Each VM has its OS and applications. A VM, also called a guest machine, is a virtual representation or software emulation of a hardware platform that provides a virtual operating environment to guest OSs.

Moving a VM from one physical hardware environment to another is called migration. If the migration is carried out, so that connected clients perceive no service interruption, it is considered a live VM migration. For example, database consolidation is made easier if VM's are not shut down before they are transferred. The method is also used for administrative purposes. For example, if a server needs to be taken offline for some reason, live transferring of VM's to other hosts can be used to pass running VM's between cloud sites over wide-area communication networks.

The research gap identified the need to improve the security of live VM migration while providing an efficient operation; aspects such as reducing the total migration time, minimising service interruption during migration, and enhancing migration security, have been essential issues since the proposal of live VM migration. For instance, the next stage of live migration begins after copying the working set to the destination server. In this phase, any modified memory pages are transferred to the destination server. This step is referred to as the memory copy process, where the remaining modified memory pages are replicated for the 'test VM' on the destination server. It is important to note that this replication carries a risk of potential compromise. This article focuses on mitigating this aspect of live VM migration in terms of data integrity and availability. The main beneficiaries of the enhancements proposed are organisations running multi-tenant data centres offering VM services.

This article also explores the potential for migrating VM's either while they are being transit or when they are located at the source and destination points in the live VM migration procedure. Based on inquiries, we present a novel framework to ensure the secure live VM migration in real-time. Our proposed framework incorporates a vTPM agent along with six additional, namely Input/Output, Data Plane, Integrity Analyser, Data organisation, Go Agent, and Libvirt Agent. While existing studies [7–10] has developed a framework for ensuring the integrity of cloud systems through live VM, a closer analysis of various research types has revealed a gap in empirical evidence and understanding. There is a need for more information to determine which specific issues hold the greatest significant in theses domains.

Additionally, the objective of this article is to assess the importance of the identified issues to answer two specific research inquiries.

Research Question 1	How do we design, implement, and establish a live VM migration framework to protect the integrity of cloud systems and evaluate its effectiveness?
Research Question 2	How might the information revealed by the above question affect the framework's integrity?

The overarching aim is to develop and design a secure live VM migration framework that improves the integrity and availability of live VM migration from one VM to another in the same platform, using the same hardware features and the same hypervisor (Xen hypervisor).

The remainder of this article is structured as follows: Section 2 discusses related work and the research's motivation. Section 3 explains the system architecture and details the components of the proposed framework (Kororā). Section 4 presents an evaluation of the system architecture. Section 5 elaborates on the implementation process and steps for Kororā. Section 6 discusses the research findings. Finally, section 7 includes the conclusions and a brief discussion about future work.

2. Related work and research motivation

Critical concerns for cloud users include protecting workloads and data, as well as ensuring security and integrity for VM images launched on CSPs [11,12]. To achieve live VM and workload data protection, cloud-user organisations require a framework for securely placing and using their workloads and data in the cloud. Current provisioning and deployment frameworks typically involve storing the VM and application images and data in the clear (i.e., unencrypted) or having these images and data encrypted keys controlled by a service provider, usually uniformly for all tenants.

In a multi-tenant cloud environment, VM images, which effectively serve as containers for OS and application images, configuration files, data, and other entities require confidentiality protection. These images must be encrypted and decrypted by keys under tenant control in a transparent manner to the CSP. The critical step in a planned migration is to take snapshots that preserve the state and data of a VM at any given time. These snapshots are copies of the VM in each state and are stored for later use. During migration, the snapshot is then migrated to the destination cloud, where the hypervisor creates a new VM with the same configuration as the snapshot. Once the target VM is up and running, the source cloud redirects the incoming traffic of its VM to the destination VM [13,15,15].

Some of the research relevant to this field is described below.

Data Deduplication is a useful process for live VM migration that helps reduce the migration time by transferring only the altered memory material on the source server [7]. It involves two main components: the Dirty Block Tracking (DBT) mechanism and the Diff format. DBT records all the operations that cause changes in the VM disk image, while the Diff format stores the reported data. DBT labels each changed disk page as a dirty file, and only those pages identified by the DBT are migrated to the storage leaving the rest behind. This technique is particularly useful for VMs undergoing multiple migrations, resulting in multiple destination servers. This research employs the data deduplication process, which effectively eliminates redundant data copies and substantially reduces the storage capacity needed. Enabling this process in the proposed framework is applicable at the volume level, and the volume can be made available using either cluster shared volumes file system or NTFS. However, it is important to note that the performance is not guaranteed, and the process is only supported for cold data, which refers to files that are not currently open). In the implementation phase of this research, the process assumes that live VMs are built using diff-disks, where a read-only golden image serves as the parent. The setup of this process within the proposed framework results in considerable storage space savings. However, when Kororā is employed to replicate the live VMs, each chain of diff-disks is treated as a single entity and replicated as such. Consequently, during the migration process, three copies of the golden image will be present as part of the replication. To address this, the data deduplication process becomes crucial in reclaiming the space utilized by these duplicated copies.

The WAIO (Workload-Aware Input/Output Outsourcing) scheme proposed by Yang et al. is designed to improve the efficiency of live processing during VM migration [8]. This scheme involves outsourcing the working set of the VM to surrogate devices and creating a separate I/O path to serve VM I/O requests. During the live storage migration process, VM I/O requests from the original storage are outsourced to the surrogate device, which services them separately and more efficiently. The WAIO prototype implementation and experiments conducted by Yang et al. demonstrate that this approach improves I/O performance during the VM migration process and enables faster VM migration without sacrificing I/O performance. This piece of work contributes to this research proposed framework significantly and helps Kororā resolve the performance-related issues to detect and pinpoint as several layers of a highly complex physical and virtual hardware stack are involved. As it comes to Kororā performance, for most mission-critical live VM migrations, mainly databases or email solutions, the expectation is very high to get the best VM performance possible as it directly impacts the accuracy of service. The research proposed framework was inspired by the WAIO prototype's method of utilizing I/O requests from the primary storage. The goal of this framework is to address issues related to running VM migration, such as achieving accurate runtime and minimising disruptions. Otherwise, alternative approaches would involve

increasing the number of CPU cores or adding more virtual processes for each VM or upgrading the hardware of the host machine with a lower frequency.

Riteau et al. [9] proposed a live VM migration system, called Shrinker which allows VM clusters to migrate between data centres linked via a network. This system operates on the principles of handling distributed information and allows chunks of VMs to be migrated to multiple data centres across different servers. Shrinker differs from traditional live VM migration methods as it allows source and destination server hypervisors to interact during migration. By integrating data duplication and cryptography hash functions, Shrinker reduces the data that needs to be migrated. The work on opportunistic replay aims to reduce the amount of data migrated in low-bandwidth environments [14]. This approach keeps a record of all types of user events that occur during the execution of the VM. Then, this information is transferred to an identically manufactured VM and put into effect to produce almost the same state as the VM source. In addition, the changes that were made after the reply was transferred and applied resulted in an identical surrogate VM. The utilization of the Shrinker method enables this research to employ an integrated approach that involves data duplication and a cryptographic hash function. This approach facilitates the seamless migration of running VMs from one host to another within the proposed framework, minimising perceived downtime. As a result, Kororā becomes more flexible, and the running VMs are not solely dependent on a single host machine. This leads to a highly available and fault-tolerant framework for Kororā, albeit with a certain level of percentage.

According to Zheng et al. [10], they introduce a fresh scheduling algorithm aimed at enhancing the performance of I/O storage during wide-area migration. This algorithm stands out because it takes into account various factors related to the storage I/O workload of individual VMs, including temporal location, spatial location, and popularity characteristics. By considering these aspects, the algorithm calculates optimal schedules for data transfers, resulting in improved efficiency. This research inspired by Zheng et al. put forward an algorithm that delivers substantial performance advantages for a diverse set of benefits across a wide range of popular VM workload, reduce latency, and improve overall resource utilization.

Berger et al. [18] discuss a vTPM that provides trusted computing for multiple VM's running on a single platform. The key to this process is finding a way to store vTPM data encrypted in the source platform and restore them safely in the in-destination platform, as well as a way to protect the integrity of the transferred data in the process of live vTPM-VM migration, where it is vulnerable to all the threats of data exchange over a public network. These include leakage, falsification, and loss of sensitive information in the VM and vTPM instances.

This article presents an improved live VM migration framework that utilizes distinctive values, referred to as "flag", to indicate problematic data within the new system. In other words, when users find a flag in a specific record, they know that the migrated record contains information that could not be loaded immediately. For each such example, the original data from the legacy system is preserved in a standard format and linked to the new record. The user can quickly check the source to interpret the data meaningfully.

By inspiring from Berger's research, the proposed framework enables the acquisition of the target VM's working set data while migrating to the Kororā platform. This feature provides the Kororā continuous access to the dataset during the migration, while the I/O migration process focuses on interacting with the original disk. As a result, it becomes possible to significantly reduce the traffic between the I/O processes and the Kororā platform, ultimately enhancing the overall integrity of live VM migration.

3. System architecture

The IT security framework is supported by tools that enable service providers to bridge the gap between control requirements, technical issues, and business risks. The proposed framework, called Kororā, is capable of measuring and preserving the integrity of live VM migration and increasing the level of integrity among various physical hosts. Kororā allows users to check malicious files against three different malware providers' engines, and it can compare indicators such as hashes, URLs, IP addresses, and domains from other resources.

This section aims to explain the system requirements, represented from a design perspective, by using an intermediate model of logical architecture. The goal of the Kororā system architecture is to fulfil the following system elements requirements.

- *System Element 1 – Integrity of configuration files:* In this case, the VM image structure can represent a complete file system for a given platform's integrity (e.g. 'vbox' files in virtual box or '. vmx' files in VMware). However, both of these files can be edited by a third party to make changes in the configuration of VM's.
- *System Element 2 – Virtual hard disk integrity:* The VM image's life cycle consists of various states such as creation, starting suspension, stopping, migration, or destruction. Typically, VM images are loaded from a storage location, such as a hard disk drive, and run directly from a VMM that does not understand the quality of integrity (e.g. '. vmdk', '. vdi', '. ova' files). As such, third parties can make changes to these files after running them in their environment. Since the actual OS holds the file, it would be easy to place a Trojan or any malicious program in the file.
- *System Element 3 – The integrity of the data files on the VM, including all confidential and system files:* The VM is loaded from the storage location, and the VM image may not comply with the intended settings and configurations needed for proper implementation of each environment. Furthermore, the VM image itself could be distorted (perhaps by an insider) or maliciously modified. This research found two ways to analyse these files before migration – 'supply the data files' and 'system files hashsum' – and then check them after migration.

3.1. System architecture requirements

The Kororā system architecture focuses on a hypervisor that preserves metadata using cryptography and hashing algorithms. The protected live VM migration framework based on this hypervisor was designed to identify possible attacks and perform an independent secure migration process.

The approaches of live VM migration are generally divided into three classes: 1) Migration of the process; 2) Migration of memory; 3) Suspend/resume migration. In this research, live VM migration means migrating a VM from a source host to a destination host while protecting against four key migration attacks (detailed in section 6.1). These requirements must be incorporated into the secure live VM migration platform.

Prior to commencing the migration process, it is crucial to ensure that the source hosts, destination hosts, and VMs satisfy the migration requirements that Kororā aims to meet. It is also important to verify the accuracy of the target destination and the effectiveness of the access control policies, including the cryptography rule, to protect the live VM migration process. In case an unauthorized user or role initiates the live VM process and the migration process, access control lists in the hypervisor can prevent unauthorized activities.

During the migration process, an attacker may initiate Man-in-the-Middle (MiTM) attacks using route hijacking or Address Resolution Protocol (ARP) poisoning techniques. The source and destination platforms need to perform mutual authentication during live VM migration to avoid MiTM attacks (authentication). An encrypted network must be set up so that no data can be accessed from the VM content by an intruder, and any software alteration can be detected correctly. This will help prevent active attacks on live migration, such as memory

manipulation, and passive attacks, such as sensitive information leakage (confidentiality and integrity). An intruder may intercept traffic and later replay it for authentication in the process of the VM migration. Therefore, the method of live VM migration should be immune to replay attacks. For example, nonces in Java applications can help with the password for migration authorisation, as well as the public key to the machine where the user is sitting to provide the correct command that is transmitted to the server during migration, preventing playback attacks (reply to resistance). The source host cannot deny the VM migration activity, and this feature can be achieved by using public key certificates (source non-repudiation).

This framework is orthogonal to existing live migration approaches – including the Zehang et al. [9], and Mashtizadeh et al. [16] live migration patents, and the Fan Peiru [17] vTPM-VM live migration protocol – and it is a secure boost layer for most, if not all, VM live migration schemes. In addition, this framework can improve the security of other VM tasks, such as those associated with the virtualisation and the virtual networking layers, which may experience the same data integrity problem as VM live storage migration. This research framework and the three frameworks named above exploit the secure live migration characteristics, but they improve the VM migration security in different ways. For example, the scheme of Zheng et al. [9] aims to significantly reduce the total amount of data transferred by exploiting the workload of the VM's locality. Rarely updated data blocks are differentiated from frequently updated data blocks in virtual disk images by analysing the workload position. The rarely updated data blocks are transferred in the migration before the frequently updated data blocks; therefore, the re-transmissions of data blocks are minimised, thus reducing the total amount of data transmissions. While this current research framework secures the live VM migration, its methodology is entirely different from that of Zehang [9].

The responsibilities of the seven agents are as follows.

- *Virtual Trusted Platform Module Agent:* A vTPM agent provides trusted computing for multiple VM migrations on a single platform [18]. Moving the vTPM instance data along with its corresponding VM data is essential to keep the VM security status synched before and after a live vTPM-VM migration process. The key to this process is creating ways to safely store and restore the vTPM instance data encrypted in the source system and destination platform. In addition, it needs to protect the integrity of the transferred information in the process of live vTPM-VM migration, as the migration of a VM over the internet is vulnerable to all the threats of data exchange over a public network. Current live VM migration schemes only check the hosts' reliability and integrity, neglecting the verification process for the vTPM-VM to be moved and the vTPM-VM container. This poses a considerable security risk for vTPM-VM migration. To solve this problem, the proposed framework uses vTPM to securely boot the VM(s) over the hypervisor (Xen hypervisor) (see Fig. 1, Label 1).
- *Input/Output Agent:* The I/O agent redirects the necessary I/O requests to the replacement device from the operating VM itself. It redirects all write requests on the replacement device to minimise I/O traffic to the original replacement device [19]. Meanwhile, the I/O redirects all the popular read requests identified by the *Data Plane Agent* (Next Agent) to the replacement device. Suppose the replacement device has only partial data for a request. In this case, the I/O issues read requests to the original replacement device and merge the original device's data into the replacement device. Either the original storage device [19] or the replacement device can be redirected to the read requests from the migration module. While the original storage device generates most of the virtual disk images, the replacement device provides the modified chunks (units of information containing either control information or user data). Because of the VM workload locality, most requests will be routed to the original storage device (see Fig. 1, Label 2).
- *Data Plane Agent:* This module moves different memory contents from one host to another host (e.g., kernel states and application data). Therefore, the transmission channel must be secured and protected from attack. All migrated data are transferred as precise data without encryption in the live VM migration protocol. Hence, an attacker may use one of the following techniques to position himself in the transmission channel to execute a MiTM attack: ARP spoofing, DNS poisoning, or route hijacking [20,21]. These attacks are not theoretical. Tools such as Xensploit work against Xen and VMware migration [22] (see Fig. 1, Label 3).
- *Integrity Analyser Agent:* This agent aims to determine standard migration processes and decompose them into operational-level activities to make the migration process more transparent. This agent provides the core mechanism of integrity verification to assist

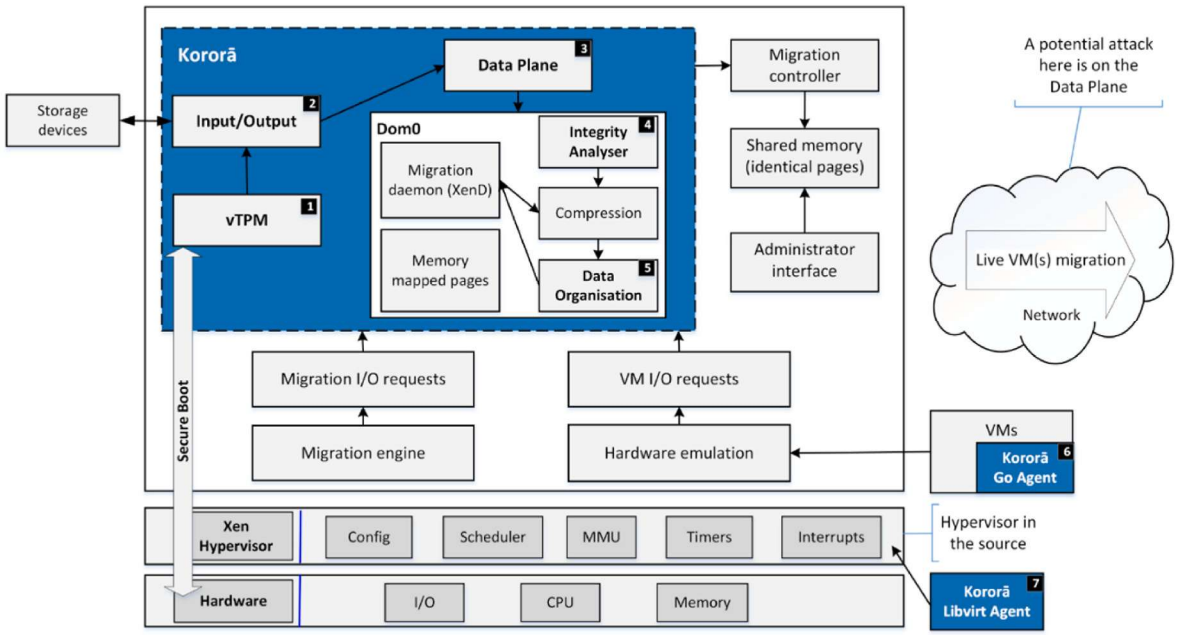

Fig. 1. System architecture of the proposed framework.

Kororā, particularly with migrating live VM data to the cloud. This agent uses the Clark and Wilson (CW) [23] security model as a fundamental theory for specifying and analysing an integrity policy for the proposed Kororā. It adopts the CW model to live VM migration, focusing on the subjects, objects (see Section 4), and their data exchange of users' applications to enhance the live VM migration mechanism's security level and provide user convenience (see Fig. 1, Label 4).

- *Data Organisation Agent:* In the virtual disk images, the data organisation agent monitors reading request's popularity from the live VM itself. Only the popular data blocks that will be read are outsourced to the replacement device. Since the replacement device serves all write requests, monitoring the popularity of write requests is not required. Each virtual disk image of the running VM is divided into fixed-size chunks, and the data organisation agent records each chunk's access frequency. If the access frequency exceeds a predefined threshold for a particular chunk, the entire chunk will be outsourced to the replacement device. The replacement device will serve all the subsequent accesses to this chunk, which removes their I/O involvement with the migration process. The migration module usually scans the entire virtual disk files by submitting read-only requests. Most of these requests will only be issued once, except for requests that read dirty data blocks (see Fig. 1, Label 5).
- *Go Agent:* The Go Agent is a secure, lightweight process that manages the VM interaction with the hypervisor controller. It has a primary role in enabling and executing hypervisor VM extensions, which allow the post-deployment configuration of the VM, such as installing and configuring software. In addition, VM extensions enable recovery features such as resetting the administrative password of a VM. Without the Go Agent, VM extensions cannot be run in Kororā. Go Agent in Kororā is like the Azure VM Agent's role, which is to install by default on any Windows- or Linux-based systems and provides valuable features, such as local administrator password reset and script pushing (see Fig. 1, Label 6).
- *Libvirt Agent:* Kororā uses Libvirt Agent as its application programming interface (API). This package adds support for virtualised systems to install and manage large numbers of Unix system configurations automatically. It is particularly suitable for sites with very diverse and rapidly changing configurations. Further, the Kororā system includes synchronisation markers that allow the host physical machine to force a guest VM back into synch when issuing a command; as Libvirt Agent already uses these markers, guest VM's are able to discard any earlier pending undelivered responses safely (see Fig. 1, Label 7).

To conclude this section, several secure, small, and innovative live migration framework designs such as TrustVisor and CloudVisor have been proposed to address migration security. However, these designs either have reduced functionalities or pose substantial restrictions to the VM's. In contrast, Kororā relies on a trusted hypervisor to provide the security guarantee (integrity) and has several unique characteristics.

- Kororā does not reduce functionalities or pose substantial restrictions to the VMs
- Kororā is addressing the threats from a complex hypervisor to VM data
- Kororā reduces VM's and hypervisor TCB based on a microkernel approach
- It is not required to reimplement the VMs and hypervisor from scratch, which is challenging to maintain
- Kororā saves VMs migration features and does not allow the migration features to be lost
- Kororā supports encryption-based protection
- Kororā validates features like Paravirtual I/O
- Kororā uses seven agents running on the Xen privileged 'dom0' and communicating with the Xen hypervisor.

In addition, when applying the system design agents in the Xen hypervisor environment, it is essential to take into account the following requirements.

- 64-bit x86 computer with at least 1 GB of RAM (a server, desktop, or laptop) and a trusted platform module chipset on the motherboard. The TPM hardware must be activated through the BIOS.
- Intel's virtualisation technology or AMD-V support (optional for paravirtualisation [PV], required for hardware VM and some PV optimisation).
- Sufficient storage space for the Kororā framework's dom0 installation.
- Extensible firmware interface –helps the hardware layer select the OS and get clear of the boot loader. In addition, it helps the CSP to protect the created drivers from a reverse-engineering (back-engineering) attack.
- Software requirement cmake – this is the main additional product necessary for compiling a vTPM. To manage domains with vTPM, libxl should be used rather than 'xm' which does not support vTPM.
- Linux host (Ubuntu 12.4) must be installed on the machine.

4. Evaluation of the system architecture

One primary aim of our integrity framework is to consider the entire cloud integrity environment and capture all potential integrity attributes and elements as evidence, including functional and non-functional elements. Evaluation is a crucial analytical process for all intellectual disciplines and different evaluation methods can provide information about the CSP's complexity and ubiquity [26]. This article aims to identify a set of necessary evaluation components and apply them to evaluate the Kororā migration framework method, reviewing the secure establishment framework and analysing its weaknesses and strengths. The evaluation of the Kororā system architecture provides a theoretical foundation for developing a secure live VM migration framework [27]. The evaluation process is shown in Fig. 2, representing an overview of the evaluation components and their interrelations and helping to establish a clear pathway for this research.

The main objective of using the evaluation theory in this study is to achieve a comprehensive and reliable integrity level in live VM migration processes. Additionally, this theory provides a clear and formal description of the evaluation components, as depicted in Fig. 3.

The above concepts are discussed in detail below.

- *Target*: ensuring integrity between CSPs and cloud service users (CSUs).
- *Criteria*: integrity elements of the CSPs and CSUs that need to be evaluated.
- *Yardstick/standard*: the ideal secure live VM migration framework measured against the current secure live VM migration framework.
- *Data-gathering techniques*: critical or systematic literature review is necessary to obtain data to analyse each criterion.
- *Synthesis techniques:* to be used to access each criterion and, therefore, access the target, obtaining the evaluation result.
- *Evaluation processes:* a series of tasks and activities used to perform the evaluation.

4.1. Layered system architecture

The proposed framework is illustrated in Fig. 1 and includes subjects, objects, access attributes, access matrix, subject functions, and object functions (see Fig. 4). Several terms are used in the proposed layered model: Identifier (I) is either an 'HTTP' or 'HTTPS' uniform resource identifier or an extensible resource identifier; Relying Party (RP) is an action that a CSP takes to obtain proof that the end-user controls an identifier; OpenID Provider (IDP) is an authentication server on which

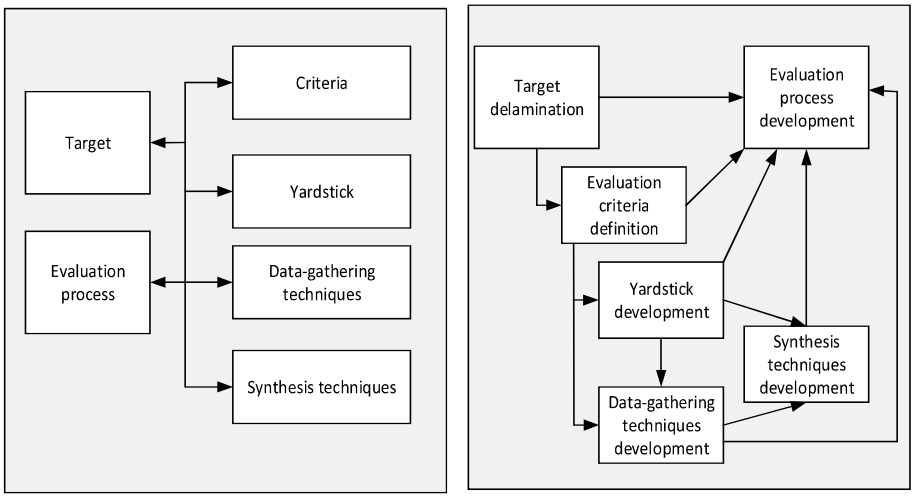

Fig. 2. Components of evaluation and the interrelationships between them [27].

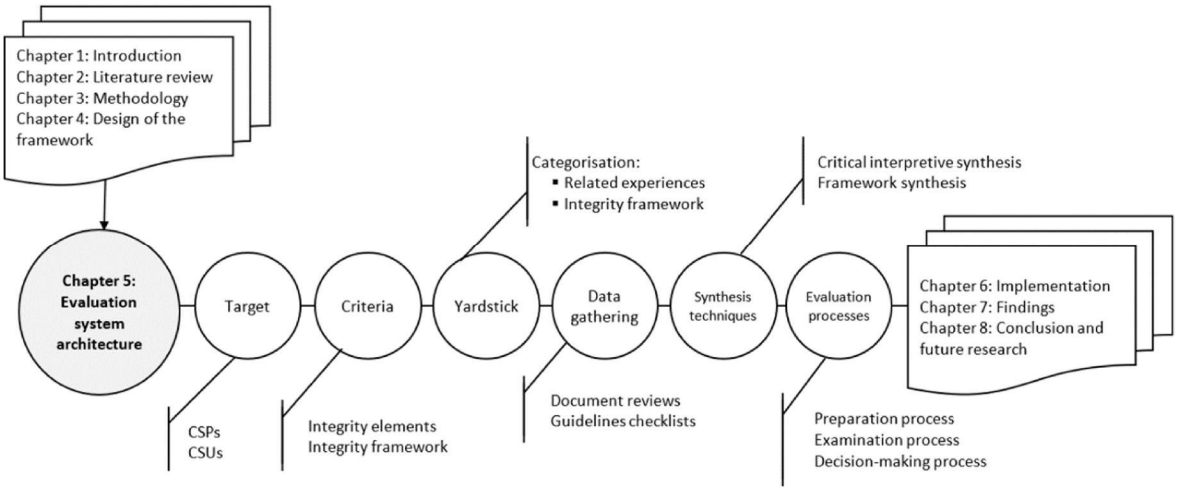

Fig. 3. The concepts of evaluation theory in this study's development of the Kororā framework.

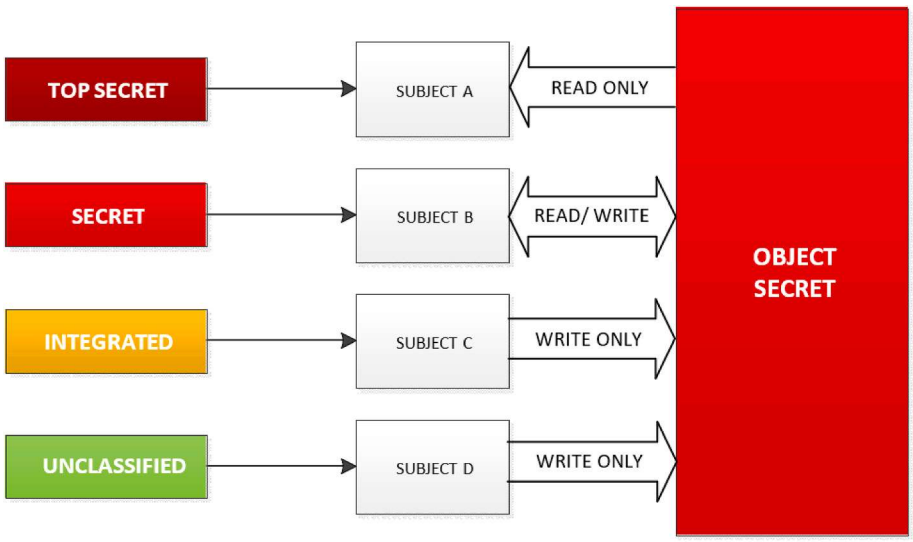

Fig. 4. The relationships between the objects and subjects.

an RP relies if the end-user controls an identifier; User Agent is the end-user that runs a VM migration process called UA; Trust Authority (TA); and Endpoint URL (IEU). Access attributes are defined as Read, Write, Read/Write, and Execute. In the access matrix, each member represents the access authority of the subject to the object.

The proposed layered model works as follows.

1) t ∈ T, where T has sorted Quaternion, element of T is denoted using "t"
2) $T = (a, B, c, D)$, where
3) $a \subseteq (S \times O \times A)$.
4) B is an access matrix, where $B_{ij} \subseteq A$ signifies the access authority of s_i to o_i.
5) c ∈ C is the access class function, denoted as $c = (c_s, c_o)$.
6) D signifies the existing hierarchy of the proposed framework
7) S is a set of Subjects
8) O is a set of Objects
9) A = [r, w, a, e] is the set of access attributes
10) e*e*: R × T → I × T shows all the roles in the proposed framework, in which "e" is the system response and the next state, R is the requests set, and I is the arbitrary set of requests, which is [yes, no, error, question]. In this study, the question is important because if the response is equal to the question, it means that the current rule cannot deal with this request.
11) ω = [e_1, e_2, …, e_s], ω is the list of exchange data between objects:

W (ω) ⊆ R × I × T × T.
(R_k, I_m, T*, T) ∈ W (w).

if $I_m \neq$ Question and exit a unique J, 1≤ j ≤ s, it means that the current rule is valid, subject and object also are valid because the object verifies the vTPM of the other object (attester) by request (challenge) for integrity checking. Consequently, the result is,

(I_m,$t*$) = e_i(R_k,t), which shows for all the requests in the "t" there is a unique response, which is valid

Where, a ⊆ (S × O × A) where S is a set of Subjects, O is a set of Objects, and A = [r, w, a, e] is the set of access attributes.

12) c_s is the security level of the subject (includes the integrity level $c_1(S)$ and category level $c_4(S)$). Fig. 4 shows the security level in the proposed framework and the relationships between the subjects and objects. c_o signifies the security function of objects. Fig. 4 illustrates the relationship among the subjects, objects, security functions, and the proposed framework's security level.
13) Under the presumption that the hardware (the lowest layer) comprising the user agents (the highest layer) is valid, the integrity of user agents can be guaranteed if and only if the integrity of the lower layer, by using vTPM, is checked and the transition to higher layers occurs only if the integrity check on those layers is complete. Therefore, the integrity level is $c_1(TPM)$, $c_2(TA)$, $c_3(IDP)$,$c_4(RP)$, and level $c_5(UA)$; this study should prove that each state of the proposed framework is secure. It has been assumed that each state is secure except for state 3, called Data Plane (see Fig. 1). Therefore, if state 3 is secure, all the states are secure.
14) Σ (R, I, W, z_0) ⊂ X × Y × Z
15) (x, y, z) ∈Σ (R, I, W, z_0), if (z_t,y_t,z_t,z_{t-1}) ∈ W for each t ∈ T, where z_0 is the initial state. Based on the above definition, Σ (R, I, W, z_0) is secure in all system states; for example, (z_0,z_1,…,z_n) is a secure state.
16) The CW model has several axioms (properties) that can be used to limit and restrict state transformation. If the arbitrary state of the system is secure, then the system is secure. The simple-security property (SSP) [28] is adopted in this study. This property states that an object at one level of integrity is not permitted to read an object of lower integrity.
17) t = (a, B, c, D)
18) Satisfies SSP if,

For all s ∈ S, s ∈ S ⇒ [(o∈ a (s: r, w)) ⇒ ($c_s(s)$, > $c_o(o)$)]
i.e., $c_1(s) \geq c_2(o)$, $c_3(s) \supseteq c_4(o)$..
$c_1(G) \geq c_2(vTPM)$, $c_1(IEU) \geq c_2(RP)$.

Based on Figs. 1 and 4, and the SSP axiom, all the objects of Kororā use two primary concepts to ensure the enforcement of security policy: well-informed transactions and separation of duties. The integrity axiom is 'no read down' and 'no write up', which means a subject at a specific classification level cannot read and write to data at a lower or higher classification, respectively. Other models such as the Star property, Discretionary security, and Compatibility property can also be used to limit and restrict state transformation, and these models will be utilized in future work.

5. Kororā implementation: agent-based detection and reaction system

Kororā allows users to check malware files against three different malware providers' engines, and it can check indicators of comparison details of hashes, URLs, IP numbers and domains from various resources.

5.1. Secure resource management and integration plan

Kororā treats resources as black-box entities, and the protection plane components are considered standards and 'as is'. They can only be accessed through vendor-specific APIs to send alerts from protected resources to the security manager and submit commands to alter the protected resources' actions or internal state. The system manager is directly connected to Kororā agents to translate their APIs to Kororā APIs.

5.2. Agent plan: functionality enforcement and policy refinement

The core objectives structure of the gent layer is applied differently depending on whether agents refer to Kororā APIs, other agents, or a system manager. The detection and reaction system hierarchy are based on root agents that construct slave objects recursively. Different functions are defined for enforcing multiple agent-related functionalities as described in the framework (see Fig. 1), including 1) applying the alert aggregation policy to received alerts; and 2) refining the reaction policies, as follows.

a) Kororā's alert aggregation policy is implemented through the handler's alert handling feature. This callback is triggered every time a slave object sends a warning message. Various activities are possible, such as forwarding the 'raw warning' to the parent entity or 'correlating multiple warnings' before notifying the parent; and
b) The refinement of policy (defined in the policy agent function) is implemented whenever an agent receives an alert from its parent.

Agent can interact with a system manager or other agents, and the interactions with the security plane depend entirely on the commodity API of its components. As a result, there is a one-to-one mapping between the system manager APIs and agent callbacks. Additionally, interactions among agents are generally described as dependent on time, and this brings about synchronisation aspects that are vital to ensure the safety of systems. Although agents are independent, they may require the results of other agents' computations, whether collaboratively or competitively, to reach the outputs, such as transforming policies into sub-policies for slave objects to follow.

Kororā has a detector agent, and any failure of a detector agent directly impacts the framework's security. In the dispatcher case, issued warnings are aggregated, combined, and then forwarded to Kororā. Similarly, Kororā can refine the reaction policy chosen by an agent using the callback function as a decision-making handler to enforce the