computers

MDPI

Article

The Use of Reactive Programming in the Proposed Model for Cloud Security Controlled by ITSS

Dhuratë Hyseni [1], Nimete Piraj [2], Betim Çiço [3] and Isak Shabani [1,*]

[1] Faculty of Electrical and Computer Engineering, University of Prishtina, St. Bregu i Diellit p.n., 10000 Prishtina, Kosovo; dhurate.hyseni@uni-pr.edu
[2] Faculty of Computer Science, University of Prizren "Ukshin Hoti", St. Rruga e Shkronjave, nr. 1, 20000 Prizren, Kosovo; nimete.piraj@gmail.com
[3] Department of Computer Engineering, Epoka University, St. Tiranë-Rinas, 1032 Tirana, Albania; bcico@epoka.edu.al
* Correspondence: isak.shabani@uni-pr.edu

Abstract: Reactive programming is a popular paradigm that has been used as a new solution in our proposed model for security in the cloud. In this context, we have been able to reduce the execution time compared to our previous work for the model proposed in cloud security, where the control of security depending on the ITSS (IT security specialist) of a certain organization based on selecting options. Some of the difficulties we encountered in our previous work while using traditional programming were the coordination of parallel processes and the modification of real-time data. This study provides results for two methods of programming based on the solutions of the proposed model for cloud security, with the first method of traditional programming and the use of reactive programming as the most suitable solution in our case. While taking the measurements in this paper, we used the same algorithms, and we present comparative results between the first and second methods of programming. The results in the paper are presented in tables and graphs, which show that reactive programming in the proposed model for cloud security offers better results compared to traditional programming.

Keywords: reactive programming; cloud provider; reactive architecture; reactive extensions; encryption and decryption file; data sources; IT security specialist

Citation: Hyseni, D.; Piraj, N.; Çiço, B.; Shabani, I. The Use of Reactive Programming in the Proposed Model for Cloud Security Controlled by ITSS. *Computers* **2022**, *11*, 62. https://doi.org/10.3390/computers11050062

Academic Editors: Leandros Maglaras, Helge Janicke, Mohamed Amine Ferrag and Francisco J. Aparicio-Navarro

Received: 20 March 2022
Accepted: 22 April 2022
Published: 25 April 2022

1. Introduction

In cloud computing, security has been a challenge because a third party has access to our data, and the data should be trusted to a third party. Recent trends in cloud security have played an important role in attracting organizations/companies to deploy sensitive data on the cloud. In this paper, the IT Security Specialist, ITSS, is referred to as the person responsible for a company/institution who realizes the "configuration" of security for other users in the organization.

In this context, the proposed model offers different scenarios based on the level of sensitivity of data [1]. From another point of view, this increases the reliability of clients in cloud computing. This reliability will be increased by offering data security controls to end-user ITSS. Our model, shown in Figure 1, was proposed in [1] with the same philosophy for controlling security in the cloud, which is based on two objectives: the control of security depending on the ITSS of a certain organization and the possibility of selecting options based on different algorithms. The increase in requirements in programming for performing tasks simultaneously in traditional programming has encountered requirements that cannot be covered by the increase in resources, such as multi-processor processing. After all this, another method of programming is required that will function asynchronously; thus, in order to realize these requirements, we use reactive programming.

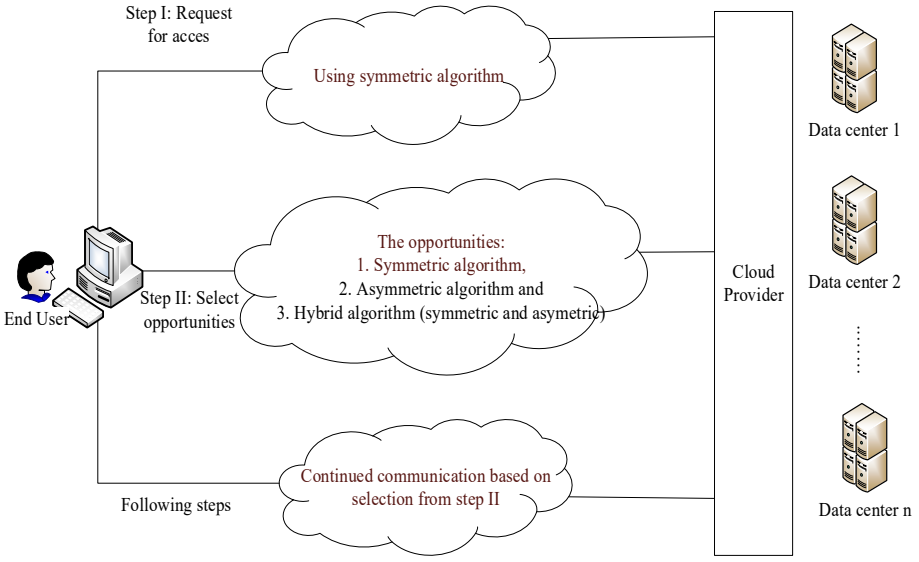

Figure 1. Proposed model for security in cloud computing controlled by the ITSS [1].

The focus of the study will be the evaluation of the effect of reactive programming on the proposed model for cloud security. The effect will be evaluated from the aspect of performance but also from the aspect of the application development process. The language that will be used for this paper is C #, as it is a well-established language with libraries that will facilitate the development, and our aim will be to compare the synchronous programming that developed the solution offered in [1] and the reactive one that is realized in this study.

There is a great deal of research that has been conducted on reactive programming [2–4] that shows the ideas of reactive programming from a theoretical point of view as well as the mathematical models that stand behind the reactive model, which are sometimes difficult concepts to understand, especially for those who have programmed only with traditional techniques. A programming paradigm refers to the programming style, which means building the structure and elements of the program. There are some characteristics that define a programming paradigm, such as modularity, objects, interruptions or events, control flow, etc. [5]. Everyone has an opinion as to which of the programming paradigms, or coding styles, is best. Sometimes, a single model is not the only solution to the problem and must be combined with different models to solve the same problem, which we propose in our paper.

Using commands such as threads, locks, and events that refer to asynchronous programming, we can achieve the required result, which we completed during this work.

Functional reactive programming enables us to preserve the asynchronous nature of modern applications while preserving the deterministic nature of traditional programming. All these applications are supported by Reactive Add-ons, which allow us to handle asynchronous sources of information, such as events. Data streams are sequences of continuous events sorted by time. They can indicate three different things: a value (of a certain type), an error, or a signal indicating "completion". It should be noted that the signal indicating "completion" occurs, for example, when the window or the current view containing that button is closed [6].

In general, reactive programming is constructed on two main concepts, which are:

Data source: The component that emits or transmits data to other components. This component transmits an indefinite amount of data, including transmitting even when we do not have any value. The broadcast can be completed successfully, which means the broadcast is closed or terminated, and it can stop when we have a potential error. There are cases when the data source does not stop transmitting data: even if we do not have any subscribers, some cases where the transmission of data does not stop include

information about stocks, Twitter, meteorological information, etc. Data sources may have many subscribers at the same time, but they may equally not have any.

Subscribers: The component that monitors changes or monitors data sources when transmitting data. The moment data are transmitted from the data source, the onNext command is executed on each subscriber, while now, when the data source stops transmitting the data without any error, the onComplete command is executed for each subscriber. However, at the time of transmission completion, if we encounter any error, the onError command is executed on each subscriber.

This study is organized into five main parts. The first and second provide introductory information related to the paper and the purpose of using reactive programming in the proposed model for cloud security as well as providing information to resolve different problems when using reactive programming.

Furthermore, the third and fourth sections give a clear view on the proposed model in the cloud and the form of implementation of the solution while using reactive programing. In the last part, we present the results of the two solutions we have focused on: the first from implementing traditional programing and the second using reactive programming.

2. Literature Review

2.1. Reactive Programming Aspects

Reactive programming has aroused tremendous interest because it offers numerous possibilities for solving problems. These methods have been used and implemented in various systems and programming languages, from low-level to high-level languages. Reactive Programming with Node.js [7] presents the techniques of reactive programming in Node.Js, while in Programming Reactive Extensions and LINQ [8], we are dealing with the implementation of Reactive Programming in C # and LINQ. We see that reactive programming methods and techniques find use in different languages and systems.

Bonér, Jonas, and Viktor Klang, in their publication "Reactive Programming vs. Reactive Systems" [9], pointed out the separation of the concepts of reactive programming and reactive systems where they tried through examples to present these two concepts. They also showed that functional reactive programming is one of the concepts that has been misunderstood and mixed with other concepts. One of the reasons for the separation of reactive programming and reactive systems is the communication method they use. Reactive programming is event-driven, while reactive systems are message-driven.

In "Reactive Programming: A Walkthrough" [10], by Salvaneschi, Guido, et al., the uses of Reactive Programming are presented, starting from the simplest to its implementation in the Internet of Things and integrated devices. The main idea of using reactive programming is because of the fast response of the system.

Recent research shows that reactive programming is the most appropriate current method for implementing real-time systems. Thus, according to performance analyses in several programming languages and systems, works have been written in different languages and libraries for the use of reactive programming [2–4], while research carried out shows how reactive programming is also adequate for constructing and manipulating visual images in GUI [11,12].

Reactive programming is a programming paradigm oriented around data streams and the distribution of changes. This means that it should be possible to easily show data streams statically or dynamically in the programming languages used and that the execution model will automatically propagate the changes through the data stream [13–15]. It decomposes the problem into different and unique steps where each can be executed asynchronously and in a non-blocking way, and then, the workflow is carried out—there may be limits to its inputs or outputs. The main benefits of reactive programming are the increasing use of informatics resources in multicore and multi-CPU hardware and increased performance by reducing serialization points.

The other benefit is developer productivity, as traditional programming paradigms have stalled to provide a straightforward and maintainable approach to handling asyn-

chronous and non-blocking and I/O computing [16,17]. Reactive programming solves most of the challenges here, as it usually eliminates the need for coordination between active components. Reactive programming excels at creating components and composing workflows. To take full advantage of asynchronous execution, the inclusion of back pressure is crucial to avoid excessive utilization or unlimited consumption of resources.

Although reactive programming is a very useful component when building modern software, to justify building a system at a higher level, reactive architecture should be used—the process of designing reactive systems. Furthermore, it is important to know that there are many programming paradigms, and reactive programming is just one of them, just as every paradigm is not intended for all use cases.

2.2. Cloud Security Aspects

The literature has suggested different models to increase security in cloud. For all the customers who are still struggling to migrate to the cloud environment, virtual machine monitoring for addressing security and privacy in the cloud is offered as a choice [18].

The authors from [19] proposed the Trusted Cloud Computing Platform (TCCP), which allows the calculation of the integrity and confidentiality of data sent by the provider. This platform informs the CSC (Cloud Service Client) if the data have been accessed by others or the CSC and then decides whether to interrupt the VM or to continue detecting unauthorized access. The TCCP's main task is to ensure that nothing intervenes between the CSC, CSP (Cloud Service Provider), and VM. Finally, it is the TCCP approach, based on the plain form of the traditional TERRA belief [20,21], which provides the integrity and confidentiality of the data with respect to multiple hosts. The authors of [22] offer a platform for cloud computing: Private Virtual Infrastructure (PVI). This platform manages, monitors, and combines the Trusted Platform Modules (TPMs) and the Locator Bot (LB) that provide security measurements to the properties they own; it also provides the database and offers continuous monitoring in the cloud. From this point of view, it can be said that the data security falls like the CSC and the CSP. Starting from the SLA, there are no proper roles for the security responsibilities for all participants in the cloud environment [23,24]. The authors of [23,25,26] argue that cloud service reliability is achieved through the security level of workload to facilitate service quality assurance.

Security level determination is based on the following requirements: Workload State Integrity, Guest OS Integrity, Zombie Protection, Denial of Service attacks, malicious resource exhaustion, platform attacks, and backdoor protection. In this scientific paper, there is no clear method to determine the level of security, as it is claimed, so it is not apparent how this security level information is sent to the CSC and CSP.

The authors of [27] proposed a private cloud monitoring and management scheme called the PCMONS. Despite many differences from traditional technologies and the cloud environments, it is argued that it is possible that these resources (inherited network and distributed management methods, etc.) have potential for reuse in the private cloud. The PCMONS is focused on centralized architecture described by these features:

- A node with accumulated information, which is responsible for collecting local information for the next node;
- Cluster Data Integrator—a collection of data for the next layer (collected by monitoring);
- Monitoring Data Integrator, which collects information and stores it only for archives in the database and provides these data to the configuration generator;
- A Virtual Machine whose task is to transfer data from the VM to the overall monitoring system;
- A Configuration Generator whose task is to obtain information from the database; and
- A Monitoring Tool Server is used for the sole purpose of gathering information from the monitoring of different resources from a database that is populated with notes from the Configuration Generator and the Monitoring Data Integrator.

The PCMONS projection responds to the private cloud requirements, with only space to provide a security environment for the CSC and the CSP. The whole architecture of this proposal is based on some interesting features that can be used in similar architectures with

the sole purpose of providing a safe environment for all parties. One of the features that can be exploited is the monitoring data integrators, which can be exploited by the CSC as a feature to provide information in the cloud environments. From other research, it can be said that researchers have provided some security solutions in the cloud environment, some of which have been implemented. All of the focus has been on the issue of creating a credible environment and monitoring virtual equipment driven only by the CSP, not by all the participating parties, and the ability to monitor all these parties. The goal is for the entire monopoly to be implemented by the CSP. In the paper [28], the authors discussed the advantages of implementing Blockchain and Using Honeypot Network for detecting new attacks as well as optimizing and exploiting resources, which we believe will increase PCMONS performance.

All the work is based on this slogan [29]: "if the provider does not need to read the information, why should he be allowed to?" The belief in this article was achieved by dividing the information and then controlling the scattered parts.

In their work, the author identified five cloud computing models designed to increase cloud security:

- The separation model separates the storage of data from the processing of data at different providers;
- The availability model ensures that there are at least two providers for each of the data storage and processing tasks and defines a replication service to ensure that the data stored at the various stage providers remain consistent at all times;
- The migration model defines a cloud data migration service to migrate data from the stage provider to another;
- The tunnel model defines a data tunneling servile between a data processing service and a data stage service, introducing a layer of separation where a data processing service is oblivious of the location (or even identity) of a data storage service; and
- The cryptography model extends the tunnel model by encrypting the content to be sent to the stage provider.

By using these models, which allow duplication and task sharing, they reduce their integrity, availability, and confidentiality by encrypting the data storage [16,30]. According to [31], the EU confidence increases when exposure and access to sensitive data is banned.

Considering this, a solution is provided where the client requires sensitive data to be processed only in the system where their placement is required to know exactly where they are located to be within the EU. However, this is a dubious point because the cloud service providers have the right to operate all over the world.

The authors from [32] proposed the pi-cloud (personal secure cloud), which includes the cloud resource management resources that are interrelated to each other for end users. The pi-cloud objective is the last user to format the IT Infrastructure without losing control over its data. The cloud federation refers to a personal combination of the user for private and public cloud sources.

Up to this proposal, in the cloud environments, three facilities of success are jeopardized: availability, integrity, and confidentiality.

The pi-cloud works by sharing trustworthy and untrustworthy sources. The user adapts the cloud based on their needs, providing data flow and execution of services. The pi-cloud is controlled by the pi box, and the task of this gate is to divide sensitive data from the public data. The pi box consists of four main components:

- The cockpit;
- The service controller;
- The data controller; and
- The resource manager.

3. Overview of Proposed Model Analysis for Cloud Security Controlled by End User—ITSS

We closely analyzed important elements for the proposed model for security in the cloud and the method of storing data by the organizations after these data are sent to cloud [21], and then, we compared them with possibilities to enhance or improve our proposed model for security in the cloud, shown in Table 1.

Table 1. Data in relation to our proposed model for security in cloud, referring to study [21] and with the most important elements shown.

Elements	Support of the Proposed Model
Does your company have an encryption strategy?	Our proposed model gives responses to the question where data are partitioned and encrypted and then sent to the cloud, of which 50% say their organizations have an overall encryption plan that is applied consistently across the entire enterprise, but others have a limited encryption plan, or they did not have at all.
Industries that have shown interest in developing a better encrypting strategy	In this part, our proposed model offers support for every field we mentioned above.
Possible attacks on sensitive data	After processing the data, 53% of attacks on sensitive data come from the employees. In this context, our proposed model does not allow employees to make such interferences. It uses security configurations realized by the IT specialist (employee who is qualified in IT), and then, the entire communication is developed on that configuration of a particular employee.
What keys pose difficulties when managing in the encrypting process?	The proposed model makes it possible to store the data by the used strategy. This strategy depends on the type of algorithm used, the cryptography keys, the method of partitioning, etc. This file is stored locally to the last user.

The elements that will be offered by the proposed model for cloud security are presented in Table 1. As an initial part, the "configuration" of the security of the organization will be realized by the IT Security Specialist. Based on data sensitivity, we select the options mentioned below (Proposal 1, Proposal 2, and Proposal 3).

The proposed model is based on two main actors: the possibility of categorizing the level of security based on the combination of security algorithms [33,34] and all the control of security depending on the end user, the ITSS of a certain organization.

The proposed model enables an increase in security in the cloud (Figure 1), offering three proposals of choice depending on the sort of sensitive data:

- Proposal 1: Security is based on the choice of the end user, the ITSS, depending on the information the proposed model offers. Based on the system proposals, no other specific factors are considered except for the faster way that the system offers.
- Proposal 2: Based on the features of the file, possible algorithms are proposed, along with the length of keys, to the user, and then, the user makes a choice. Based on the system proposals that consider the features of the file, the most suitable and fastest solution offered by the model is proposed.
- Proposal 3: Security is based on the file cryptography by the client and by keys generated locally to the client. Thereafter, the file is partitioned and encrypted in particular parts (P1, P2, . . . , Pn); each part can be stored in different clouds. A new po file contains the selected algorithm, along with the indexing and the position of the file. The po file is significantly smaller, encrypted by a more powerful algorithm, and can

be stored anywhere in cloud. This solution depends entirely on the selection of ITSS, not taking into account the suggestions that the model can make; the security "configuration" is performed by ITSS in case the data sensitivity is too high, and the safest path is determined using algorithms specific for data encryption and fixed/random partition number.

In this work, it can be seen how the client and the server communicate, and we have defined an executable protocol that verifies the data requested by the client. In previous research [17], Proof of Retrieval (POR) protocols have been proposed, whose task is for large files to prove that the file was not deleted or modified during the communication process. This encryption technique takes the file and then encrypts blocks to randomly encrypt the file. After this step, the client requires its data to depend on their sensitivity to determine the manner of their archiving.

In addition, communication with cloud providers is realized according to [23,24] service level agreement (SLA) mutually agreed upon between all parties to provide the quality of services (QoS), which uses our proposed model for cloud security, as shown in Figure 1.

To provide a secure communication in the communication channels, we use cryptography. As technology improves, the need to provide new encrypting strategies increases as well [6,13,14], as the only reason that data can be transferred from point A—the end user—to point B—the database—is to be secured. For data encryption, we use two techniques of cryptographies: a symmetric algorithm, where the same key is used for encryption and decryption, and an asymmetric algorithm; however, different keys are used for encryption and decryption. Our proposed model not only uses these two techniques of encryption but also uses the combination of both symmetric and asymmetric algorithms as well, known as hybrid algorithms [23].

Our proposed model's distribution of file was completed based on the number of available cloud providers, the space they have, and random distribution in Figure 2.

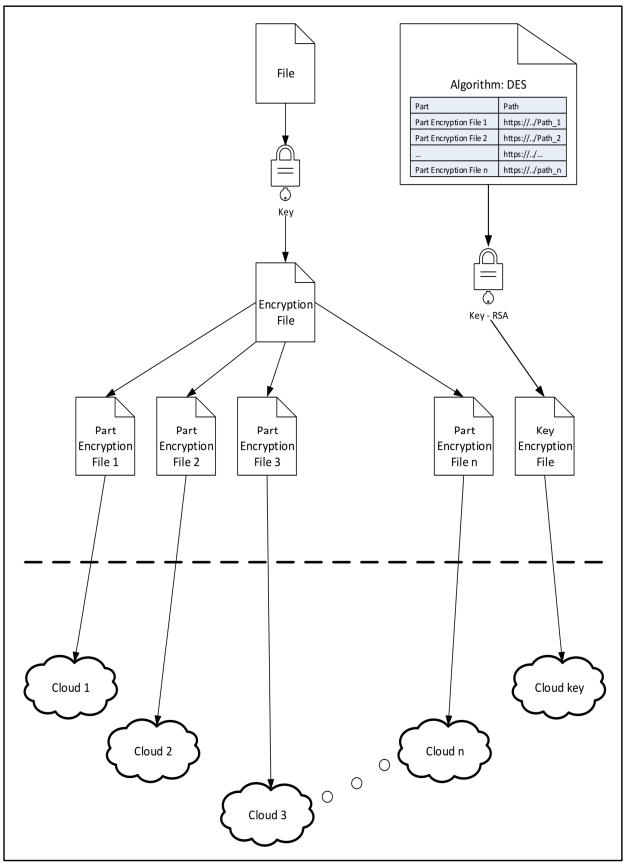

Figure 2. Partitioning and distribution of files to the cloud computing.

Despite segments being divided depending on the partitioning method we used, another file with the suffix ".enc", as in Figure 2, was generated, which offers general information on the type of algorithm, the method of partitioning, and the names of the files before and after the encryption (naming of partitioning is random) as well as the keys used for cryptography. In Figure 2, this file is named as the Key Encryption File.

Figure 3 shows a demonstrated case where a test.docx file is used for testing, and then, we used the first level and selected the RSA algorithm. Partitioning is completed randomly, and then, the partitions are encrypted. Figure 3 shows the case of return, description, and the merging, and as a result of this option, we obtained the file test_merge.docx, which takes us to the first step. Additionally, in part of Figure 3, we present the file test.docx..enc, which contains the information we explained above.

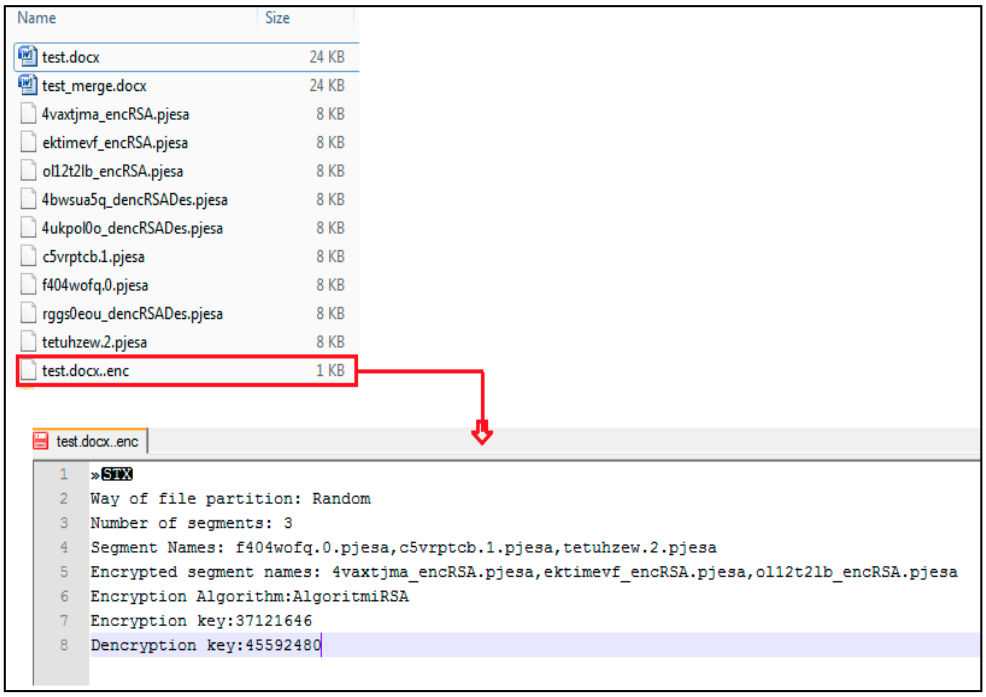

Figure 3. Content of the file .enc.

4. Utilization of Reactive Programming in the Proposed Model for Cloud Security

In the proposed model for cloud security proposed in [1,12], reactive programming was used in the partition distribution by the client to the cloud providers. Figure 4 presents the earlier method of programming, which is presented in [1], while in Figure 5, the proposed method of using reactive programming in the proposed model for cloud security is presented.

We used Reactive Extensions (Rx) to use reactive programming, and the advantage of using Rx is that it offers a new way to build and integrate asynchronous events, such as coordinating multiple data streams as they arrive asynchronously from the cloud. With Rx, these streams can be "flattened" in a single method, making the code very simple.

As an example, the classic async model in .NET is to initiate each call with a BeginXXX method and end it with an EndXXX method, also known as the Begin/End model. If more than several simultaneous asynchronous calls are made, data stream control quickly becomes impossible.

However, with Rx, the Begin/End model is summarized in a single method, making the code much cleaner and easier to be changed. Reactive Add-ons can be seen as a library for building or developing asynchronous and event-based software that uses observable collections.

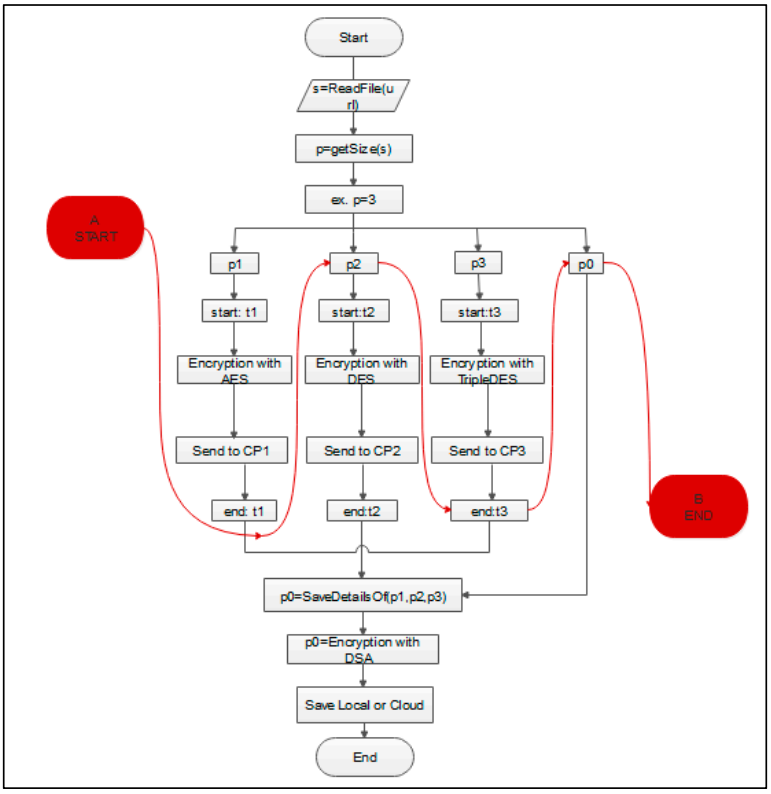

Figure 4. Traditional programming used for model proposed for security in cloud.

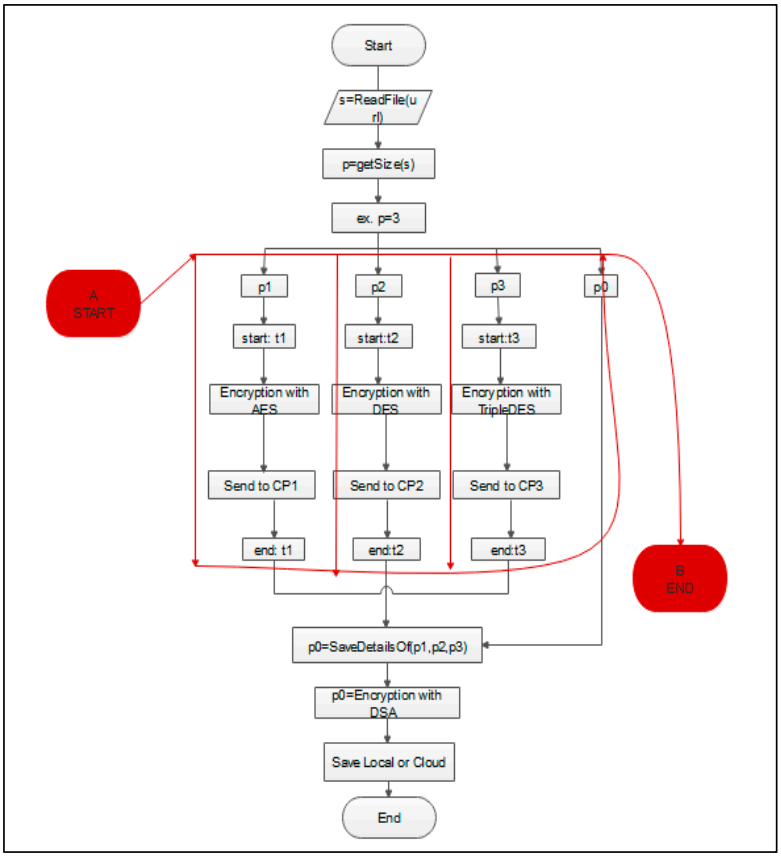

Figure 5. Reactive programming used in proposed model for cloud security.

For the measurements, we followed the path from point A to point B, which is presented in red in Figures 4 and 5.

Figure 4 presents the traditional programming applied in the proposed model for cloud security, while Figure 5 presents the new solution realized in this study for the proposed model for cloud security.

In Figure 5, for the continuation of our work, we used reactive programming realized in the model for cloud security. The reason why we decided to use reactive programming in this part, Figure 5, is that, after the analysis of the delay time, it was seen that, when sending and receiving partitions, there are delays, and the use of reactive programming speeds up this process.

In the algorithm from Figure 6, the algorithm we used for programming the proposed model in the cloud is presented. During this algorithm, reactive programming was used with the sole purpose of faster and more accurate communication. The algorithm shows that it starts with the question of whether the providers are active as to where the partitions will be sent/received, pn (p1, p2, p3..pn). If yes, then the next step Enc. (Alg.n)/Dec. (Alg.n) is performed at the same time for all partitions. After the completion of this step, the last step is performed, which is the population of the file with the communication rules, which, in the algorithm, we have called p0. The algorithm in Figure 6 is realized using LINQ communication as part of ASP.NET, as shown in Figure 7.

Using pseudocode for partitioning files and encrypting them at the same time

Require:
Communication between end user and providers (C1,C2, C3..Cn)

Ensure:
Communication is ready, *Cn*, and the file is partitioned into *pn(p1,p2, p3..pn)* partitions, and calling algorithms for encryption/ decryption, it is possible *Enc.(alg.1, alg.2, alg.3,,,alg.n)/ Dec.(alg.1, alg.2, alg.3,,,alg.n)*

while *Cn* is active, and *pn* is ready
    do
    If (*tn* is active)
        Continue encryption/decryption with *Enc.(alg.n)/Dec.(alg.n)* for *pn*
    else
        Delete(*tn*)
    end if
end while
until all partitions are encrypted/decryption
Enc.(alg.n) all *pn* and also send to the *Cn* in same time
    if (t1, t2..tn end)
        return *p0* with rules that have been used
    else
        return *p0* empty
    end if

Figure 6. Pseudocode for partition file and encryption/decryption used in Reactive Programming.

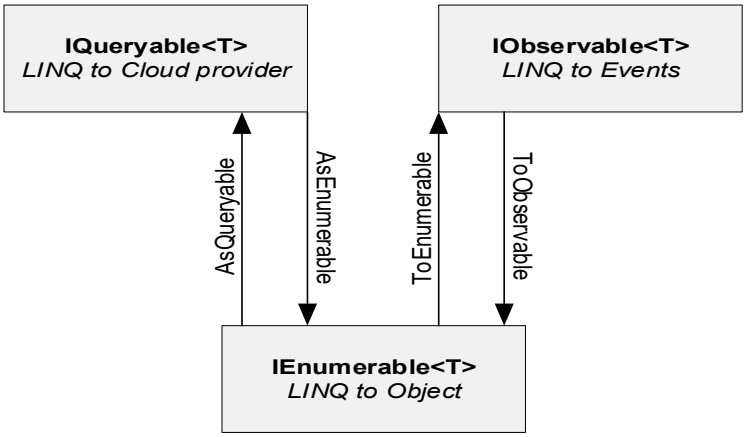

Figure 7. LINQ communication as part of ASP.Net used in reactive programming.

In Figure 7, the communication that is presented using LINQ as part of ASP.Net is introduced, which is a set of technologies based on the integration of query capabilities directly into the C # language and which is used in our model, Figure 5. In IQueryable, ready providers are called to place the partitions; for each of them, Objects are created, and through the events, the resources of reactive extensions are called.

5. Results and Analysis

To be able to compare two different methods of programming, we used the same executable environment as well as the same files as in Table 2. In order to have accurate comparisons of the results, we relied on the same work environment for two methods of programming:

- Processor: Initial (R) Celeron (R) CPU 1005 M 1.90 GHz;
- Network details: ping: 55 ms, download: 15.46 Mbps, upload: 2.22 Mbps;
- We used the same files for the experiments, Table 2.

Table 2. Files used for measurement referred from the paper [1].

Type	Size	Comment
.doc	2969 KB	Large
.doc	606 KB	Medium
.pdf	606 KB	Medium
.png	606 KB	Medium
.mov	454 KB	Medium

The structure of the data analysis is the same as in previous work [1]; for all measurements, we used the measuring unit of time of execution in milliseconds. Additionally, for every case of measurements, there were two methods of measurements for Upload and Download. These measurements are realized using the same schemas for every type of algorithm (symmetric, asymmetric, and hybrid) for Old Execution Time, Figure 4, and for New Execution Time, Figure 5. Moreover, in this part, the execution time of measurement starts after the file is partitioned, and special measurements (i.e., t_1,t_2 ... t_n) are made for every partition. Additionally, the total time in general is taken for T, while in partitions, t_1,t_2 ... t_n. For measurements, we used three symmetric algorithms, namely AES, DES, and TripleDES, and three asymmetric algorithms, namely RSA, Diff-Hellman, and El Gamal, as well as hybrid algorithms (combination of both symmetric and asymmetric algorithms).

In Table 3 and Figure 8, the results of measurements are shown, and the time of calculation is carried out as in Equation (1), while by using reactive programming, it will be found by Equation (2). The MaximumOfTime() function calculates the longest processing time (upload + download) of the partition to a provider, and that is the time it takes

to process all the partitions (upload + download) for a file (because it is processed the execution at the same time for all file partitions).

$$T_{\text{Old Execution Time}} = t_1 + t_2 + \cdots + t_n \quad (1)$$

$$T_{\text{New Execution Time}} = \text{MaximumOfTime}\,(t_1, t_2, \ldots t_n) \quad (2)$$

Table 3. Information table with old and new execution time [1].

File Type	File Size	Algorithm	Old Execution Time	New Execution Time
.doc	2969 KB	AES | DES | TripleDES	43,217	26,927
.doc	2969 KB	RSA | Diffie-Hellman | ElGamal	175,436	74,337
.doc	2969 KB	RSA | DES | ElGamal	133,085	74,337
.doc	606 KB	AES | DES | TripleDES	27,057	11,255
.doc	606 KB	RSA | Diffie-Hellman | ElGamal	42,542	15,322
.doc	606 KB	AES | Diffie-Hellman | TripleDES	32,256	15,322
.pdf	606 KB	AES | DES | TripleDES	26,551	10,056
.pdf	606 KB	RSA | Diffie-Hellman | ElGamal	45,357	17,559
.pdf	606 KB	DES | TripleDES | ElGamal	37,137	17,559
.png	606 KB	AES | DES | TripleDES	25,215	10,121
.png	606 KB	RSA | Diffie-Hellman | ElGamal	40,975	15,553
.png	606 KB	RSA | Diffie-Hellman | TripleDES	35,543	13,764
.mov	454 KB	AES | DES | TripleDES	15,658	11,021
.mov	454 KB	RSA | Diffie-Hellman | ElGamal	39,056	13,696
.mov	454 KB	DES | Diffie-Hellman | ElGamal	32,716	13,696

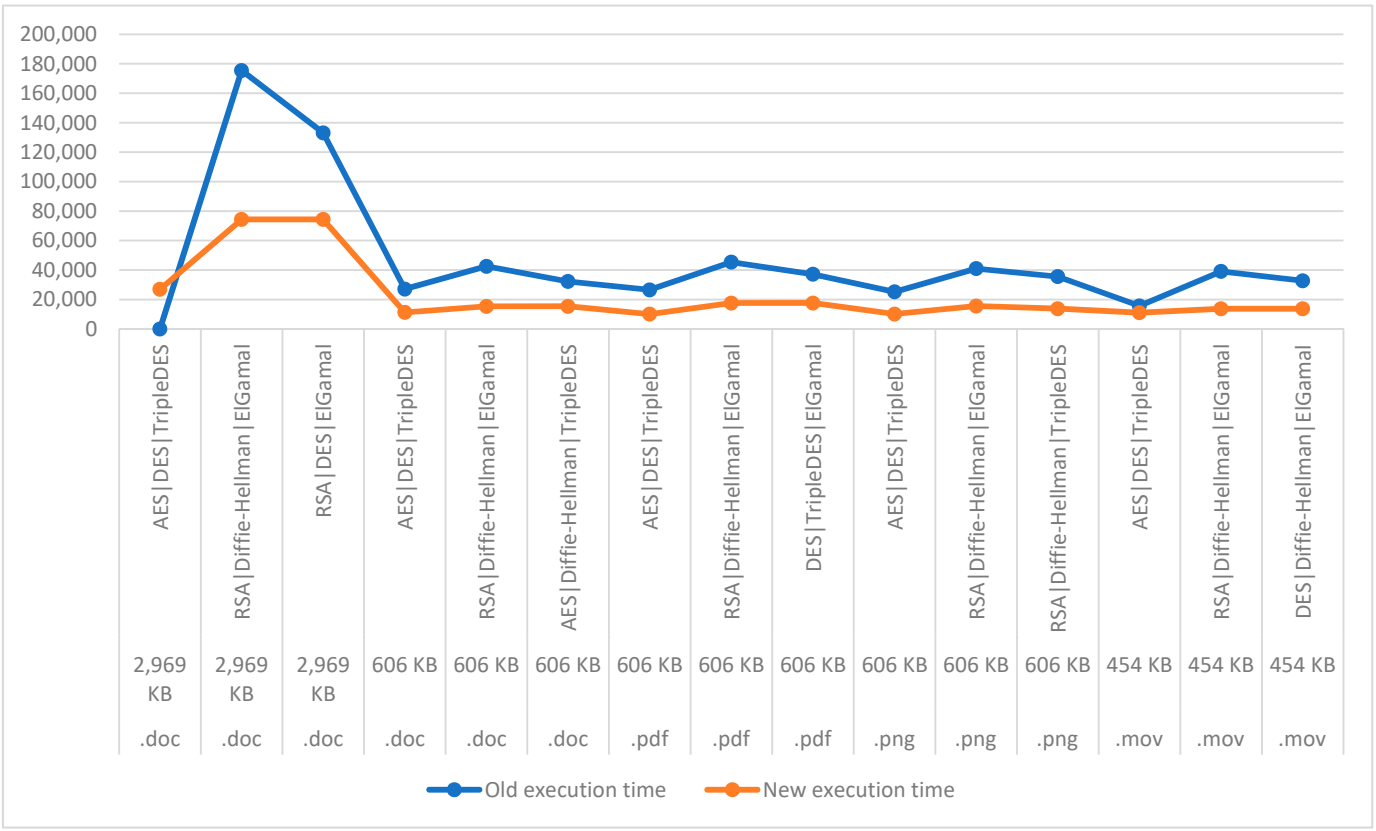

Figure 8. Results of measurements from old and new execution time.

6. Conclusions

In this study, we have presented the model proposal for cloud security and the possibility of categorizing the level of security based on the combination of security algorithms,

and then, all the control of the security depends on the end user, the ITSS, of a certain organization, which is a solution that was proposed in our previous work. For the implementation of this solution, we used traditional programming, but we experienced longer delays that came up when files were being partitioned and during the merging of those partitions from the cloud providers.

During this study, we proposed a solution for solving this problem by using reactive programming. As presented in Figure 6, the use of reactive programming in this case speeds up the processes, e.g., pn(p1,p2, p3..pn), Enc.(alg.1, alg.2, alg.3,,,alg.n), Dec.(alg.1, alg.2, alg.3,,,alg.n). After using this technique of reactive programming, we realized measurements for the same data to come to a more accurate comparison between these two programming techniques for the proposed model for security in the cloud.

The measurements we made are based on the two equations presented in the study: the time calculation is completed as in Equation (1), and using reactive programming, as in Equation 2, is based on the results shown in Table 3. It can be seen that the difference of execution time exists both during the partitions and during the merging, which is also emphasized in all data types.

From the realized measurements that are presented in Figure 8, we can conclude that the effect of using reactive programming is presented in each file type and by each algorithm used, reducing the execution time in the proposed model for cloud security.

In conclusion, it can be said that reactive programming use is probably more difficult to apply than the traditional programming that has been used so far, but the execution time is faster, especially in our case.

In the future, it would be good for the proposed model with the changes implemented in this paper to be tested in other environments and with other data in order to have different results and then to improve where it shows delays.

However, in the future, we plan to use FPGA, as a frontage connected with business (IT Security Specialist), to increases the level of security and speed of the encryption and decryption process (as a hardware and software solution). Additionally, in the future, it will be interesting for this model to advance more with testing and provide more cloud storage security managed with data dynamics, including quantum key cryptography.

Author Contributions: Methodology, D.H., I.S. and B.Ç.; validation, N.P. and I.S.; formal analyses, D.H. and B.Ç.; resources, D.H., B.Ç. and I.S.; writing—original draft preparation, D.H.; writing—review and editing, I.S. and B.Ç.; visualization, D.H. and N.P.; supervision, B.Ç. and I.S.; project administration, D.H.; funding acquisition, D.H. and N.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

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

References

1. Hyseni, D.; Luma, A.; Selimi, B.; Cico, B. The Proposed Model to Increase Security of Sensitive Data in Cloud Computing. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 203–210. [CrossRef]
2. Mogk, R.; Salvaneschi, G.; Mezini, M. Reactive programming experience with rescala. In Proceedings of the Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France, 9–12 April 2018; pp. 105–112.
3. Ignatoff, D.; Cooper, G.H.; Krishnamurthi, S. Crossing state lines: Adapting object-oriented frameworks to functional reactive languages. In *International Symposium on Functional and Logic Programming*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 259–276.
4. Cunha, G.T. Reactive vs. Synchronous Performance Test with Spring Boot 2.0. 2018. Available online: dzone.com/articles/spring-boot-20-webflux-reactive-performance-test (accessed on 12 January 2022).

5. Programming Paradigms. Available online: https://theailearner.com/2018/09/26/programming-paradigms/ (accessed on 25 February 2022).
6. Staltz, A. The Introduction to Reactive Programming You've Been Missing. 2014. Available online: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 (accessed on 21 August 2017).
7. Doglio, F. Functional Reactive Programming. In *Reactive Programming with Node*; Apress: Berkeley, CA, USA, 2016; pp. 25–45.
8. Liberty, J.; Betts, P.; Turalski, S. *Programming Reactive Extensions and Linq*; Apress Media, LLC: New York, NY, USA, 2011.
9. Bonér, J.; Klang, V. *Reactive Programming vs. Reactive Systems*; Lightbend, Inc.: San Francisco, CA, USA, 2016.
10. Salvaneschi, G.; Margara, A.; Tamburrelli, G. Reactive programming: A walkthrough. In Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 16–24 May 2015; Volume 2, pp. 953–954.
11. Czaplicki, E.; Chong, S. Asynchronous functional reactive programming for GUIs. *ACM SIGPLAN Not.* **2013**, *48*, 411–422. [CrossRef]
12. Hyseni, D.; Cico, B.; Shabani, I. The proposed model for security in the cloud, controlled by the end user. In Proceedings of the 2015 4th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 14–18 June 2015; pp. 81–84.
13. Hyseni, D.; Cico, B.; Luma, A.; Selimi, B.; Shemsedini, E. Different methods of distribution data in the cloud—Controlled by IT security specialist. In Proceedings of the 2018 7th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 10–14 June 2018; pp. 1–5.
14. Hyseni, D.; Selimi, B.; Luma, A.; Cico, B. The strategy of cryptography for the proposed model of security in cloud computing. In Proceedings of the 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), Chennai, India, 21–22 September 2017; pp. 24–28.
15. Hussain, M.E.; Hussain, R. Cloud Security as a Service Using Data Loss Prevention: Challenges and Solution. In *International Conference on Internet of Things and Connected Technologies*; Springer: Cham, Switzerland, 2021; pp. 98–106.
16. Mishra, S.; Sharma, S.K.; Alowaidi, M.A. Analysis of security issues of cloud-based web applications. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 7051–7062. [CrossRef]
17. Juels, A.; Kaliski, B.S., Jr. Pors: Proofs of retrievability for large files. In Proceedings of the 2007 ACM Conference on Computer and Communications Security (CCS 2007), Alexandria, VA, USA, 2 November–31 October 2007; ACM Digital Library: New York, NY, USA, 2007; pp. 584–597.
18. Sen, J. Security and privacy issues in cloud computing. In *Architectures and Protocols for Secure Information Technology Infrastructures*; IGI Global: Hershey, PA, USA, 2013; pp. 1–45.
19. Santos, N.; Gummadi, K.P.; Rodrigues, R. Towards Trusted Cloud Computing. In Proceedings of the 2009 Conference on Hot Topics in Cloud Computing (HOTCLOUD), San Diego, CA, USA, 15 June 2009; USENIX: Berkeley, CA, USA, 2009.
20. Garfinkel, T.; Pfaff, B.; Chow, J.; Rosenblum, M.; Boneh, D. Terra: A virtual machine-based platform for trusted computing. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 19–22 October 2003.
21. Ponemon Institute Research Report. Global Encryption Trends Study. 2021. Available online: https://www.entrust.com/-/media/documentation/reports/2021-global-encryption-trends-exec-summary-re.pdf (accessed on 12 March 2022).
22. Krautheim, F.J. Private Virtual Infrastructure for Cloud Computing. In Proceedings of the 2009 Workshop on Hot Topics in Cloud Computing, HotCloud 2009, San Diego, CA, USA, 15 June 2009.
23. Kumar, S.; Karnani, G.; Gaur, M.S.; Mishra, A. Cloud security using hybrid cryptography algorithms. In Proceedings of the 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, UK, 28–30 April 2021; pp. 599–604.
24. Dhirani, L.L.; Newe, T.; Nizamani, S. Federated Hybrid Clouds Service Level Agreements and Legal Issues. In *Third International Congress on Information and Communication Technology*; Springer: Singapore, 2019; pp. 471–486.
25. Ouedraogo, M.; Mignon, S.; Cholez, H.; Furnell, S.; Dubois, E. Security transparency: The next frontier for security research in the cloud. *J. Cloud Comput.* **2015**, *4*, 12. [CrossRef]
26. Arshad, J.; Townend, P.; Jie, X. Quantification of Security for Compute Intensive Workloads in Clouds. In Proceedings of the 15th International Conference on Parallel and Distributed Systems, Shenzhen, China, 8–11 December 2009.
27. De Chaves, S.A.; Uriarte, R.B.; Westphall, C.B. Towards an architecture for monitoring private clouds. *IEEE Commun. Mag.* **2011**, *49*, 130–137. [CrossRef]
28. Sangui, S.; Ghosh, S.K. Cloud Security Using Honeypot Network and Blockchain: A Review. In *Machine Learning Techniques and Analytics for Cloud Security*; Scrivener Publishing LLC: Beverly, MA, USA, 2021.
29. Jaatun, M.G.; Zhao, G.; Vasilakos, A.V.; Nyre, Å.A.; Alapnes, S.; Tang, Y. The design of a redundant array of independent net-storages for improved confidentiality in cloud computing. *J. Cloud Comput. Adv. Syst. Appl.* **2012**, *1*, 13. [CrossRef]
30. Dorairaj, S.D.; Kaliannan, T. An adaptive multilevel security framework for the data stored in cloud environment. *Sci. World J.* **2015**, *2015*, 601017. [CrossRef] [PubMed]
31. Wolters, P.T.J. The security of personal data under the GDPR: A harmonized duty or a shared responsibility? *Int. Data Priv. Law* **2017**, *7*, 165–178. [CrossRef]
32. Mosch, M.; Groß, S.; Schill, A. User-controlled resource management in federated clouds. *J. Cloud Comput.* **2014**, *3*, 10. [CrossRef]
33. Nagarajan, G. Comparative Analysis of Public Cloud Security Based Schemes and Cryptographic Algorithms. *Turk. J. Comput. Math. Educ. (TURCOMAT)* **2021**, *12*, 2114–2127.
34. Awaysheh, F.M.; Aladwan, M.N.; Alazab, M.; Alawadi, S.; Cabaleiro, J.C.; Pena, T.F. Security by design for big data frameworks over cloud computing. *IEEE Trans. Eng. Manag.* **2021**. [CrossRef]