SortAlgo-Metrics: Identification of Cloud-Based Server Via a Simple Algorithmic Analysis

Samuel D Baba, Supriya Yadav
School of Engineering and Digital Arts
University of Kent
Canterbury, UK
sb2097@kent.ac.uk
sy227@kent.ac.uk

Gareth Howells
School of Engineering and Digital Arts
University of Kent
Canterbury, UK
W.G.J.Howells@kent.ac.uk

***Abstract*— This paper introduces a novel technique to detect spoof or fake software systems via the generation of a unique digital signature based on a direct analysis of the construction of the system. Specifically, we model a novel mechanism referred to as Sort*Algo-Metrics analysis* to identify cloud-based servers. Experimentally, we deployed four cloud-based servers to run four sorting algorithms in order to extract features that are employed to perform statistical analysis upon with the aim to obtain their metrics which has further underpin the investigation of their behaviours. The model has been validated by comparing training data and unknown data, and the result has shown server 2-4 have a strong identification with 96% probability, while server 1 with 55%, it is surmised that is could be as the result of insufficient sample data. However, if such a simple model can produce a result with this high probability, this shows that with more complex features and sufficient data pulled from cloud-based servers, *SortAlgo-Metrics* model could generate a higher degree of basis numbers for ICMetrics technology entropy key generation and other complex systems.**

Keywords—***Keywords: Cloud computing, Cloud-based servers, Spoofing, attacks, ICMetric Technology, SortAlgo-Metrics Analysis, Spoofing.***

I. INTRODUCTION & RELATED WORKS

Identifying cloud-based servers has become crucial as hackers are utilising the advantage of the vulnerabilities and threats associated with the clustered servers in cloud computing networks which has made it a complex system, and the benefits of cloud computing has also contributed to the significant malicious activities which include server spoofing, IP address, Datagram, Address Resolution Protocol (ARP)[1],[2],[3],[4],[5],[6],[7].

Several enterprises have adopted cloud system in order to heighten the operational performance. However, our studies have revealed that the spoofing of the above-stated complex components of cloud technology could result to the companies' data confidentiality, integrity, and availability been corrupted, hijacked and made unavailable [1],[8],[9]. Consequently, this may lead to the loss of billions of pounds (GBP).

Fig I below, have demonstrated how a spoofed or fake web server could intercept a genuine web server while establishing a connection to other servers which include Domain Name System (DNS), Dynamic Configuration Host Protocol (DHCP) which consequently might result in distributed denial of service (DDoS) attack, and its effects could be capable of causing a total shutdown of an Information Technology (IT) infrastructure [1],[5],[8].

The subsequent sections of this paper are structured as follows; Section II Introduces ICMetrics Technology, its unique attributes, and areas of application, while section III discussed Metrics Analysis and its application, Section IV also discusses Sorting Algorithm its application, Section V Presents the details of SortAlgo-Metrics experiment VI section VII discussed the Mapping Methodology and finally, the paper is concluded in section VIII.

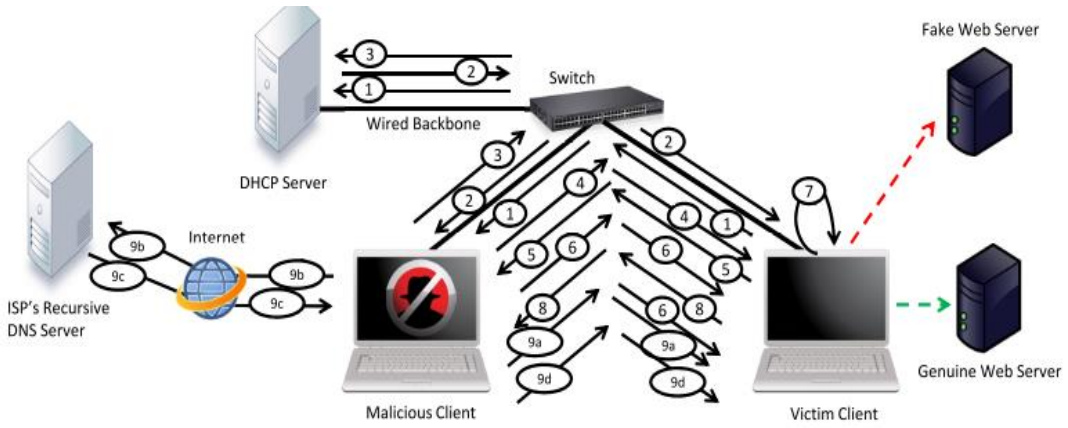

FIG I: TYPICAL IP SPOOFING TO FLOOD THE CLOUD SERVER [5]

Our previous studies have revealed several researchers developed various models with diverse techniques to detect spoofed servers running in the cloud platform [1],[2],[3],[4],[5],[6],[7]. Conversely, still much work is needed in that domain. Therefore, it would be advantageous however to develop a system that could be employed on relatively simple software constructions that could subsequently be scaled up to generate a high degree of discernment for a complex system capable of exhibiting a high degree of entropy.

With the current level of clustered servers in the cloud environment, it makes it quite challenging to identify a server running in the cloud network and propose a security mechanism to secure it. In that note, we are motivated to develop an efficient technique that will allow us to extract potential properties or features from servers running in the cloud platform. After having conducted significant research in sorting algorithms and metrics analysis in [4]-[10]. Consequently, we derived a mechanism referred to as SortAlgo-Metrics analysis, which is the synthesis of Sorting Algorithm and Metrics Analysis. The term was derived base on the critical investigation, observation and analysis conducted on the roles played by these components in various applications, and the evident attributes in general [4]-[10].

Below is the typical layout of SortAlgo-Metrics Mechanism

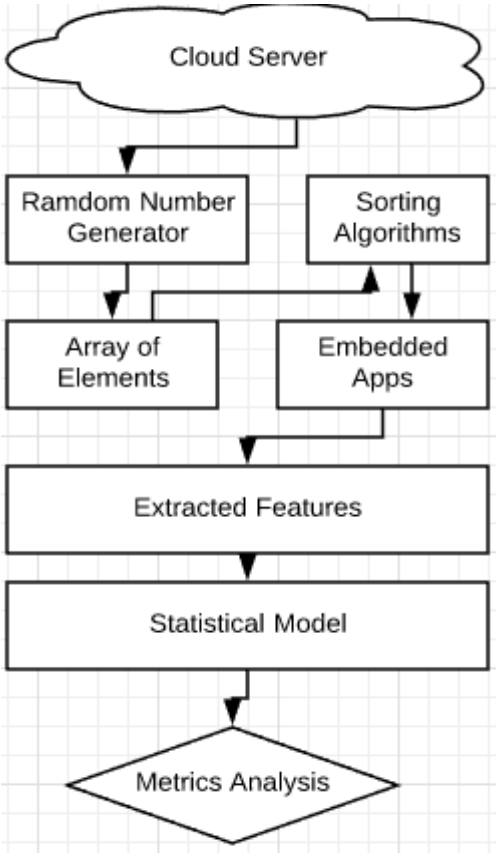

FIG II. TYPICAL SortAlgo-Metrics Mechanism

In Fig II above, we demonstrated SortAlgo-Metric Mechanism processes. Firstly, we deployed four sorting algorithms in the cloud servers and generate random numbers to be stored in the cloud, and those algorithms running in the cloud server sorts the elements in descending order to enable features extraction. Finally, we compute the features using the statical model to arrive at metrics evaluation.

II. ICMETRICS TECHNOLOGY

This section presents ICMetrics technology and its characteristics

A. ICMetrics technology

ICMetrics has a technique of generating a unique encryption key from the operational characteristics of both software and hardware systems [4],[11],[12]. Most importantly, it only generates its digital signatures during the run time.

Below are the succinct advantages of the technology;

- The technology does not store its templates in a device.
- ICMetrics do not allow back door creation
- It cannot be compromised
- It could regenerate the algorithm automatically when tampered with, maliciously or not.
- The private key is not stored in the system but is rather regenerated on demand.

ICMetrics-based security system utilises the metrics derived from the extracted features of a software, hardware or the combination of both through a statistical model to generate a unique identifier for a device which would consequently be employed as a basis of a digital signature or an encryption key.

III. METRICS ANALYSIS

Metrics are measurements, in order to evaluate these measurements, they are required to be analysed. In this section, we looked at how Metrics formed an essential part of several experiments. [13] employed Metrics analysis to validate data across Fortran Projects including "*Software Science metrics, cyclomatic complexity, and various traditional program.* A metrics analysis has been employed by [14] and explains that due to an exponential increase in power consumption a model referred to Power Usage Effectiveness (PUE) metrics is employed to evaluate power consumption at Datacenters. Furthermore, the authors explained that the metrics obtained will help the field of research gain an understanding of the trend of power consumption in the data centres. In this metrics analysis, the authors presented a metric data analysis on symmetric key cryptographic technique to analyse the line of code (LOC) values between the different algorithms in order to demonstrate the quality of the software from its process through the software analysis [10]. An experiment carried out by [4], Metrics analysis are employed by statistically analysing the properties extracted from nine servers which are simulated in the cloud environment to measure their numeric mapping. The author applied a multi-level algorithm to convey abnormal distribution into a normal Gaussian state, and *"multi-dimensional normalization map generation algorithm programmed to generate a multi-dimensional normalisation map"*. Consequently, the author developed a "*multi-dimensional binary key mapping algorithm to map a measured data from multi-dimensional space to a key vector"*. [2] applied metric analysis in an experiment performed with an attempt to observe certain properties and behaviours of Android applications. The measurements of the extracted features were conducted in different stages include the probability of density function of the targeted applications, intra-sample variance, and the correlation. However, the result was categorised into two, the Low and High intra-sample variance, and the final result of the experiment has shown that ICMetric technology was able to detect the spoofed application, which the authors referred to as "*identical light sensor".*

IV. SORTING ALGORITHM

Primarily, a sorting algorithm is a method used to order an unordered list of elements in an array, which could take a specific order. However, with the proliferation of technology, unordered data is also growing exponentially. Therefore, sorting algorithms are becoming crucial to sort these data. In this section, we looked at various sorting algorithms, applications and complexity succinctly. [4] deployed three sorting algorithms; Bubble sort, Cocktail sort and Merge sort on the cloud servers to investigate and analyses correlation and causalities of the features under observation running on those

servers as well as the time complexity of those algorithms. [15] demonstrated an algorithm referred to as a new Modified sorting algorithm, in [15] explained that the difference new Modified sorting algorithm and the generic algorithms, is the speed and has the efficacy to count both negative and positive numbers. [16] also demonstrated the application of the sorting algorithm to achieve "*Super-Elastic Motion Behavior"* by deploying provot particle position correction.

V. EXPERIMENTAL TEST BED SETUP

We setup an experimental testbed which involved different phases as shown in Fig III below. We used the ownCloud platform to create four virtual servers and deployed four algorithms as listed in Table II with embedded applications to generate a random array of numbers and other applications to create files and store the extracted features pulled from the cloud servers as presented in Table II below.

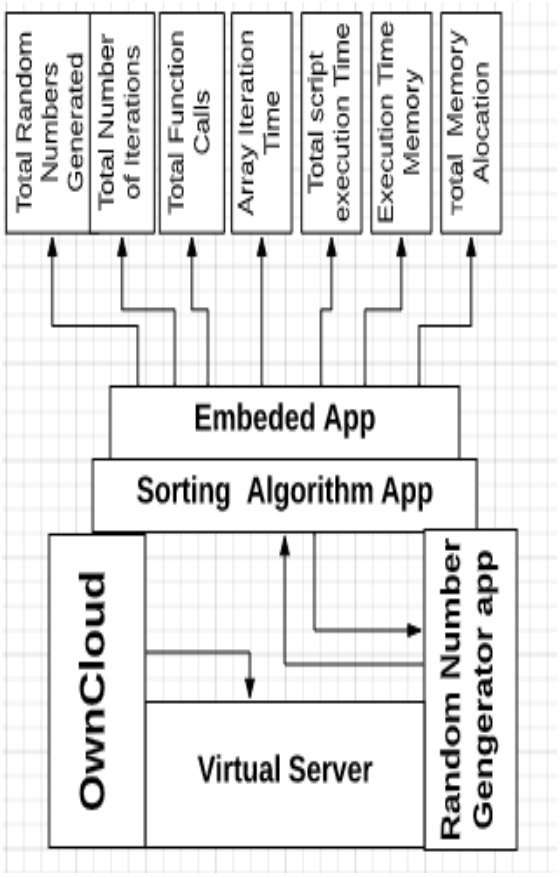

FIG. III. EXPERIMENT FRAMEWORK

TABLE I. FOUR SORTING ALGORITHMS, TIME AND SPACE COMPLEXITY [17],[4],[16]

Algorithms	Average	Worse	Space
Insertion	$0(n^2)$	$0(n^2)$	$0(1)$
Quick	$0(n \log n)$	$0(n^2)$	$0(\log n)$
Bubble	$0(n^2)$	$0(n^2)$	$0(1)$
Merge	$0(n \log n)$	$0(n \log n)$	$0(n)$

A. *The composition of feature extraction application*

To adequately extract features from the cloud-based server as shown in Table III below, a range of programming was involved as demonstrated in Fig III. Including the following stages;

- The logical design of the experimental framework for the entire application
- Programming an embedded function call for each algorithm
- Programming an embedded memory runtime usage and total memory capture
- A program to capture both random numbers generated per execution and keep track of a number of pass per each execution.
- Finally, an embedded program to generate files to collect the extracted features respective files
-

B. ***Cloud server configuration***

To enable us to create the cloud server instances, we deployed a VirtualBox machine.[13] on Windows10. Consequently, we created four instances of cloud servers and configured them to run the four proposed algorithms as shown in Table I. Hence, this has allowed us to remotely access the servers for simulation. As previously mentioned the selections of the algorithms in Table I: was motivated due to the significant roles the algorithms play in computer science such as to sort or organise homogenous dataset. Having said that, each of the server instances runs each algorithm to allow the extraction of features for statical analysis to compare the behaviours of each server as shown in Table VIII and IX below.

C. *Feature extraction procedures*

This section introduces the extraction processes of the features as presented in Fig. III. We deployed four algorithms in the cloud-based servers as shown in Table I. For each single remote script execution, random numbers are generated and stored in an embedded file, which serves as the input file to the sorting algorithms as Fig III above. Therefore, each time the script is executed, the servers run the algorithms and seven features are generated and extracted into seven files as shown in Fig III, in Table IV-VII below are the extracted values metrics of the feature. Those values are further analysed statistically in section VIII. The details description of those features is shown in Table II, and the selected features used in this analysis are shown in Table III which are consequently used to produce basis number to identify servers from one another.

TABLE II. SEVEN FEATURES EXTRACTED

S/N	Features	Description	Type of features	Number of Samples
1	Random number generated	The embedded program calculates the total array of random numbers generated in the ownCloud server	Discrete random variable	1000
2	Number of iterations	The embedded program calculates the total array of random numbers iterated by the sorting algorithm running in the ownCloud server	Discrete random variable	1000
3	Function calls	The embedded program calculates total function calls to sort the array by the sorting algorithm running in the ownCloud server		1000
4	Array iteration time	The embedded program calculates the amount of time it takes to iterate the random number stored in the array by the sorting algorithm running in the ownCloud server	Gaussian	1000
5	Script execution time	The embedded program calculates the amount of time taken to sort an array of random numbers by the sorting algorithm running in the ownCloud server	Gaussian variable	1000
6	Execution memory allocation	The embedded program calculates the amount of memory allocated to sort an array of random number by sorting algorithm running in the ownCloud server	Gaussian/ Discrete random variable	1000
7	Total memory allocation	The embedded program calculates the total memory allocated to the entire sorting algorithm script running in the ownCloud server	Gaussian/ Discrete random variable	1000

TABLE III. ANALYSED FEATURES

S/N	Features	Description	Type of features	Number of Samples
3	Function calls	The embedded program calculates total function calls to sort the array by the sorting algorithm running in the ownCloud server	Discrete random variable	1000
4	Array iteration time	The embedded program calculates the amount of time it takes to iterate the random number array by the sorting algorithm running in the ownCloud server	Gaussian	1000
5	Script execution time	The embedded program calculates the amount of time taken to sort an array of random numbers by the sorting algorithm running in the ownCloud server	Gaussian variable	1000
6	Execution memory allocation	The embedded program calculates the amount of memory allocated to sort an array of random number by sorting algorithm running in the ownCloud server	Gaussian/ Discrete random variable	1000

TABLE IV. INSERTION SORT.

	A	B	C	D	E	F	G
1	Insertion Sort						
2	N/Elemts	N/Itrts	F/Calls	rts R/Tim	S/RTime	Run TM	Tot M/A
3	1073	2146	436736	3.278899	6.94	552.98	768
4	2087	4174	2146162	17.29255	32.5566	822.65	1
5	810	1620	298736	2.188106	4.4251	475.32	768
6	1408	2816	848498	5.686229	11.2758	631.48	768
7	1867	3734	1676428	10.94663	23.2684	739.05	1
8	1569	3138	1091376	7.410487	14.7825	669.22	1
9	1499	2998	975500	6.551238	12.902	652.83	1
10	560	1120	152232	1.040794	2.0533	416.73	512
11	433	866	97202	0.680653	1.3362	378.96	512

TABLE V. QUICKSORT.

	A	B	C	D	E	F	G
1	Quick Sort						
2	N/Elemts	N/Itrts	F/Calls	trts R/Time	S/RTime	Run TM	Tot M/A
3	380	553	1110	0.04225111	0.088	371.05	1.25
4	74	107	218	0.00726295	0.0135	293.3	512
5	1551	2273	4550	0.192348	0.3735	669.51	2.75
6	1309	1925	3854	0.14840698	0.2988	612.79	2
7	367	539	1082	0.03580308	0.0722	368.04	1
8	2259	3267	6538	0.31179094	0.6104	867.47	6
9	1098	1603	3210	0.15012908	0.3012	563.34	2.75
10	114	153	310	0.01064491	0.0214	302.66	512
11	1160	1677	3358	0.13666105	0.2818	577.88	2.25

TABLE VI BUBBLE SORT.

	A	B	C	D	E	F	G
1	Bubble Sort						
2	N/Elemts	N/Itrts	F/Calls	Itrts R/Time	S/RTime	Run TM	Tot M/A
3	721	1439	1442	0.012188911	0.0239	455.41	768
4	1749	3495	3498	0.036273956	0.0751	712.36	1
5	42	81	84	0.001019955	0.0021	281.2	512
6	1543	3083	3086	0.033885002	0.0702	664.09	1
7	1331	2659	2662	0.030143023	0.0613	614.41	768
8	201	399	402	0.004703999	0.0088	321.53	512
9	982	1961	1964	0.023321867	0.0505	516.59	768
10	1428	2853	2856	0.029571056	0.0617	637.12	768
11	1906	3809	3812	0.041615009	0.0833	749.15	1

TABLE VII: MERGE SORT

	A	B	C	D	E	F	G
1	Merge Sort						
2	N/Elemts	N/Itrts	F/Calls	Itrts R/Time	S/RTime	Run TM	Tot M/A
3	2602		15608	1.928580046	3.8618	952.19	1.75
4	955	1907	5726	0.585073948	1.087	518.13	768
5	2662	5321	15968	1.98760581	3.9678	966.24	1.75
6	2661	4589	15962	2.027411938	4.0285	966.1	1.75
7	2296	559	13772	1.656507015	3.2466	880.5	1.5
8	281	1511	1682	0.133626223	0.2717	352.18	512
9	757	3759	4538	0.434869766	0.8303	471.73	768
10	1881	317	11282	1.266334057	2.4747	751.19	1.25
11	160	3949	956	0.063338995	0.1336	319.78	512

VI. MAPPING METHODOLOGY

This section introduces the algorithm for generating an encryption key which has the following four example; Function calls, Iteration run time, Script run time, Run time memory as shown in Table III. To produce an encryption key, it is essential to develop suitable methods for combining selected features to produce unique basis number - an initial binary number unique to the servers from which actual encryption keys may be derived [9],[19]. This basis number may consequently be employed to generate encryption keys to authenticate servers. To achieve a high entropy, values pulled out of multiple features are combined [20]. Feature Combination; the aim of the feature combination is to generate server specific identification numbers with low intra-sample variance (the values produced for the same circuit) but high inter-sample variance, (the values produced for different circuits) with the ideal case being no inter-sample overlap of potential basis numbers. The effectiveness of the combination plan depends on the stability of the basis numbers generated. To achieve stability in the basis numbers, stable bits are chosen from the monitored signals in the calibration phase. Due to the fact that, for any given server, a subset of the features the basis numbers generated will deviate from their ideal values, and the following properties will be typically observed for a simple addition of the values: The low order bits will vary widely since these will be governed by any feature values which have not measured within the ideal interval. The higher order bits will, in contrast, tend to be stable for a given server (intra-sample) but significantly dissimilar for differing servers (inter-sample). These stable bit are employed to form the required basis number as they have low intra-sample variance but high inter-sample variance. The basis number generated may be employed to generate the encryption key.

B. *Feature quantization and normalization*

The proposed system operates in a two-phase process, first analysing typical feature values for known servers to produce a normalization map for each feature and subsequently employing the normalization maps to produce a code for a potentially unknown server. The basic concept of the normalization map is to map a measured series of feature data into a multidimensional space. In our

previous work [20], normalization maps are linear based, mapping each individual feature to a vector and concatenating them together. The goal of quantization is to normalize feature data, so the best quantization interval should exhibit the biggest inter-sample variance between devices.

C. *Multimodal distributions*

After quantization and normalization, the next step is to establish the form of the probability distribution, for example, Gaussian, bimodal or multimodal in nature. It is possible that a set of data from a particular feature is mostly multimodal in nature, making it difficult to generate a basis number. Feature values that are multi-modal in distribution require careful consideration with regards to generating a stable key. Using multi-modal distribution, where values are chaotically positioned, we can normalise the feature distributions so we can treat the features as Gaussian. Following on from this is the operation phase, which executes when any device or service is required to generate its ICMetric.

This consists of observing features that can distinguish devices to gather their raw values. Once the raw values have been collected the normalised maps that were generated during the calibration phase can be applied and this returns values that can be combined together to derive an ICMetric. Many features have values that change whilst the device is in operation and when values are not static, the range of values a feature can produce must be mapped to some selected arbitrary value, which allows the key that is derived to be stable enough to be used.

A simple approach to this problem is to apply a peak-trough detection algorithm to the distribution, where the troughs split the multimodal distribution into separate Gaussian distributions with the peaks forming the modes [19]. Table VIII shows the modes after applying the peak-trough algorithm.

In TABLE VIII, the analysis has shown different servers, first, we calculate frequency distribution of all servers and then we apply a peak-trough detection algorithm to the distribution. Here the peak-troughs split the multimodal distribution into separate Gaussian distributions with the peaks forming the modes and then we use multivariate normal probability density function to calculate the probability of the sample associated with that mode [19].

In our experiment, the features from all servers have bimodal and multimodal distribution. For calculating the probability, we take the samples from each server, calculate the mean and covariance of the modes within the distribution of the current servers. For example, if the server has bimodal distribution then we have two modes and each mode has mean and covariance. Therefore, we first determine which of the mode the current sample falls into, then we calculate the probability of the sample and repeat the same process subsequent modes accordingly. We then take the same sample from another server and see if that sample from other server lies in which mode of the first server and then we calculate the probability of the sample. If the probability from the second server is low as compared to the first server, that means the first server is correctly identified based on probability and we repeat the same process for' n' servers. As it is shown in Table IX, the percentages go up according to the quantity of sample input. Consequently, we observed that the training data and testing data were able to differentiate between servers with 96% result based on the probability of servers except for S1, with the percentages of 50%, 53% and 55% proportional to the sample input. In our opinion, this could be as the result of insufficient data but that has not been established yet. However, in our next phase of this experiment, we are planning to pull more complex features and sufficient data from the cloud-based servers in order to further evaluate the efficiency of SortAlgo-Metrics model.

TABLE VIII. MODES OF THE FOUR DIFFERENT FEATURE

Feature Set	Server1 (Modes)	Server2 (Modes)	Server3 (Modes)	Server4 (Modes)
F3	3999.0, 7793.0	1825480.0	2946.0, 5738.0	5999.0, 11690.0
F4	0.104575872	23.04747	0.5668571	.559297, 2.72663
F5	0.2792	37.335299	0.99319999	5.91230
F6	520.9,758.7	520.4,758	25.019, 762.84	529.375, 767.190

TABLE IX. RESULTS BASED ON DIFFERENT SERVERS DATA

Servers	Random Samples	% Result
S1	100	50
	200	53
	1000	55
S2	100	95
	200	97
	1000	98
S3	100	79
	200	82
	1000	89
S4	100	96
	200	97
	1000	96

VII. CONCLUSION

This paper has reviewed different types of methods employed to detect spoofing in a cloud computing environment. We proposed a SortAlgo-Metrics model capable of differentiating between the servers fours with about 96%, except S1 (server one) with 55% as the highest percentage in which we surmised that it could be as the result of insufficient data, although, no proof yet. However, we have further observed that the model could produce a high degree of probability when it takes in a high volume of data, as shown in Table IX. This study has also shown SortAlgo-Metrics model could be capable of producing a basis number for ICMtrics technology for generating a unique digital signature and other systems that are key encrypted-based security. Therefore, in our next phase of this experiment, we shall pull out more data and complex features from the cloud-based servers and employ an advanced multidimensional space analysis in order to evaluate the efficiency of the SortAlgo-Metrics model.

REFERENCES

[1] F. Guo, J. Chen, and T. C. Chiueh, "Spoof detection for preventing DoS attacks against DNS servers," *Proc. - Int. Conf. Distrib. Comput. Syst.*, vol. 2006, 2006.

[2] M. Haciosman, B. Ye, and G. Howells, "Protecting and identifiying smartphone apps using icmetrics," *Proc. - 2014 Int. Conf. Emerg. Secur. Technol. EST 2014*, pp. 94–98, 2014.

[3] H. S. Kang, J. H. Son, and C. S. Hong, "Defense technique against spoofing attacks using reliable ARP table in cloud computing environment," *17th Asia-Pacific Netw. Oper. Manag. Symp. Manag. a Very Connect. World, APNOMS 2015*, no. 2, pp. 592–595, 2015.

[4] B. Ye, G. Howells, M. Haciosman, and F. Wang, "Multi-dimensional key generation of ICMetrics for cloud computing," *J. Cloud Comput.*, vol. 4, no. 1, 2015.

[5] O. A. Osanaiye and M. Dlodlo, "TCP/IP header classification for detecting spoofed DDoS attack in Cloud environment," *Proc. - EUROCON 2015*, pp. 1–6, 2015.

[6] H. Basim and T. Ahmed, "An Improved Strategy for Detection and Prevention IP Spoofing Attack," *Int. J. Comput. Appl.*, vol. 182, no. 9, pp. 28–31, 2018.

[7] N. Tripathi, M. Swarnkar, and N. Hubballi, "DNS spoofing in local networks made easy," *11th IEEE Int. Conf. Adv. Networks Telecommun. Syst. ANTS 2017*, no. January, pp. 1–6, 2018.

[8] O. A. Osanaiye, "Short Paper: IP spoofing detection for preventing DDoS attack in Cloud Computing," *2015 18th Int. Conf. Intell. Next Gener. Networks, ICIN 2015*, pp. 139–141, 2015.

[9] R. Tahir, H. Hu, D. Gu, K. McDonald-Maier, and G. Howells, "A scheme for the generation of strong cryptographic key pairs based on ICMetrics," *Internet Technol. Secur. Trans. 2012 Int. Conferece*, pp. 168–174, 2012.

[10] B. Dhanuja and G. Sathiya, "Software metric LOC data analysis for symmetric key cryptographic technique," *2017 Innov. Power Adv. Comput. Technol. i-PACT 2017*, vol. 2017-Janua, pp. 1–5, 2018.

[11] Y. Kovalchuk and K. Mcdonald-maier, "Overview of ICmetrics Technology – Security Infrastructure for Autonomous and Intelligent Healthcare System," *Sci. Technol.*, vol. 4, no. 3, pp. 49–61, 2011.

[12] S. Yadav and G. Howells, "Analysis of ICMetrics features/technology for wearable devices IOT sensors," *Proc. - 2017 7th Int. Conf. Emerg. Secur. Technol. EST 2017*, pp. 175–178, 2017.

[13] V. R. Basili, R. W. Selby, and T. Y. Phillips, "Metric Analysis and Data Validation Across Fortran Projects," *IEEE Trans. Softw. Eng.*, vol. SE-9, no. 6, pp. 652–663, 1983.

[14] R. C. Zoie, R. D. Mihaela, and S. Alexandru, "An analysis of the power usage effectiveness metric in data centers," *Proc. - 2017 5th Int. Symp. Electr. Electron. Eng. ISEEE 2017*, vol. 2017-Decem, pp. 1–6, 2017.

[15] Y. Gugale, "Super Sort Sorting Algorithm," *2018 3rd Int. Conf. Converg. Technol. I2CT 2018*, pp. 1–5, 2018.

[16] H. Han, J. Wang, K. Liu, and L. Zhou, "Particle Sorting Algorithm Based on Stretching Tensor," *2016 12th Int. Conf. Comput. Intell. Secur.*, pp. 689–692, 2016.

[17] Y. Yang, P. Yu, and Y. Gan, "Experimental study on the five sort algorithms," *2011 2nd Int. Conf. Mech. Autom. Control Eng. MACE 2011 - Proc.*, pp. 1314–1317, 2011.

[18] Vasudevan M. S., B. R. Mohan, and D. K. Damodaran, "Performance Measuring and Comparison of VirtualBox and VMware," *Int. Proc. Comput. Sci. Inf. Technol.*, vol. 27, no. Icicn, pp. 42–46, 2012.

[19] Y. Kovalchuk *et al.*, "Investigation of

properties of ICmetrics features," *Proc. - 3rd Int. Conf. Emerg. Secur. Technol. EST 2012*, pp. 115–120, 2012.

[20] G. Howells, E. Papoutsis, A. Hopkins, and K. McDonald-Maier, "Normalizing discrete circuit features with statistically independent values for incorporation within a highly secure encryption system," *Proc. - 2007 NASA/ESA Conf. Adapt. Hardw. Syst. AHS-2007*, no. Ahs, pp. 97–102, 2007.