Proof of Encryption: Enforcement of Security Service Level Agreement for Encryption Outsourcing

Sultan Alasmari
SIS Department
UNC Charlotte
Charlotte, NC US
salasma1@uncc.edu

Weichao Wang
SIS Department
UNC Charlotte
Charlotte, NC US
wwang22@uncc.edu

Tuanfa Qin
Multimedia Key Lab
Guangxi University
Nanning, China
tfqin@gxu.edu.cn

Yu Wang
CIS Department
Temple University
Philadelphia, PA
wangyu@temple.edu

***Abstract*—With the popularity of cloud and edge computing, various types of service level agreement (SLA) are used to enforce the amount of resources that service providers commit to end users. However, the security SLAs (SSLA) are usually hard to verify since their execution results are related to the intensity and frequency of cyber attacks. In this paper, we investigate the proof of encryption problem. Restricted by available resources and hardware capabilities, some end users need to outsource the strong encryption operations of data to external service providers. Verification of the correct execution of encryption is a problem since it is directly related to data safety. We first define the expected properties of the Proof of Encryption (PoE) mechanisms. We propose two mechanisms based on whether or not the service requester is aware of the encryption key. They allow an end user to verify that corresponding encryption algorithm and key strength are honored. We describe the details of the verification procedures. Simulation and experiments show that our approaches can detect a dishonest service provider with high probability.**

I. Introduction

With the fast development and deployment of cloud and edge computing, a large portion of data processing and storage operations are actually outsourced to various types of service providers. For example, according to Forbes, about 77% of enterprises have at least one application or a portion of their enterprise computing infrastructure in the cloud. Service providers and cloud customers often use Service Level Agreement (SLA) to determine the committed resources or responsibilities [1]. Since service providers often charge end users based on the amount of resources that they use (e.g. CPU cycles, network bandwidth, and memory), verification mechanisms have been designed so that claims from service providers can be verified [2]–[4].

In parallel to the proliferation of cloud computing is Security-as-a-Service (SaaS) [5], [6]. Here end users depend on third parties to scan their network traffic, hard drive, or even execution states in systems to detect vulnerabilities and defend against on-going attacks. However, SaaS faces a serious challenge, which is the verification of services that are actually delivered. While the resources in a traditional SLA such as CPU cycles can be roughly audited based on the computation speed [7], the execution of security SLAs cannot be verified based on only the detection results.

Below we describe an example. Assuming that an end user out-sources data storage to a service provider. The Security SLA requires that data must be encrypted by a symmetric encryption algorithm not weaker than AES-256. Restricted by available computation resources and remaining energy, the data source cannot conduct encryption by itself. Therefore, it needs to outsource the operations. However, the service providers have motivations to use a weaker encryption algorithm (or even store just plain text) to reduce the computation cost and electricity bill. Please note that this challenge is different from the Proof-of-Retrievablity [8] problem since we care about the format in which data is stored instead of whether or not the provider has it. We need a new approach to solve this problem.

In this paper, we propose to design mechanisms to achieve proof of encryption (PoE). We assume that the end customer has relatively weak computation resources and cannot accomplish all encryption operations by itself (it may or may not have resources to encrypt selected data blocks). The service providers will encrypt data with a predetermined algorithm and key strength. To prevent a service provider from cheating, the end user needs to verify some of the encryption results. There are several expected properties, such as randomness during data selection and configurable overhead, of the PoE mechanisms. We propose two mechanisms based on whether or not the service requester (aka end user) knows the encryption key. Through both analysis and experiments, we evaluate the proposed approaches on detection probability, overhead, and robustness to false results. Our evaluation shows that the proposed approaches allow end users to achieve proof-of-encryption with high probability and configurable overhead.

The contributions of the paper can be summarized as follows. Firstly, we identify that compared to other types of SLAs, it is fairly difficult to verify the execution of security SLAs. As a special example, we discuss the problem of proof of encryption. Secondly, we define the expected properties of PoE mechanisms. Thirdly, we design two mechanisms for PoE that allow end users to verify the encryption operations by service providers. Last but not least, we evaluate the proposed approaches through both analysis and experiments.

The remainder of the paper is organized as follows. In

978-1-7281-2319-6/19/$31.00 ©2019 IEEE

Section II, we describe related work that we can benefit from. In Section III, we first define expected properties of proof-of-encryption mechanisms. We then present the details of the verification mechanisms. They suit different scenarios based on whether or not the end user knows the encryption key. We identify different parameters that can impact the detection of a dishonest service provider and analyze the detection probability. Section IV presents the quantitative evaluation results. Finally, Section V concludes the paper.

II. Related Work

Since many users of cloud computing are middle size or small size business, they cannot afford a special security team for the companies. Therefore, outsourcing the security services becomes a natural choice. Several efforts have been made to pave the way for auditing of the services. For example, in [9], the authors propose to firstly map security SLAs to Service Level Objects (SLO). Cloud service providers (instead of end customers) need to assess the fulfillment of the SLOs. Similarly, in Cloud Trust Protocol (CTP) [10], the cloud providers build an open API set to enable end customers to query providers about the security level of their services. Another factor that the authors in [9] identify for security SLA enforcement is standardization. From this perspective, ISO/IEC 19086 [11] and EU's General Data Protection Regulation (GDPR) [12] could become pioneers. Cloud companies often pay close attention to data storage encryption. For example, Concentra Health agreed to a settlement of $1.7 million for failing to meet industry-standard data encryption expectations [13].

Another thread of research in this domain is the establishment of ontology for security feature understanding [14]. This scheme provides a formal framework for representing knowledge for potential logic inference. While the approach allows end users to understand the security agreements with a provider, its auditing efforts focus on compliance with federal regulations, instead of fulfillment of security operations. Based on the analysis, we can see that there is not much effort in enforcement of security SLAs in cloud computing environments, especially for the fulfillment by cloud providers. More research is needed to solve this problem.

III. Proof of Encryption: Proposed Approaches

In this section, we will present the details of the proposed approaches. We will first analyze the expected properties of a Proof-of-Encryption mechanism. We will then describe the working procedures of the protocols. We will also analyze the detection capability of the proposed approaches.

A. Expected Properties of Proof-of-Encryption Algorithms

Randomness: The goal of the algorithm is to prevent a service provider from violating the SSLA by selectively encrypting only a portion of data, using a weaker encryption algorithm, or skipping encryption at all. To avoid incurring too much overhead at the end user, it can only verify encryption results of a small portion of data. Therefore, the selection procedure must be robust against pre-computation or guessing attacks. In other words, a service provider cannot predict the segments of data that the requester will challenge.

Robust against encryption on the fly attack: Since the requester will challenge the service provider and the response must be calculated based on both the challenge and the data encryption results, the approach must be robust against encryption on the fly attacks. In other words, the requester must be able to identify whether or not the encryption operation is conducted only after the provider receives the challenge.

Configurable overhead: Since a service requester usually has very limited computation and communication resources, the proposed approach needs to support configurable overhead. Quantifiable analysis needs to be provided to maintain balance between detection accuracy/capability and its commitment to resources for verification.

Support changes to data: This property is directly related to the data applications. For example, if a small portion of data is updated, the PoE operations should not be heavily impacted. Otherwise, the adoption of the approach will be greatly restricted.

From the description above, we can see that the first two properties are used to prevent the service provider from cheating. The last two properties focus on the obstacles of adoption. These properties are not bound to any specific encryption algorithms or cloud architectures. Therefore, the proposed approach can be applied to various scenarios.

B. System Assumptions

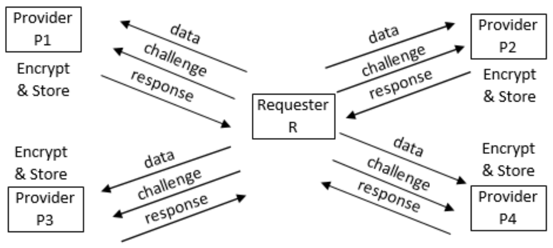

Fig. 1. Application scenarios of PoE approaches.

Figure 1 shows the application scenario of the proposed approaches. We assume that a service requester R will outsource the storage of data to multiple providers P_i, $i = 1 \cdots s$. Here the providers P_i may belong to the same owner or multiple owners. Restricted by available computation resources, R cannot afford to encrypting all data with a strong algorithm by itself. It has to outsource the encryption operations to P_i. Please note that R has concerns on data confidentiality for attackers who can compromise the storage devices of P_i. Therefore, it needs to verify that P_i actually accomplishes the encryption of all data with the agreed algorithm and key strength as defined in SSLA. To achieve that, R will generate random challenges and P_i has to respond to them based on the challenges and encryption results.

Since the verification of data encryption happens between the service providers and data owners, we need to differentiate between two cases: whether or not the service requester R has enough information (about the key) and hardware/software resources to encrypt selected data blocks by itself. In the first case, if the requester R knows the encryption key used by P_i and has the CPU power to encrypt some data with the key (it may not be power efficient for R to encrypt all data), the PoE problem becomes straightforward. The two parties R and P_i need to verify that: (a) they know the same information (the encryption results), and (b) the information is not generated on the fly. A variation of the real-time data flow verification scheme [15] can be adopted to achieve the goal.

However, in real life under many cases, the storage providers will not share the encryption keys with the requester. Therefore, even if R has a powerful CPU and corresponding resources, it cannot verify the cipher-text directly. We must design a different approach to the problem. In this paper, we propose a cross-comparison based scheme to detect a dishonest service provider.

The attacker is a dishonest service provider that tries to save encryption operations through violating the SSLA. It may have a super fast computer that can encrypt data on the fly. However, to allow such attacks, the attacker must be able to transmit data from storage to the computer through a link with very short delay. Similarly, if multiple service providers are involved, they may try to collude to cheat the requester.

The following table summarizes the symbols.

R	service requester (end user)
P_i	service providers
$h(x)$	secure hash function known to both parties
m_j	jth data message sent by R to P
L	each data chunk contains L data blocks
t	length of specific patterns in encryption results to trigger verification operation
q_j	the probability that Provider P_j skips encryption of a data block
$p_{j,k}$	the percentage of data blocks provided to P_j that are also provided to P_k

C. Proposed Mechanism for Scenario 1

In scenario 1, we assume that both the service provider P_i and the requester R know the encryption key. At the same time, R has the software/hardware resources to accomplish encryption on a selected group of data blocks. Therefore, the requester can randomly select a group of data blocks and challenge P_i for encryption results. In this way, the property of "randomness" is satisfied automatically. At the same time, since R can determine the amount of data that it will challenge P_i with, it can control the committed resources.

The major challenge is to prevent P_i from encrypting the data blocks after it receives the request (encryption on the fly). To differentiate the situation in which encryption is pre-accomplished from the case in which data is encrypted after the request is received, we can measure the network delay between the request is submitted and the response is received. However, a dishonest provider may choose to hide the computation time in the network transmission delay. For example, based on BearSSL [16], on an Intel Xeon CPU 3.1GHz, AES-256 CBC mode can encrypt about 125MB/sec. Based on this measurement result, for a 1KB data block, the service provider needs only 8μsec to accomplish the encryption. It can easily hide the delay under the round trip communication time of tens of millisecond between R and P_i.

To detect this cheating behavior, we propose the following approach. Assume that the data blocks that P_i needs to encrypt are represented as m_j, j = 1 to n. These data blocks are divided into chunks with the size L. For example, m_1 to m_L form Chunk 1, m_{L+1} to m_{2L} form Chunk 2, so long and so force. The corresponding cipher-text blocks are represented as c_1, c_2, $\cdots$, c_n. Within each chunk, CBC (cipher block chaining) mode is used for data encryption. The initial vector (IV) for encryption can be jointly determined by R and P_i. In this way, we guarantee that within a chunk, multiple blocks cannot be encrypted in parallel.

When the requester R wants to verify that proper encryption operation has been applied to the data blocks, it will randomly choose one block m_{aL+j} ($a = 0 \cdots \lfloor \frac{n}{L} \rfloor$) in a chunk and provide a random number r as the challenge. The service provider P_i needs to return the MAC (message authentication) code $hash(r, hash(r, c_{(a+1)L}, c_{aL+j}, c_{aL+1}))$. We can see that the hash result covers the first block, the last block, and requested block in the chunk.

If P_i has properly encrypted all data blocks and stored them, it just needs to concatenate the corresponding cipher text and return the results. On the contrary, should P_i have violated the SSLA and stored the data in other formats, it has to re-encrypt the blocks with the secret key determined in the SSLA. Since CBC mode is adopted, it cannot use parallel computing to accelerate the procedure. It has to encrypt the blocks one by one. Since the hash result puts $c_{(a+1)L}$ as the first entry, the hash calculation cannot start until the encryption of the whole chunk is accomplished. This operation will drastically increase the response delay.

C.1 Analysis

The chunk size L provides a user configurable parameter that can impact the increases in response time should the service provider conduct encryption-on-the-fly attacks. The larger is L, the longer time P_i needs to encrypt the whole chunk. In this way, R can adjust the choice of L based on the network connection quality and speed between it and P_i. Please note that the goal of initial vector and chain mode is to prevent an attacker from conducting random encryption of the data. It is not bound to AES or any other specific encryption function as long as the overhead can be enforced.

Another manipulation P_i can play is to store only the cipher text of the first and last blocks of each chunk. This operation is not attractive to the service provider because of the following reasons. Firstly, to calculate and store the cipher-text of the last block in a chunk, it needs to accomplish the encryption of the whole chunk since CBC mode is used. From this point of view, even if P_i decides to store the data in other formats, the cost of encryption has already been paid. Secondly, since the challenge of verification covers the cipher text of c_{aL+j}, if P_i stores only c_{aL+1}, it needs to re-encrypt the data blocks to get the value of c_{aL+j}. The introduced delay could still be detected by the requester.

D. *Proposed Mechanism for Scenario 2*

In real life scenarios, very frequently the service provider of encryption and storage and the service requester will belong to different organizations. Therefore, they will not trust each other on encryption key generation and storage. For example, should the data confidentiality be compromised because of key disclosure, it will be very hard to determine accountability if multiple parties know the key. Therefore, more frequently we need to deal with scenario 2 in which the requester does not know the encryption key.

Since R does not know the encryption key, it cannot directly challenge P_i for data result. Therefore, we need a new approach to randomly select the data blocks and verify their encryption results. Without losing generality, we assume that R will choose three different service providers P_1, P_2, and P_3 to accomplish the encryption and storage tasks. Some of the data blocks that R sends to the three parties will overlap so that R can cross compare the encryption results. In addition to R and P_i, we also assume that a certificate authority (CA) exists in the system and all four parties trust it.

D.1 Overview of Approach

In this part, we will provide an overview of the proposed approach. More details will be provided in subsequent parts. Firstly, R will notify P_1, P_2, and P_3 that they are chosen for its encryption and storage task. Each of them will work with CA to get a new public/private key pair. Please note that this new key pair is associated with a pseudo ID of each party. In this way, through looking at only the public keys, they cannot figure out the real identities of the service providers.

After getting the public/private key pairs, the service providers will run a multi-party Diffie-Hellman key generation protocol [17] with the help from R. Again all communications among the providers will go through R. In this way, they cannot figure out the real identities of each other. At the same time, the authenticity of exchanged information is protected by digital signatures with the new private keys.

Once the key generation procedure is accomplished, R can distribute different chunks of data to different providers. Since the providers know only there are other service providers but not their identities, they cannot collude to manipulate the encryption results. Since the requester R does not know the encryption key, it cannot directly verify the encryption results. However, it can cross compare the encryption results from multiple parties since they use the same key. R can either challenge different service providers with the same data blocks or use another method to randomly choose data blocks. An overview of the approach is shown in Figure 2.

D.2 Detailed Description

Step 1 : Generation of Encryption Key

As shown in Figure 2, R decides to choose P_1, P_2, and P_3 to help it encrypt data. It will notify them and also provide the identity of the CA. Now each provider will contact CA and acquire a new public/private key pair for its temporary

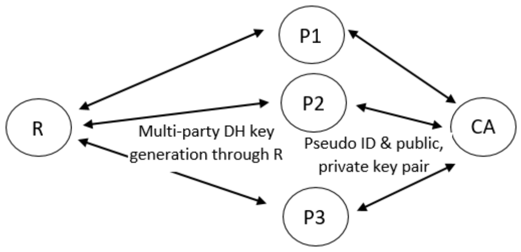
P1
P2
P3
R
CA
Multi-party DH key generation through R
Pseudo ID & public, private key pair

Fig. 2. Overview of the approach for scenario 2.

identity P_1', P_2', and P_3'. Please note that the temporary IDs are not linked to their real IDs. Once receiving the new public key certificates, P_1, P_2, and P_3 will share them with R. R will then distribute the certificates to all providers. At this time, only R and CA know the real IDs of the providers.

R will choose a finite cyclic group G of order w and a generating element g in G. R will distribute the generator g and the large prime number P to all three providers. They will then run the multi-party Diffie-Hellman protocol. Specifically, each provider will generate its own random number $r_{P_i'}$, calculate $g^{r_{P_i'}}$ mod P, and send it to R. The integrity of the information will be protected by the digital signature associated with the new temporary ID.

As shown in Figure 2, all providers will send their digitally signed shares to R. R will then forward them to other parties that have not added their shares. Throughout the procedure, the requester R serves as the center of communication and transmits packets between the providers. In this way, a provider cannot learn the identities of other providers and collude with them. At the end of the key generation procedure, each provider will learn the group key $g^{r_{P_1'} \times r_{P_2'} \times r_{P_3'}}$ mod P. The key is protected by three digital signatures from the providers. The requester R, although passes all traffic, cannot learn the group secret.

Step 2 : Selection of Challenged Data Blocks

In this part, we will introduce two methods to select data blocks for encryption result challenge. In the first method, R will send the same group of data blocks to two or even more encryption providers. These blocks will blend into other groups of blocks that are sent to them. Since the providers do not know the identities of each other, they cannot conduct off-line collusion. After confirming that the data blocks have been encrypted, R will challenge two or more parties with the same group of data blocks for encryption results. While the challenge procedure could be similar to that of scenario 1, R needs to measure two parameters. Firstly, it needs to measure the delay between the request is sent and the result is received. This is to prevent P_i from conducting encryption-on-the-fly attacks. Secondly, R needs to cross compare the encryption results of the same group of blocks from different providers. If a provider has chosen to violate the SSLA and uses other encryption algorithms or weaker keys, its encryption results will be different from those of others. Therefore, R will be able to verify the encryption results even if it does not know the key.

Letting R select data blocks for challenge, although achieving the randomness property, has several limitations. Firstly, the distribution of the verified data blocks is very unbalanced. If a group of blocks are chosen, thousands of consecutive blocks will be verified. On the contrary, if a group of blocks are not selected, non of them will be verified. Secondly, the data access delay could vary a lot depending on the storage medium. For example, after encryption, a provider may move the data to external hard drive, or under an extreme condition, a tape drive. Therefore, when a challenge is received, the provider may need an extended period of time to get data back into memory. This may lead to increases in false alarms.

To solve these problems, we need to design a mechanism that selects data blocks for challenges as the encryption algorithm is running. In this way, the data has not been moved to the external devices. At the same time, the selection procedures cannot be controlled by service providers. A potential solution is the criteria that have been used for mining in BlockChains. Assume that the requester R and provider P_i have determined the encryption algorithm, encryption mode, and key strength. R will now choose a pattern in the encryption result of a data block. If the result satisfies the pattern (e.g. the first t bits are all '0'), P_i needs to report the index of the block to R. This mechanism has the following advantages.

Firstly, Randomness of Selected Data for Verification

Since the criteria are applied to the encryption results, the provider P_i cannot predict which blocks to encrypt beforehand. The usage of initial vector (IV) prevents P_i from predicting the encryption results based on similarity of data blocks. From this point of view, the selection procedure is random. Even the requester R cannot predict which blocks will be selected. A provider may choose to lie about the encryption results or the index of the blocks. However, there are other providers and it does not know to which provider which blocks are shared. Therefore, should it choose to lie, it will face the risk that the same block may be encrypted by another provider.

Secondly, Configurable Overhead

Through adjusting the pattern criteria, R can control the probability that any block is challenged. For example, we can approximate any probability p with the sum of a group of negative powers of 2:

$$p = a_0 \times \frac{1}{2^0} + a_1 \times \frac{1}{2^1} + a_2 \times \frac{1}{2^2} \cdots + a_l \times \frac{1}{2^l} \cdots ; \quad (1)$$

Here the probability p has the value between [0, 1], and the value of a_i is either 0 or 1, depending on the probability we want to approximate. The accuracy of approximation is determined by the length of the encryption results.

E. Analysis of Detection Accuracy and Overhead

In this part, we will analyze the detection capability and overhead of the proposed approaches. We will discuss two scenarios when the provider P_i is benign or malicious, respectively. If P_i is benign, it will encrypt all data blocks based on the SSLA. Therefore, the only factors that impact the frequency of data verifications are the choices of encryption algorithm, the initial vector, the key, and the data blocks.

Without losing generality, we assume that when the first t bits of the encryption results are all '0', it will trigger the verification procedure. For each block, the probability that it does not trigger verification is $(1 - \frac{1}{2^t})$. Here we propose to divide data into windows with the size of L consecutive blocks. If at least one block in a window satisfies the criteria, we say that the whole window is verified. Therefore, the probability that no block in v_t consecutive windows triggers verification is $(1 - \frac{1}{2^t})^{v_t \times L}$.

Here we will follow the adjustment practice of blockchain mining. To prevent P_i from cheating, we require that if v_t windows of consecutive blocks do not trigger any verification operations, we will reduce t by 1 bit. In other words, we double the probability that a block satisfies the verification criteria. Similarly, if u_t consecutive windows have all triggered the verification procedure, we will increase t by 1 so that fewer blocks will be verified by R. Therefore, the requester R can adjust the choices of L, v_t, and u_t to control the probability that a block is verified.

If the provider P is malicious, it needs to find a trade-off between the percentage of data that it encrypts and the probability that it is detected by the requester. Assume that the provider has the probability q to not encrypt a data block. Since it does not encrypt the block, it will not report the index to the requester even if the encryption result could have triggered verification. Since for each block, the probability that it triggers verification is $\frac{1}{2^t}$, the probability that the provider misses the block is $q \times \frac{1}{2^t}$. Note that if another provider encrypts this block and discovers that the encryption results satisfy the criteria, it will report to the requester and this violation of SSLA will be detected. Therefore, the probability that in x consecutive blocks the malicious provider does not miss any blocks that could trigger verification is $(1 - \frac{q}{2^t})^x$.

In real life the situation will be more complicated. It is possible that two providers both skip the block. Therefore. even if it could have triggered verification, none will report to the requester. Let us assume that for Provider P_1 the parameters have the values (q_1, t_1) (P_1 has probability q_1 to not encrypt a block, and the trigger pattern covers the first t_1 bits.) Similarly, for provider P_2 the parameters have the values (q_2, t_2). Without losing generality, we assume that $t_1 \geq t_2$. In other words, if the encryption results satisfy the pattern of provider P_1, it will also satisfy the pattern of P_2.

For any data block that R provides to both P_1 and P_2, the scenarios in which a violation of SSLA by either party is detected can be described as follows: (1) P_1 chooses to encrypt the block but P_2 decides to skip it; or (2) P_2 chooses to encrypt it while P_1 skips it. Now let us examine their probability.

If P_1 encrypts the block and the results satisfy its pattern but P_2 skips the block, the probability is

$$(1 - q_1) \times q_2 \times \frac{1}{2^{t_1}}; \quad (2)$$

Here since $t_1 \geq t_2$, the encryption results will automatically satisfy the pattern of P_2. Under this case, P_2's violation of SSLA will be detected.

On the other hand, if P_2 encrypts the block and the results satisfy its pattern but P_1 skips the block, the probability is

$$q_1 \times (1 - q_2) \times \frac{1}{2^{t_2}}; \quad (3)$$

Here we face a problem. An encryption result that satisfies the pattern of P_2 may not satisfy the pattern of P_1. To differentiate between the two cases, we need the provider P_2 to send back not only the index of the block but also the encryption result. In this way, we can tell whether or not the violation of P_1 can be detected. So the probability that P_1's violation can be detected is:

$$q_1 \times (1 - q_2) \times \frac{1}{2^{t_1}}; \quad (4)$$

When we investigate the equations, we can learn the following information. Firstly, since the requester R does not know the encryption key, it can detect an SSLA violation only when the same data block is sent to more than one provider and their encryption results differ. Therefore, the higher percentage of blocks that are sent to multiple providers, the better chance to detect violations. If R can afford it, it should provide the same data blocks to as many providers as possible. This also supports the requirements on data randomness and the prevention of collusion between providers.

Secondly, although R could choose different patterns for different providers (e.g. t_1 '0's for provider P_1 and t_2 '1's for P_2), the analysis above shows that R has the best chance of detecting violations when the patterns of different providers are similar. At the same time, the detection probability is determined by the longer pattern between two providers. On one side, the longer is the pattern, the lower chance that the encryption result triggers verification. Therefore, the detection probability will be low. On the other side, the longer is the pattern, the lower communication volume between the providers and requester. Therefore, it makes sense to keep the pattern length of different providers close to each other. In this way, we reduce the communication overhead without impacting the detection capability.

Thirdly, we want to analyze the relationship between the detection capability and the value of q. In other words, what is the best strategy of a dishonest service provider to avoid detection. If we look at only Equation (2), for P_2 to avoid being detected, it needs to reduce the value of q_2, and increases the value of q_1. In other words, it needs to encrypt more data blocks and expects P_1 to skip more. Similarly, if P_1 wants to protect itself from being detected, it needs P_2 to increase q_2. If we jointly consider the sum of the two detection probabilities, we get $(q_1 + q_2 - 2 \times q_1 \times q_2) \times \frac{1}{2^{t_1}}$. Analysis shows that the joint detection probability has the double-saddle shape. The probability is low when both q_1 and q_2 are very close to 0 or 1. In other words, if we look at only P_1 and P_2, they can avoid detection by either encrypting all data blocks or encrypting none at all. For the latter option, however, they will be caught very soon since the trigger pattern will become very short (thus many blocks should trigger verification). If there is at least one honest service provider in the system, it will report data blocks satisfying the criteria and the dishonest ones will be caught as well.

F. Managing the Probability of Sharing Blocks

The analysis in Section III-E considers only the data blocks that have been submitted to both P_1 and P_2. In real applications, restricted by the cost, only a part of data blocks will be submitted to more than one encryption service provider, as shown in Figure 3. Without losing generality, we use the probability $p_{1,2}$ to represent the ratio between the number of blocks that are known to both P_1 and P_2 and those known to only P_1. Therefore, for the requester to detect an SSLA violation by P_1 through the report of P_2, the probability is

$$p_{1,2} \times q_1 \times (1 - q_2) \times \frac{1}{2^{t_1}}; \quad (5)$$

If P_1 skips the encryption of v data blocks, the probability that it is detected by R is

$$1 - (1 - p_{1,2} \times q_1 \times (1 - q_2) \times \frac{1}{2^{t_1}})^v; \quad (6)$$

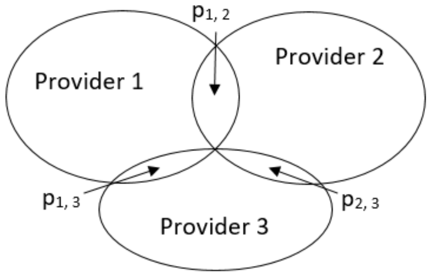

Fig. 3. Only data blocks submitted to multiple providers can be used for detection of SSLA violations.

IV. Quantitative Results

In this part, we will conduct experiments to collect quantitative results about the proposed approaches. The experiments include both data encryption and access operations on real networked devices and the simulation of the impacts of different parameters on the detection capability.

A. Detection of Encryption-On-the-Fly Attack

As we describe in Section III-C, if a dishonest service provider chooses to encrypt data blocks after receiving a verification request, it will introduce extensive delay into the response procedure. To verify that we can detect such attacks through measuring the network delay, we choose three could service providers and implement encryption functions on a virtual machine hosted by them. A client machine will issue a verification request. If the virtual machine has already accomplished data block encryption, it will reply with the hash results as described in Section III-C. Otherwise, if the virtual machine has not encrypted the data blocks, it will have to conduct encryption and hash in sequence. We choose two sites, one on-campus and one off-campus, as the client to conduct the experiments. The delays are measured in four consecutive days during both day time busy hours and late in the night. The data blocks that are challenged have the size of 1K Byte.

The window size is 10,000 blocks. Five groups of experiences are conducted and their average delay is shown in Figure 4.

From the figures, we can see that the on campus connections with the service providers are very stable. If the virtual machines have to encrypt the 10M Byte data on the fly, it will take somewhere between 80 to 100 ms. The increases in delay obviously deviate from the normal network conditions. A user will be able to detect such changes. We assume that a dishonest service provider will choose to use the customized environments such as AES-NI [18] to avoid detection. Based on [16], the AES 256 CBC mode encryption speed will increase about 4 times if AES-NI is adopted, which will bring the extra delay down to 20 to 25 ms. Such increases are still detectable compared to the normal measurement results.

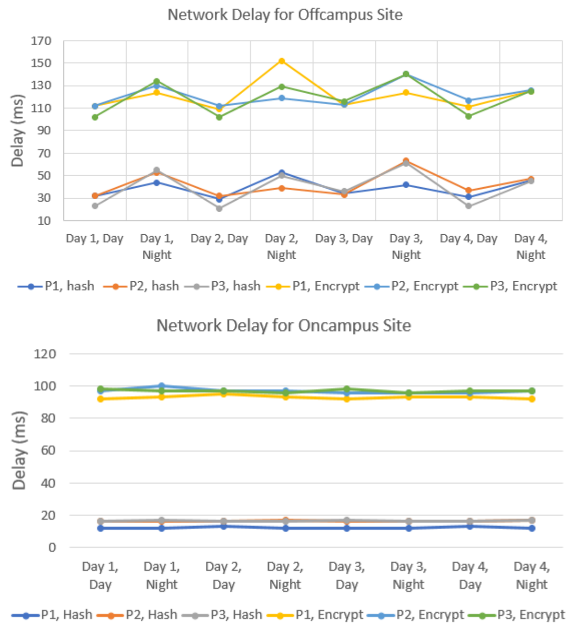

Fig. 4. Detection of on-the-fly encryption attacks through changes in network delay.

B. Cross-Comparison Detection

data sharing probability $p_{i,j}$ between two providers	ratio b/w data at each provider and overall data volume
0%	33.3%
10%	37%
20%	41.6%
30%	47.6%
40%	55.5%
50%	66.7%

In the second group of experiments, we will use simulation to investigate the impacts of different parameters on the detection capability of the requester when it does not have a copy of the encryption key. To simplify analysis, we assume that the requester chooses three encryption service providers. The probability $p_{i,j}$ represents the percentage of data blocks of provider P_i that are also known to provider P_j. Please note that as $p_{i,j}$ increases, the probability that a dishonest provider is detected also increases. As the price to pay for improved security, each provider has to encrypt more data blocks. The table above shows as $p_{i,j}$ increases from 10% to 50%, the changes of encryption load at each provider. We can see that when the sharing probability reaches about 30%, each provider needs to encrypt about half of overall data blocks.

The simulation results are shown in Figure 5. Here we experiment with the data block sharing probability between two providers $p_{i,j}$ = 10% and 20%. The probability that a provider skips encrypting a data block ranges from 5% to 15%. In other words, a service provider will randomly pick 1 to 3 data blocks to skip encryption in every 20 blocks. The length of the pattern in the encryption results to trigger data verification ranges from 7 bits to 13 bits. In other word, on average, one block in every 128 to 8192 data blocks could trigger a verification operation. In all six sub-figures, on the X-axis we show the number of data blocks that are provided to a provider and it needs to encrypt all of them. On the Y-axis, we show the probability that a dishonest provider is caught if it skips encryption of some data blocks. Please note that the number of blocks on X-axis is roughly in logarithmic scale.

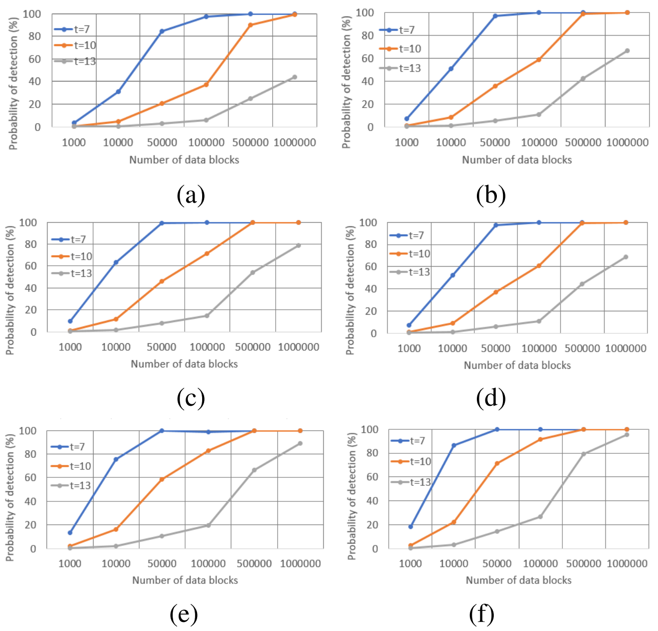

Fig. 5. Detection probability as the number of data blocks changes. (a) $p_{i,j}$=10%, q_i=5%; (b) $p_{i,j}$=10%, q_i=10%; (c) $p_{i,j}$=10%, q_i=15%; (d) $p_{i,j}$=20%, q_i=5%; (e) $p_{i,j}$=20%, q_i=10%; (f) $p_{i,j}$=20%, q_i=15%.

From the figures, we can see that as the number of data blocks increases, the probability that a dishonest service provider is caught increases very fast. When R provides only 10% of the data blocks of the provider P_1 to $P2$ and P_1 skips only 5% of the encryption operations, when the number of data blocks reaches 500,000, P_1 has 20% chance to be caught. Please note that if we choose the data block size to be 1K Byte, that is only 500M Byte data that R sends to P_1. Considering the amount of data that is uploaded to the cloud everyday, this is a very small number. The service providers do not have a strong motivation to cheat when they are aware of the high probability of detection.

Another feature we can see from the simulation results is that the length of the pattern in encryption results to trigger verification has a large impact on the detection capability. When we use a longer pattern, the probability that a data block will trigger verification decreases exponentially. An end user needs to maintain balance between the communication overhead and the detection capability when it outsources the encryption operations.

C. Discussion on Security of Approaches

In this part, we will discuss the safety of the approaches. We are especially interested in the following aspects.

Frame an honest service provider

In Scenario 2 of the proposed approaches, the requester R does not know the encryption key. Therefore, it can detect discrepancy between multiple service providers only through cross-comparison of the encryption results. It is possible for one service provider to falsely report a data block satisfying the pattern to frame another competitor. For example, the provider P_2 can randomly select a data block and report to R that it satisfies the trigger condition of verification (e.g., the first t_2 bits of the encryption results are '0'). If this block is provided only to P_2, no other providers can verify it and R has to take it as is. However, if the same block is also provided to P_1 and it does not report it, R must identify who is not telling the truth. To defend against such attacks, one way R can adopt is to provide the same group of blocks to the third encryption server P_3 and wait for the feedback to determine who is the attacker.

Robustness against off-line collusion

Although the proposed approach tries to hide the real identities of the service providers during key generation, some of them may still have off-line agreement to cover up for each other. For example, P_2 and P_3 may exchange the hash results of an encryption key to determine whether or not they are assigned to the same task. If the answer is 'yes', they can then identify the data blocks shared between them. However, the knowledge that the colluding parties can learn is limited. For example, if there is at least one provider that does not collude with them, they face the same probability of detection as we analyze in Section III-E. To hide the information on data distribution among different encryption providers, R can involve more parties during the key generation procedures. After the key is determined, it will provide data blocks to only a subset of the providers. In this way, a dishonest provider cannot derive out the number of encryption servers solely based on the key generation procedures.

V. Conclusion

In this paper we investigate the problem of proof of encryption. Specifically, when a data source asks multiple providers to encrypt its data with specified algorithms and key strength, there must be a way for it to verify the execution of the SSLA. We discuss two scenarios based on whether or not the requester knows the encryption key. The designed approaches can detect the violations of SSLA on encryption. We conduct both experiments and analysis to investigate the mechanisms and their probability to detect a dishonest service provider.

When we put the research problems of the paper in a bigger view, the goal is to allow end users to verify the execution of security service level agreement (SSLA). Different from the SLAs that focus on resource usage aspects (e.g. CPU cycles, available bandwidth), security related SLAs are harder to verify since there is not always a security incident available for detection. We plan to design a comprehensive suite of mechanisms to cover more features in security enforcements. We are especially interested in the services such as malware scanning and effectiveness of firewalls. The completeness of the approaches can be assessed with an ontology framework.

References

[1] N. Sfondrini, G. Motta, and L. You, "Service level agreement (sla) in public cloud environments: A survey on the current enterprises adoption," in *International Conference on Information Science and Technology (ICIST)*, 2015, pp. 181–185.

[2] S. Akter and M. Whaiduzzaman, "Dynamic service level agreement verification in cloud computing," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 15, no. 9, 2017.

[3] E. Giachino, S. de Gouw, C. Laneve, and B. Nobakht, "Statically and dynamically verifiable sla metrics," in *Theory and Practice of Formal Methods. Lecture Notes in Computer Science, vol 9660*, E. Abraham, M. Bonsangue, and E. Johnsen, Eds. Springer, Cham, 2016.

[4] M. Krotsiani and C. K. a. G. Spanoudakis, "Validation of service level agreements using probabilistic model checking," in *IEEE International Conference on Services Computing (SCC)*, 2017, pp. 148–155.

[5] Y. Khettab, M. Bagaa, D. L. C. Dutra, T. Taleb, and N. Toumi, "Virtual security as a service for 5g verticals," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1–6.

[6] Y. Sun, S. Nanda, and T. Jaeger, "Security-as-a-service for microservices-based cloud applications," in *IEEE International Conference on Cloud Computing Technology and Science*, 2015, pp. 50–57.

[7] Z. Zhou, H. Zhang, X. Yu, and J. Guo, "Audit meets game theory: Verifying reliable execution of sla for compute-intensive program in cloud," in *IEEE International Conference on Communications (ICC)*, 2015, pp. 7456–7461.

[8] C. B. Tan, M. H. A. Hijazi, Y. Lim, and A. Gani, "A survey on proof of retrievability for cloud data integrity and availability: Cloud storage state-of-the-art, issues, solutions and future trends," *Journal of Network and Computer Applications*, vol. 110, pp. 75–86, 2018.

[9] J. Luna, N. Suri, M. Iorga, and A. Karmel, "Leveraging the potential of cloud security service-level agreements through standards," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 32–40, 2015.

[10] CloudTrust Protocol Working Group, "Cloudtrust protocol data model and api," Cloud Security Alliance, 2015.

[11] EC Cloud Select Industry Group (C-SIG), "Cloud service level agreement standardization guidelines," European Commission, 2014.

[12] Erkuden Rios and Eider Iturbe and Xabier Larrucea and Massimiliano Rak and etc., "Service level agreement-based gdpr compliance and security assurance in (multi)cloud-based systems," *IET Software*, 2019.

[13] HIPPA, "Encryption almost prevents humana data breach in wisconsin," HIPAA Journal, 2015.

[14] C. Lee, K. M. Kavi, R. A. Paul, and M. Gomathisankaran, "Ontology of secure service level agreement," in *IEEE International Symposium on High Assurance Systems Engineering*, 2015, pp. 166–172.

[15] D. Puthal, S. Nepal, R. Ranjan, and J. Chen, "Dlsef: A dynamic key-length-based efficient real-time security verification model for big data stream," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 2, pp. 51:1–51:24, 2016.

[16] T. Pornin, "Bearssl: A smaller ssl/tls library," https://bearssl.org, 2018.

[17] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater, "Provably authenticated group diffie-hellman key exchange," in *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001, pp. 255–264.

[18] S. Gueron, "Intel advanced encryption standard (intel aes) new instructions set," Intel Whitepaper, 323641-001, 2012.