CORP: An Algorithm to Prevent Unauthorised Data Modification Using Collaborative Nodes

Alan T Litchfield
alan.litchfield@aut.ac.nz
Service and Cloud Computing Research Lab
Auckland University of Technology

Monjur Ahmed
Monjur.Ahmed@wintec.ac.nz
Centre for Information Technology
Waikato Institute of Technology

ABSTRACT

The COllaborative Redundant Processing (CORP) algorithm is an approach to prevent unauthorised modification of data in a decentralised and distributed computing environment. Built on Ki-Ngā-Kōpuku, a distributed and decentralised security model for Cloud Computing, where redundant nodes are functionally identical, the nodes collectively maintain consistency and integrity of processed data. If a single node is compromised and acts maliciously to modify data, other nodes detect the action. CORP extends the functionality of Ki-Ngā-Kōpuku and is developed mainly for a Cloud Computing context, but the concept can be used in any distributed and decentralised environment to provide consistency, integrity, and availability.

CCS CONCEPTS

• **Security and privacy → Digital signatures**; **Distributed systems security**; • **Software and its engineering → Cloud computing**.

KEYWORDS

Collaborative Processing; Cloud Computing; Cloud Computing Security; Cloud Security Model; Distributed Security; Distributed Computing; Redundancy

ACM Reference Format:
Alan T Litchfield and Monjur Ahmed. 2019. CORP: An Algorithm to Prevent Unauthorised Data Modification Using Collaborative Nodes. In *IEEE/ACM 12th International Conference on Utility and Cloud Computing Companion (UCC '19 Companion), December 2–5, 2019, Auckland, New Zealand*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3368235.3368873

1 INTRODUCTION

In this paper is presented the COllaborative Redundant Processing (CORP) algorithm that is built on Ki-Ngā-Kōpuku principles, a decentralised and distributed software architecture for Cloud Computing. Ki-Ngā-Kōpuku is designed to deliver a secured computing application framework for development by decentralising and distributing application components across a server network. Ki-Ngā-Kōpuku practically makes it difficult for an attacker to identify a specific target because application components are distributed somewhat randomly (no specific or pre-defined pattern) and the with a high level of redundancy, the application framework attains a very high level of resilience and availability. Furthermore, Ki-Ngā-Kōpuku provides no system core or centralised management system [3]. Therefore, a Ki-Ngā-Kōpuku system consists of nodes that host application components, that collectively define the scope of the system. No single node is more capable than any other node, no does any node have higher priority.

CORP addresses issues around system security and unauthorised data modification [7] to provide data consistency and integrity. One of the major aims in deploying security measures is to protect data from being modified by unwanted parties [17], especially in centralised Cloud Computing environments and in distributed and decentralised computing. In the latter case, where more than one server or node may process data, a mechanism to prevent unwanted and unauthorised modification is required for the participating nodes.

As an approach to prevent unauthorised modification of data CORP takes advantage of highly redundant systems, employing multiple nodes that carry out the same task. While processing transactions, nodes determine collectively if processed data is consistent. If a node gets compromised or acts in a malicious way to modify data, other nodes detect the unusual behaviour.

The rest of the paper is structured as follows: Section 2 provides a background for Ki-Ngā-Kōpuku, Section 3 summarises relevant literature, Section 4 that defines the problem, Section 5 outlines the solution to the problem and provides an illustrative example and a logical model. Possible future enhancement of the algorithm is discussed in Section 6.

2 BACKGROUND

While Cloud Computing is an accepted form for the distribution of computing resources and services, the openness of the Cloud architecture leaves many systems open to successful attack. This is shown almost weekly from some very large scale thefts and intrusions, through minute by minute attacks on systems. Tracking and categorising attack vectors provides a realistic view of how and where attacks emerge, highlighting the need for greater resilience in systems architectures and removing the human element where insider actions (whether by accident or malicious) lead to serious data harm or loss).

Primary threat vectors to Cloud Computing appear as technological factors (hardware and software) and human factors [2]. These can be further broken down into the following categories:

UCC '19 Companion, December 2–5, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-7044-8/19/12…$15.00
https://doi.org/10.1145/3368235.3368873

Technological factors
- Hardware: Computing services, internal infrastructure, external network (mobile and fixed)
- Software: Local and Cloud platforms, network protocols, virtualisation, software tools, web services, security systems

Human factors Trust, compliance, regulation, competence, specialisation, SLA misinterpretation, the social context.

From these, systems that provide security apparently have common issues: That exploits are more vulnerable to those that have knowledge of the system architecture and its weaknesses, and that systems typically provide identifiable targets. That is, the greatest threat to most systems are internal actors (whether by accident or malicious) and those people are able to exploit weaknesses in the single point of failure. Therefore, to provide a solution to these issues one needs to address both points effectively.

There are issues in Cloud Computing are not easily addressed, for example multi-tenancy and specifically the common weakness of all Cloud servers; shared technologies that enable successful Denial of Service (DoS), cross VM cache side channel, hypervisor escape, and hyper-jacking attacks [16]. In most cases, a successful attack results in the loss of confidentiality, integrity, and/or availability. Thus, the following terms are defined:

Vulnerability A weakness or flaw in a system that can be exploited in an attack, which is a specific event or series of actions.

Threat An exploitation of any known vulnerability that can result in serious loss of data and information.

Data breach Unauthorised access to systems and networks, such that there is a loss of confidentiality of data within the system (stored or in transit).

Data loss Deliberate or unintentional deletion or manipulation of data such that it loses integrity or availability.

The means by which attacks may be successful include hijacking user account details after obtaining user credentials and information, for example via phishing, DoS, and Man-in-the-Middle attacks. Most sophisticated attacks involve combinations of these methods. These examples illustrate some of the many social engineering opportunities exploited by attackers. Technological weaknesses may be exploited by identifying vulnerabilities in shared technologies, that is, shared CPU cores, high level cache, storage devices, and network interface cards in multi-tenancy environments. Since all Cloud Computing services rely on hypervisors for virtualisation, it is assumed that the hypervisor will provide sufficient isolation. However, all hypervisors contain vulnerabilities (known and unknown) and these provide pathways for attack vectors in exploitation scenarios. For example, unauthorised access can be gained to the hypervisor through the exploitation of code vulnerabilities or configuration flaws. Once inside the network, the attacker can gain control over the network and cloud delivery platform, even giving access to the Virtual Machines (VMs) of other users. The platform characteristics that enable Cloud Computing also provide the opportunities for attackers to hijack user credentials, eavesdrop on information, and control other users' VMs.

It is not the purpose of Ki-Ngā-Kōpuku to provide a solution to the human related factors, but to create a security system that makes it very difficult for an attacker to successfully exploit a cloud services platform. Ki-Ngā-Kōpuku does that by presenting no single point of failure and is self-healing. The former is achieved by applying two approaches: Eliminating the concept of a monolithic software program, and breaking the program down into parts or components, then ensuring a high level of redundancy of the parts so that the program is never without some function when required; the second approach is having no central core or administration module, that is, to decentralise the software architecture and distribute the system's resources across a server network. These approaches provide resilience under attack and ensure availability. Not addressed specifically in this paper, the self-healing property is provided through the use of drone nodes, which are nodes that have no predefined functionality but exist to ensure rapid deployment in the event of an attack. Therefore, this paper presents a solution to the question of how Ki-Ngā-Kōpuku addresses data confidentiality and integrity and shows how collaborative processing enables an effective means for messaging between nodes.

CORP relates to Ki-Ngā-Kōpuku, so a brief description is provided (Figure 1). Any application that is run on Ki-Ngā-Kōpuku is componentised, with program fragments inside node wrappers. A cloud infrastructure is comprised of Cloud Instance of Operating Systems (CIOS) that may be hosted on bare metal or hosted hypervisors and that provide a virtualised environment. The decentralised approach ensures that everything except a user interface exists within the BEC. The types of CIOS are the Front-end CIOS (FEC) and Back-end CIOS (BEC). The FEC provides interfaces for user interaction, application programming interfaces, and data assembly to aggregate data that comes from the Ki-Ngā-Kōpuku nodes. The FEC provides primary and secondary functions and in any event, the functions are triggered by user interactions via devices that may be local or accessed via Internet services. Requests from the FEC are broadcast to the BEC, where they are received by relevant application components and processed. At this stage, integrity verification is required. The BEC aggregates node functions, and provides a system interface to unicast messages and data to the originating FEC. In some instances, secondary FEC requests may be divided into smaller requests by the primary FEC. The divided requests are processed individually and then recombined by the Data Assembler before transmission back to the secondary FEC.

Some rules have been established to ensure Ki-Ngā-Kōpuku achieves the required outcomes. The virtualised CIOS exist on Virtual Machines (VM), which contain nodes. The number and distribution of nodes is not fixed or preplanned, nor are the distribution and number of components within each node. Thus, there must always be a minimum number of component copies available, no single node may contain all application components, no single VM may contain all application components, a node may not contain an application component, and a VM may not contain an application component. Furthermore, the environment is dynamic, therefore provided the rules are maintained, components may be delivered to nodes at any time.

3 RELATED STUDY

In this section, prior research related to decentralisation and collaborative processing are presented. The research presents a mature

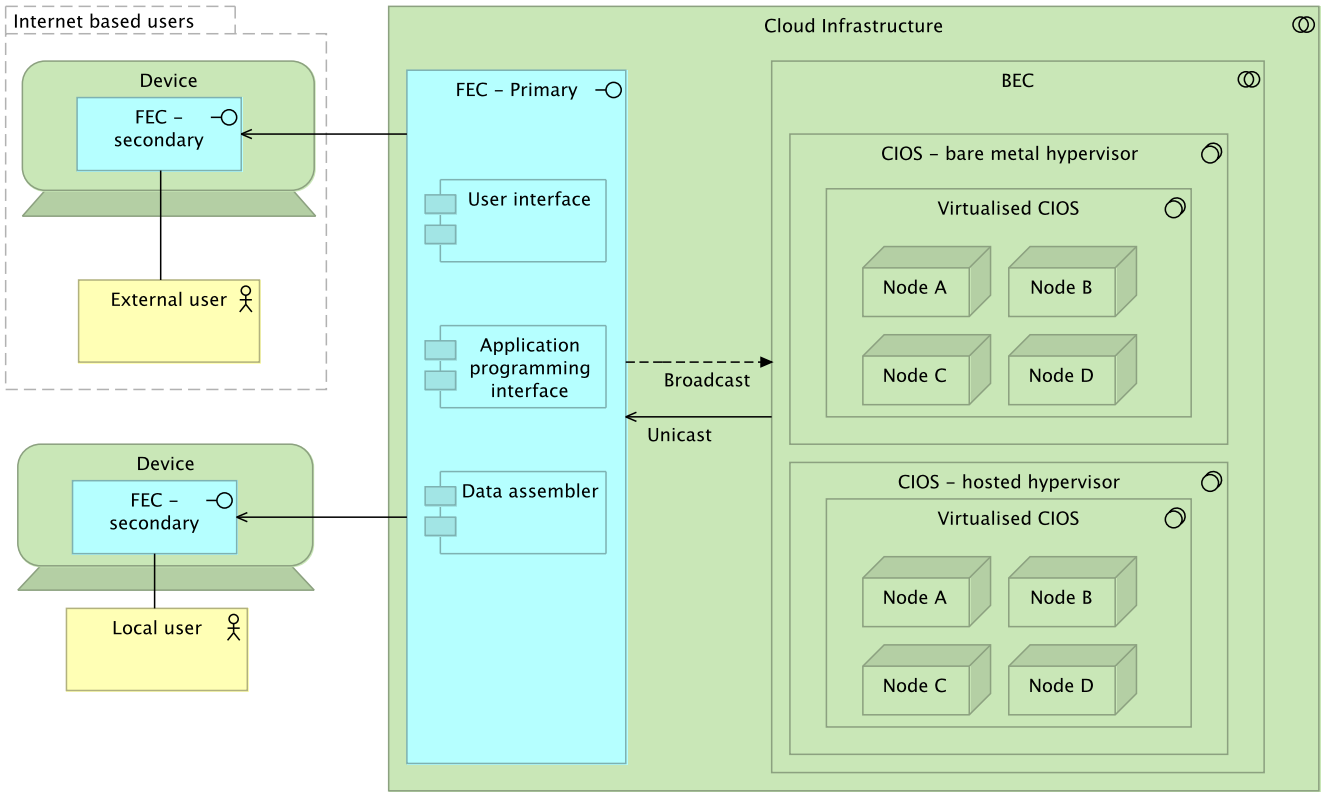

Figure 1: Ki-Ngā-Kōpuku reference architecture

computing landscape, with a wide range of approaches and technologies that are designed to meet the requirements of specific or niche problems. Collaborative computing environments have emerged with the growth of decentralisation and the Internet of Things (IoT), in which low powered edge computing devices rely on other devices to share processing loads.

The concept of collaborative processing and related algorithms cover aspects of data processing on hardware and software. For example, the algorithm proposed by Pedrycz [12] explores the data structure of a data site by comparing sites and to discover structural commonalities or anomalies. While this approach provides evidence of data variation and therefore, a probable attack has occurred, Li et al. [11] discuss a host-based collaborative detection mechanism for distributed computing to counter False Data Injection (FDI) attacks.

In examples that demonstrate how a collaborative approach between system nodes realise sophisticated algorithms, a collaborative content dissemination method for multimedia Cloud is proposed by Chunlin et al. [4] that aims to improve user multimedia experience. Cuomo et al. [5] present an IoT based collaborative reputation system that evaluates and classifies users' behaviour using IoT systems. Also, collaborative processing is often encountered in IoT and sensor networks that are developed to provide research outcomes, such as the decentralised reactive clustering for collaborative processing proposed by Xu and Qi [19] and within the context of sensor networks, Xu and Qi [20] discuss collaborative processing in a mobile agent based computing environment.

Fundamental issues exist at the manifest level of networks, which means that collaborative systems may not access nodes effectively or efficiently. Various solutions apply a statistical or probabilistic approach to the management of node messaging. An approach may be for each node to estimate in advance their energy requirements, based on their local topology, by applying a learning algorithm that uses kernel-linear least-squares regression estimators Predd et al. [14]. A further problem in decentralised systems is complexity, where the level of complex relationships between nodes increases rapidly. Khalili et al. [8] address this issue by applying second order statistical information in the complex domain and uses variations the actual distribution of nodes, plus affine projection learning rules within the nodes, to track variations in statistical information. Kumar and Zaveri [9] present how to use clustering to make energy use more efficient across a heterogeneous IoT environment. They apply a three level approach in which there exists a cluster head that coordinates nodes and allows the exchange of semantics. The nodes do not communicate directly with each other but through this top layer. This approach would not be effective in Ki-Ngā-Kōpuku because Ki-Ngā-Kōpuku is more homogenous, each node is expected to communicate directly, and it is not an IoT sensor network. The concept of a cluster head would make message processing less efficient in Ki-Ngā-Kōpuku.

The collaborative approach in CORP is enhanced by the decentralisation and distribution of processes, as opposed to centralised computing [18]. This approach offers ubiquity, ensures no single point of failure, and provides a solution to a possible problem with communication bottlenecks [6]. Existing solutions demonstrate that decentralised control is suitable for dynamic and distributed large-scale systems, for example Schlegel [15], and are sufficiently robust [10].

4 PROBLEM DEFINITION

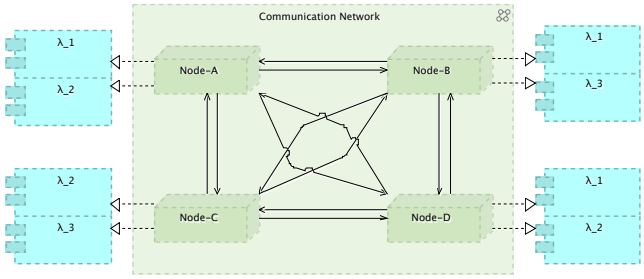

Figure 2: Ki-Ngā-Kōpuku transactional requirements

As previously described, any application running on Ki-Ngā-Kōpuku is divided amongst the nodes on a CIOS. As a form of decentralised computing, CORP works in a computing architecture that has no core controller and therefore, single point of failure. This is illustrated in Figure 2, as is the important point that communications between nodes are negotiated on an almost ad hoc basis because there is no prior knowledge of where application components exist. This makes it difficult for an attacker to gain knowledge of where and how to attack the system.

An Ki-Ngā-Kōpuku environment comprises N_n number of nodes and an application can be divided into λ_n number of components. There must be at least λ_{n+1} copies of each component. Provided the rules for Ki-Ngā-Kōpuku are not broken, there must be sufficient number of nodes to contain and distribute the components. If an application is divided into three components, $\lambda_1, \lambda_2, \lambda_3$ and the components are distributed to a more or less random number of nodes, then the number of nodes must be greater than the total number of components (Figure 2). For example, if an application is made up of 3 components, there must be at least 4 nodes in existence. This is the lower limit of the nodes, and there is no theoretical upper limit. How new nodes are added to the scenario, or how the components are distributed to a new node are out of the scope of this paper, and thus not discussed.

The components are distributed such that no single node may contain all the components, and there must be redundant copies of each component. The components work collaboratively and when a process is associated with a component, for example λ_1, all the nodes are available for processing. Consider the scenario in Figure 2, a process associated with λ_1 may take place in Node-A, Node-B, and Node-D but not in Node-C (the last one does not contain λ_1). Thus it is required that all the nodes containing λ_1 return the same output, ensuring transactional integrity and data consistency.

5 PROBLEM SOLUTION

Traditionally, the most straightforward approach to validating data output is by providing an encrypted hash output that may be compared with other outputs that are reputed to be from the same source, for example, CRC. Any variation in the checksums indicates that the comparison file differs from the source because of file corruption or some other cause. CORP applies this principle to validate transactions processed on nodes. The concept of CORP is that more than one node will undertake the same transaction and that the output from each must be the same. CORP defines the mechanism for data integrity checks across all nodes. This identifies if any node is compromised or attempts to maliciously modify a transaction to produce different output.

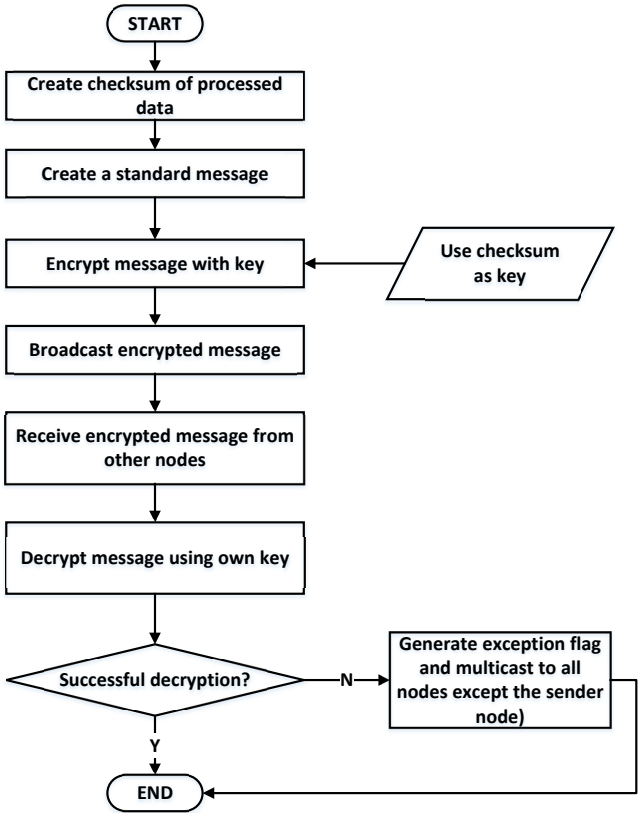

Figure 3: CORP Algorithm

To verify data consistency and integrity, all nodes engaged with the transaction initiate the CORP process. Figure 3 illustrates the algorithm steps. The node first creates a checksum of the processed data using a standard checksum generator algorithm (as there are significant variations in security and efficiency, we assume the choice of algorithm ought to be flexible). The node then creates a standard pre-defined message and uses the checksum as the key to encrypt the message. The encrypted message is then multicast to all other nodes that also processed the transaction. At the same time, the node receives an encrypted message sent from the other nodes and attempts to decrypt the message using its own checksum as a key. Provided data integrity is maintained, the nodes do not

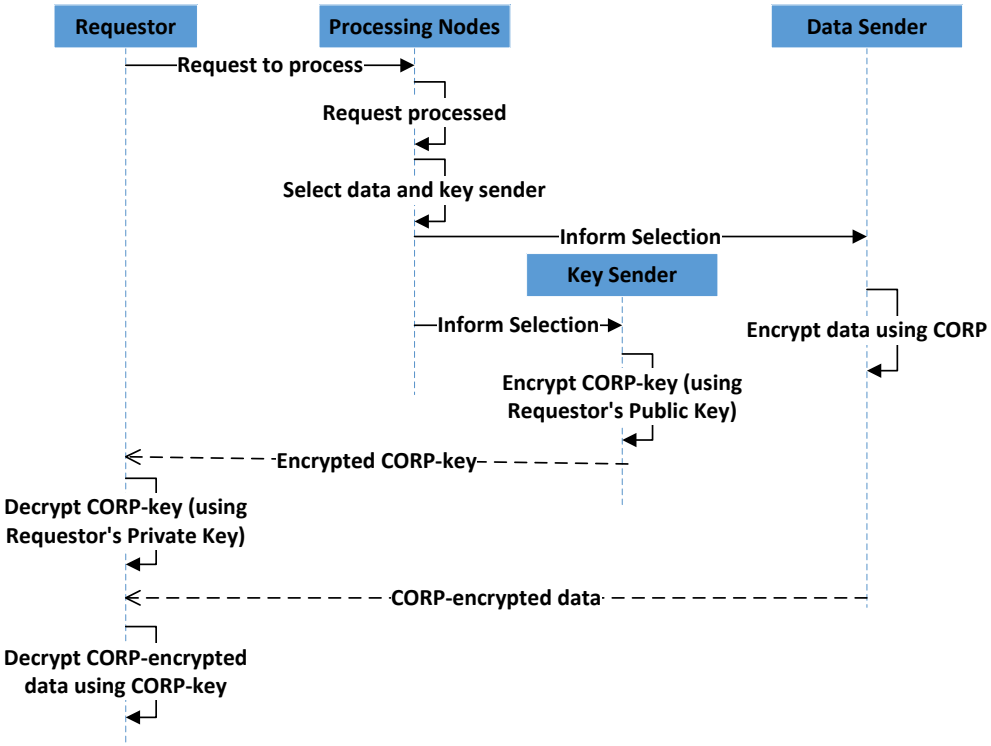

Figure 4: Collaborative Processing using CORP

need to share the key. Thus, successful decryption is achieved if there has been no unauthorised data modification. If a message cannot be decrypted, it would indicate a possible attempt to modify data and the node would generate an exception flag as an alarm. The exception flag is then multicast to all the nodes within Ki-Ngā-Kōpuku, including the nodes not involved in the transaction.

If no exception flag is generated during the CORP process, then no attempt of unauthorised data modification took place. Otherwise, the decision of what to do with the suspected node needs to be made. This is not addressed in this paper, as the mechanism that will deals with a malicious node is out of the scope of CORP algorithm.

CORP is a technologically agnostic algorithm that can cope with changing technological manifestations. It defines the steps to identify unauthorised data modification in generic steps. For example, data can be encrypted using any cryptographic approach. In addition, while Though Figure 3 presents the checksum as a key, the CORP algorithm is not constrained to use it. Alternatives exist, depending on requirement, therefore Figure 4 illustrates the ideal case in which there is no exception flag generated. In the case, multiple nodes (Requestor, Processor, Key Sender, and Data Sender nodes) are process a transaction. The Requestor node initiates a process request, the Processing Nodes process the request and the Key Sender and Data Sender nodes send key and data respectively. From a client-server perspective, the Requestor node acts as the client and the other nodes act collectively. Upon receiving a process request from a Requestor (could an end-user or another system), the Processing Nodes carry out the process and collectively select one node as Key Sender and one node as Data Sender. Finally, the Requestor receives the encrypted data from one node and the key to decrypt the data from another node. The integrity is maintained by using data and the key from different sources.

Referring to Figure 2, let us assume that the component λ_1 in involved in processing a transaction. Since Node-A, Node-B, and Node-D contain λ_1, these three nodes will complete the same process. Once processing is done, the nodes will create a checksum of the processed data. Since all the nodes are working on the same data and carrying performing the same transaction, the created checksums must match. At this point, all the nodes encrypt a pre-defined standard message by symmetric encryption, where the checksum is the key. The encrypted message is multicast across the network by each of the three nodes, Node-A, Node-B and Node-D and received by their counterparts without sharing the key.

In the event of data corruption, if Node-A receives encrypted messages from Node-B and Node-D, and if Node-A successfully decrypts the message from Node-B, but the message from Node-D cannot be decrypted. Then Node-A will suspect an attempt of unauthorised data modification has taken place in Node-D and generates an exception flag. Node-A multicasts the exception flag to Node-B and Node-C. Thus, the process of CORP algorithm ends with either of the two possible outcomes; exception flag or no exception flag.

A logical model that expresses the algorithm illustrated in Figure 3 is presented here. The model assumes that the algorithm completes successfully and that is, that there is no exception flag as in Equation 1a, or there is a flag generated as in Equation 1b. Furthermore, the following conditions are defined for the model:

$CORP$ = The Algorithm
N_n = Number of nodes
D = Successful decryption
$N_n(D)$ = Successful decryption in node N_n
R = Exception Flag

$$\forall N_n : D \Rightarrow R' \tag{1a}$$
$$\exists N_n : D' \Rightarrow R \tag{1b}$$

Equation 1a states that for all the nodes involved in the transaction, if the message is decrypted successfully, then no exception flag is generated. But if an exception flag exists for any node, then some error in processing has occurred either accidentally or maliciously and an exception flag is generated, Equation 1b. According to the condition, even a single exception flag may mean some malicious act, therefore data integrity is maintained if and only if there is no failed decryption and no exception flag.

$$\begin{aligned} \exists N_n(D) &\Leftrightarrow \neg N_n(D') \\ &\Leftrightarrow R' \end{aligned} \tag{2}$$

Thus, the algorithm is expressed by Equation 3.

$$CORP := \forall N_n(D) \Rightarrow R' \oplus \exists N_n : D' \Rightarrow R \tag{3}$$

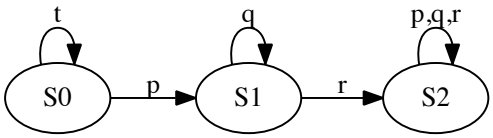

Figure 5: CORP model NFA

The model is evaluated using NFA (Figure 5). The NFA has three states $s0$, $s1$ and $s2$ and four input functions t, p, q, and r. The states are:

$s0$ = nodes with no processing request initiated
$s1$ = processing request initiated, nodes create key and standard message and encrypts the message with key
$s2$ = nodes broadcast and receive encrypted data, decrypts to determine data integrity. This is the acceptance state from which point further processing may carry on.

and the inputs are:

t = no change, this is the initial idle state with no request to process data
p = processing request
q = key creation, encryption of data
r = broadcast, receive and decryption of data.

The NFA can be represented as $M = (Q, Z, R, q, F)$ where:

M = the NFA
q = initial state $s0$
Q = states of the NFA $s0, s1, s2$
F = acceptance state $s2$
Z = input symbols t, r, p, q
R = transition $R : (Q \times Z) \rightarrow Q$

6 FURTHER WORK

At this stage, CORP is not fully optimised and there are some processing overheads associated with CORP. For example, involving all the nodes in processing will make the complexity of the algorithm grow exponentially. To select a random number of nodes for processing instead of involving all nodes, a sub-algorithm is planned. Technological constraints and performance overheads related factors are also to be considered in future developments of CORP.

7 CONCLUSION

CORP is a research in progress. The security model Ki-Ngā-Kōpuku may incorporate CORP to add data security on top of its current features. However, CORP is not constrained to be used only by Ki-Ngā-Kōpuku and can be used in any distributed computing setting. CORP introduces redundant processing which is a transactional overhead, but the trade-off depends on the sensitivity of the data. CORP may be a good fit in the contexts that deal with highly sensitive data that requires security assurance.

REFERENCES

[1] M. Ahmed. 2018. *Ki-Ngā-Kōpuku: A Decentralised, Distributed Security Model for Cloud Computing.* Ph.D. Dissertation. Auckland University of Technology, Auckland, New Zealand. Advisor(s) A. Litchfield and B. Cusack.
[2] M. Ahmed and A.T. Litchfield. 2016. Taxonomy for Identification of Security Issues in Cloud Computing Environments. *Journal of Computer Information Systems* 58, 1 (2016), 79–88. https://doi.org/10.1080/08874417.2016.1192520
[3] M. Ahmed, A. Litchfield, and C. Sharma. 2016. A Distributed Security Model for Cloud Computing. In *Twenty-second Americas Conference on Information Systems.* AIS, San Diego, CA, USA, 1–10.
[4] L. Chunlin, L. Yanpei, L. Youlong, and Z. Min. 2017. Collaborative content dissemination based on game theory in multimedia cloud. *Knowledge-Based Systems* 124, 1 (2017), 1–15.
[5] S. Cuomo, P. De Michele, F. Piccialli, A. Galletti, and J.E. Jung. 2017. IoT-based collaborative reputation system for associating visitors and artworks in a cultural scenario. *Expert Systems With Applications* 79, 1 (2017), 101–111.
[6] A. Farinelli, A. Rogers, A. Petcu, and N.R. Jennings. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, Vol. 2. International Foundation for Autonomous Agents and Multiagent Systems, Estoril, Portugal, 639–646.
[7] L. Ferretti, F. Pierazzi, M. Colajanni, M. Marchetti, and M. Missiroli. 2014. Efficient detection of unauthorized data modification in cloud databases. In *IEEE Symposium on Computers and Communication.* IEEE, Funchal, Portugal, 1–6.
[8] A. Khalili, A. Rastegarnia, and W.M. Bazzi. 2015. Incremental augmented affine projection algorithm for collaborative processing of complex signals. In *International Conference on Information and Communication Technology Research.* IEEE, Abu Dhabi, 60–63. https://doi.org/10.1109/ICTRC.2015.7156421
[9] J.S. Kumar and M.A. Zaveri. 2016. Clustering for collaborative processing in IoT network. In *Urb-IoT'16.* ACM, Tokyo, Japan, 95–97. https://doi.org/10.1145/2962735.2962742
[10] V. Lesser, C.L. Ortiz Jr, and M. Tambe. 2012. *Distributed sensor networks: A multiagent perspective.* Springer Science & Business Media, New York.
[11] B. Li, R. Lu, W. Wang, and KK.R. Choo. 2017. Distributed host-based collaborative detection for false data injection attacks in smart grid cyber-physical system. *Journal of Parallel Distributed Computing* 103, 1 (2017), 32–41.
[12] W. Pedrycz. 2002. Collaborative fuzzy clustering. *Pattern Recognition Letters* 23, 1 (2002), 1675–1686.
[13] B. Pon. 2016. *Blockchain will usher in the era of decentralised computing.* London School of Economics and Political Science. Retrieved 9 Sept., 2019 from https://blogs.lse.ac.uk/businessreview/2016/04/14/blockchain-will-usher-in-the-era-of-decentralised-computing/
[14] J.B. Predd, S.R. Kulkarni, and H.V. Poor. 2007. A Collaborative Training Algorithm for Multi-Sensor Adaptive Processing. In *2nd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing.* IEEE, St. Thomas, VI, USA, 297–300. https://doi.org/10.1109/CAMSAP.2007.4498024
[15] T. Schlegel. 2010. *Decentralised Agent-based Resource Allocation in Open and Dynamic Environments.* Ph.D. Dissertation. PhD Thesis, Faculty of Computer Science and Information Technology ....

[16] A. Shahzad and A. Litchfield. 2015. Virtualization Technology: Cross-VM Cache Side Channel Attacks make it Vulnerable. In *Australasian Conference on Information Systems*. AIS, Adelaide, Australia, 1–16.
[17] B. Singh. 2012. *Network Security and Management*. PHI Learning, New Delhi.
[18] G. Stevenson, L. Coyle, S. Neely, S. Dobson, and P. Nixon. 2005. ConStruct. A Decentralised Context Infrastructure for Ubiquitous Computing Environments. In *Information Technology and Telecommunications (IT&T) Annual Conference*. Ulster University, Cork Institute of Technology, Ireland, 26–27.
[19] Y. Xu and H. Qi. 2004. Decentralized Reactive Clustering for Collaborative Processing in Sensor network. In *Proceedings of the Tenth International Conference on Parallel and Distributed Systems*, Nian-Feng Tzeng (Ed.). IEEE, Newport Beach, CA, USA, 54–61. https://doi.org/10.1109/ICPADS.2004.1316080
[20] Y. Xu and H. Qi. 2006. Mobile Agent Migration Algorithms for Collaborative Processing. In *Proceedings of the Wireless Communications and Networking Conference 2006*. IEEE Communications Society, Los Vegas, NV, USA, 2324–2329.