 \leftrightarrow N_1 \leftrightarrow N_2 \leftrightarrow N_3$). Then, each of the intermediate nodes (N_1 and N_2) removes one layer of encryption and at the end forwards the onion to N_3. Finally, once the exit node N_3 receives the two onions from the two CSPs, it removes the last layer and sends the messages to B.

In the first process P_1, CSP_1 and CSP_2 sends two tokens t_1 and t_2, respectively. Once the exit node N_3 removes the last onion layer, it sends the message to B on a public channel ($t_1||t_2$). In the second process P_2, the two tokens are swapped; such that CSP_1 and CSP_2 sends t_2 and t_1, respectively. Similarly, N_3 publishes the message on a public channel ($t_2||t_1$). The CSPs anonymity is preserved if an attacker cannot distinguish between the two messages and hence, cannot learn which token is sent by which CSP. Nodes transfer messages to each other using a public channel.

If the two defined processes are observationally equivalent ($P_1 \approx P_2$), then we say that the attacker cannot distinguish between t_1 and t_2, which means the attacker is unable to distinguish when the message changes. Hence, the attacker is not able to link two communication streams of the same CSP, and thus cannot learn which message is sent by which CSP.

Theorem 3: The observational equivalence of P_1 and P_2 defined in equation 2 holds ($P_1 \approx P_2$).

$$P_1(t_1, t_2) \approx P_2(t_2, t_1) \tag{2}$$

6 Implementation and Evaluation

Implementation. *QRES* is implemented in Java, using Apache-Tomcat 9.0 and Amazon

DynamoDB [4] database, which is a NoSQL database service. The DynamoDB supports both string and key value store models which we used to store the encrypted tokens (B side). We used Tomcat web server on our Ubuntu machine to transmit the encrypted data to Amazon DynamoDB (the CSP side). We implement the hash functions using HMAC based on SHA-256. The symmetric encryption scheme is implemented based on AES 128. Moreover, we use $\beta = 128$ for nonces and 8 bytes per token. Furthermore, we modified Yao's garbled circuit coding [17] to fit our encryption scheme. Finally, we implemented two Java Server Page (JSP) files to send the tokens and to search for the keywords. The complete source code along with a detailed explanation of setting up and using the *QRES* can be found at [46].

Performance Evaluation. We evaluate the performance of the *QeSe* protocol running between the CSP and the B using two use-cases. First, we use *QReS* with one CSP which is offering various tokens (SLOs) in its secSLA . Note that, the CSP is offering *one service level for each SLO* in its secSLA. Each token (8 bytes) is encrypted and sent to Amazon DynamoDB. Figure 6 shows the amount of time spent by a broker to search for the customer keywords (i.e. 5,10, 20 and 50 keywords) over 10, 20, 50 and 150 SLOs provided by one CSP. The use-case shows that the time to search for the customer's different keywords over various number of SLOs is almost the same despite the number of the offered SLOs. This is expected as the most time consuming part of the computation, that is the computation of the circuits, is the same despite the amount of offered SLOs.

Further, we explore the case in which a customer compares various CSPs based on their advertised secSLAs. Figure 7 shows the amount of time a broker spends to search for the customer requirements (keywords) over 1, 10 and 30 CSPs' secSLAs where each secSLA consists of 150 SLOs. For each CSP's secSLA, we extracted 150 tokens (8 bytes each) which are then encrypted and sent to Amazon DynamoDB (in our current implementation we saved each CSP token in a different table). The use-case shows a linear time progression as the broker searches for more keywords in each CSP's table.

The SLOs are extracted from the public STAR repository [21]. The rationale for using STAR repository is that (i) to the best of our knowledge no other cloud SLA repositories are publicly available and (ii) major CSPs are still in the process of restructuring their SLAs by leveraging the recently published ISO/IEC 19086. Currently, the STAR contains reports with CSP's answers to Consensus Assessments Initiative Questionnaire [19] with yes/no answers. Furthermore, we utilized other requirements defined in multiple research projects such as A4cloud [41], CUMULUS [42], and SPECS [45].

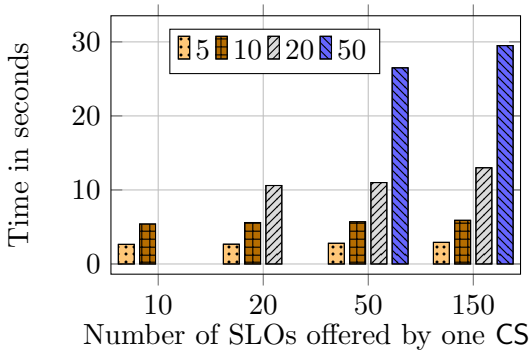

Figure 6: Time used by a customer to search for her/his different keywords over varied number of SLOs offered by one CSP.

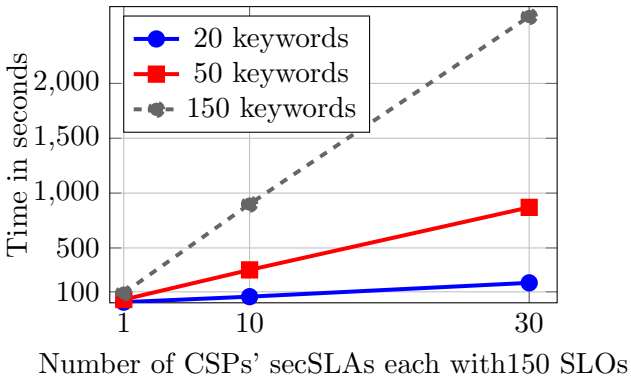

Figure 7: Time used by the customers to search for their different keywords over 150 SLOs offered by varied CSPs.

7 Related Work

In this section we describe the related work which falls into four categories: security quantification, secured computation, searchable encryption, and formal analysis.

Security quantification: Security requirements for non-cloud scenarios have been addressed by Casola et al. [18], who propose a methodology to evaluate security SLAs for web services. In [18], the authors propose a methodology to assess security SLAs for web services. In [26] and [34], the authors propose a technique to aggregate security metrics from web services security SLAs. However, in contrast to our work, the previous work did not empirically validate the specified metrics. In [3], the authors propose the notion of evaluating cloud SLAs by introducing a metric to benchmark the security of a CSP based on categories. However, all of the previously mentioned approaches do not allow the CSPs to specify their security posture.

Secure computation: Fully homomorphic encryption (FHE) [28] and other general functional encryption [27] schemes can be used to compute functions over encrypted data. However, they do not address all our desired security properties, and they are prohibitively slow for the selected usecase. Some recent systems such as CryptDB [43] and Mylar [44] support secure computation efficiently. However, these systems enables certain types of search over the encrypted data which are not matching our required properties. In [49], the authors utilize secure computation on encrypted data for deep packet inspection. However, the existing two part computation over encrypted data schemes do not provide the desired property and functionality of *QRES*.

Searchable encryption: There exist different kinds of searchable encryption schemes named deterministic [10] and randomized [33, 50]. In [49], Sherry et al. introduced an encryption scheme that achieves both the detection speed of the deterministic encryption and the security of the randomized encryption.

Formal analysis: In [8], the authors provide practical repudiation for autonomous communication networks by tracing back the selected outbound traffic to the predecessor node. They conduct a formal security analysis of the OR protocol using ProVerif by formalizing anonymity and no forward traceability as observational equivalence relations, and backward traceability and no false accusation as trace properties. In [9] an abstraction of non-interactive zero-knowledge proofs within the applied pi-calculus is presented. The authors transform their abstraction into an equivalent formalization that is accessible

to ProVerif. The authors in [15] study the formal security properties of well-established protocols for secure file sharing on untrusted storage. The protocol modeling and properties verification are studied using the automatic protocol verifier ProVerif.

8 Conclusion

Cloud Service Providers are a lucrative target because of the amount of data they process. A disruption of their services due to a hack, can cause them a terrible financial loss and many problems for the customers whose data are compromised. The inclusion of security implementations information in the Service Level Agreements is risky, as it would enable malicious entities to better orchestrate their attacks and easier discover vulnerabilities. Hence, many providers do not include them in their SLAs. The SLAs however, are the agreement between customers and providers for the services offered, and the security implementations is an important factor for choosing a cloud provider.

We tackle this problem with a system called *QRES*. Our system enables CSPs to create security SLAs and publish them encrypted. With the help of an intermediate node, customers can find the cloud provider better matching their security needs by contrasting their requirements against the encrypted secSLAs. Our system is utilizing the *QeSe* protocol, a searchable encryption scheme protected by secure two party computation. The inputs of every party remain private while the output is only learned by the broker and consequently the customer.

We implement *QRES* and formally verify its security and privacy properties using ProVerif. In our real word tests, using existing standardized SLAs by the latest industrial standard, the system proved to be fast for the required use case. In our measurements, it requires less than 30 seconds to privately search for 50 customer keywords at a CSP.

QRES is the first step towards providing security assurance and "transparency" between cloud customers and CSPs, while at the same time ensures the confidentiality of the CSPs sensitive information. Moreover, we aim to extend the system model presented in this paper by checking whether general functional encryption [27] schemes can address all the desired security properties of our model.

References

[1] R. Popa, J. Lorch, D. Molnar, H. Wang, and L. Zhuang. Enabling security in cloud storage SLAs with CloudProof. In *Proc. of USENIX ATC*, 2011.

[2] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *In Sigplan Notices*, 36(3):104–115, 2001.

[3] Mohamed Almorsy, John Grundy, and Amani Ibrahim. Collaboration-based cloud computing security management framework. In *Proc. of Cloud Computing*, pages 364–371, 2011.

[4] Amazon. Amazon DynamoDB, 2012. Accessed: 2017-12-15.

[5] Alessandro Armando, David Basin, Yohan Boichut, et al. The avispa tool for the automated validation of internet security protocols and applications. In *Proc. of CAV*, pages 281–285, 2005.

[6] Taylor Armerding. The 16 biggest data breaches of the 21st century. Technical report, 2017.

[7] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proc. of SIGSAC*, pages 535–548, 2013.

[8] Michael Backes, Jeremy Clark, Peter Druschel, Aniket Kate, and Milivoj Simeonovski. Introducing accountability to anonymity networks. *In CoRR arXiv:1311.3151*, 2013.

[9] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Proc. of Security and Privacy*, pages 202–215, 2008.

[10] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Proc. of CRYPTO*, pages 535–552, 2007.

[11] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of CSFW*, pages 82–96, 2001.

[12] Bruno Blanchet. From secrecy to authenticity in security protocols. In *Proc. of SAS*, pages 342–359, 2002.

[13] Bruno Blanchet. Automatic verification of correspondences for security protocols. *In Journal of Computer Security*, 17(4):363–434, 2009.

[14] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *In Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[15] Bruno Blanchet and Avik Chaudhuri. Automated formal analysis of a protocol for secure file sharing on untrusted storage. In *Proc. of Security and Privacy*, pages 417–431, 2008.

[16] Dan Boneh, Giovanni Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Proc. of EUROCRYPT*, pages 506–522, 2004.

[17] Michael Brenner. Yao's garbled circuits implementation, 2011.

[18] Valentina Casola, Antonino Mazzeo, Nicola Mazzocca, and Massimiliano Rak. A SLA evaluation methodology in Service Oriented Architectures. *In Quality of Protection*, pages 119 – 130, 2006.

[19] Cloud Security Alliance. The Consensus Assessments Initiative Questionnaire v3.0.1, 2017.

[20] Cloud Security Alliance. The Open Certification Framework. 2017.

[21] Cloud Security Alliance. The Security, Trust & Assurance Registry (STAR). 2017.

[22] Cas Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Proc. of CAV*, pages 414–418, 2008.

[23] Cas Cremers, Pascal Lafourcade, and Philippe Nadeau. Comparing state spaces in automatic security protocol analysis. In *Proc. of Formal to Practical Security*, pages 70–94, 2009.

[24] Danny Dolev and Andrew Yao. On the security of public key protocols. *Transactions on information theory*, 29(2):198–208, 1983.

[25] International Organization for Standardization. Information Technology, Security Techniques, Code of Practice for Information Security Management. Technical Report ISO/IEC 27002:2013, 2013.

[26] Ganna Frankova and Artsiom Yautsiukhin. Service and protection level agreements for business processes. In *Proc. of ICSOC*, pages 38 – 43, 2007.

[27] Sanjam Garg, Craig Gentry, Shai Halevi, et al. Candidate indistinguishability obfuscation and functional encryption for all circuits. *In SICOMP*, 45(3):882–929, 2016.

[28] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. of STOC*, pages 169–178, 2009.

[29] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[30] Giles Hogben and Marnix Dekker. Procure secure: A guide to monitoring of security service levels in cloud contracts. Technical report, 2012.

[31] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *Proc. of CRYPTO*, pages 18–35. 2013.

[32] International Organization for Standardization. Information Technology - Cloud Computing - Service Level Agreement (SLA) Framework and Terminology. Technical Report ISO/IEC 19086, 2014.

[33] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proc. of CCS*, pages 965–976, 2012.

[34] Leanid Krautsevich, Fabio Martinelli, and Artsiom Yautsiukhin. A general method for assessment of security in complex services. In *Proc. of European Conference on a Service-Based Internet*, pages 153–164, 2011.

[35] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Proc. of EUROCRYPT*, pages 52–78, 2007.

[36] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[37] Jesus Luna, Ahmed Taha, Ruben Trapero, and Neeraj Suri. Quantitative reasoning about cloud security using service level agreements. *In TCC*, (99), 2017.

[38] Mandy Messenger. Cyber-security: Why would I tell you? Technical Report 3, Research briefing report by Mandy Messenger, 2006.

[39] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *Proc. of CRYPTO*, pages 573–590, 1999.

[40] National Institute of Standards and Technology. Security Controls for Federal Information Systems. Technical Report NIST SP-800-53, 2013.

[41] David Nunez and Carmen Gago. D:C-5.2 Validation of the Accountability Metrics. Technical report, Accountability For Cloud and Other Future Internet Services (A4Cloud), 2014.

[42] Alain Pannetrat, Giles Hogben, Spyros Katopodis, George Spanoudakis, and Carlos Cazorla. D2.1 Security-Aware SLA Specification Language and Cloud Security Dependency model. Technical report, sep 2013.

[43] Raluca Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proc. of SOSP*, pages 85–100, 2011.

[44] Raluca Popa, Emily Stark, Jonas Helfer, Steven Valdez, Nickolai Zeldovich, Frans Kaashoek, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *Proc. of NSDI*, pages 157–172, 2014.

[45] SPECS Project. Report on requirements for Cloud SLA negotiation - Final. Technical Report Deliverable 2.1.2, oct 2014.

[46] QRES. Quantitative reasoning on encrypted security slas, 2017.

[47] Michael Rabin. How to exchange secrets with oblivious transfer. In *Proc. of IACR Cryptology ePrint Archive*, volume 2005, page 187, 2005.

[48] Michael Reed, Paul Syverson, and David Goldschlag. Anonymous connections and onion routing. *In Selected areas in Communications*, 16(4):482–494, 1998.

[49] Justine Sherry, Chang Lan, Raluca Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. *In SIGCOMM*, 45(4):213–226, 2015.

[50] Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proc. of S&P*, pages 44–55, 2000.

[51] Ahmed Taha, Patrick Metzler, Ruben Trapero, Jesus Luna, and Neeraj Suri. Identifying and utilizing dependencies across cloud security services. In *Proc. of AsiaCCS*, pages 329–340, 2016.

[52] US Government. Federal Risk and Authorization Management Programme - Concept of Operations. Technical report, 2012.

[53] Andrew Yao. How to generate and exchange secrets. In *Proc. of Symposium on IEEE*, pages 162–167, 1986.

Appendix

Security Definition

A function $\nu()$ is negligible in an input parameter k if for every positive polynomial $p()$ and all large k, $\nu(k) \leqslant 1/p(k)$. An Encryption Scheme ES is a tuple of polynomial time algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ where:

- $\mathsf{k} \leftarrow\!\!\$\, \mathsf{KGen}(1^n)$: It takes as input a security parameter $1^n$and outputs a secret key k.

- $c \leftarrow\!\!\$\, \mathsf{Enc}(\mathsf{pk}, m)$: It takes as input a public key pk, a plaintext m, and outputs a a ciphertext c.

- $m \leftarrow_{\$} \mathsf{Dec}(\mathsf{sk}, c)$: It takes as input a secret key sk, a ciphertext c, and outputs a a plaintext m.

Note that, for symmetric encryption scheme $\mathsf{pk} = \mathsf{sk}$. We denote $s \leftarrow_{\$} \mathsf{Sig}(\mathsf{sk}, m)$ for signing m with a secret key; verify the signature using the public key $\mathsf{Vf}(\mathsf{pk}, s)$ and recover m.

A Symmetric searchable encryption SSE allows an entity to encrypt data in such a way that it can later generate search tokens to search over the encrypted data and find the required files only. Based on the SE schemes definition in [33,49], we define an SSE scheme as a tuple of polynomial time algorithms (KGen, Enc, KeyEnc, Search) where:

- $\mathsf{k} \leftarrow_{\$} \mathsf{KGen}(1^n)$: It takes as input a security parameter $1^n$and outputs a secret key k.

- $c^{t^1}, \ldots, c^{t^n} \leftarrow_{\$} \mathsf{Enc}(\mathsf{k}, t^1, \ldots, t^n)$: It takes as input a secret key k and a set of n substrings, and outputs a set of n encrypted substrings.

- $c^w \leftarrow_{\$} \mathsf{KeyEnc}(\mathsf{k}, w)$: It takes as input a secret key k and one substring, and outputs an encrypted substring.

- $d \leftarrow_{\$} \mathsf{Search}(c^w, c^{t^1}, \ldots, c^{t^n})$: It searches for the encrypted substring from $c^{t^1}, \ldots, c^{t^n}$ which matches c^w. It takes as input $n+1$ encrypted substrings and outputs the matched encrypted substring index. If there is no any encrypted substring matching c^w, it outputs $\perp$.

Operational semantics

The rules that define the operational semantics of applied pi-calculus and ProVerif are adapted from [13]. The identifiers a, b, c, k and similar ones range over names, and x, y and z range over variables. As detailed in [13], set of symbols is also assumed for constructors and destructors such that f for a constructor and g for a destructor. Constructors are used to build terms. Therefore, the terms are variables, names, and constructor applications of the form $f(M_1, \ldots, M_n)$.

We use the constructors and destructors defined in [13] as an initial step to represent the cryptographic operations as depicted in Figure 8. We added other different constructors/destructors which are used to define our protocol. Constructors and destructors can be public or private. The public ones can be used by the adversary, which is the case when not stated otherwise. The private ones can be used only by honest participants.

The operational semantics used are presented in Figure 9. A semantic configuration is a pair $\mathcal{E}$,$\mathcal{P}$ where the $\mathcal{E}$ is a finite set of names and $\mathcal{P}$ is a finite multiset of closed processes. The semantics of the calculus is defined by a reduction relation $\rightarrow$ on semantic configurations as shown in Figure 9. The process $event(M).\mathsf{P}$ executes the event $event(M)$ and then executes P. The input process $in(M, x).\mathsf{P}$ inputs a message, with x bound to it, on channel M, and executes P. The output process $out(M, N).\mathsf{P}$ outputs the message N on the channel M and then executes P. The nil process 0 does nothing. The process $\mathsf{P}|\mathsf{Q}$ is the parallel composition of P and Q. The replication $!\mathsf{P}$ represents an unbounded number of copies of P in parallel. $(newa)\mathsf{P}$ creates a new name a and then executes P. The conditional if $M = N$ then P else Q executes P if M and N reduce to the same term at runtime; otherwise, it executes Q. Finally, let $x = M$ in P as syntactic for $\mathsf{P}\{M/x\}$ which is the process obtained from P by replacing every occurrence of x with M. As usual, we may omit an else clause when it consists of 0.

Symmetric enc/dec:
Constructor: encryption of x with the shared secret key k, $senc(x, k)$
Destructor: decryption $sdec(senc(x, k), k) \rightarrow x$
Asymmetric enc/dec:
Constructor: encryption of x with the public key generation from a secret key k, $pk(k)$, $aenc(x, pk(k))$
Destructor: decryption
$adec(aenc(x, pk(k)), k) \rightarrow x$
Signatures:
Constructors: signature of x with the secret key k, $sign(x, k)$
Destructors: signature verification using he public key generation from a secret key
$k, pk(k), verify(sign(x, k), pk(k)) \rightarrow x$
One-way garbling function:
Constructors: garbling of x with the key $k, garble(x, k)$
Evaluation function:
Constructors: evaluation function of garbling of variables x, y, and z with the key k, $evaluate(garble(x, k), garble(y, k), garble(z, k))$
Commitment:
Constructors: committing x with a fresh nonce n $k, commit(x, n)$

Figure 8: Constructors and destructors

Protocol modelling and properties verification

In this section we model the *Qese* protocol depicted in Figures 4, then verify the fairness property.

$- \mathsf{P_B}(sk_\mathsf{B}, pk_\mathsf{B}, m_\mathsf{B}) = !in(c, m).(newb)$
$event(e_1(commit(m_\mathsf{B}, b))).$
$out(c, commit(m_\mathsf{B}, b)).in(c, m'').let((x_\mathsf{B}, x_f, x_\mathsf{CSP}, m_x) =$
$adec(m'', sk_\mathsf{B}))inif m_x = commit(m_\mathsf{B}, b)then$
$event(e_\mathsf{B}(commit(m_\mathsf{B}, b), x_\mathsf{B}, x_f, x_\mathsf{CSP}, evaluate(x_\mathsf{B}, x_f, x_\mathsf{CSP}))).$
$out(c, evaluate(x_\mathsf{B}, x_f, x_\mathsf{CSP}))$
$- \mathsf{P_{CSP}}(pk_\mathsf{B}, m_f, m_\mathsf{CSP}) = in(c, m').let((y_\mathsf{B} = garble(m', k))|$
$(y_f = garble(m_f, k))|(y_\mathsf{CSP} = garble(m_\mathsf{CSP}, k)))in$
$event(e_2(m', y_\mathsf{B}, y_f, y_\mathsf{CSP})).$
$out(c, (senc((y_\mathsf{B}, y_f, y_\mathsf{CSP}, m'), sk_\mathsf{CSP}))$
$.in(c, m''')$
$if m''' = evaluate(y_\mathsf{B}, y_f, y_\mathsf{CSP})then$
$event(e_\mathsf{CSP}(m', y_\mathsf{B}, y_f, y_\mathsf{CSP}, m'''))$
$- \mathsf{P}(newm_f)(newm_\mathsf{CSP})(newm_\mathsf{B})(newsk_\mathsf{B})letpk_\mathsf{B} = pk_{sk_\mathsf{B}}in$
$out(c, pk_\mathsf{B}).\mathsf{P_B}(sk_\mathsf{B}, pk_\mathsf{B}, m_\mathsf{B})|\mathsf{P_{CSP}}(pk_\mathsf{B}, m_f, m_\mathsf{CSP})$

The channel c is public so that the adversary can send, replay and get any messages sent over it. We use a single public channel and not two or more channels because the adversary could take a message from one channel and relay it on another channel, thus removing any difference between the channels. The process P begins with the creation of the secret and public keys of B, and the creation of messages $m_f, m_\mathsf{CSP}, m_\mathsf{B}$ The public key is output on channel c to model that the adversary has it in its initial knowledge. Then the protocol itself starts: $\mathsf{P_B}$ represents B, $\mathsf{P_{CSP}}$ represents the CSP. Both principals can run an unbounded number of sessions, so $\mathsf{P_B}$ and $\mathsf{P_{CSP}}$ start with replications.

We consider that B first inputs a message containing the encrypted tokens and then starts the protocol run by choosing a nonce b, and executing the event $e_1(commit(m_\mathsf{B}, b))$, where m_B is initially added to the B knowledge. Intuitively, this event records that B sent $Message_1$ of the

$(Nill)$ $\mathcal{E}, \mathcal{P} \cup \{0\} \rightarrow \mathcal{E}, \mathcal{P}$

$(Repl)$ $\mathcal{E}, \mathcal{P} \cup \{!\mathsf{P}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, !\mathsf{P}\}$

(Par) $\mathcal{E}, \mathcal{P} \cup \{\mathsf{P}|\mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, \mathsf{Q}\}$

(Par) $\mathcal{E}, \mathcal{P} \cup \{\mathsf{P}|\mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, \mathsf{Q}\}$

(New) $\mathcal{E}, \mathcal{P} \cup \{(newa)\mathsf{P}\} \rightarrow \mathcal{E} \cup \{a'\}, \mathcal{P} \cup \{\mathsf{P}\{a'/a\}\}$ where $a' \notin \mathcal{E}$

(I/O) $\mathcal{E}, \mathcal{P} \cup \{out(c, M).\mathsf{Q}, in(c, x).\mathsf{P}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{Q}, \mathsf{P}\{M/x\}\}$

$(Cond1)$ $\mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } \mathsf{P} \text{ else } \mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}\}$ if $M = N$

$(Cond2)$ $\mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } \mathsf{P} \text{ else } \mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{Q}\}$ if $M \neq N$

(Let) $\mathcal{E}, \mathcal{P} \cup \{\text{let } x = g(M_1, \ldots, M_n) \text{ in } \mathsf{P} \text{ else } \mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}\{M'/x\}\}$
if $g(M_1, \ldots, M_n) \rightarrow M'$

Figure 9: Operational semantics [14]

protocol. Event e_1 is placed before the actual output of $Message_1$; this is necessary for the desired correspondences to hold: if event e_1 followed the output of $Message_1$, we would not be able to prove that event e_1 must have been executed, even though $Message_1$ must have been sent, because $Message_1$ could be sent without executing event e_1, as stated in [13]. The situation is similar for events e_2, e_{B} and e_{CSP}.

Next, B receives the garbling of CSP's inputs as well as the garbling of the committed messages encrypted with its public key. B decrypts the message using its secret key sk_{B}. If decryption succeeds B checks if the message has the right form using the pattern-matching construct $let((x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}}, = m_{\mathsf{B}}) = adec(m'', sk_{\mathsf{B}}))in$. Then B executes the event $e_{\mathsf{B}}(commit(m_{\mathsf{B}}, b), x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}},$
$evaluate(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}}))$, to record that it has received $Message_2$ and sent $Message_3$ of the protocol. Finally, B sends the last message of the protocol $evaluate(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}})$.

After sending this message, B executes some actions needed only for specifying properties of the protocol. When the received message $m_x = commit(m_{\mathsf{B}}, b)$, that is, when the session is between B and CSP, B executes the event $e_{\mathsf{B}}(commit(m_{\mathsf{B}}, b), x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}},$
$evaluate(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}}))$, to record that B ended a session of the protocol, with the participant (CSP), which is authenticated using the authentication key. B also outputs the evaluation function output $evaluate(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}})$.

The process $\mathsf{P}_{\mathsf{CSP}}$ proceeds similarly: it executes the protocol, with the additional event $e_2(m', y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}})$ to record that $Message_1$ has been received and $Message_2$ has been sent by CSP, in a session with the participant of public key pk_{B} and the received message m'. After finishing the protocol itself, when $m''' = evaluate(y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}})$, that is, when the session is between B and CSP, $\mathsf{P}_{\mathsf{CSP}}$ executes the event $e_{\mathsf{CSP}}(m', y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}}, m''')$, to record that CSP finished the protocol, and outputs m'''.

The events will be used in order to formalize fairness. For example, we formalize that, if CSP ends a session of the protocol $e_{\mathsf{CSP}}(m', y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}}, m''')$, then (a) B has

started a session of the protocol by committing m_{B} with the nonce n_{B}, and (b) CSP outputs the evaluation function $evaluate(y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}})$. Furthermore, B ends a session of the protocol, then (a) CSP has already garbled the B's committed input message, and (b) B outputs the evaluation function $evaluate(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}})$.

Next, we formally define the correspondences in order to verify the fairness property. We prove correspondences in the form of *if an event e has been executed, then events $e_1, \ldots, e_m$ have been executed*. These events may include arguments, which allows one to relate the values of variables at the various events. We can prove that each execution of e corresponds to a distinct execution of some events, and that the events have been executed in a certain order. We assume that the protocol is executed in the presence of an adversary that can listen to all messages, compute, and send all messages it has, following the so-called Dolev-Yao model [24]. Thus, an adversary can be represented by any process that has a set of public names $Init$ in its initial knowledge and that does not contain events.

As presented in system model, the correspondence event $e_{\mathsf{CSP}}(x_1, x_2, x_3, x_4, x_5) \rightsquigarrow e_1(x_1) \wedge e_2(x_1, x_2, x_3, x_4) \wedge e_{\mathsf{B}}(x_1, x_2, x_3, x_4, x_5)$ means that, if the event $e_{\mathsf{CSP}}(x_1, x_2, x_3, x_4, x_5)$ has been executed, then the events $e_1(x_1), e_2(x_1, x_2, x_3, x_4)$ and $e_{\mathsf{B}}(x_1, x_2, x_3, x_4, x_5)$ have been executed, with the same value of the arguments x_1, x_2, x_3, x_4, x_5.

Table 4: Notation and Preliminaries

Term	**Definition**
o	Service Level Objective (SLO)
C	Customer
B	Broker
m	Plaintext
c	Ciphertext
k	Symmetric encryption/decryption key.
pk_X	Public key of party X.
sk_X	Private key of party X.
$cert_\mathsf{X}$	X.509 certificate of party X.
$\mathsf{k}_\mathsf{X}^{auth}$	Authentication secret generated and sent to party X.
$N_\mathsf{X} \in \{0,1\}^\beta$	Random number generated by party X from the set of all binary strings of length β.
$y \leftarrow\!\!\$\ \mathsf{A}(x)$	On input x, algorithm A binds the output to variable y.
$m_1 \| \| m_2$	Concatenation of m_1 and m_2.
KGen	key generation.
Enc	Encryption algorithm
Dec	Decryption algorithm
$c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m)$	Takes as input a public key pk, a plaintext m, and outputs a ciphertext c.
$m \leftarrow\!\!\$\ \mathsf{Dec}(\mathsf{sk}, c)$	Takes as input a secret key sk, a ciphertext c, and outputs a plaintext m.
$s_m \leftarrow\!\!\$\ \mathsf{Sig}(\mathsf{sk}, m)$	Signs m with a secret key sk, and outputs a signed message s_m.
$b \leftarrow\!\!\$\ \mathsf{Verify}(\mathsf{pk}, s_m)$	Verifies the signature using the public key pk and returns 1 if the signature is valid for the given message and 0 if not.
$y \leftarrow\!\!\$\ \mathsf{H}(x)$	Hash function which takes as input x, and responds with a random outputs y. Ideally, a hash function is defined as a random oracle that responds with a random output $y \in \mathcal{Y}$ to each given input for $x \in \mathsf{X}$. Where X is the set of possible messages, $\mathcal{Y}$ is a finite set of possible digests.
$t^1, \ldots, t^n$	The SLA XML file can be split into n substrings. For example, the substrings generated from Listing 1.1 are: "$level3\|\|3$" and "$level2\|\|4$" (cf., Section 4)
$t^1_{\mathsf{CSP}_1}, \ldots, t^n_{\mathsf{CSP}_1}$	Substrings generated by CSP_1 and are named tokens, where n is the number of tokens
$w^1_\mathsf{C}, \ldots, w^n_\mathsf{C}$	Substrings generated by the customer C and are named keywords, where n is the number of keywords