

CephVault: A Secure Key Management System (KMS) for Ceph

by

Subhabrata Rana

Bachelor of Technology in Computer Science and Engineering,
Maulana Abul Kalam Azad University of Technology, India, 2010

A Thesis Submitted in Partial Fulfilment of
the Requirements for the Degree of

Master of Computer Science

in the Graduate Academic Unit of Computer Science

Supervisor(s): Kenneth B. Kent, PhD., Faculty of Computer Science

Examining Board: Kalikinkar Mandal, PhD., Faculty of Computer Science
Saqib Hakak, PhD., Faculty of Computer Science
Argyri Panezi, PhD., Faculty of Law

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

March, 2024

© Subhabrata Rana, 2024

Abstract

Organizations are leveraging cluster storage solutions to address expansive storage requirements. Ceph is a reliable and massively scalable cluster solution that supports object, block, and file storage capabilities on commodity hardware without a single point of failure. Despite growing popularity, the absence of native object encryption support in Ceph raises concerns about potential security vulnerabilities and data compromise. *CephArmor*, a cryptography interface, was previously developed to provide data confidentiality in Ceph while data is at rest. In this work, we propose a secure Key Management System (KMS), *CephVault* that can support key generation for various encryption schemes and key lengths required by *CephArmor*. *CephVault*, which supports twelve phases of a KMS life cycle, is developed as an intrinsic component of Ceph. We demonstrate that the proposed solution provides better features and security than other KMSs, making *CephVault* a competitive and preferable choice to many existing KMSs available in the Ceph ecosystem.

Dedication

I dedicate this thesis to my beloved wife, Rituparna, who has been the cornerstone throughout my academic journey. She is the one who believed in me and motivated me to pursue a research-based master's degree, which I could hardly dare to dream of. Without her constant support, patience, unconditional love, and guidance, I would not have come this far.

Acknowledgements

I would like to express my gratitude to Dr. Kenneth B. Kent for agreeing to supervise me and allowing me to work with and learn from the exceptional researchers at the Center for Advanced Studies (CAS)-Atlantic. His constant support, encouragement, valuable advice, and profound knowledge have been the guiding beacon for my research. He has been my inspiration throughout my academic journey, and I am fortunate to be supervised by him.

I am incredibly grateful to Stephen MacKay, who interviewed me and provided me with an opportunity to be part of the CAS-Atlantic group. His technical acumen, prowess in reviewing manuscripts, and patience with lengthy esoteric drafts have been vital for my thesis. I thoroughly enjoyed working with him, and I can confirm that he is one of the best managers I have ever worked with.

I am equally thankful to Brett Kelly, the product manager from 45Drives, for his generous support, patience, and technical guidance and for providing opportunities to learn Ceph in depth.

I would like to thank Dr. Kalikinkar Mandal and Dr. Saqib Hakak for their teachings, which are extremely helpful for this research work. They have been very helpful in clearing my doubts on many topics, and I could employ some of the learning in

my research works. They have been very supportive and provided me with valuable advice, and they have always been reachable, even beyond the scheduled classroom time.

I want to take this opportunity to thank all the researchers at CAS-Atlantic who have provided me with valuable guidance, especially Md. Alvee Noor, who has been my true mentor even before I joined UNB. His valuable advice on time management helped me greatly. I am thankful to Georgiy Krylov, who helped me to be a better programmer; Hassan Arafat, whose advice helped me greatly to understand critical topics; Fatemeh Khoda Parast, who guided me throughout the research and helped me to be a better writer; Maria Patrou, who helped me greatly to cope with technical and psychological challenges during the research.

I would like to thank the system admins, DeVerne Jones and Aaron Graham, for their timely support in resolving various infrastructure-related problems. I would like to acknowledge the support of Mitacs and 45Drives for funding this research. With their generous support, this research was conducted.

Table of Contents

Abstract	ii
Dedication	iii
Acknowledgments	iv
Table of Contents	vi
List of Tables	xi
List of Figures	xiii
Abbreviations	xv
1 Introduction	1
2 Background	4
2.1 Key Management Life cycle	4
2.2 Basic Principles of Information Security	7
2.3 Fundamental Security Design Principles	8
2.4 Cryptographic Primitives	10
2.4.1 KMS for Symmetric Cryptography	10
2.4.2 Envelope Encryption	11
2.4.3 KMS for Asymmetric Cryptography	12
2.4.4 KMS for Hybrid Cryptography	13

2.5	Hardware Accelerated AES Instructions	15
2.6	Hardware-Based Cryptography Modules	16
2.6.1	Benefits of Using Hardware-Based Cryptography Modules	16
2.6.2	Various Hardware-Based Cryptography Modules	17
2.7	Key Generation	18
2.8	Common Security Vulnerabilities and Attacks	21
2.9	Ceph Architecture	23
2.10	Our Research Contribution	26
2.11	Federal Information Processing Standards	27
2.11.1	Security Requirements for a Cryptography Module Adhering to FIPS 140-2	28
2.11.2	Security Requirements for a KMS Adhering to FIPS 140-2	29
2.12	Threat Model	31
2.13	Summary	31
3	Related Work	32
3.1	Research Method	32
3.2	KMS Solutions Supporting Ceph (Officially)	33
3.3	KMS Solutions Not Supporting Ceph (Officially)	33
3.4	Summary	33
4	Design and Implementation	37
4.1	CephVault Architecture	37
4.2	CephVault Threat Model	39
4.3	Programming Language For CephVault	41
4.4	Design Pattern for CephVault	42

4.5	CephVault Modules	43
4.6	CephVault Configurable Operations	53
4.6.1	Options for CephVault Modes of Operation	53
4.6.2	Options for Location of the KEK	55
4.6.3	Options for Envelope Encryption	56
4.7	Processing of Cryptography Requests	57
4.7.1	Data Encryption Request (Per-User)	57
4.7.2	Data Encryption Request (Per-Object)	62
4.7.3	Processing Decryption Requests (Per-User and Per-Object)	62
4.8	Summary	65
5	Performance Analysis	66
5.1	Experimental Setup	66
5.2	Evaluation Methodology	67
5.3	Performance Analysis of CephArmor with CephVault	67
5.4	Performance Analysis of CephVault	68
5.4.1	Dependency on Data Sizes	68
5.4.2	Dependency on Modes of Operation	71
5.4.3	Dependency on Location of the KEK	74
5.4.4	Dependency on Envelope Encryption Schemes	75
5.4.5	Dependency on CephVault's Internal Events	76
5.4.6	Dependency on Use Cases	80
5.4.7	Dependency on Number of Records in the Database	83
5.5	Summary	83
6	Storage Analysis	85

7	Security Analysis	87
7.1	Information Security Principles	87
7.2	Security Design Principles	87
7.3	Defense Against Various Attacks	89
7.4	Comparative Analysis of Popular KMSs	89
7.5	Evaluation of CephVault’s Cryptography	
	Module Adhering FIPS 140-2	95
7.6	Evaluation of CephVault as a KMS Adhering FIPS 140-2	95
7.7	Security Analysis of Location of Keys	97
	7.7.1 User’s Home Directory	97
	7.7.2 Configuration File	98
	7.7.3 Ceph	99
	7.7.4 TPM	100
7.8	Threat Analysis	101
7.9	CephVault Risk Assessment	103
7.10	Summary	106
8	Future Work	107
8.1	Future Directions For Development	107
8.2	Future Direction for Evaluation	111
8.3	Summary	114
9	Conclusion	115
	Bibliography	138
A	Codes	139
A.1	Integration of CephVault with CephArmor	139
A.2	Scalability of CephVault	141

Vita

List of Tables

2.1	Comparative Analysis: AES Encryption Modes	12
2.2	Various Hardware Modules Supporting Cryptography Operations . .	17
2.3	Summary of Security Requirements for a Cryptographic Module [1] .	30
3.1	KMSs Supporting Ceph: Contributions, and Limitations	34
3.2	KMSs Not Supporting Ceph: Contributions, and Limitations	35
3.3	KMSs Not Supporting Ceph: Contributions, and Limitations (Con- tinued)	36
4.1	Countermeasures to Language Vulnerability	42
7.1	Adherence to Information Security Principles	88
7.2	Adherence to Information Security Principles (Continued)	89
7.3	Adherence to Fundamental Security Design Principles	90
7.4	Adherence to Fundamental Security Design Principles (Continued) . .	91
7.5	Countermeasure Against Various Attacks	92
7.6	Countermeasure Against Various Attacks (Continued)	93
7.7	The Column Names and their Respective Representations	94
7.8	Comparative Analysis of KMSs with CephVault	94
7.9	Evaluation of Cryptography Module in CephVault for FIPS 140-2 . .	96
7.10	Evaluation of Cryptography Module in CephVault for FIPS 140-2 (Continued)	97
7.11	CephVault Risk Assessment	104

7.12 CephVault Risk Assessment (Continued) 105

List of Figures

2.1	KMS for Symmetric-Key Cryptography	10
2.2	Hybrid KMS with Symmetric Encryption Scheme and Hash Function	14
2.3	Hybrid KMS with an Asymmetric Encryption Scheme, a Symmetric Encryption Scheme and a Digital Signature	15
2.4	Key Generation in TPM [2]	20
2.5	Ceph Architecture (Vanilla)	24
2.6	Integration of Ceph with CephArmor and CephVault	27
4.1	CephVault Architecture	38
4.2	CephVault Threat Model	39
4.3	CephVault Key Destruction Process	48
4.4	CephVault Commands to Get Status	52
4.5	CephVault Configurable Options	54
4.6	Software-Supported Envelope Encryption	56
4.7	Hardware-Supported Envelope Encryption	58
4.8	Processing of Data Encryption Request: Per-User	59
4.9	Processing of Data Encryption Request: Per-Object	63
4.10	Processing of Data Decryption Request: Per-User and Per-Object . .	65
5.1	Dependency on Data Sizes on Encryption Operations	69
5.2	Dependency on Data Sizes on Decryption Operations	70
5.3	Comparative Analysis: Per-User Vs. Per-Object	73
5.4	CephVault Events During Cryptography Operations	78

5.5	Dependency on Use Cases	82
5.6	Dependency on Number of Records	84
6.1	Storage Requirements for CephVault	86
7.1	Shamir's Secret Shares	99
7.2	Security of Pool Contains Shamir Secret Shares	100

Abbreviations

AD	Active Directory
AES	Advanced Encryption Standard
AWS	Amazon Web Services
CBC	Cipher Block Chaining
CD	Compact Disk
CephFS	Ceph File System
CRUSH	Controlled Replication Under Scalable Hashing
CRC	Cyclic Redundancy Check
CM	Configuration Management
CSP	Critical Security Parameter
CSPRNG	Cryptographically-Secure Pseudorandom Number Generator
DES	Data Encryption Standard
DEK	Data Encryption Key
DSA	Digital Signature Standard
EAL2	Evaluation Assurance Level-2
ECC	Elliptic-curve cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standards
FPGA	Field Programmable Gate Array
GlusterFS	GlusterFS Gluster File System
GKMP	Group Key Management Protocol
HSM	Hardware Security Module
HDD	Hard Drive
IV	Initialization Vector
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
K8s	Kubernetes
KDC	Key Distribution Center
KDF	Key Derivation Function
KMS	Key Management System
KMIP	Key Management Interoperability Protocol
KEK	Key Encryption Key

LDAP	Lightweight Directory Access Protocol
LUKS	Linux Unified Key Setup
LAN	Local Area Network
MON	Ceph Monitor
MTA	Mail Transfer Agent
MitM	Man-in-the-Middle
NAS	Network Attached Storage
NIST	National Institute of Standards and Technology
NRBG	Non-Deterministic Random Bit Generator
NVDATA	Non-Volatile Data
OSD	Object Storage Daemons
OPT	One Time Password
OOP	Object-Oriented Programming
PAM	Pluggable Authentication Modules
POSIX	Portable Operating System Interface
PPs	Protection Profiles
PKI	Public Key Infrastructure
PRNG	Pseudorandom Number Generator
PPs	Protection Profiles
PRE	Proxy Re-encryption
PKI	Public Key Infrastructure
PRNG	Pseudorandom Number Generator
PUF	Physical Unclonable Function
RSA	Rivest Shamir Adleman
RGW	RADOS Gateway (RGW)
RADOS	Reliable Autonomic Distributed Object Store
RBD	RADOS Block Device
RNG	Random Number Generator
SDS	Software-Defined Storage
SELinux	Security-Enhanced Linux
SP	Security Provider
SAN	Storage Area Network
SIEM	Security Information and Event management
SSD	Solid State Drive
SSP	Secure Session Parameter
Triple DES	Triple Data Encryption Standard
TTP	Trusted Third Party
TPM	Trusted Platform Module
UE	Updatable Encryption
UAF	Use-After-Free

Chapter 1

Introduction

Every year, data breaches are rising, costing millions of dollars [3]. A strong encryption mechanism is crucial for safeguarding data-at-rest and data-in-transit. The National Institute of Standards and Technology (NIST) approved different security mechanisms, such as symmetric-key algorithms and asymmetric-key algorithms, that can provide data security [4] [5]; however, adopting a strong encryption mechanism is not sufficient for data protection. Encryption algorithms are not kept secret; instead, encryption keys are kept secure from adversaries [6]. Well-known encryption algorithms such as Data Encryption Standard (DES), Triple Data Encryption Standard (Triple DES), Advanced Encryption Standard (AES), Rivest Shamir Adleman (RSA), Elliptical Curve Cryptography (ECC), etc. [7] are difficult to break, so adversaries find it convenient to attack the Key Management System (KMS) to retrieve encryption keys. A weak KMS can lead to unauthorized access to the organization's sensitive data and the compromise of trade secrets and business-critical data, which can eventually cause the collapse of the entire organization. An organization's safety, security, and data confidentiality depend on the proper safeguarding of the encryption keys. The KMS should not only safeguard keys from external threats but also be resilient against inside attackers and accidental exposure of the keys. The respon-

sibility of a KMS is to manage the entire life cycle of the keys, which comprises key generation, key registration, key usage, key storage, key distribution, key deletion, key destruction, key rotation, key revocation, and key monitoring.

There is a massive growth in data generated by organizations worldwide, which is expected to grow beyond 180 zettabytes by 2025 [8]. To address the rising demand for massive storage, organizations employ cluster storage solutions to manage and store data. Ceph [9] was developed by Sage A. Weil in 2007 to address the growing need for scalable, high-performance storage solutions. Ceph is a software-defined storage (SDS) technology that supports block, file, and object storage. It is highly reliable as there is no single point of failure. Ceph's Controlled Replication Under Scalable Hashing (CRUSH) algorithm is a storage solution that separates data and metadata, which replaces the need for maintaining a centralized metadata table to record the location of the saved data. The CRUSH algorithm computes the location of the data to be read or to be written in real time, making the read-write operation fast and fault-tolerant. Ceph also employs self-managing and self-healing features. When there is a failure, Ceph can identify the particular component responsible for the failure, and it can recover the data without administrator's intervention.

Despite providing a robust storage solution, Ceph lacks native object encryption. Ceph offers software encryption of the disk level, but the objects themselves are not encrypted. Ceph does not support highly secure encryption schemes, and Ceph does not have a reliable Key Management System (KMS). It solely relies on access control and authentication as the default security mechanism for its data security, making it vulnerable to data leakage in cases where any adversary compromises data [10][11].

A lightweight cryptography API, *CephArmor* [12] was developed by our research

team to encrypt and decrypt data in Ceph. *CephArmor* requires a robust key management system to safeguard data at rest. The primary objective of this research is to develop a secure and performant KMS, *CephVault* that would support various encryption schemes and different lengths of encryption keys required by *CephArmor*.

Chapter 2

Background

This chapter includes the necessary information that is instrumental for understanding *CephVault*.

2.1 Key Management Life cycle

The key management life cycle includes various operations that are performed during the lifetime of the encryption keys. The number of phases in the KMS life cycle depends on applications and use cases [13]. A typical KMS life cycle is described with their applicability and related research below [14].

- **Key Generation:** Key generation is the first phase of the KMS life cycle. This is one of the most challenging phases since the generated key must be of proper length and sufficiently unpredictable to foil adversaries [15]. Keys can be generated in a deterministic or non-deterministic fashion [16] or from biometric attributes such as DNA sequence, fingerprint, or the iris of the eye [17].

- **Key Registration:** Encryption keys are random numbers, and prior to the use of the encryption keys, useful metadata, such as key ID, creation date, deletion date, key version number, and name of the key creator or modifier, must be associated with the keys. Keys also can be registered with the registration authorities [18].

- **Key Usage:** Encryption keys are used as input to cryptography modules performing encryption-decryption and hashing operations. Key usage policies must be in place to regulate a key's usage and to protect against unauthorized access to the data. For different cryptography operations or for different applications, separate keys should be used [19].

- **Key Storage:** Once encryption keys are generated, they must be stored in a location completely separate from the encrypted data [20]. Based on the usage type, keys can be pre-operational keys, which are generated and stored prior to being used in cryptography operations, or operational keys, which are actively used for cryptography operations. Pre-operational keys are saved offline in memory, in hard copy, or on disk. Since operational keys are accessed in real-time, they must be encrypted before storing them in the storage system. Key wrapping techniques [21] are generally employed to encrypt keys before storing them.

- **Key Distribution:** Keys are distributed using different methods, such as group key distribution [22], Diffie-Hellman key distribution [23], or chaos synchronization [24]. For multicast communication, the Group Key Management Protocol (GKMP) is employed [25]. For the distribution of encryption keys, a Key Distribution Center (KDC) can act as an intermediary between group members. In pairwise key distribution, each group member participates in encryption key generation and distribution [26]. The primary challenge in the key distribution process is that the recipients may not be honest, and adversaries can act as recipients of the keys, so it is imperative to establish trust between the sender and the receiver prior to key distribution. Since accepting encryption keys from untrusted sources poses a security threat, dishonest participants must be prevented [27][28].

- **Key Rotation:** Keys must be changed periodically as using a single key for a longer period increases the risk of being identified and hence compromises the entire system [29]. One of the major issues during key rotation is that the encrypted data

needs to be decrypted with the old key before being encrypted with the new key; however, an updatable encryption technique [30, 31, 32] can change the old ciphertext to a new one without decrypting the data.

- **Key Backup:** Keys can be replicated after their creation in an offline or online manner. Backed-up keys can help in the event of accidental key loss, destruction, or deletion. Key backup represents an important stage of the KMS life cycle as it plays a pivotal role in ransomware prevention [33].

- **Key Recovery:** The key recovery process, also known as key escrow, grants trusted parties access to the encryption key [34]. The trusted parties can be KMS providers, government establishments, or business organizations. The key recovery process is crucial for recovering encrypted data whose keys are lost or stolen. The recovery key is generated before the actual usage of the key [35], and a trusted party can be given the recovery key [36]. There are different ways key escrow can be implemented [37], such as storing the recovery information with the encrypted data [38] or splitting the recovery information among multiple recovery agents [39].

- **Key Revocation:** A revocation procedure must be performed if the key is compromised by either replacing the compromised keys or updating the compromised key so that the old one becomes ineffective [40]. During the key revocation and key replacement processes, applications, devices, and users using the compromised keys can be negatively affected.

- **Key Destruction:** When keys are no longer needed, they should be destroyed along with any backup to adhere to the data reservation policies implemented by international authorities [41]. Keys can be programmed to self-destruct after the expiration of the predefined timeline [42].

- **Key Audit Trail:** An audit trail is required for organizations to be compliant with international regulations. An audit trail may contain sensitive information that can leak the user's privacy, so the audit trail should be encrypted, and a search func-

tionality should be enabled on the encrypted audit logs [33].

- **Key Monitoring:** The entire life cycle of KMS should be monitored for any unauthorized access to the system. Monitoring different phases can prevent the installation and distribution of corrupt keys. A corrupt key can jeopardize the KMS, and data loss can be irreversible. Apart from the security aspect, monitoring can help to improve system performance.

2.2 Basic Principles of Information Security

An application that implements various phases of the KMS life cycle described in Section 2.1 should follow various security principles to safeguard data [43]. The basic principles of information security that influence the design of *Ceph Vault* are discussed in this section. The National Institute of Standards and Technology (NIST) standard Federal Information Processing Standard (FIPS) 199 categorized confidentiality, integrity, and availability as the CIA triad, which is the fundamental principles of information security used for evaluating the security objectives of a system, where C stands for confidentiality, I stands for integrity, and A stands for availability. However, the extended CIA triad included authenticity and accountability for comprehensibility [44].

- **Confidentiality:** Confidentiality ensures data is private and not disclosed to any unauthorized person. Without confidentiality, data can be compromised.

- **Integrity:** Integrity protects data against improper modification. Without integrity, data can be altered by an unauthorized person.

- **Availability:** Availability facilitates timely data access. Without availability, data is deemed to be unreliable and could lead to service disruption.

- **Authenticity:** Authenticity ensures the genuineness of the user by employing authentication mechanisms, such as username, password, biometrics, etc. Authenticity

verifies users and ensures that the data transmission is genuine. Unauthenticated users can leak confidential information.

- **Non-repudiation:** Non-repudiation ensures that the origin of the data can be verified by a third party. With non-repudiation, an entity can not deny its involvement in an activity that was done by that entity previously.

- **Accountability:** Accountability tracks the activity of an entity and ensures that the entity is responsible for its actions.

2.3 Fundamental Security Design Principles

Developing a secure application mandates the following design principles to countermeasure software vulnerabilities and prevent potential attacks. *CephVault*'s design is based on the following fundamental security design principles listed by The National Centers for Academic Excellence for Information Assurance/Cyber Defense (a collaboration effort between the U.S. National Security Agency and the U.S. Department of Homeland Security) [44].

- **Economy of Mechanism:** Make the design simple and small to perform thorough tests. A complex design can lead to attackers exploiting security vulnerabilities in the design.

- **Open Design:** Make the design of an application available to all rather than keeping it secret, which enables security experts to review the design and identify potential vulnerabilities in the design.

- **Separation of Privilege:** Divides an application into various modules and different privileges are employed in various modules to perform the desired task.

- **Least Privilege:** Ensures that a method, process, or user can perform a task with the least privileges and absolutely necessary privileges. Role-based authentication is an example of the least privilege principle.

- **Least Common Mechanism:** Ensures minimum shared functions between users, reducing unintended operations and creating the shortest communication path between tasks.
- **Psychological Acceptability:** Provides guidelines for user adaptability of an application without facing any additional challenges. Security mechanisms developed for the system must not hinder the primary duty of the user who interacts with the system. Without psychological acceptability, the system may become burdensome to users, and users are likely to commit mistakes while using the application.
- **Isolation:** Keeps sensitive data in a restrictive enclosure (can be software or hardware). Isolation restricts public access or unauthorized access to the data, processes, and memory space.
- **Encapsulation:** Binds data and methods together in a single unit. Encapsulation provides an additional layer of security, where data from one class is hidden from other classes and can be exposed only to the intended classes through proper access specifiers.
- **Modularity:** Groups common functionalities in separate, secured modules. Modularity provides reusable functionalities and easy update or upgrade options.
- **Layering:** Secures data with overlapping security mechanisms. Failure of one mechanism should protect the data.
- **Least Astonishment:** Designs an application so that users feel comfortable using the application without being surprised. The design should be transparent for the user to understand the security goals that are mapped to the security features developed for the applications.

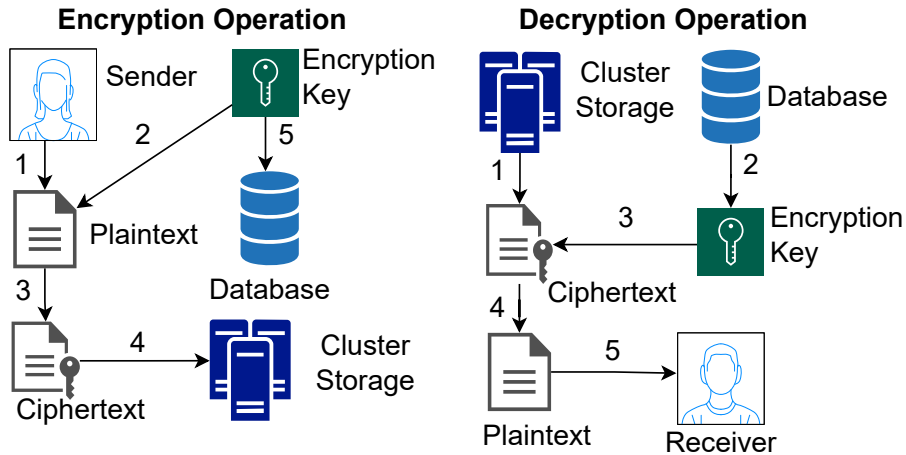


Figure 2.1: KMS for Symmetric-Key Cryptography

2.4 Cryptographic Primitives

Cryptography primitives can be employed in a key management system to enforce information security principles. For example, an encryption scheme can provide security, a hash function can provide integrity, and a digital signature scheme can provides non-repudiation, and integrity to a KMS. A KMS can support various cryptography schemes, such as symmetric key encryption schemes, asymmetric key encryption schemes, or hybrid encryption schemes, where a combination of different cryptography and non-cryptography schemes are supported by the KMS. Some of the KMS based on the various cryptography primitives that are relevant to *CephVault* are discussed below.

2.4.1 KMS for Symmetric Cryptography

In a symmetric-key cryptography-based KMS, the same key, a Data Encryption Key (DEK), is used to encrypt and decrypt the data at rest [45]. The encryption and decryption operations for data are depicted in Figure 2.1.

For encryption, the plaintext provided by the sender is encrypted with the encryption key to generate the ciphertext. The ciphertext is stored in the cluster storage, and

the encryption key is stored in the database for future decryption requests. In the decryption process, the decryption key, retrieved from the database, is used for ciphertext decryption.

2.4.2 Envelope Encryption

In Figure 2.1, we depicted symmetric-key cryptography-based KMS, where the DEK is stored in the database in plaintext. Storing the DEK in plaintext in the database reveals the key, so the key must be kept in the database always in the encrypted format. In an enveloped encryption scheme, the DEK is encrypted with another encryption key called Key Encryption Key (KEK). We can again encrypt the KEK with another encryption key, and so on, and with decryption, we need to unwrap the encrypted key. Symmetric encryption algorithm AES is secure for post-quantum cryptography, provided the key size should be increased, making the AES algorithm a better choice for encryption of the DEK, with future extensibility [46]. AES can be operated in various modes. Some of the modes pertaining to this thesis are discussed below.

- **AES-CBC (Cipher Block Chaining):** In AES-CBC (Cipher Block Chaining) mode, the plaintext is divided into fixed-size blocks (16 bytes), and each block of the plaintext is encrypted with an AES block cipher. When the plaintext is not a multiple of 16 bytes, the last block is padded before the encryption. In this mode, the previous block's ciphertext influences the next block's encryption, such that the previous block's ciphertext is XORed with the next block's plaintext, making it a chain of blocks. The use of padding in AES-CBC mode can be vulnerable to padding oracle attacks [47].

- **AES-CTR (Counter):** AES-CTR (Counter) mode converts the AES into a stream cipher, and it divides the plaintext into fixed-sized blocks of 16 bytes. The encryption process requires a unique Initialization Vector (IV) and a fixed counter

Table 2.1: Comparative Analysis: AES Encryption Modes

Attributes	AES-CBC	AES-CTR	AES-GCM
Plaintext is divided into blocks of 16 bytes.	Yes	Yes	Yes
Dependency on the previous block for encryption	Yes	No	No
Provides confidentiality	Yes	Yes	Yes
Provides integrity	No	No	Yes
Padding required for the last block	Yes	No	No
Vulnerable to padding oracle attacks	Yes	No	No
Block cipher transformed into stream cipher	No	Yes	Yes

value, which can be zero. Since each block does not depend on the previous block for encryption, parallelization can be possible. If the plaintext is not a multiple of 16 bytes, AES-CTR mode encrypts the partial block of plaintext, IV, and counter value the same as it encrypts the entire block of plaintext. Since no padding is used, AES-CTR mode is not vulnerable to padding oracle attacks.

- **AES-GCM (Galois/Counter Mode):** In AES-GCM (Galois/Counter Mode), the plaintext is divided into fixed-sized blocks; however, unlike AES-CBC and AES-CTR mode, the block cipher is transformed into a stream cipher. Unlike AES-CBC, AES-GCM mode encrypts each block independently. Unique data, known as Additional Authentication Data (AAD), provides integrity, where, like IV, AAD can be public. AES-GCM mode does not require padding even for the last block of the plaintext, which is not 16 bytes, making it secure to padding oracle attacks. A comparative analysis of various AES encryption modes are described in Table 2.1.

2.4.3 KMS for Asymmetric Cryptography

An asymmetric-key encryption scheme, also known as public-key cryptography [48], uses a private-public key pair bounded by a mathematical principle for cryptography operations [49]. When the public key encrypts the data, the resulting ciphertext can only be decrypted by the corresponding private key. Asymmetric encryption is slower than symmetric encryption schemes and generally used to encrypt data-in-

transit [50].

2.4.4 KMS for Hybrid Cryptography

Employing traditional symmetric or asymmetric cryptography to secure data encounters many limitations associated with these algorithms [51] [52]. Researchers introduced efficient approaches [53] that are more secure than the traditional symmetric and asymmetric algorithms [54].

Researchers employ a combination of symmetric encryption schemes, such as Blow-Fish [55], AES [56], DES [57], or IDEA [58], with asymmetric encryption schemes, such as ECC [59], RSA [60], El Gamal [61], or Paillier [62] to implement a hybrid scheme. The specific choice of algorithm depends on the system requirements and efficiency of the algorithm. In hybrid cryptography-based KMS, the DEK is generated by the symmetric encryption scheme, and the public-private key pair of the asymmetric encryption scheme is used as the KEK.

- **KMS for Hybrid Cryptography with Hash Function:** In the hybrid cryptography-based KMS, a hash function can be employed with various encryption algorithms to provide data integrity. A hash function takes the data of variable length as input and performs complex operations to produce a fixed-length output known as a hash code, a hash value, a hash, or a digest. It is computationally difficult to deduce the original data from a hash and to identify the two different inputs producing the same hash values [63][64]. Figure 2.2 represents the hash-based hybrid KMS. In encryption, a plaintext message is encrypted using the pre-shared secret key. At the same time, the plaintext message is sent to the hash function to generate a checksum. Then, the encrypted message and the checksum are sent to the receiver. In decryption, the encrypted message at the receiver's end is decrypted using the pre-shared secret key, and the plaintext is used to generate a hash value. Then, the generated checksum is compared with the received checksum to identify

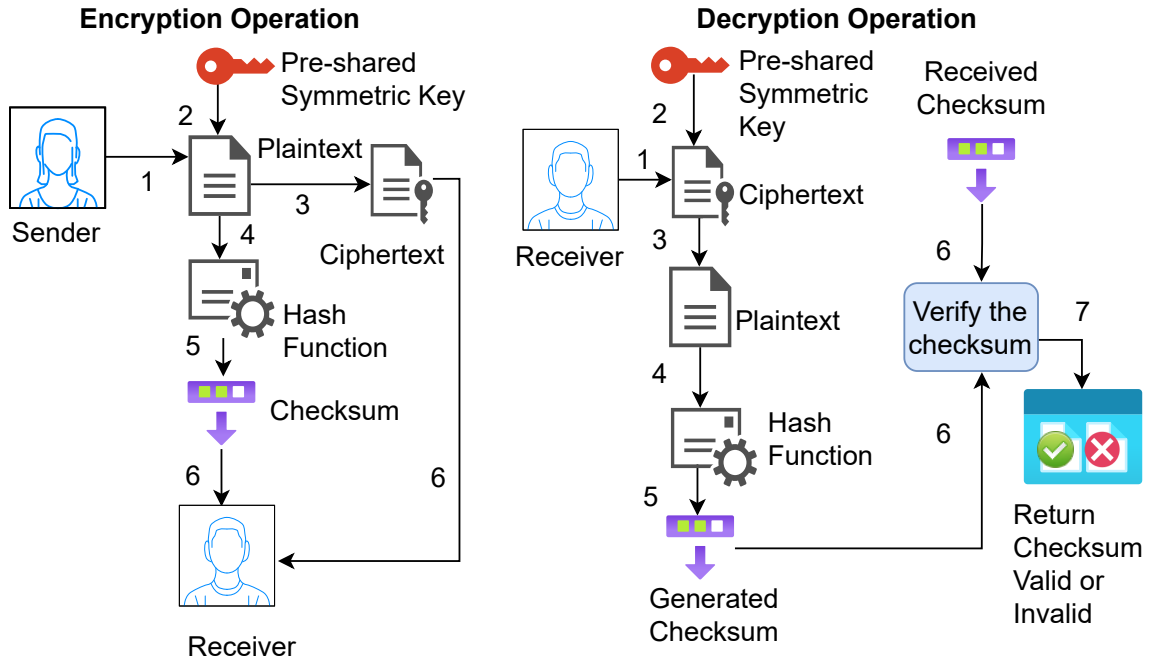


Figure 2.2: Hybrid KMS with Symmetric Encryption Scheme and Hash Function

whether the checksums match or not.

- KMS for Hybrid Cryptography with Digital Signature:** In the hybrid cryptography-based KMS, a digital signature can be employed with various encryption algorithms to provide data integrity and non-repudiation. A digital signature generation algorithm receives the sender's private key along with the data and produces a digital signature, and a digital signature verification algorithm receives the digital signature, data, and the sender's private key to verify whether the signature is valid or not. Figure 2.3 represents the digital signature-based hybrid KMS.

In encryption, a plaintext message is encrypted using a symmetric key pre-shared between the sender and the receiver. The ciphertext is then used as input to the digital signature generation algorithm together with the sender's private key to generate a digital signature. Then, the sender's public key, the digital signature, and the encrypted message are sent to the receiver. In decryption, the encrypted message at the receiver's end is provided to the digital signature verification algorithm along with the sender's public key and the digital signature. A valid digital signature,

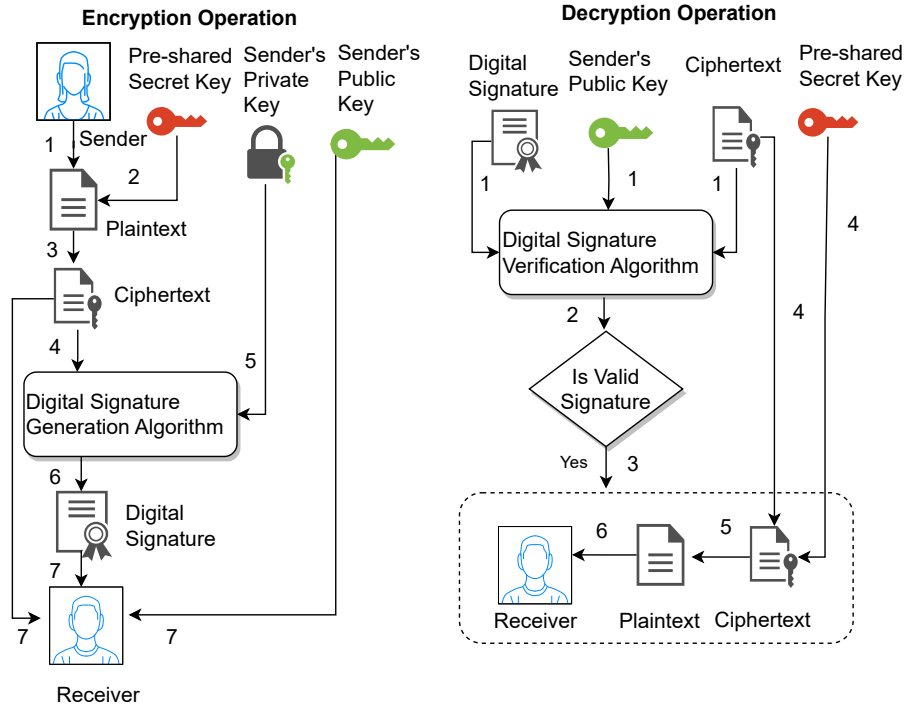


Figure 2.3: Hybrid KMS with an Asymmetric Encryption Scheme, a Symmetric Encryption Scheme and a Digital Signature

confirmed by the digital signature verification algorithm, ensures that the message integrity and non-repudiation are upheld, or an invalid digital signature affirms a compromised message [65, 66, 67]. Once the digital signature is verified, the ciphertext is decrypted with the pre-shared secret key, and the plaintext is provided to the receiver.

2.5 Hardware Accelerated AES Instructions

AES encryption is one of the accepted, widely used, and secure symmetric encryption schemes that adhere to the FIPS. AES New Instruction Set (AES-NI) [68] is a hardware-accelerated AES instruction set introduced by Intel, to perform encryption and decryption operations to improve performance and security.

Benefits of Using AES-NI: AES-NI, supported by Intel and AMD processors, can benefit as below. *Hardware Acceleration:* Execution of the AES-NI instructions is accelerated by the CPU's hardware, unlike software-based AES, which is slower than AES-NI due to the consumption of more CPU cycles. *Parallelism:* Since modern CPUs support parallel processing, multiple data blocks can be processed in parallel by AES-NI instructions. *Energy Efficiency:* AES-NI instructions consume less power than software-based implementations. *Reduced CPU Overhead:* Since cryptographic operations are performed on the dedicated hardware, the CPU becomes available to perform other tasks, making it performant. *Lower Latency:* With lower latency, AES-NI is a better choice for real-time applications than a software-based AES implementation. *Improved Security:* AES-NI is more secure than a software-based AES implementation since AES-NI is less susceptible to side-channel attacks.

2.6 Hardware-Based Cryptography Modules

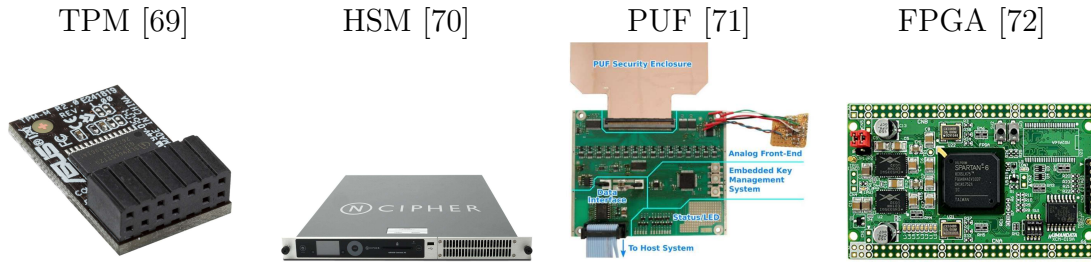
Dedicated hardware can be employed to perform various cryptography operations, generate and store cryptography keys, and provide tamper-resistant enclosure.

Hardware-based cryptography modules provide better security than entirely software-based key management systems. The benefits of using hardware-based cryptography modules are described below.

2.6.1 Benefits of Using Hardware-Based Cryptography Modules

The advantages of using the hardware-based cryptography modules are described below, followed by some examples. *Cryptographic Operations:* Hardware provides a secure execution environment for encryption and decryption operations. *Random Number Generation:* a Hardware-based cryptography module acts as a True Random

Table 2.2: Various Hardware Modules Supporting Cryptography Operations



Number Generator (TRNG). *Secure Storage*: Secrets can be physically stored in the hardware, making it secure even if the hardware is compromised. *Physical Security*: A tamper-resistant enclosure comprises sealing that detects any physical attack on the hardware. *Compliance*: A hardware-based cryptography module is required for FIPS 140-2. *High Performance*: Hardware-based cryptography modules can provide better performance or cryptography operations.

2.6.2 Various Hardware-Based Cryptography Modules

Various hardware component types (depicted in Table 2.2) that support secure cryptography operations are described below.

- **Trusted Platform Module (TPM)**: A Trusted Platform Module (TPM) is a cryptographic coprocessor that can be directly integrated with the motherboard of a device, server, computer, or IoT device. A TPM provides a secure boot, tamper-resistant secure environment, key storage, key generation, and cryptography operations [73].
- **Hardware Security Module (HSM)**: Hardware Security Module (HSM) is a dedicated device that provides various cryptography operations, generation, and storage of digital signatures, certificates, and encryption keys. HSMs facilitates authentication and authorization functionalities, backup of the secrets, archiving, and logging mechanism. HSMs support zeroization of the secrets, where secrets are deleted if any attempt to tamper with the hardware is detected.

- **Physical Unclonable Function (PUF):** Physical Unclonable Function (PUF) is a physical entity that leverages the production variable of the hardware, such as the physical microstructure of the hardware introduced during the manufacturing process, to generate hardware-specific random numbers. Since the output is unique to that hardware, and it is highly unlikely two devices can have the same output, the output is considered as the device fingerprint. A PUF is considered unclonable since variations in the hardware property cannot be controlled from the outside, so the output keys are different in various hardware. However, when the key is generated, the key can be cloned [74].

- **Field Programmable Gate Array (FPGA):** A Field-Programmable Gate Array (FPGA) is a programmable hardware device that can be used to perform cryptographic operations and store encryption keys [75]. FPGAs can be used to accelerate AES encryption and prevent side-channel attacks. FPGAs, which are resource-constrained devices, can provide a few megabits of storage. FPGAs might need external storage if cryptographic operations need more storage, making them inefficient for faster data access [76].

2.7 Key Generation

Key generation is the first stage in the KMS life cycle process. The strength of the encryption algorithm depends on the randomness and length of the keys. The KMS generates different types of keys, such as private keys or public-private key pairs. While the private keys are used by symmetric encryption schemes, the public-private key pairs are used by the public encryption schemes.

Private Key Generation using Random Number: Encryption keys are random numbers, which can be generated in two ways: Pseudorandom Number Generation Process or True Random Number Generation Process.

- **Pseudorandom Number Generator (PRNG):** A Pseudorandom Number Generator (PRNG) is a computer algorithm that generates a sequence of random numbers, which is completely dependent on the seed or an initial value. The seed can have true random numbers harnessed from the various entropy sources such as mouse movements or system interrupts. PRNG is also known as a Deterministic Random Bit Generator (DRBG).

- **Cryptographically-Secure Pseudorandom Number Generator:** A random number generated by PRNG, which is apt for cryptographic operations, is called a cryptographically secure Pseudorandom Number Generator (CSPRNG). The bits in the random number generated by CSPRNG should have high uncertainty and uniform distribution of binary values, making it challenging to predict the bit sequence and resistant to various cryptanalysis attacks.

- **True Random Number Generator (TRNG):** True Random Number Generator leverages physical properties or noise sources, such as thermal noise, photoelectric effects, radioactive decay, or quantum effects, to generate a random number. TRNG is also known as Non-Deterministic Random Bit Generator (NRBG).

Private Key Generation using Threshold Cryptography: The *threshold encryption scheme* is employed to generate and safeguard private keys [77]. In this scheme, the secret key is split into multiple shares, with the final secret key generated by combining the threshold number of shares [78]. Even if some shares are compromised by an adversary (up to some predefined number), the entire key generation process can be attack-tolerant. Shamir's secret sharing is one of the most recognized, and well-established threshold cryptography scheme for generation of an encryption key [79].

Key Generation In TPM: A TPM employs a hierarchical structure of keys, where different hierarchies are used for various purposes. For example, Endorse-

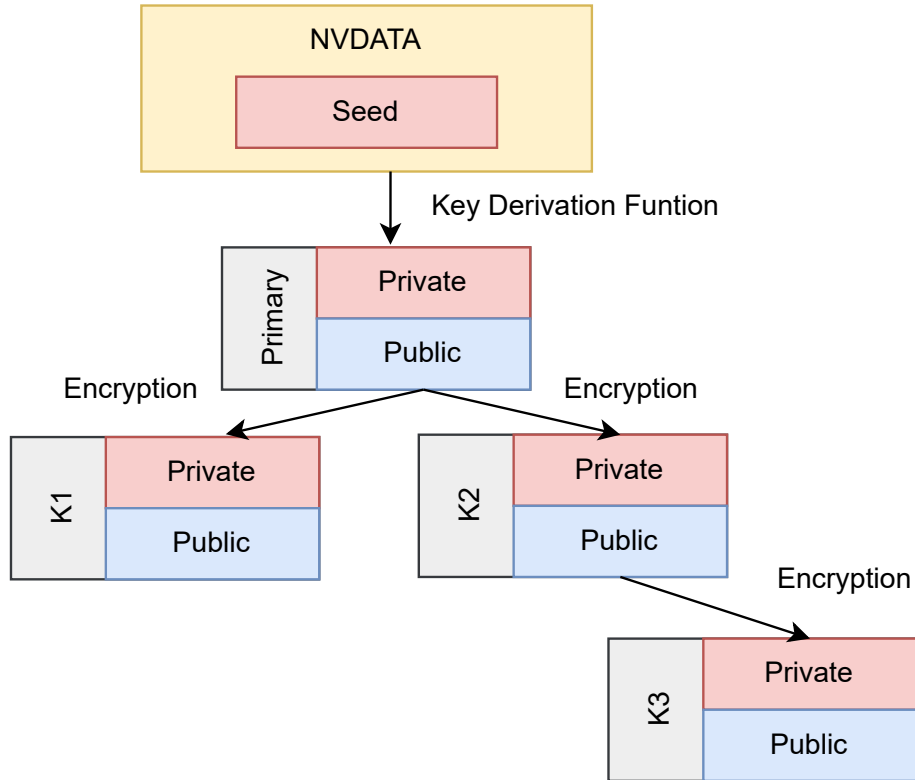


Figure 2.4: Key Generation in TPM [2]

ment Hierarchy (EK) is used for attestation, and Storage Hierarchy (SH) is used for managing keys used for data encryption [73]. A hierarchy is created by TPM keys, where parent keys encrypt or wrap child keys. The TPM employs asymmetric key cryptography, where the private key material never leaves the TPM in plaintext, and the public key is used to encrypt the private part of the child key. The root key in the TPM is known as the primary key, which has no parent. The primary key is generated using a Key Derivation Function (KDF) from the seed stored in the Non-Volatile Data (NVDATA) section. In Figure 2.4, the primary key’s public part encrypts the private parts of its children $K1$ and $K2$. Similarly, the private part of key $K3$ is encrypted by $K2$, creating the hierarchy.

2.8 Common Security Vulnerabilities and Attacks

CephVault employed countermeasures against common security vulnerabilities and attacks, which are described in this section. The countermeasures employed in *CephVault* are discussed in the security analysis of *CephVault* in Section 7.3.

- **Buffer Overflow:** A program employs a temporary storage area in the memory, known as a buffer, to store data, and an attacker can overflow the buffer by sending more data than the buffer intended to store. Buffer overflow can lead to arbitrary code execution, unauthorized access, system crash, and compromising the entire system [80].

- **SQL Injection:** Attackers can inject malicious SQL code into the database query by bypassing the application’s input validation, leading to unauthorized access to sensitive data, altering data, and deleting data [69] [81].

- **Directory Traversal Attacks:** In directory traversal attacks, attackers can gain access to the systems directory and files outside of the authorized directory by providing malicious special character patterns (example, ../) and malformed relative path patterns (example, ../../), leading to unauthorized access to the system [82].

- **Side Channel Attacks:** In a side-channel attack, instead of exploiting cryptography algorithms, which are difficult to break, attackers target the physical implementation of the algorithms, such as the time taken to perform encryption-decryption operations, electromagnetic radiation emitted by the devices, or power consumption during cryptography operations [83].

- **Physical Attacks:** Attackers can gain physical access to the computers, servers, or cryptography modules and compromise sensitive data. Attackers can gain access to various ways, such as physical storage devices, can tamper system hardware, install keyloggers, and clone biometric data [81] [84].

- **Insider Attacks:** An unauthorized person inside an organization can intentionally or unintentionally compromise sensitive data. Insider attackers can be malicious

internal employees, developers, architects, contractors, or disgruntled former employees possessing access to the organization's sensitive data [85] [81].

- **Man-in-the-middle Attacks:** In a Man-in-the-Middle (MitM) attack, attackers intercept the communication between two users or applications and perform malicious activities, such as gaining unauthorized access, modification of communication, and injecting malformed content [86].

- **Brute Force Attacks:** Attackers can guess passwords or encryption keys using different possible combinations of characters, words from a dictionary, or known sources. Attacks can effortlessly compromise weak passwords, leading to unauthorized access to sensitive data and revealing sensitive data [87].

- **Rainbow Attacks:** Various systems employ hash functions to store their users' passwords in a hash format instead of storing the passwords as plaintexts, and a rainbow attack is launched to break the password hashes. Rainbow tables, which are precompiled tables comprising a massive number of hashes of probable passwords, are employed to match the password hashes retrieved from the systems. This attack is more efficient in breaking passwords than brute force attacks since systems can employ mechanisms to authenticate after a certain number of tries, which can defer brute force attacks [88].

- **Chosen Plaintext Attacks:** Attackers can select plaintexts of their choice and inspect their corresponding ciphertexts. If the same plaintext is always encrypted with the same key, it produces the same ciphertext. For this scenario, the encryption scheme is vulnerable to indistinguishability under a chosen plaintext attack (IND-CPA). Attackers can observe the patterns in the ciphertexts, which could lead to key recovery, the encryption algorithm used for ciphertext generation, and the relationship between plaintexts and ciphertexts [89].

- **Padding Oracle Attacks:** In symmetric key encryption, generally, a plaintext is divided into multiple blocks, with a block of 16 bytes to perform encryption op-

erations on each block. If the plaintext is not a multiple of 16 bytes, the last block needs padding. Attackers can manipulate the padding to decrypt ciphertext [90].

2.9 Ceph Architecture

Ceph is an open-source, distributed, software-defined file system that can provide a storage solution for the object, block, and file types [91]. Ceph is a high-performing, efficient storage solution that circumvents the issues with traditional storage solutions, such as Network Attached Storage (NAS), Storage Area Network (SAN), and Gluster File System (GlusterFS), where read-write operations on the disk are performed with the help of the centralized metadata table. Ceph introduces a new approach, Controlled Replication Under Scalable Hashing (CRUSH), where, rather than performing a lookup operation, the client can compute the location of the read-write operation using the CRUSH algorithm. Some of the Ceph concepts relevant to our research are discussed in this section (Figure 2.5).

- **CRUSH:** CRUSH is fully aware of the logical and physical infrastructure. Ceph creates a CRUSH map, which is a hierarchy comprised of the cluster’s physical topology, rules for data placement policy, storage capacity, information regarding nodes, data-centers, pools, racks, switches, domains, buildings, rooms, etc. [92]. The CRUSH map contains information on the failure domain, so in case of component failure, the CRUSH map is automatically updated with the failed component. Thus, Ceph achieves a self-healing capability. Since the CRUSH map is shared among both the client and Ceph, when clients want to store data in the cluster using Ceph, the client can deduce the exact location using the CRUSH map. This way, Ceph circumvents the need for lookup operations on a centralized metadata table.
- **Object Storage:** In Ceph, data is divided into smaller units of data called objects, which can be uniquely identified by their respective IDs and Pool names. Objects,

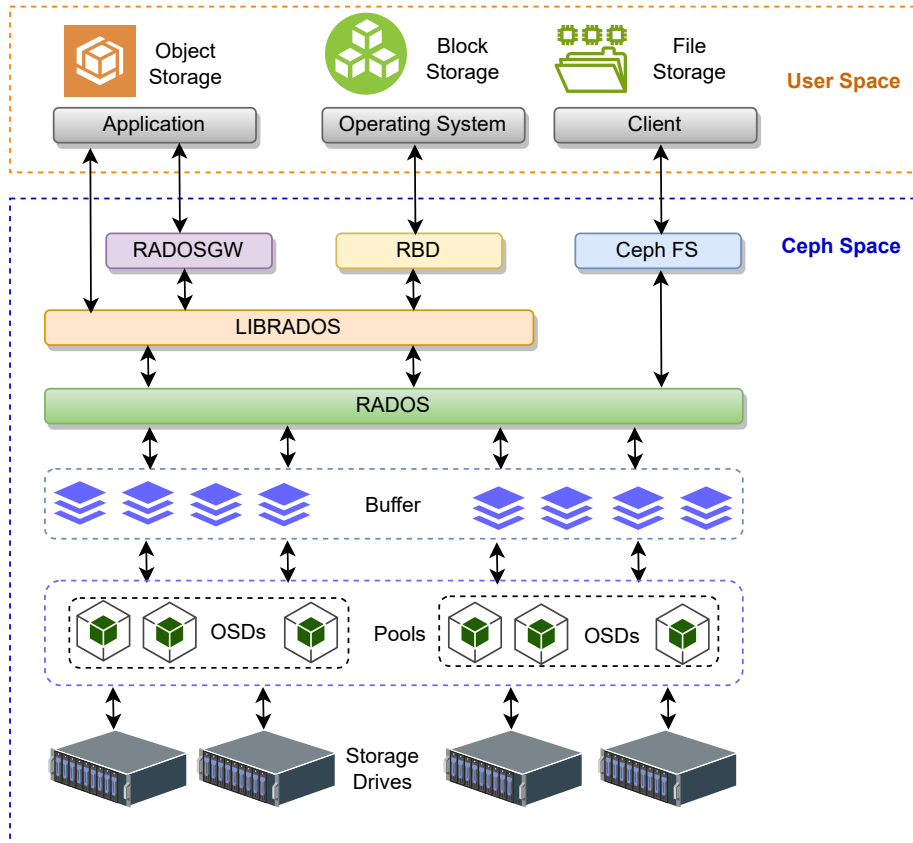


Figure 2.5: Ceph Architecture (Vanilla)

which can be accessed by HTTP(S) RESTful APIs, store unstructured data, such as pictures, videos, documents, and images.

- **Reliable Autonomic Distributed Object Store Gateway (RADOSGW):** Reliable Autonomic Distributed Object Store (RADOS) Gateway, which is known as RADOSGW, or Ceph Object Gateway, provides an interface to store objects in Ceph.
- **Block Storage:** A block storage is a sequence of bytes, striped over several objects, which can be mapped to the operating system [93]. Ceph block storage facilitates thin provisioning, a technique to allocate storage on-demand for writing data in Ceph.
- **RADOS Block Device (RBD):** RADOS block device (RBD) provides an interface for the virtual machines and containers to store data on various storage media,

such as Hard Drives (HDDs), Solid State Drives (SSDs), Compact Disks (CDs), and floppy disks.

- **File Storage:** Ceph file storage facilitates distributed file systems to be mounted and accessed as traditional file systems. It is apt for the Portable Operating System Interface (POSIX) compliant environment, where files are shared across a distributed environment.

- **Ceph File System (CephFS):** Like traditional file systems, where data is categorized in a directory and file structure, Ceph File System (CephFS) provides a POSIX interface to access files.

- **LIBRADOS:** LIBRADOS is a library that acts as an intermediary between Ceph's storage interfaces (RADOSGW and RBD) and Reliable Autonomic Distributed Object Store (RADOS), allowing object-level operations, such as data management, reading or writing operations.

- **Reliable Autonomic Distributed Object Store (RADOS):** Reliable Autonomic Distributed Object Store (RADOS) is the primary component of the Ceph cluster. Ceph facilitates data access via various data access functionalities, such as RBD, CephFS, RADOSGW, and LIBRADOS, which operate on the top of the RADOS layer. When the user sends a request to write data, the CRUSH algorithm calculates the location where data should reside. Then, data is sent to the RADOS layer, which breaks the data into small chunks, namely objects, and distributes the data among all the clusters. Finally, objects are stored in Object Storage Devices (OSDs) [93].

- **Buffer:** Buffer comprises Buffer List, a data structure of a sequential list that provides a contiguous data area for transferring data between various layers in Ceph.

- **Object Storage Device (OSD):** Object Storage Device (OSD) stores the objects on the physical devices of the cluster nodes. During a write operation, data is stored in the form of objects, and during the write operation, the data is retrieved

from the object and provided to the user.

- **Ceph Pool:** A Ceph pool provides a logical partition to store objects. Ceph distributes each and every single pool across cluster nodes, making Ceph resilient. Erasure Coding (EC) is a mechanism where objects are replicated to desired chunks, and when needed, only a subset of the chunks are required to recreate the data, making data available. Also, EC provides data protection since the fragmented data is encoded and distributed across the cluster nodes.

- **Ceph Namespace:** A Ceph namespace is defined as a logical group of objects with a pool, and each object can be assigned to a namespace. Namespaces are used to control access to the pool, such that read and write operations can be confined only within the namespace, and a user can access an object associated with a namespace within the pool only if the user has access to the namespace. Permissions, which are also known as “capabilities,” can be granted to users authenticated with Ceph. “Capabilities” can be employed to provide restricted access to data inside a pool or a namespace within a pool [94].

2.10 Our Research Contribution

Ceph’s default architecture, as depicted in Figure 2.5, does not contain any cryptography component that could store data as an encrypted format. Our research team developed capabilities to safeguard data in Ceph. The contributions of our research team, which are depicted in Figure 2.6 (highlighted in yellow background), are described below.

- **CephArmor [12]:** A cryptographic API, developed by our research team, that encrypts and decrypts data-at-rest in Ceph.
- **Security Provider (SP) [12]:** The Security Provider (SP) employs an interface with the various required cryptographic operations. *CephArmor* uses various SP

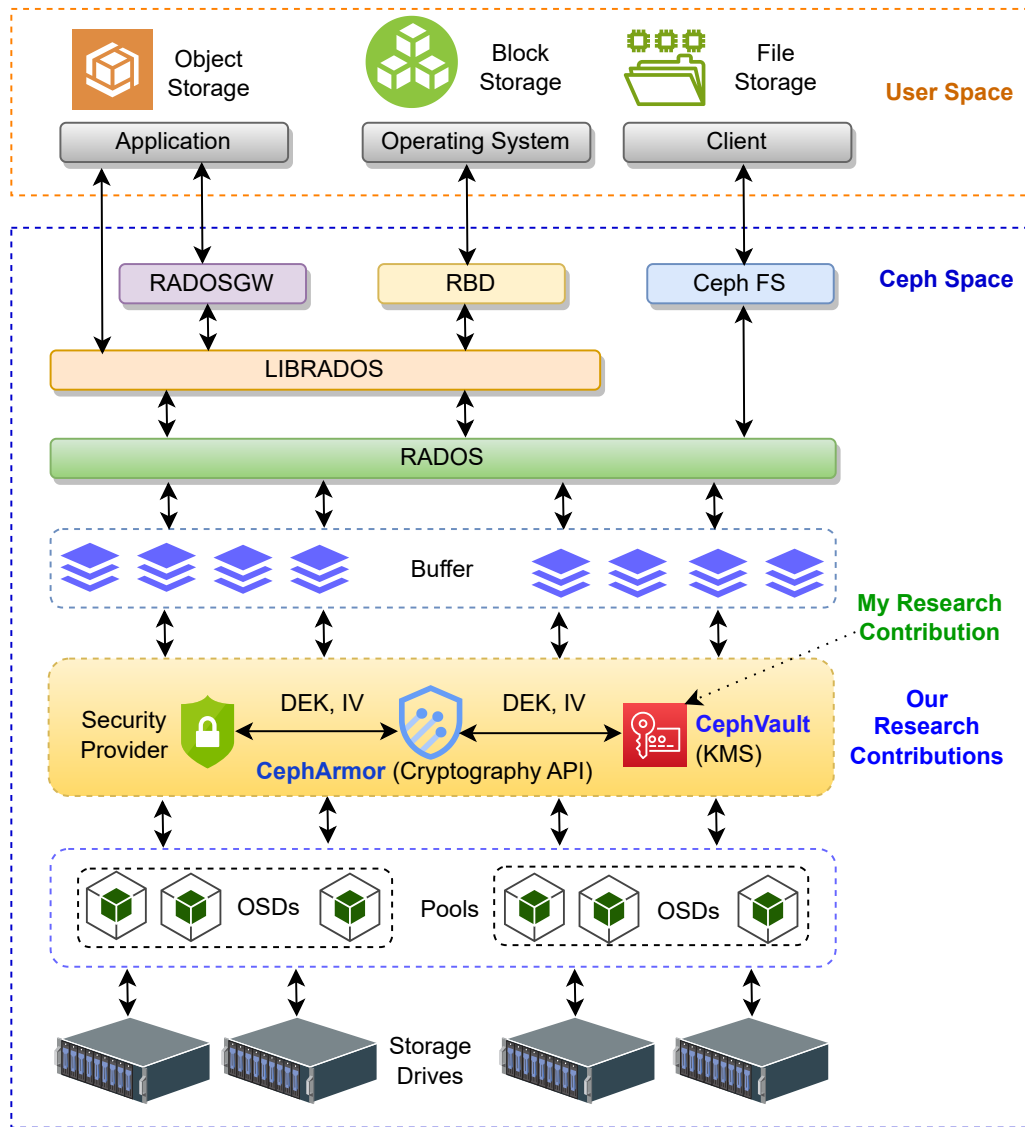


Figure 2.6: Integration of Ceph with CephArmor and CephVault

methods to provide encryption-decryption operations.

- **CephVault:** The key management system for Ceph, which I proposed and developed in this thesis.

2.11 Federal Information Processing Standards

The guidelines and standards proposed and designed by NIST are known as FIPS. FIPS 140-2 is an established standard that focuses on the security requirements

of a cryptographic module [1]. FIPS 140-3, which is the successor to FIPS-2, was approved on March 22, 2019 [95]. Although FIPS 140-3 standards are published at the time of writing this thesis, various open source libraries and other components are still in the process of FIPS 140-3 validation. For example, the OpenSSL library described the plan for FIPS 140-3 validation, which is anticipated to be completed by 2024 [96]. So, we focused our research for achieving FIPS 140-2 standards, instead of FIPS 140-3.

2.11.1 Security Requirements for a Cryptography Module Adhering to FIPS 140-2

A cryptography module comprises a software or hardware component, or a combination of software and hardware components, that facilitates various cryptography operations, key generation, and storage. A key management system can employ a cryptography module and other components, such as a database, to manage the life cycle of a key. FIPS 140-2 has a total of 4 levels of security to cover a wide area of environments and their security requirements. This thesis will focus on achieving FIPS 140-2 level 2. We discuss only levels 1 and 2 for the sake of brevity; details of levels 3 and 4 can be found on the FIPS website [1].

- **Security Level 1:** In this level, the minimum requirement is to use one approved encryption algorithm or approved security function. There is no requirement to employ a physical security mechanism. A personal computer (PC) encryption board can be considered a cryptography module at this level. The cryptographic module's software component can be run on the unevaluated operating system.

- **Security Level 2:** In this level, all requirements of level 1 are included. The cryptographic module must have a tamper-evident coating, seal, and pick-resistant lock to ensure the integrity of the cryptography module. Role-based authentication is employed to ensure authenticated users have a specific role in performing any op-

eration on the cryptographic module. The operating system must be trusted so that when the general-purpose computing platform is used by the cryptographic modules for generating keys, these computing platforms can be considered equivalent to a dedicated hardware-based cryptographic module.

A summary of the security requirements for a cryptographic module adhering to FIPS 140-3 Level 2 is specified in Table 2.3.

2.11.2 Security Requirements for a KMS Adhering to FIPS 140-2

This section depicts the security requirements for a KMS adhering to FIPS 140-2 (Levels 1, 2, 3, and 4) [97].

- **Documentation Requirements:** To adhere to the FIPS 140-2 (Levels 1, 2, 3, 4), KMS documentation should contain the details pertaining to all cryptographic keys and Critical Security Parameters (CSPs) used by a cryptographic module.
- **Random Number Generators (RNGs):** KMS can only employ an approved RNG (deterministic and non-deterministic) to generate cryptographic keys.
- **Key Generation:** Similar to the guideline regarding the random number generator, only approved Random Number Generators (RNGs) can be used in the key generation process.
- **Key Storage:** Cryptographic modules can store keys either in plaintext form or encrypted form.
- **Key Usage:** This is called separation of duty, where a single key can only be used for a single use (for example, encryption, key wrapping, random number generation, or digital signatures).
- **Labeling of Cryptographic Information:** To provide better information about a key, various labels may be employed.
- **Electronic Key Distribution:** Prior to the distribution of the key electronically,

Table 2.3: Summary of Security Requirements for a Cryptographic Module [1]

	Security Level 1	Security Level 2
Cryptographic Module Specification	Specification of the cryptographic module, cryptographic boundary, Approved algorithms, and approved modes of operation. Description of the cryptographic module, including all hardware, software, and firmware components. Statement of module security policy.	
Roles, Services, and Authentication	Logical separation of required and optional roles and services.	Role-based or identity-based operator authentication.
Physical Security	Production grade equipment.	Locks or tamper evidence.
Operational Environment	Single operator. Executable code. Approved integrity technique.	Referenced PPs ^a evaluated at Evaluation Assurance Level 2 (EAL2) with specified discretionary access control mechanisms and auditing.
Cryptographic Key Management	Key management mechanisms: random number and key generation, key establishment, key distribution, key entry/output, key storage, and key zeroization. Secret and private keys established using manual methods may be entered or output in plaintext form.	
Cryptographic Module Ports and Interfaces	Required and optional interfaces. Specification of all interfaces and of all input and output data paths.	
Mitigation of Other Attack	Specification of mitigation of attacks for which no testable requirements are currently available.	
Self-Tests	Power-up tests: cryptographic algorithm tests, software/firmware integrity tests, critical functions tests. Conditional tests.	
Design Assurance	Configuration management (CM). Secure installation and generation. Design and policy correspondence. Guidance documents	CM system. Secure distribution. Functional specification.

^a Protection Profile: an implementation-independent set of security requirements for a category of Targets of Evaluation (TOEs) that meet specific consumer needs.

a key encrypting key can be used for key wrapping.

2.12 Threat Model

Threats can adversely affect a system, and understanding threats helps organizations plan for threat mitigation. A threat model represents attackers, their capabilities, trust boundaries, and vulnerabilities the adversary can exploit. The STRIDE model is a popular choice for analyzing threats. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege [98]. *Spoofing* refers to a technique where the attacker impersonates a legitimate user. *Tampering* refers to unauthorized modification of data. *Repudiation* involves the denial of actions which already occurred. *Information Disclosure* refers to the unauthorized exposure of data. *Denial of Service* involves disrupting the usual services or functions by systems. *Elevation of Privilege* refers to gaining unauthorized access to the higher privilege level.

2.13 Summary

In this chapter, some of the key concepts, such as the life cycle of a key management system, various security primitives, and software and hardware-based cryptography modules, were discussed. We also discussed the internal layers of Ceph to provide an overview of our research contribution and applicability. Then, we discussed information security principles, secure design principles, various known attacks, and FIPS 140-2 guidelines since *CephVault* will be evaluated against these attributes in Chapter 7. In the next chapter, we will discuss various key management systems that are supported by Ceph and some open-source solutions that could be used for Ceph. Also, a comparative study on *CephVault* with other solutions will be provided.

Chapter 3

Related Work

Various open-source and enterprise KMSs support different applications, for example, Venafi manages SSL/TLS certificates [99], or Thales CipherTrust Cloud Key Manager secures data across cloud environments [100]. In this section, we will discuss relevant KMS solutions to identify their applicability to Ceph. We will also compare these solutions with *CephVault*.

3.1 Research Method

We searched for research papers, scientific journals, scholarly articles, and technical white papers for key management systems in scientific search engines. However, only a few publications were identified that described key management solutions for cluster storage comprising the life cycle of a KMS. We found research papers on KMSs for novel encryption schemes [101, 102, 103], group key management protocols [104, 105, 106, 107], or wireless sensor networks [108, 109, 110]. We also found research papers describing various security principles, security challenges and their countermeasures, and best practices required for a secure KMS [111, 112, 113]. However, we identified a gap in the literature that did not provide a detailed description of KMSs, such as the architecture of KMSs or the techniques employed to

generate, store, or rotate keys safely. On the other hand, we searched for KMS solutions on organizations' official websites and code-sharing platforms, such as GitHub. We identified some popular KMS solutions and analyzed their features for possible integration with Ceph.

3.2 KMS Solutions Supporting Ceph (Officially)

Although Ceph does not provide native support for a key management system, KMSs developed by other organizations can be incorporated to encrypt data in Ceph. According to Ceph's official website, the two KMSs that support Ceph are: OpenStack Barbican [114], and Hashicorp Vault [115]. Table 3.1 provides a detailed description of contributions and limitations of KMSs supporting Ceph.

3.3 KMS Solutions Not Supporting Ceph (Officially)

We evaluated some popular open-source KMS solutions, although they do not support Ceph officially, which means these solutions are not included as supported key management solutions on Ceph's official website. The reason for evaluation was to identify possibilities of integration with Ceph. Table 3.2, and Table 3.3 provide a detailed description of contributions and limitations of KMSs not supporting Ceph.

3.4 Summary

In this chapter, we analyzed various enterprise and open-source key management solutions and provided a comparative analysis with *CephVault* to understand the applicability of our research.

Table 3.1: KMSs Supporting Ceph: Contributions, and Limitations

KMS	Contributions	Limitations
Barbican [116]	Barbican is a REST API that provides users with a container to manage and store certificates and keys. It supports pluggable backends, simple crypto, and PKCS#11, a public key cryptography standard. OpenStack Swift can be used as a storage backend, and an HSM can be used as a cryptography backend.	For authentication and authorization, Barbican leverages OpenStack Keystone, which stores passwords in the config file in plaintext [117] [118]. Similarly, the KEK is stored in the config file [119] [120]. Also, Redhat confirmed that Barbican could only be used as a preview, not to be used in the production systems [121]. Unlike <i>CephVault</i> , which is embedded with Ceph, Barbican requires a separate KMS server. Keys are not created per-object. The object, pool, and namespace are not considered for key generation, making it vulnerable to accidental overwrite of duplicate data in Ceph.
Hashicorp Vault [122]	Hashicorp Vault, a mature secrets management solution (more than 1250 contributors to its open-source project), provides a wide variety of secrets, such as keys, IVs, certificates, database passwords, session keys, and tokens. The secret management solution is a well-documented and popular choice among the community. Vault uses AES256 in GCM mode to encrypt and decrypt secrets internally. It supports hardware and software cryptography modules that use Shamir's secret shares to generate the KEK.	Hashicorp Vault open source is not FIPS compliant; the final key (unseal key) needs to be stored using external services, such as Azure key vault. So, another third-party key vault is required. Development mode runs on localhost without SSL, and keys generated in memory are not retained if the KMS server shuts down. It is expensive; the Hashicorp Vault Plus version can be USD \$82,397/year, with an additional USD \$1349/user each year [123]. The enterprise version is charged on case-by-case basis. It does not support per-object key generation.

Table 3.2: KMSs Not Supporting Ceph: Contributions, and Limitations

KMS	Contributions	Limitations
Ansible Vault [124]	Supports the encryption of variables and files.	Plaintext password is stored in file systems or environmental variables, with no envelope encryption. Only key creation and key rotation are supported.
Cloudflare Red October [125]	A software-based TLS server that encrypts and decrypts files. Encryption keys are split into two shares for security.	Only 128-bit AES encryption is supported. The DEK is derived from the user password, where the strength of the encryption key depends on the password's complexity. Also, the KMS mandates a local file to store the key.
Codahale Sneaker [126]	Stores secrets on AWS leveraging Amazon S3. Employs AES-GCM for envelope encryption.	Codebase is not reviewed by security professionals. Requires Amazon S3 subscription. Keys and encrypted data are stored together in Amazon S3, which is not secure.
Square Keywhiz [127]	Supports a wide variety of secrets, such as TLS certificates, keys, API tokens, and database credentials. Secrets are managed using JSON APIs. Administrators can use web API, CLI, or REST API to manage KMS. HSM is employed to store the KEK, which is generated using HKDF, a key derivation function (KDF) based on the HMAC message authentication code.	Developed in Java, so it will not provide as seamless integration as <i>CephVault</i> , which is developed in C++, and deployed as an intrinsic component of Ceph. Secrets are cached in memory, which is vulnerable to memory-scraping attacks, side-channel attacks, and cold-boot attacks. Does not provide high availability. The project is deprecated.

Table 3.3: KMSs Not Supporting Ceph: Contributions, and Limitations (Continued)

KMS	Contributions	Limitations
XOR Data Exchange Crypt [128]	Command line tool helps to store and retrieve encrypted configurations from <i>etcd</i> , a distributed key-value store.	Keys are not secured with envelope encryption. Does not support various modes of operation. Keys are stored in the configuration file as a plaintext.
Flix- Keeto [129]	It is a module in the OpenSSH server that enables secure distribution of distribution of OpenSSH keys.	Does not support secure hardware storage, and envelope encryption scheme. Clients are responsible to manage their keys [130].
Oleiade Trousseau [131]	Provides a key-value pair to store various passwords in a file, where a file is encrypted by a password. It is easy to import and export the file, making it easy to transport the file from one environment to another.	No envelope encryption is supported. Users are responsible for generating and storing the key that encrypts the file containing the key-value pairs. Does not support the KMS life cycle, CSPRNG or TRNG.
Cyberark Conjur [132]	Used for securing credentials and secrets required by applications. Supports secrets: docker credentials, service accounts tokens - Kubernetes, login pass pairs. Employs role-based access policy. Key rotation is supported (every 10 minutes/every day/every two weeks etc.).	Stores the DEK in the environment variable, which is an insecure way to store the DEK. If the variable is overwritten, the encrypted data will be unrecoverable. Except for key rotation, other phases of the KMS life cycle are not supported. The database connection is stored in the environment variable.

Chapter 4

Design and Implementation

This section discusses the design and implementation of *CephVault*. *CephVault* is developed to support the entire life cycle of a key management system, described in Section 2.1. *CephVault*'s design is based on information security principles, defensive coding principles, and secure design principles.

4.1 CephVault Architecture

CephVault's architecture is depicted in Figure 4.1. It is logically divided into components described below.

- **Software-Based Cryptography:** Cryptography operations are supported by *CephVault*'s software-based component (encircled with the dotted line in Figure 4.1). Overall, *CephVault* follows a modular design, where each KMS life cycle stage is developed as a module. For security reasons, most of the methods in a module are kept private only to the module, whereas some of the methods are kept public to interact with the other modules of *CephVault*.

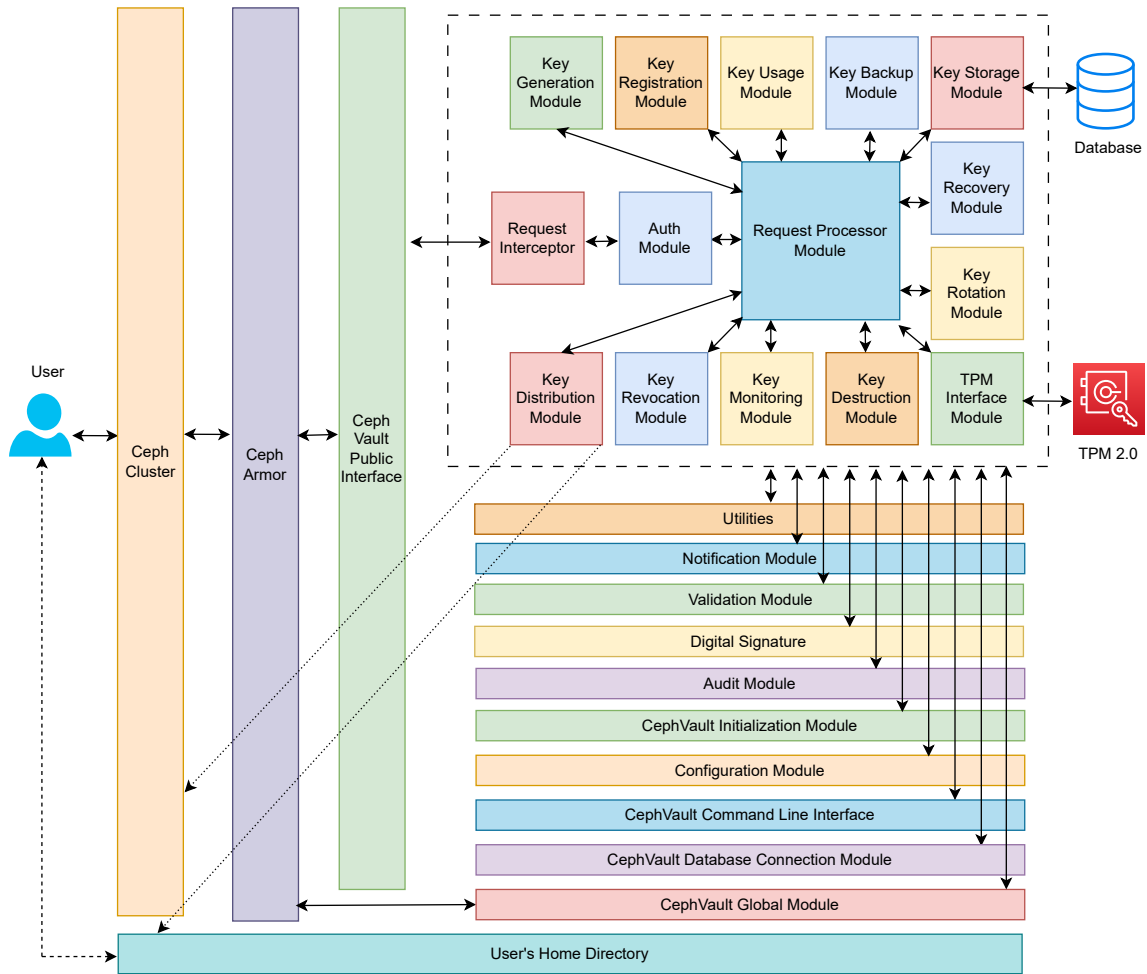


Figure 4.1: CephVault Architecture

- **Hardware-Based Cryptography:** *CephVault* employs TPM 2.0 for cryptography operations, key generation, and key storage.
- **Databases:** Databases are required to store information, such as the encrypted DEK, DEK metadata, KEK metadata, encryption types, and key rotation details. *CephVault* leverages SQLite [133] for data storage.
- **Common Modules:** To support the entire life cycle of *CephVault*, along with software-based and hardware-based cryptography modules, other modules, such as *Audit Module*, *Notification Module*, and *Utilities* are required to support the KMS. These modules have common methods which are accessible by all the modules. Only *CephVault Public Interface Module* and *CephVault Global Module* are exposed to the

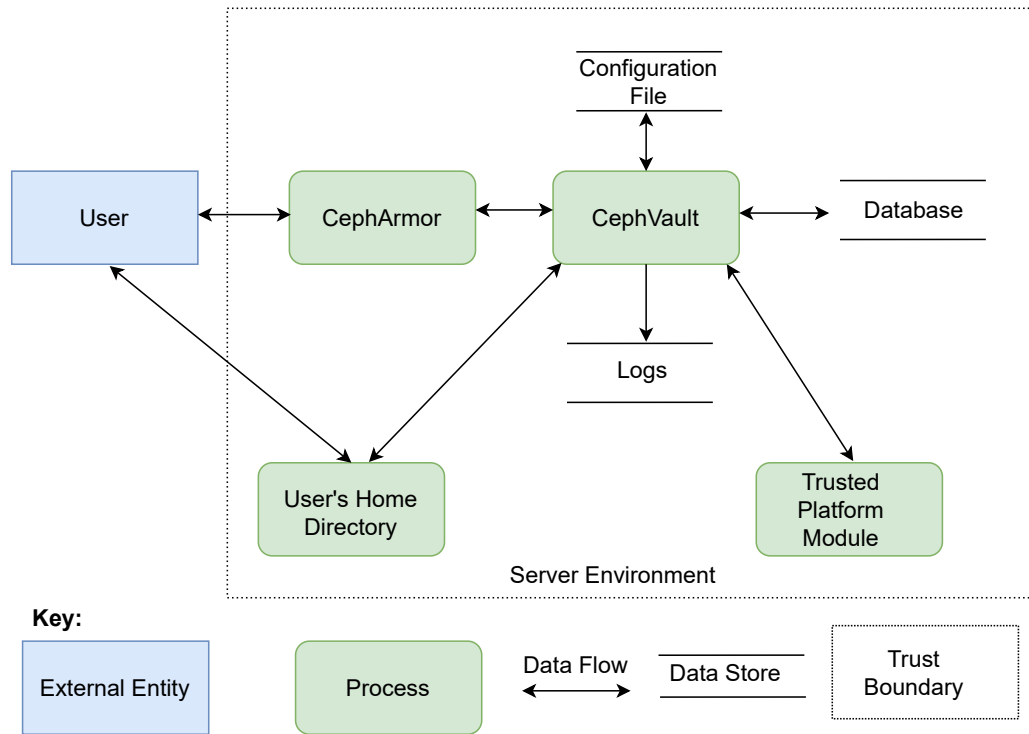


Figure 4.2: CephVault Threat Model

outside world (*CephArmor*), whereas all other modules are accessible within *CephVault* only.

- User's Home Directory:** Each user who interacts with Ceph to encrypt data has a unique home directory in the server, and *CephVault* employs the user's home directory to store a pre-shared symmetric key, which is used to encrypt the DEK when it is sent to *CephArmor*. This pre-shared key is different than the DEK or KEK. Also, the user's home directory contains keys for the digital signature required to sign the encrypted DEK during key distribution. The security implication of storing encryption keys in the home directory is discussed in Section 7.7.1.

4.2 CephVault Threat Model

Figure 4.2 depicts the threat model of *CephVault* using a Data Flow Diagram (DFD) [134].

- **Adversarial Capabilities:** *CephVault*'s threat model considers two types of attackers: 1) A malicious attacker capable of deviating from any established protocol and using unpredictable strategies to attack the system. 2) An insider who is honest but curious. The insider has the knowledge of *CephVault*'s architecture and implementation. The insider follows the protocol and may not intentionally compromise the system. However, the insider is motivated to learn information regarding other users.

- **Trust Boundaries:** *CephVault* employs the trust boundary as the server environment. *CephArmor* and *CephVault* are incorporated within Ceph. Further, the logs, configuration files, and databases reside within the same trust boundary, where all components interact within the Ceph boundary. The user's home directory and the TPM reside within the server boundary. The user can interact with *CephArmor* and the user's home directory.

- **Assumptions:** We assume that the root user of Ceph is not compromised and that Ceph's default access control mechanism is secure. Encryption algorithms, AES, and RSA employed in *CephVault* for envelope encryption are strong enough to withstand external attacks. Also, the desired key length is selected for encryption algorithms [135]. Further, communication between Ceph and the user over the network is secured by Ceph's *secure* protocol [136]. Ceph's *secure* mode of communication encrypts all data transmitted through the network, providing authentication during connection establishment, cryptographic integrity checks, and encryption of all post-authentication traffic. *Secure* mode provides security against man-in-the-middle attacks, ensuring end-to-end data protection. We further assume that the server is secured with an antivirus, firewall, Intrusion Detection System (IDS) [137], and Intrusion Prevention System (IPS) [138]. The server is also protected by ransomware detection and prevention software [139] to protect the KEK stored in Ceph. Furthermore, the disk is encrypted with disk encryption software [140]. While *CephVault* is

responsible for generating application logs, a Security Information and Event Management (SIEM) [141] tool is employed to apply rules and analyze logs for possible security incidents. We also assume that software is updated periodically.

4.3 Programming Language For CephVault

To develop *CephVault*, we decided to use C++ as the programming language despite being a more insecure language than Java or C# [143].

- **Vulnerabilities in C++:** Some of the common vulnerabilities in C++ are discussed: Since C++ does not support built-in bounds checking for pointer access and arrays, it is vulnerable to buffer overflow, off-by one error, and memory corruption. Moreover, since C++ allows direct pointer manipulation, an inappropriate pointer operation can lead to a memory vulnerability. C++ provides support for direct memory manipulation that can lead to memory leaks. C++ does not provide type safety, and incorrect type conversion can lead to data leaks. Libraries such as `strcpy`, `scanf`, `strncpy`, and `strncpy`, `gets`, etc., are inherently vulnerable to buffer overflow and memory leaks [143].

- **C++ as a Choice of Programming Language:** Reasons for choosing C++ for development of *CephVault*: firstly, *Ceph* and *CephArmor* are implemented in C++, and *CephVault* is deployed with *CephArmor*, so C++ is the preferred choice for *CephVault*. Moreover, the seamless compatibility between C++ and *CephVault* ensures smooth integration, eliminating the need for a native programming interface for interoperability. Furthermore, opting for languages other than C++ would necessitate the creation of such interfaces, introducing additional time and complexity into the development process. Additionally, converting various data types from C++ to Java or C# can lead to performance bottlenecks. Likewise, debugging applications implemented with multiple languages can be challenging. Lastly, operating systems

Table 4.1: Countermeasures to Language Vulnerability

Security Vulnerability	CephVault Features
No Built-in Bounds Checking	<i>CephVault</i> minimizes the use of a statistical declaration of the array and pointer manipulation; instead, the vector is implemented, where dynamic memory management and automatic bounds checking are performed. Wherever an array is used in <i>CephVault</i> , bounds checking is always performed.
Support For Pointer Manipulation	Instead of using a raw pointer, <i>CephVault</i> employs a smart pointer (<code>std::shared_ptr</code>) [144] while creating <i>CephVault</i> objects, which automatically manages memory, mitigates memory leaks, and dangling pointers.
Manual Memory Management	<i>CephVault</i> employs Resource Acquisition Is Initialization (RAII) [145][146], a C++ programming principle to manage memory allocation and deallocation, ensuring automatic memory releases when the object is destroyed.
Type Safety	<i>CephVault</i> employs countermeasures to mitigate issues due to wrong type conversion. <i>CephVault</i> employs FlawFinder [147], an open-source static code analyzer, to identify security vulnerabilities in code and address issues based on the recommendation of FlawFinder.
Unsafe Standard Library Functions	<i>CephVault</i> avoids the use of vulnerable libraries, and a monitoring module periodically checks whether any identified vulnerable libraries are used in the code.

provide different support for various languages, so cross-platform compatibility can be expensive when the application’s other components are implemented in C++ and if we choose Java or C# for *CephVault*. We implemented countermeasures to address language vulnerability in C++, which is described in Table 4.1.

4.4 Design Pattern for CephVault

Design patterns are followed to provide code reusability, scalability, maintainability, optimization, better understandability, and industry best practices, helping to mitigate design issues, common mistakes, and security vulnerabilities. *CephVault* is developed with the Mediator design pattern, which provides loose coupling between

various modules or objects. The Mediator design pattern employs a centralized object, known as the mediator, that encapsulates the interaction between various other objects, which are known as colleagues, where colleagues cannot communicate with each other. Some use cases of the Mediator design pattern are various chat applications, where interaction between users is securely delivered, and air traffic control systems, where interaction between control towers and airplanes is secured. The benefit of using the Mediator pattern is manifold. *Centralized Communication:* Centralized communication enables altering the behavior of one object without impacting other objects. *Reduced Coupling:* Since there is no direct interaction allowed between colleagues, the modular and easily maintainable code produced by the mediator pattern provides better security than tightly coupled code. *Improved Maintainability:* The risk of propagating bugs and errors introduced in one object to other objects is less since the interactions are localized and contained in the respective objects. *Scalability:* Unlike tightly coupled systems, where direct interactions make code complex and difficult to manage, the Mediator pattern provides better scalability as changing one colleague has less impact on the others. *Isolation of Issues:* Since the errors are isolated in a colleague, other colleagues are unlikely to be impacted, making it more secure than tightly-coupled design patterns.

The mediator pattern has disadvantages, such as a single point of failure when the mediator is compromised. However, any design pattern is not inherently insecure, and its security depends on the underlying implementation.

4.5 CephVault Modules

CephVault employs modular design for code isolation, code reusability, enhanced security, and extensibility. In this thesis, the DEK_IV and KEK_IV indicate IVs

associated with the DEK and KEK, respectively.

- **CephVault Key Generation Module:** The *CephVault Key Generation Module* is responsible for generating DEKs and KEKs, which are symmetric keys. Keys are generated using a Cryptographically-Secure Pseudorandom Number Generator (CSPRNG) [148], a True Random Number Generator (TRNG), or Shamir's Secret Shares. We leverage OpenSSL's *rand_bytes()* to generate cryptographically secure pseudorandom numbers. *Rand_bytes()* can use the operating system as an entropy source to seed random numbers automatically. If the entropy source is unavailable, the random number generation process fails. *CephVault* also employs TPM 2.0 as a TRNG.
- **CephVault Key Registration Module:** Encryption keys are associated with the relevant metadata prior to being used. A 128-bit Universally Unique Identifier (UUID) is associated with the DEK and KEK to identify the keys uniquely. During the key registration process, the key creator name, key creation date, key rotation date, and version number are associated with the key.
- **CephVault Key Usage Module:** The *Key Usage Module* handles internal usage of DEKs and KEKs, such as encrypting a DEK with a KEK.
- **CephVault Key Storage Module:** The *CephVault Key Storage Module* possesses the logic to store the DEK and KEK metadata in the database. A DEK is stored in the database in plaintext or encrypted format based on the options selected by the administrator (Section 4.6.3). A KEK can be stored in various places, such as in a configuration file, in Ceph, or in the TPM based on the selected option (Section 4.6.2).
- **CephVault Public Interface Module:** The *CephVault Public Interface* is the first point of contact with *CephArmor*, which requests the DEK and IV for the cryptography operation. For security reasons, only two methods, one for encryption and the other for decryption, are exposed in this module. During deployment, the

header file of this module along with the *CephVault Global Module* are deployed as public, and the rest of the *CephVault* modules are deployed as private (internal to *CephVault*) so that they can not be accessed outside *CephVault*.

- **CephVault Global Module:** The *CephVault Global Module* contains shared functions that provide access to data, such as encryption types, encryption modes, and key length. This module can be accessed by *CephVault* and *CephArmor*. *CephVault* accepts enums as parameters for encryption type, encryption mode, and key length instead of a string, which provides security benefits: *Compile-time Safety:* Being strongly typed, enums reduce runtime errors by catching compile errors. When an attacker provides an incorrect enum value, the KMS stops compilation. *Limited Input Options:* Accepting a malformed string can lead to SQL injection attacks and buffer overflow attacks, whereas enums, which are a predefined limited set of inputs, provide the option to the user to select only the accepted values, making it safer. *Validation and Sanitization:* *CephVault* employs input validation to identify whether the provided enums are within the accepted range, mitigating the risk of accepting malformed strings.

- **CephVault Authentication Module:** The *CephVault Authentication Module* leverages Pluggable Authentication Modules (PAM) [149] for authenticating users. Apart from supporting traditional authentication methods (username and password), PAM also supports various other authentication mechanisms, such as One Time Password (OTP), smart-card, and biometrics, etc., making it suitable for future extensibility of *CephVault* (Section 8.1). PAM has a modular design, which provides easier extensibility for the addition or removal of authentication methods without modifying the application. Also, PAM supports most Linux distributions, making it suitable for the scalability of *CephVault*.

- **Request Processor Module:** The *Request Processor Module* acts as a mediator in the *CephVault* design. This module has access to all other modules and controls

the access when one module communicates with the other modules. This module processes user requests and decides to forward the requests to appropriate modules and functions. This module is responsible for deciding which envelope encryption should be used based on the configuration settings.

- **CephVault Backup Module:** The *CephVault Backup Module* backs up the DEK and its metadata. There are two ways to take a backup: immediate backup or scheduled backup. In the immediate backup approach, the data is stored in the primary table, and a backup is taken instantaneously for the primary table, which provides data consistency. In the scheduled backup, there is a delay in data availability in the backup table, which could lead to data loss in case the primary table is compromised or unavailable prior to taking the backup. *CephVault* employs an immediate backup that follows principles of atomicity in the database transaction: if the backup fails, KMS data is deleted from the primary table, and the entire process fails.

- **CephVault Key Recovery Module:** The KEK can be recovered by the root user. For example, if Shamir's secret shares are used to generate the KEK, it can be recovered by the root or authorized persons by providing the key shares that were earlier generated during the KEK generation process. *CephVault* does not support any recovery of the DEK by the administrator to minimize the security vulnerability; an encrypted DEK corresponding to a user can only be decrypted by *CephVault* after authentication of that user.

- **CephVault Key Rotation Module:** In the key rotation process, the old ciphertext, which was previously encrypted with the old DEK, is decrypted prior to being encrypted with the new key. The key rotation process can be initiated by an administrator or by a scheduled job, where the old DEK is retrieved from the database, which is used by the preexisting *CephArmor*'s API to decrypt the ciphertext stored in the Ceph cluster. Then, a new DEK is created, which is used by the same *CephArmor*'s API to encrypt the data.

- **CephVault TPM Interface Module:** The *CephVault TPM Interface Module* comprises all methods required to interact with a TPM. As supported by the TPM, the primary key is created using ECC. The storage key, which is used as the KEK, is created using RSA. Both the primary key and the KEK are made persistent to improve performance and retain keys during power cycles [2]. The primary key encrypts the private part of the KEK. In cryptography operations, the DEK is sent to the TPM, and the DEK is encrypted by the KEK, where the KEK never leaves the TPM.

The Trusted Computing Group (TCG) [150], which specifies the standards for TPM 2.0, includes AES as a supported encryption algorithm for TPM 2.0. However, The TPM specification does not mandate implementing a particular encryption algorithm [151]. Implementations are left to the TPM manufacturers, and AES encryption is not generally used in TPM for cryptography operations [151]. Limited software support [152] or regulatory requirements [153] can be the reason for limited support for AES encryption. Supporting AES encryption requires the TPM to support the command *tpm2_encryptdecrypt* [154]. We identified that the TPM we employed in this version of *CephVault* does not support *tpm2_encryptdecrypt*, hence AES encryption. So, for *CephVault*, we used RSA, which is the default algorithm for generation of encryption keys in the TPM [151]. In the future, other encryption algorithms can be leveraged (Section 8.1).

- **CephVault Key Destruction Module:** In a traditional data deletion process from the SQLite tables, data is stored onto the disk, and when a record is deleted from the SQLite table using *DELETE* query, the space allocated to the data is set as unallocated, but the data is still available in that space unless the data is overwritten by new data. Attackers can employ digital forensic tools to recover the data. To countermeasure the data compromise, *CephVault* employs an encrypt and update mechanism as depicted in Figure 4.3. In *CephVault*'s key destruction process,

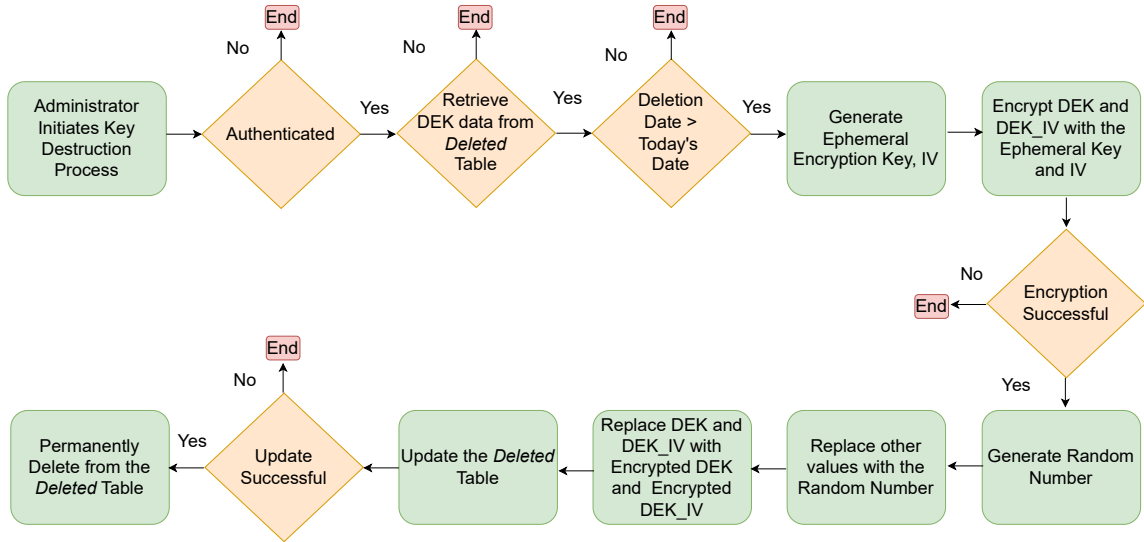


Figure 4.3: CephVault Key Destruction Process

deleted keys are stored in deleted tables temporarily, and a root user retrieves the deleted records after successful authentication. Then, each record is checked, and if the deletion date for the record is greater than the current date, the record is marked for deletion. For the record, which is scheduled for deletion, an ephemeral encryption key and IV are generated, which are used for encryption of the DEK and the DEK_IV. These ephemeral keys are not stored anywhere; their sole purpose is to encrypt the sensitive data of the record. Once the encryption process is complete, the key and IV can not be retrieved by the attacker. Upon successful encryption, a random number is generated, which is used to replace other non-sensitive or public parameters in the record. Integer values (non-sensitive) follow the zeroization process, where the values are made to zero. Once the record is updated with the encrypted values, random numbers, and zero values, the record is deleted from the table, making it a permanent deletion, where old values cannot be recovered.

- **CephVault Key Monitoring Module:** The *CephVault Key Monitoring Module* provides functionalities to monitor various activities to safeguard *CephVault*. Prior to starting processing data encryption and decryption requests from users, the checksum of the *CephVault* binaries and the KEK are generated and stored in the

database. During *CephVault*'s operations, a scheduled job checks whether there is any alteration in the binaries and the KEK, and if there are any changes, the intended recipients are notified immediately. This process safeguards *CephVault* from unauthorized modification. Also, *CephVault* monitors any use of vulnerable functions in the code.

- **CephVault Key Revocation Module:** The *CephVault Key Revocation Module* enables a root user to revoke a compromised encryption key. The *CephVault* key revocation process follows the same process as the key rotation process, with an additional step of destroying the old DEK records permanently by encrypting the compromised key with another encryption key, generated ephemerally, and updating the record prior to deleting the record from the database.

- **CephVault Key Distribution Module:** The *CephVault Key Distribution Module* is used to transmit the encryption keys and IVs to various entities. In the software approach, the KEK is distributed to a secure location in Ceph or in a configuration file. In the hardware approach, the KEK is stored in the TPM. Also, a pre-shared symmetric key and an RSA key pair used for digital signature are stored in the user's home directory.

- **CephVault Audit Module:** *CephVault* leverages SPDLog [155], an open-source, fast C++ logger, to log application specific events. There are a total of four different log levels: Debug, Info, Error, and Critical. *CephVault* admins can configure the log level based on their requirements. Admins have the flexibility to choose from the *CephVault* configuration file whether to log to the console, log to a file, or log to both console and file. In *CephVault*, each method is secured with exception handling, and logs are taken for each exception, error, and critical issue in the application. At the same time, an email is triggered to the intended recipients when an issue occurs. To provide data security, *CephVault* does not log any sensitive information such as the key and the IV. *CephVault* also lays the foundation to store the logs in the database,

which can be encrypted for more security (Section 8.1).

- **CephVault Notification Module:** The *CephVault Notification Module* leverages Postfix [156], a widely used Mail Transfer Agent (MTA), an Ubuntu server for sending an email using Gmail's SMTP server. Organizations can modify email contents and sender or requester email IDs as per their requirements without any major code changes. The *CephVault* notification system provides real-time alerts to the intended recipients and safeguards the application. Each function is secured with exception handling, and whenever there is any exception, an email is triggered. Emails are sent in various error and critical events, such as when someone tries to log in to *CephVault* and fails for the third time.

- **CephVault Validation Module:** *CephVault* accepts user inputs, such as namespace, object name, and pool name in string format. The *CephVault Validation Module* checks whether the user provides any malformed input that could lead to SQL injection attacks. This module is also responsible for validating the output of the internal functions, such as whether the database name is valid, whether the email body has malformed strings, whether folder names are valid, or whether the requested key sizes are supported by the *CephVault* supported encryption algorithms.

- **CephVault Digital Signature Module:** The *CephVault Digital Signature Module* contains the required functions to execute the digital signature process for *CephVault*. OpenSSL's EVP library [157], a CSPRNG, reliable, and widely accepted library, is employed to generate a 2048-bit RSA key pair. The 2048-bit RSA key provides a high level of security, making it difficult for malicious users to break [158]. The RSA key pair is stored in a secure location in the user's home directory, reinforced by authentication and access control, allowing users the flexibility to rotate keys at their convenience. Security analysis of storing keys in the user's home directory is described in Section 7.7.1. *CephVault* employs hybrid encryption with the RSA digital signature to provide confidentiality, integrity, and non-repudiation.

When a DEK is required to be sent to *CephArmor*, it is encrypted with a pre-shared symmetric key, and the encrypted DEK (provides confidentiality) is signed by the private key of the RSA digital signature. The encrypted DEK is signed to ensure that it is not altered by a malicious user during the transmission (provides integrity), and *CephVault* would not be able to deny that the key was generated by *CephVault* (provides non-repudiation). Upon receiving the encrypted DEK and digital signature, *CephArmor* validates the signature, and once the signature is verified, it decrypts the encrypted DEK using the pre-shared symmetric key. In *CephVault*, administrators have the flexibility of selecting whether to choose the digital signature to cater to the organization's requirements.

- **CephVault Initialization Module:** The *CephVault Initialization Module* contains functionalities required for *CephVault* to start its operations. An authenticated root user can initiate the initialization process to perform various required actions. *Install Required Packages:* Prerequisite packages are checked and installed in case the packages are missing. *Create Required Databases:* Required databases and tables are created. *Checksum on Binaries:* Checksums of the *CephVault* binaries are taken to mitigate possible modification of the files during the operations. Checksums are stored in the database, which are used by the monitoring tool to identify possible code alternations. *Configure Email Support:* Required installations and configurations are performed programmatically to support *CephVault*'s real-time notification system. *Setup SQLite Defensive Flags:* *CephVault* leverages an SQLite database to store the encrypted DEK and metadata regarding the KEK since an SQLite database does not incur any additional cost and provides required storage support. SQLite also provides security against buffer overflow memory leaks and resistance to crashes [159]. However, additional security can be employed to mitigate any unforeseen events. *CephVault* employs various defensive flags recommended by SQLite [159] that reduce the SQLite default inputs and set a high-security value.

```

● root@15drivesnode2 /local_scratch/fkhoda/ceph (CephVault_Perf)$ cephvault -status
Checking required CephVault Header Files..
cephVaultGlobalsHeader.h: Installed
cephVaultPublicInterface.h: Installed
Checking required CephVault Library installation..
libcephvault-config-noconfig.cmake: Installed
libcephvault-config.cmake: Installed
libcephvault.so: Installed
Checking CephVault Command Line Interface..
CephVault Command Line Interface: Installed
Checking required package installation..
*** DEVELOPER MODE: setting PATH, PYTHONPATH and LD_LIBRARY_PATH ***
ceph version: 15.0.0-26882-g1c6740e3e8a (1c6740e3e8abe21a9368aa408d143cb2922c956a) pacific (stable)
Ceph: Installed
SQLite 3: Installed
Postfix: Installed
PAM: Installed
Boost: Installed
Boost FileSystem: Installed
SPDLOG: Installed
● root@15drivesnode2 /local_scratch/fkhoda/ceph (CephVault_Perf)$ cephvault -help
Usage: cephvault <command> [args]

Commands                                                                 Description
-version                                                                    Shows version

```

Figure 4.4: CephVault Commands to Get Status

- CephVault Command Line Interface:** *CephVault* is developed with command line support, where authenticated root users can perform various tasks, such as key rotation, key revocation, creation of the KEK, and initialization of *CephVault*. Figure 4.4 depicts an example of the *CephVault* command.
- CephVault Configuration Module:** The *CephVault Configuration Module* is one of the most sensitive modules that contains various settings that influence the behavior of *CephVault*. The module contains various settings in the centralized code file, eliminating the risk of storing any hard-coded values in other modules. The configuration file is stored in a secure location only accessible to a root user.
- CephVault Utilities Module:** The *CephVault Utilities Module* contains the functions that are common to all other modules. Functions such as getting relative path, getting target date, and converting binary to hexadecimal string are implemented in this module.
- CephVault Database Connection Module:** The *CephVault Database Connection Module* provides functionalities to connect to the SQLite database used by other

CephVault modules. In this module, a scoped database transaction technique [160] is employed, where a block of code comprising a series of database operations is used to perform all the operations as a single transaction. Benefits of employing scoped SQLite transactions are: *Atomicity*: Either all database operations inside the block are successful, or a failure of a single operation fails the entire block as a single unit, providing data consistency. *Data Integrity*: Since a partial data update is not supported, data integrity is preserved. *Resource Cleanup*: The resource cleanup is encapsulated in the destructor, which ensures closing the open connections resources, and ensuring automatic handling of memory leaks. *Exception Handling*: Exception handling prevents the committing of erroneous transactions to be committed.

4.6 CephVault Configurable Operations

Administrators can select the various configurable options as depicted in Figure 4.5 to operate *CephVault*. Based on how cryptography requests are processed, *CephVault* can be operated with different modes of operation. For each mode of operation, the KEK can be kept in various locations, and for each location, administrators can select various envelope encryption schemes. The primary objective of supporting various configurable options is to improve performance and reduce storage requirements for *CephVault*.

4.6.1 Options for CephVault Modes of Operation

CephVault supports two modes of operation: per-user or per-object basis.

- **Per-User:** In the per-user mode of operation, a unique DEK is generated for each user; however, the same DEK is used to encrypt all objects belonging to the user. So, all unique objects, pools, and namespace combinations belonging to the user have the same DEK. When envelope encryption is enabled (Section 4.6.3), at the

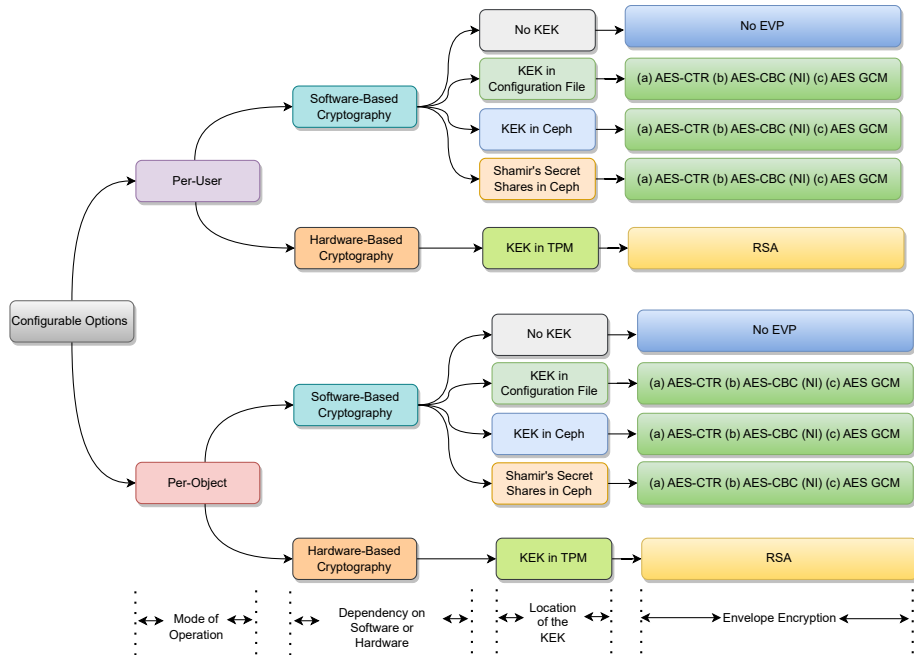


Figure 4.5: CephVault Configurable Options

time of the data encryption request, *CephVault* checks whether the requester already exists in the database, and if any record is found for the requester, the encrypted DEK is retrieved from the database, and decrypted using the KEK. However, for every request, a new DEK_IV is generated. So, even though the DEK is the same for all objects belonging to the user-based, the DEK_IV is different. On the other hand, if no record exists in the database, a new DEK is generated and encrypted using the KEK prior to sending the DEK to *CephArmor*. Since the same DEK is used for all objects belonging to the user, the encrypted DEK stored in the database is vulnerable to chosen plaintext attacks.

- **Per-Object:** In the per-object mode of operation, a unique DEK is generated for each unique object, pool, and namespace combination belonging to the user. So, all unique object, pool, and namespace combinations in the database (belonging to the same user or different users) have different DEKs. Similarly, for each request, a new DEK_IV is generated. When envelope encryption is enabled, each DEK is encrypted using the KEK prior to sending the DEK to *CephArmor*. This mode does not suffer

from chosen plaintext vulnerability as encrypted DEKs stored in the database are unique.

4.6.2 Options for Location of the KEK

Software-based or hardware-based cryptography can be selected, where the former supports software-dependent KEK generation and storage, while the latter utilizes a TPM for KEK generation and storage. Software-based cryptography options are: (a) No KEK, (b) KEK in the configuration file, (c) KEK in Ceph, or (d) Shamir's secret shares in Ceph, whereas KEK in TPM is the only choice available for hardware-based cryptography in *CephVault*.

- **No KEK:** No KEK option is automatically selected when there is no envelope encryption option selected (Section 4.6.3).
- **KEK in Configuration:** *CephVault* supports saving the KEK in a configuration file stored in a secure location in the server.
- **KEK in Ceph:** In the KEK in Ceph option, a 32-bit random number is stored in a secure location in Ceph.
- **Shamir's Secret Shares in Ceph:** *CephVault* provides an option to store Shamir's secret shares in a secure location in Ceph. A KEK is generated using these shares during cryptography operations.
- **KEK in TPM:** *CephVault* employs a TPM to store the KEK. The primary motivations behind choosing a TPM as the hardware-based storage solution for *CephVault* are security, cost-effectiveness, and compliance with FIPS 140-2 guidelines. The TPM provides a secure enclosure, which is economical (can be as inexpensive as ten dollars [151]).

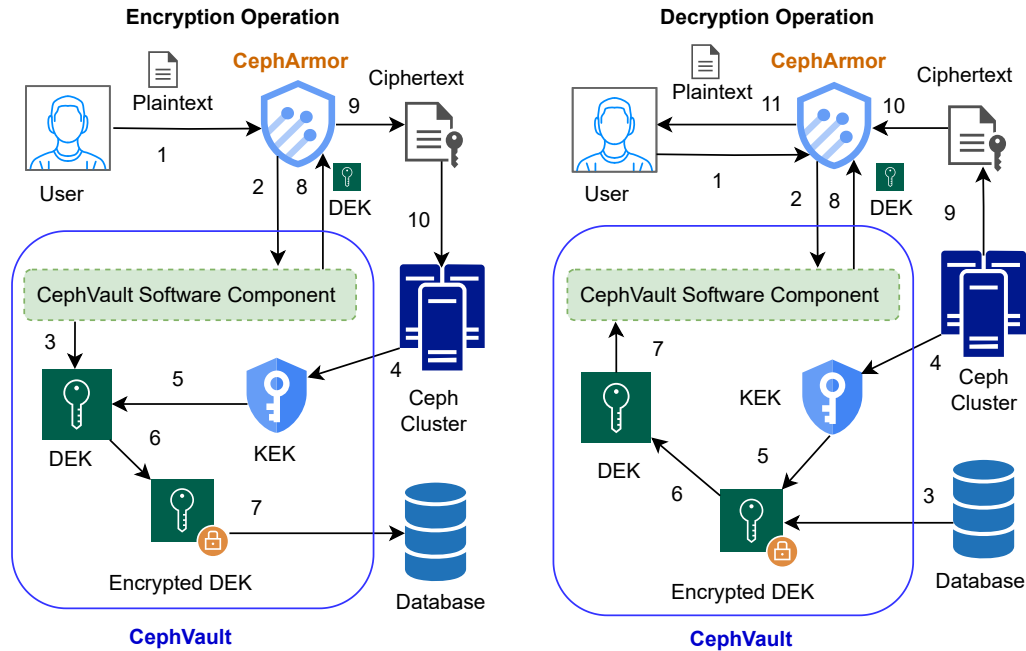


Figure 4.6: Software-Supported Envelope Encryption

4.6.3 Options for Envelope Encryption

CephVault provides various options for envelope encryption.

- **No Envelope Encryption:** *CephVault* provides an option for storing a DEK in the database without any envelope encryption. Although keeping the DEK in plaintext is vulnerable to data leakage, it provides a baseline for *CephVault*, the closest option to not having a KMS for *CephArmor*. Moreover, providing this option enables organizations to choose various levels of security based on their requirements.

- **Software-Supported Envelope Encryption:** Software-supported envelope encryption leverages computer memory to perform encryption-decryption operations. *CephVault* employs various modes of AES encryption: AES-CBC with NI support, AES-CTR, or AES-GCM for envelope encryption. For simplicity purposes, the components of Ceph are omitted, and the user interacts directly with *CephArmor*. Software-supported envelope encryption is depicted in Figure 4.6. During encryption, the user provides a plaintext, along with a Ceph namespace, an object name,

and a pool name to *CephArmor*, which in turn interacts with *CephVault*. *CephVault*'s software component accepts the request and generates the DEK. Then, the KEK is retrieved from the configuration file or Ceph (based on the selected location of the KEK described in Section 4.6.2). Next, the DEK is encrypted using the KEK, and the encrypted DEK is stored in the database. After that, the DEK is sent to *CephArmor*, which uses the DEK to encrypt the data. The ciphertext is stored in the Ceph cluster.

During decryption, the user requests to decrypt the stored data by providing a Ceph namespace, an object name, and a pool name to *CephArmor*, which interacts with *CephVault*. *CephVault*'s software component retrieves the encrypted DEK from the database. Then, the KEK is retrieved to decrypt the encrypted DEK. Next, the DEK is provided to *CephArmor*, which decrypts the ciphertext retrieved from the Ceph cluster and sends the plaintext to the user.

- **Hardware-Supported Envelope Encryption:** In *CephVault*, a hardware-supported envelope encryption leverages a TPM to perform encryption-decryption operations, as depicted in Figure 4.7. Cryptography operations are similar to the software-supported envelope encryption schemes, except the KEK is generated in the TPM.

4.7 Processing of Cryptography Requests

Cryptography requests are processed based on the mode of operation.

4.7.1 Data Encryption Request (Per-User)

Figure 4.8 depicts data encryption requests as below.

- The user requests data encryption by providing the location of the file the user wants to encrypt, the name of the object that would store the encrypted data in

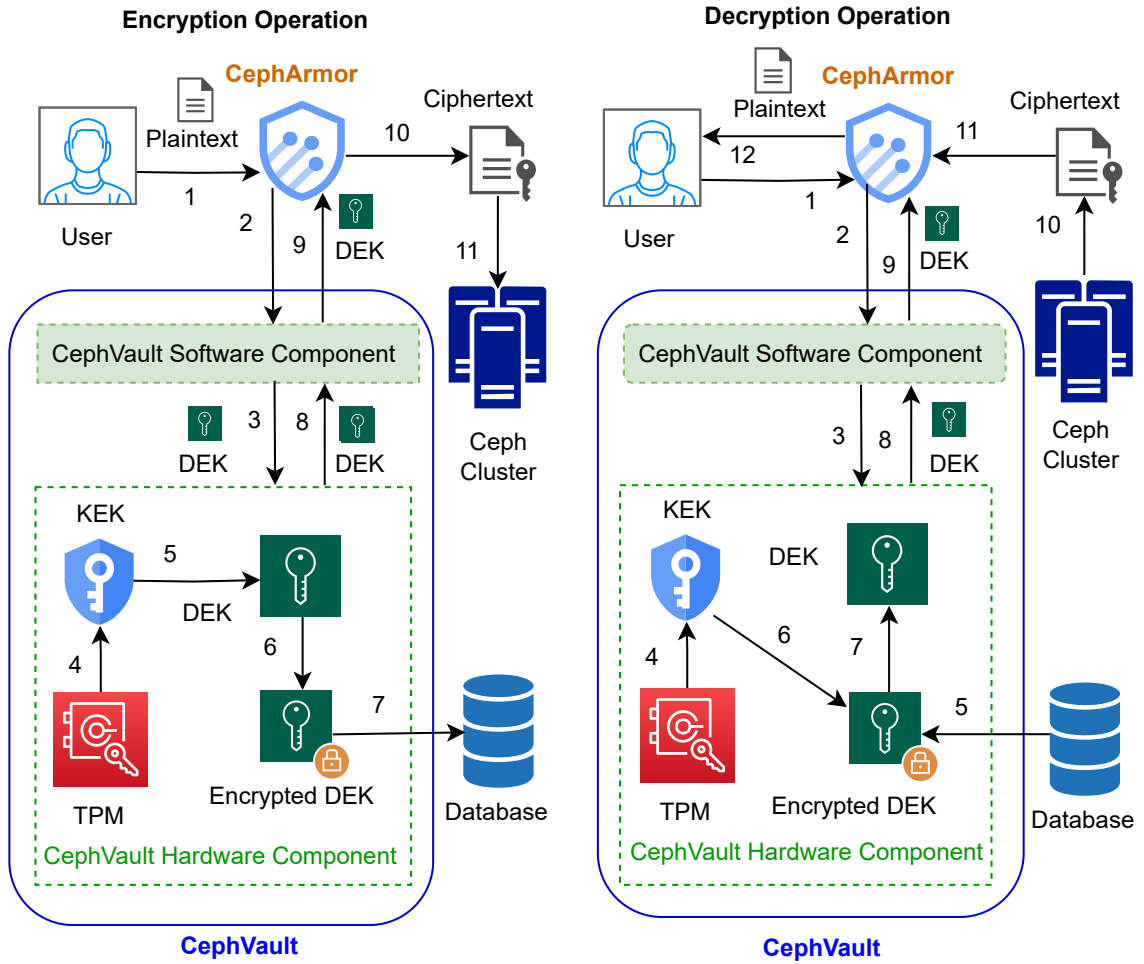


Figure 4.7: Hardware-Supported Envelope Encryption

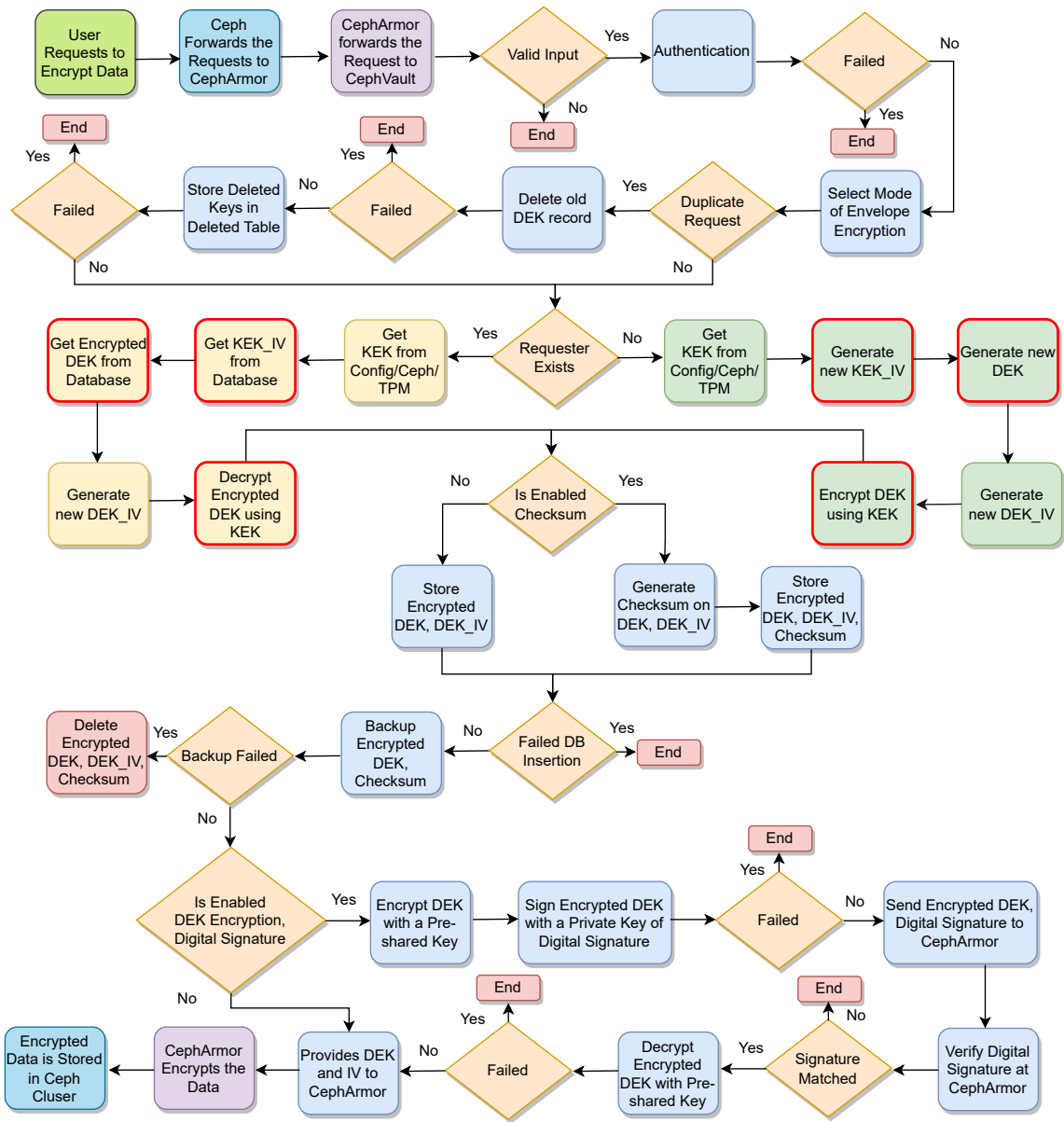


Figure 4.8: Processing of Data Encryption Request: Per-User

Ceph, the pool name where the object belongs, and the namespace associated with the pool. The user provides the command:

```
rados put objectName /path/fileName.txt -p poolName -N namespace
-e encryptionMode
```

- Ceph forwards the request to *CephArmor*, which sends the request to *CephVault*.
- *CephVault* validates the input for a malformed string, which can lead to SQL injection, unsupported encryption mode, unsupported key length, and IV length.
- Then, the user is authenticated and validated for proper access.
- When a user tries to store data in Ceph and provides an object name, pool name, and namespace, which already exists, Ceph's default behavior is to overwrite the old object and store the new one. Our research team decided to keep Ceph's default behavior the same for storing encrypted data. When the user sends the request for data encryption, the *CephVault* stores the object name, pool name, and namespace in the database for future reference. For every user request for data encryption, the request is validated in the database for a duplicate object. Based on whether there is a duplicate object present or not, the request can be processed as:

- **Duplicate Object Present:** If any duplicate entry is found in the database, the duplicate entry is deleted. Even though the duplicate entry is deleted from the primary table, the entry is not immediately removed from the databases. The deleted entry is stored in a separate table temporarily to support key recovery for accidental deletion by providing a duplicate object name. Later, the deleted keys are destroyed permanently by the key destruction process.
- **Duplicate Object Not Present:** If there is no duplicate object present in the database, the process moves to the next step.
- The next step is to validate whether the requester exists in the database or not. Based on the result, there are two ways the data encryption request is processed.

- **Requester Exists:** When a requester already exists in the database, the KEK is retrieved from the configuration file, Ceph, or TPM based on the selected envelope encryption mode. The KEK_IV and encrypted DEK are retrieved from the database. For each encryption request, even though the DEK is the same for a user, a new DEK_IV is generated for data security. Next, the encrypted DEK is decrypted using the KEK.
- **Requester Does Not Exist:** The first time a user interacts with *CephVault*, there would not be any data stored in the database, so a new DEK is generated, along with a new DEK_IV, KEK_IV, and other metadata. Next, the DEK is encrypted with the KEK retrieved from the configuration file, Ceph, or TPM.
- *CephVault* makes generation of a checksum on key and IV optional, where administrators can enable or disable the options from a configuration file. This flexibility enables organizations to select various security levels based on the requirements, where the trade-off is performance vs. security.
 - **Checksum Enabled:** A checksum is generated on the DEK and DEK_IV using SHA-256 [161], which is used for validating the integrity of the DEK and DEK_IV during decryption requests.
 - **Checksum Disabled:** No checksum is generated on the DEK and DEK_IV.
- Once the required key, IV, and checksum (conditional) are generated, they are stored in the database with the associated metadata.
- Next, a backup is taken of the DEK, DEK_IV, KEK_IV, checksum, and other metadata when the data is successfully stored in the database.
- If the backup fails, the stored DEK and other metadata are deleted from the database to remove stale data.
- Administrators are provided with the flexibility to either choose a secure key distribution to *CephArmor* or a relatively more performant option.

- **Secure Key Distribution:** When administrators select this option, a pre-shared symmetric key, generated by the user during the trust-building operation with *CephVault*, is used to encrypt the DEK. Next, the encrypted DEK is signed by the private key of the RSA-based digital signature algorithm. The digital signature provides data integrity and non-repudiation. Next, the encrypted DEK, digital signature, and DEK_IV are sent to *CephArmor*. Then, *CephArmor* verifies the digital signature, and if the digital signature is verified successfully using the public key of the digital signature, the encrypted DEK is decrypted with the pre-shared symmetric key.
- **Performant Key Distribution:** When performant key distribution is selected, the plaintext DEK and DEK_IV are sent to *CephArmor*
- Next, *CephArmor* uses the DEK and DEK_IV to encrypt the data and store the encrypted data in the Ceph cluster.

4.7.2 Data Encryption Request (Per-Object)

When per-object basis mode is enabled, the data encryption request facilitated by *CephVault* is almost the same as the one depicted in Section 4.7.1, except there is no check for an existing user in the database as a new DEK is generated for each data encryption request. Figure 4.9 depicts the data encryption request in the per-object mode, where the highlighted yellow background area demonstrates the difference between the per-user and per-object mode.

4.7.3 Processing Decryption Requests (Per-User and Per-Object)

Figure 4.10 depicts the data decryption request process applicable to all *CephVault* modes of operation.

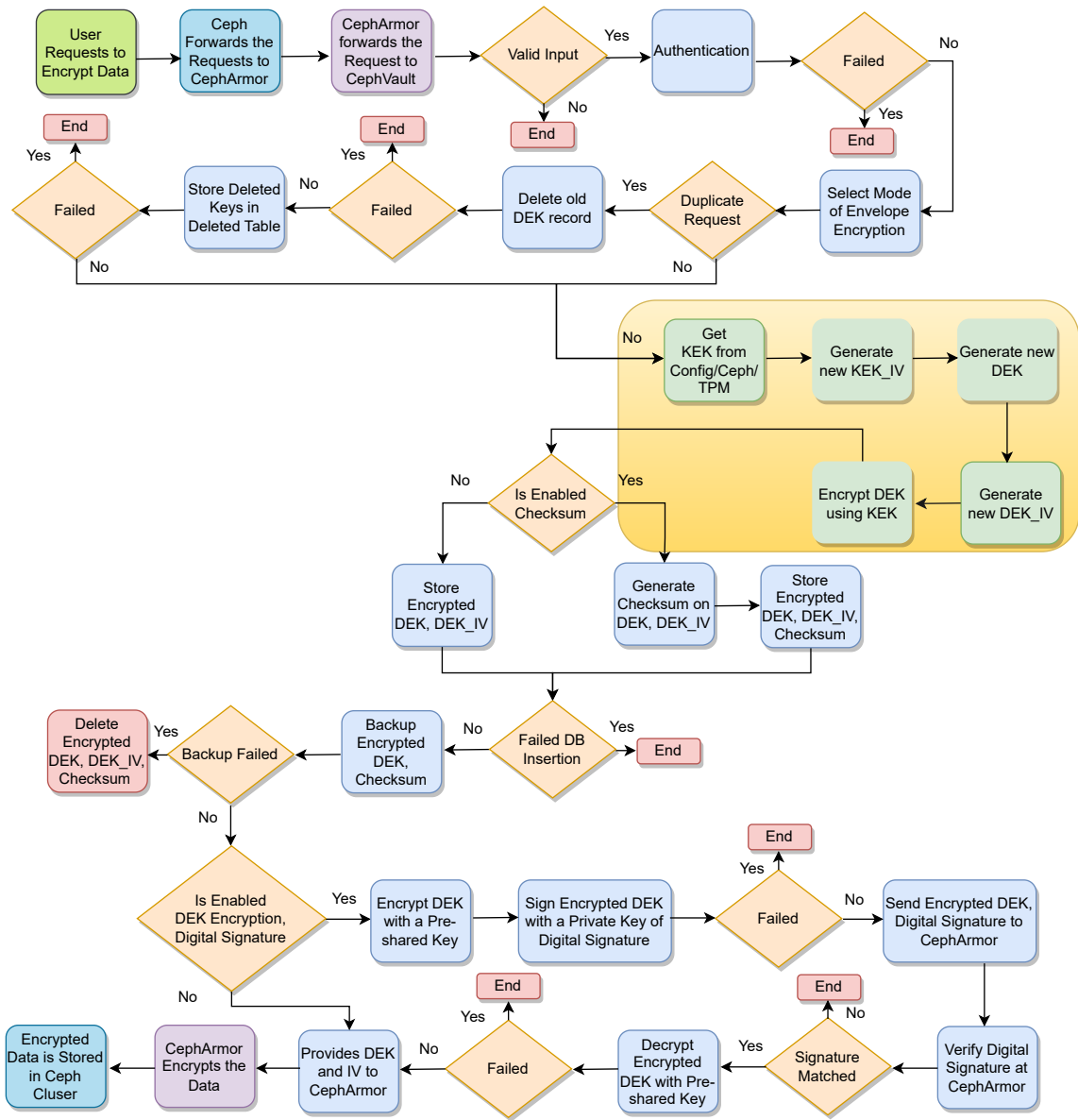


Figure 4.9: Processing of Data Encryption Request: Per-Object

- The user requests data decryption by providing the location of the file for which the user wants to view the decrypted data, the name of the object that stored the encrypted data in Ceph, the pool name where the object belongs, and the namespace associated with the pool. The user provides the command:

```
rados get objectName /path/fileName.txt -p poolName -N namespace  
-e encryptionMode
```

- Ceph forwards the request to *CephArmor*, which sends the request to *CephVault*.
- *CephVault* validates the input for a malformed string, which can lead to SQL injection, unsupported encryption mode, unsupported key length, and IV length.
- Then, the user is authenticated and validated for proper access.
- Next, the user request is validated to identify whether the record exists in the database, which implies whether the user encrypted the data previously.
- When the entry is found in the database, the master key, or the KEK, is retrieved from either Ceph or TPM (based on the choice of envelope encryption). If there is no record found, the process ends, and the user is notified.
- Then, the encrypted DEK, along with the DEK_IV, are retrieved from the database.
- Next, the encrypted DEK is decrypted with the KEK.
- Similar to the encryption request processing, checksum generation on key and IV is optional.

- **Checksum Enabled:** A checksum is generated on the DEK and DEK_IV using SHA-256. Then, the stored checksum is retrieved from the database to validate whether the generated checksum matches the stored checksum. If the checksum matches, the process goes to the next step of generation of digital signature, if enabled.

- **Checksum Disabled:** No checksum is generated on the DEK and DEK_IV, and there are no internal checks to identify whether the encrypted DEK or DEK_IV has been altered.

- Similar encryption, administrators can either choose a secure key distribution to *CephArmor* or a relatively more performant option.
- After encrypted data is retrieved from the Ceph cluster, *CephArmor* decrypts the ciphertext.

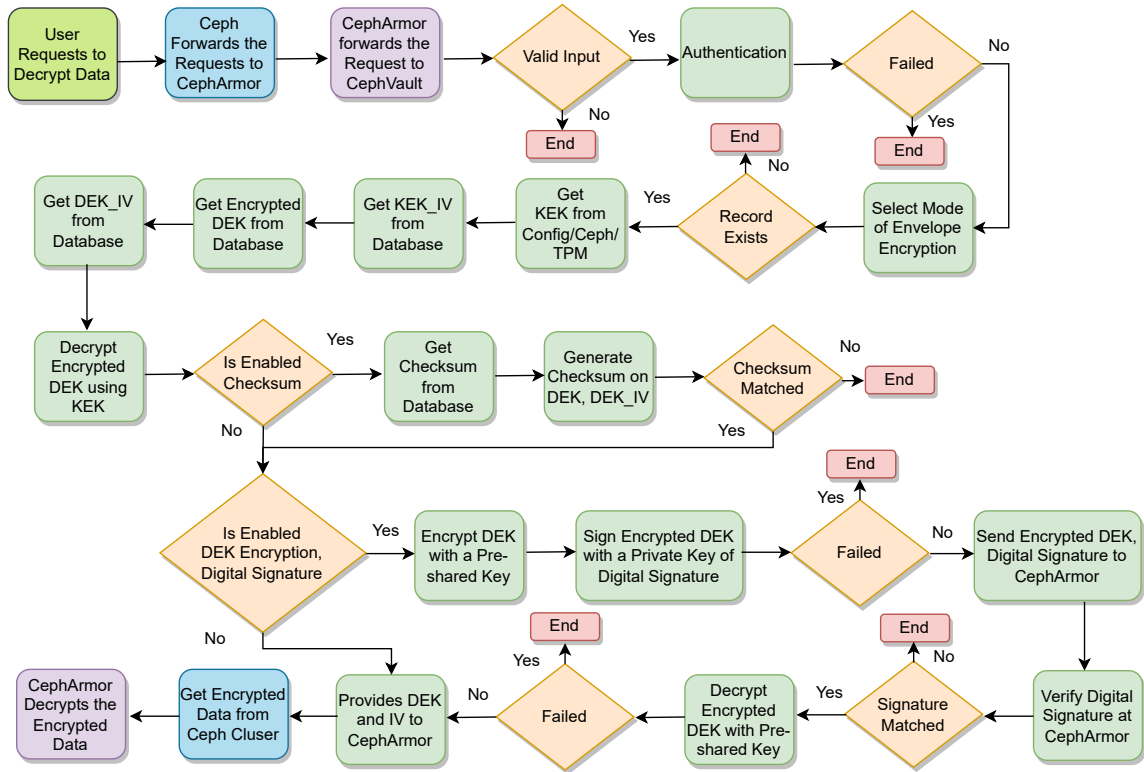


Figure 4.10: Processing of Data Decryption Request: Per-User and Per-Object

4.8 Summary

In this chapter, we discussed the architecture of *CephVault*, a brief description of *CephVault*'s modular design, followed by a detailed account of each component. Next, we delved into *CephVault*'s configurable options. Then, we discussed how encryption and decryption requests are processed by *CephVault* in per-user and per-request modes of operations. In the next chapter, we will discuss how to reproduce our work by providing the experimental details.

Chapter 5

Performance Analysis

This chapter discusses performance overhead introduced by *CephVault*. We evaluated *CephVault* with various data sizes, modes of operation, locations of the KEK, envelope encryption schemes, use cases, and number of items present in the database to identify performant options for *CephVault*.

5.1 Experimental Setup

CephVault was deployed on a 45Drives Storinator, an enterprise open-source storage server comprising forty-one hard drives and three servers. However, the evaluation of *CephVault* was performed in a single server. Having a dedicated server provided us with more accurate results without interference from other processes. However, in the future, *CephVault* can be evaluated against multiple servers in a Ceph cluster (Section 8.2). Cryptography operations were performed in Ubuntu 20.04.6 LTS. The Ceph node (server) contains a total of 14 OSDs together, providing 76 Terabytes of storage space. Also, the Ceph node connected to the user's computer facilitates data encryption and decryption commands through a local network. The latency in the network might impact the performance of the cryptography operations (Section 8.2 for future research directions).

5.2 Evaluation Methodology

Test cases were executed 30 times for each combination of mode of operation, location of the KEK, envelope encryption schemes, etc. We excluded the five best and worst outcomes and averaged the rest. We denote envelope encryption as *EVP*, the KEK in the configuration file as *KEK in Config*, the KEK in Ceph as *KEK in Ceph*, Shamir’s secret shares in Ceph as *SSS in Ceph*, absence of the KEK as *No KEK*, the KEK in TPM as *KEK in TPM*, and without any envelope encryption as *No EVP*. From the above configurable options, *No KEK*, *KEK in Config*, *KEK in Ceph*, and *SSS in Ceph* are part of software-based cryptography, and *KEK in TPM* is part of hardware-based cryptography.

When the user sends an encryption request to *CephArmor*, it is forwarded to *CephVault*. Similarly, a decryption request from the user is sent to *CephVault*. Throughout this study, the terms *encryption request* and *encryption operation* are used interchangeably. Likewise, the terms *decryption request* and the *decryption operation* are used interchangeably.

5.3 Performance Analysis of CephArmor with CephVault

We aimed to assess the overall performance of *CephArmor*, comparing scenarios with and without *CephVault*. This evaluation aimed to assist organizations in making informed decisions regarding the adoption of *CephVault* as a preferred KMS. Initially, we examined *CephArmor*’s performance across various data sizes, utilizing the hard-coded DEK and IV. Subsequently, we analyzed *CephArmor*’s performance in conjunction with *CephVault* and compared it with the previous results (Figure

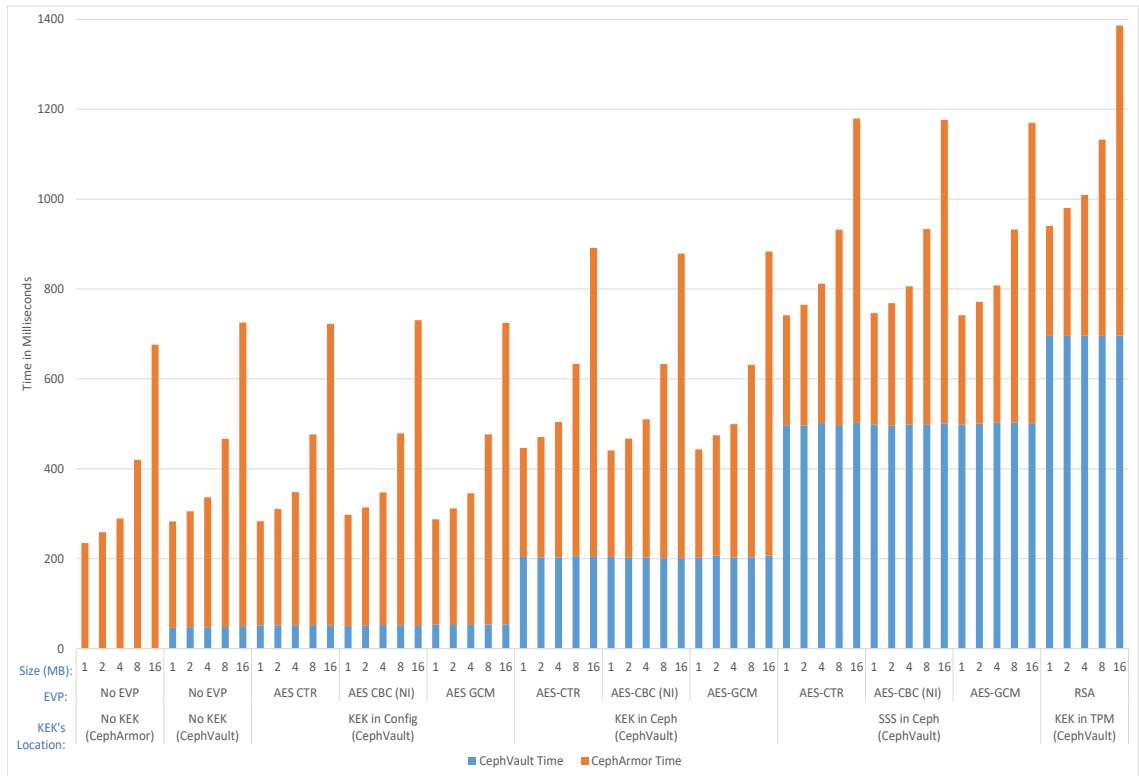
5.1 and 5.2). In this context, *CephArmor's Time* represents the duration taken by users to interact with *CephArmor's* API, coupled with the time *CephArmor* spent on cryptographic operations based on user requests. On the other hand, *CephVault's Time* indicates the time consumed by *CephVault* in performing KMS operations, encompassing key generation, usage, and storage. To ascertain the minimum additional time introduced by *CephVault* in the total execution time, we operated *CephVault* with *No KEK*, implying *No EVP* was employed. This option was the closest alternative to not having a KMS for *CephArmor*. On average, *CephVault* introduced overhead of 47.7 and 47.3 milliseconds in encryption, and 39.2 and 38.4 milliseconds in decryption, in per-user and per-object modes, respectively. With our analysis, organizations will discover that using *CephVault* alongside *CephArmor* incurs a minimal performance overhead of approximately 48 milliseconds for encryption and 39 milliseconds for decryption. *CephVault's* performance with respect to other configurable options will be discussed in the following experiments, where *CephVault's* performance will be analyzed more exclusively.

5.4 Performance Analysis of CephVault

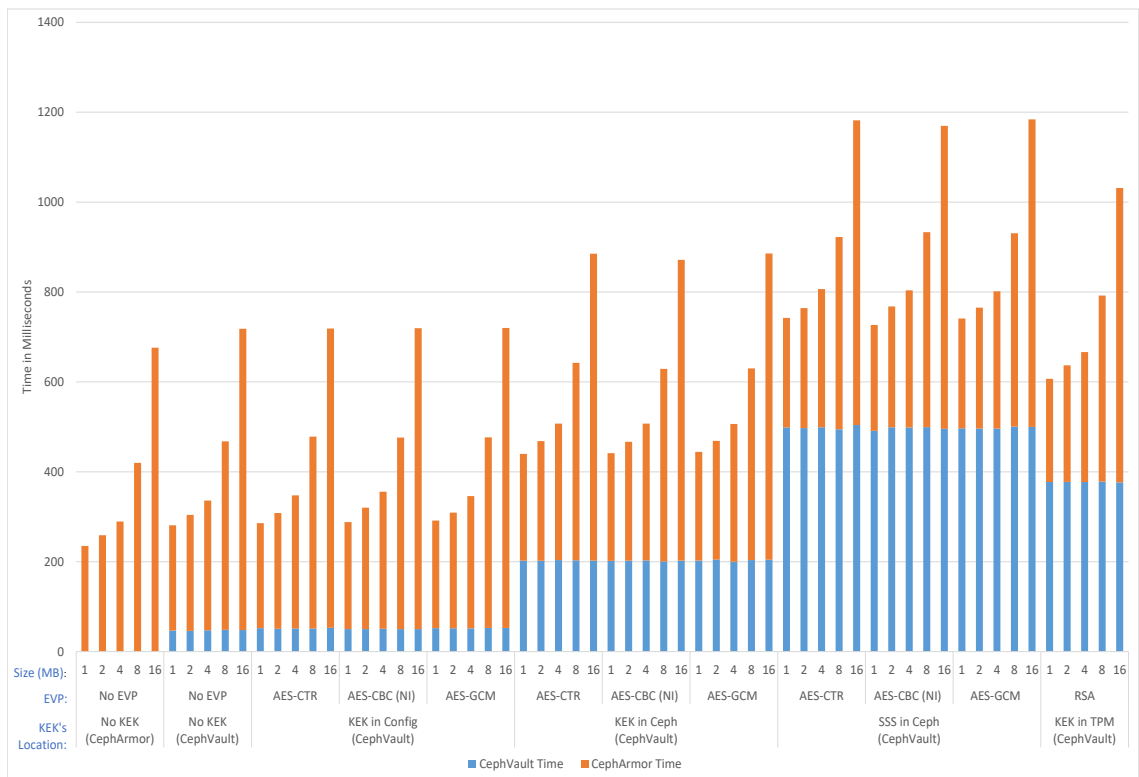
Previously, we analyzed the performance of *CephArmor* with and without *CephVault*, and determined the minimum overhead introduced by *CephVault*. In the following experiments, we will focus our analysis solely on *CephVault's* performance.

5.4.1 Dependency on Data Sizes

Since users leverage *CephArmor* to perform cryptography operations on various data sizes, we will investigate whether the performance of *CephVault* depends on data size. Our evaluation included varying data sizes (1 MB, 2 MB, 4 MB, 8 MB, and 16 MB) in both per-user and per-object modes comprising encryption (Figure 5.1) and de-



(a) Encryption Operations (Per-User)



(b) Encryption Operations (Per-Object)

Figure 5.1: Dependency on Data Sizes on Encryption Operations

ryption (Figure 5.2) operations. We considered different *EVP* configurations and locations of the KEK in our experiments. Notably, our results indicate that regardless of data size and for a specific location of the KEK, *CephVault*'s performance remains consistent. This observation holds true across different configurations such as *No KEK*, *KEK in Ceph*, *SSS in Ceph*, or *KEK in TPM*. It is evident in both modes of operation, encompassing both encryption and decryption processes. During cryptography operations, *CephVault* only accepts parameters: object, pool, namespace, and types of AES encryption. *CephVault* is responsible for generating keys or retrieving keys from the database based on these parameters. In contrast, *CephArmor* is responsible for performing cryptography operations on various data sizes.

We conclude that data sizes do not deter *CephVault*'s performance. The following experiments will discover the contributing factors for *CephVault*'s performance.

5.4.2 Dependency on Modes of Operation

In this experiment, we aim to find out how modes of operation (per-user and per-object) can impact *CephVault*'s performance. In the per-user mode, when the user exists in the database, subsequent encryption requests for the user do not create a new DEK; instead, the old encrypted DEK is retrieved from the database, and the encrypted DEK is unwrapped using an envelope decryption scheme. On the contrary, in per-object mode, even if the user exists in the database, a new DEK is generated, and the DEK is wrapped using an envelope encryption scheme for each subsequent encryption request. However, when *No EVP* is selected, there is an exception in the above process: no wrapping or unwrapping of the DEK is performed. In *No EVP*, the existing DEK in plaintext is retrieved from the database (per-user mode), or the newly generated plaintext DEK is stored in the database (per-object mode). For decryption requests, both modes of operation follow the same steps: the encrypted DEK is retrieved from the database and unwrapped to generate the DEK (exception:

No EVP, where the plaintext DEK is retrieved, and no unwrapping is performed). The experiments (Figure 5.3) were conducted when at least one user was present in the database.

Software-based cryptography operations (*No KEK*, *KEK in Config*, *KEK in Ceph*, or *SSS in Ceph*) took almost the same time in per-user and per-object modes. For example, in encryption, *KEK in Config* consumed 52.0 and 51.3 milliseconds for the per-user and per-object mode, respectively. Similarly, *KEK in Ceph* took 203.1 and 202.6 milliseconds for the per-user and per-object mode, respectively. This can be attributed to the identical steps described in Section 4.6.1, with the exception of the envelope encryption or envelope decryption step. For encryption requests, envelope encryption (per-object mode) and envelope decryption (per-user mode) took almost the same time. On the contrary, hardware-based cryptography operations (*KEK in TPM*) showed a significant difference in processing times between per-user and per-object modes, which can be attributed to how TPM handles encryption and decryption requests. TPMs are generally used for secure storage of keys, and the slow performance is an expected behavior [73][162]. In the per-user mode, the TPM took 639.0 milliseconds to perform the envelope decryption operation, contributing to 696.0 milliseconds of total execution time. On the other hand, in the per-object mode, the TPM took 323.7 milliseconds to perform the envelope encryption operation, contributing to the 377.3 milliseconds of total execution time. For decryption requests, since both modes follow the envelope decryption process, they exhibited almost the same results.

In summary, our initial assumption was to observe a significant performance improvement in the per-user mode compared to the per-object mode for encryption requests since no new DEK was generated for the per-user mode. The primary difference in

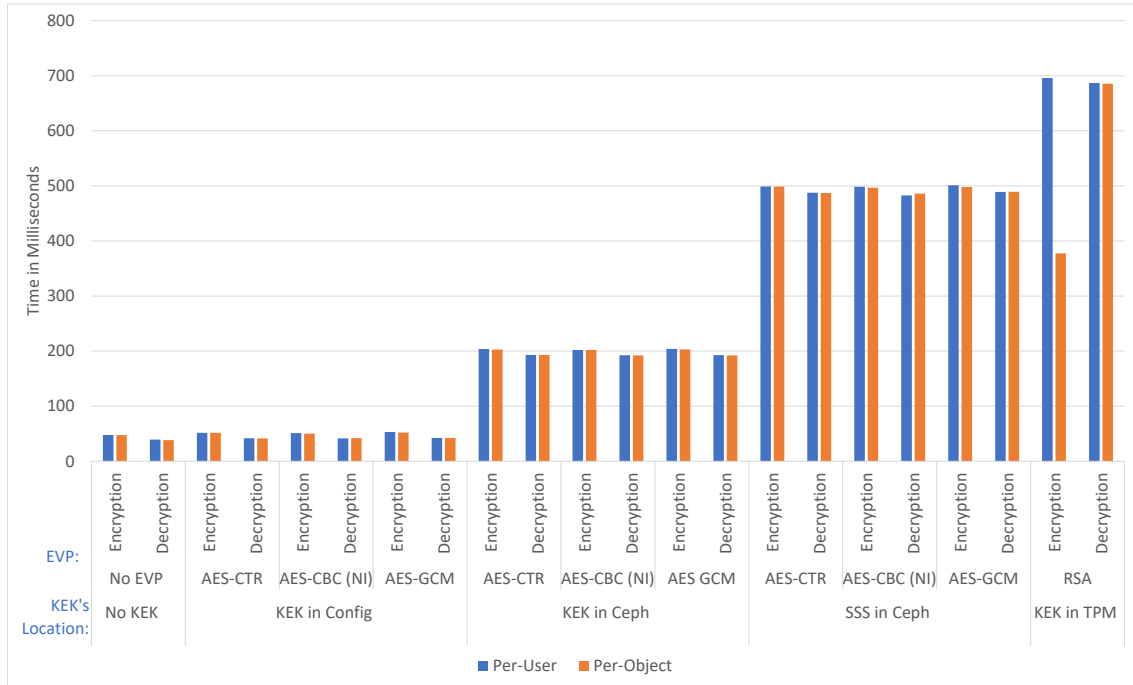


Figure 5.3: Comparative Analysis: Per-User Vs. Per-Object

both modes is that in per-user, envelope decryption is used, and in per-object, envelope encryption is employed. Our experiments show both envelope encryption and decryption consumed almost the same time in software-based cryptography. Henceforth, our initial expectation that the per-user mode would outperform the per-object mode has proven to be unfounded for software-based cryptography. The per-object mode, however, provides better security than the per-user mode since a new DEK is created for each record in the database. On the other hand, when *KEK in TPM* is selected, the per-object mode outperformed the per-user mode by 84% during the encryption operations. There is no significant performance difference during decryption operations. In the next experiment, we will determine the impact of the location of the KEK on *CephVault*'s performance.

5.4.3 Dependency on Location of the KEK

In this section, we will identify how the location of the KEK can influence the performance of *CephVault*. In this stage, we will analyze the performance for the following KEK locations: *No KEK*, *KEK in Config*, *KEK in Ceph*, *SSS in Ceph*, and *KEK in TPM*. Since in the previous experiment, we concluded that modes of operation did not significantly impact the overall performance of *CephVault*, average time will be calculated for both modes of operation to identify the impact of the location of the KEK. However, for *KEK in TPM*, modes of operation showed significant impact during encryption operations; both modes will be discussed separately. For encryption requests, the total execution time across all *EVPs*, the breakdown is as follows (Figure 5.3): *No KEK*, *KEK in Config*, *KEK in Ceph*, and *SSS in Ceph* accounted for 47.5, 51.7, 202.9, and 498.4 milliseconds for encryption requests, and 38.8, 41.9, 192.5, and 487.2 milliseconds for decryption requests, respectively. For *KEK in TPM*, the per-user and per-object mode registered duration of 696.0 and 377.3 milliseconds for encryption and 686.6 and 685.5 milliseconds for decryption requests, respectively.

Although *No KEK* is the most performant option for the location of the KEK, it is the most insecure one. *No KEK* does not offer any envelope encryption, making the DEK unencrypted in the database. *KEK in Config* shows inferior performance; however, better security than *No KEK*. *KEK in Config* provides envelope encryption for data security and faster KEK retrieval from the configuration file. Like *CephVault*, various other KMSs, such as Barbican [116] and Cyberark Conjure [132], store the encryption key in the configuration file. However, storing the KEK in the configuration file is not entirely secure [143]. *KEK in Ceph* generates a 32-bit encryption key as the KEK and stores it in a secure location in Ceph. Storing the KEK in Ceph is a more secure option than *KEK in Config*; however, it is a less performant one. During envelope encryption or decryption operation, while retrieving the KEK

from the Ceph cluster, a query is made to the database to retrieve the object and pool name that contains the KEK. Then, the request is authenticated with the Ceph Authentication List [163] before accessing the object in Ceph that contains the KEK. For the *SSS in Ceph* experiment, a total of five Shamir’s secret shares are stored in Ceph, with a minimum of three shares required to reconstruct the KEK. Dividing the KEK in five shares provides a better security than having a single KEK, since it requires at least a threshold number of shares to compromise the KEK. However, for *SSS in Ceph*, reconstruction of the KEK during envelope encryption or decryption operation requires more resource-intensive steps than *KEK in Ceph*. First, a query is made to the database that stores the names of the five objects that store the shares. Then, the threshold number of shares is randomly selected from the database before the request is made to the Ceph Authentication List for authentication. Finally, the threshold number of shares are retrieved from Ceph, and the KEK is reconstructed. In *KEK in TPM*, the KEK remains in the TPM, and the DEK is sent to the TPM for the envelope encryption operation. Similarly, the encrypted DEK is sent to the TPM for envelope decryption. *KEK in TPM* is the most secure option offered by *CephVault*. However, it is the slowest option in terms of performance.

In summary, in terms of security, the configurations—*No KEK*, *KEK in Config*, *KEK in Ceph*, *SSS in Ceph*, and *KEK in TPM*—follow an ascending order, with security improving in that sequence. However, in the context of performance, they exhibit an ascending order of execution time, signifying that performance decreases in the same sequence.

5.4.4 Dependency on Envelope Encryption Schemes

We conducted an analysis of how different *EVPs* impacted the overall execution time of *CephVault* during cryptographic operations (Figure 5.3). Our comparative analy-

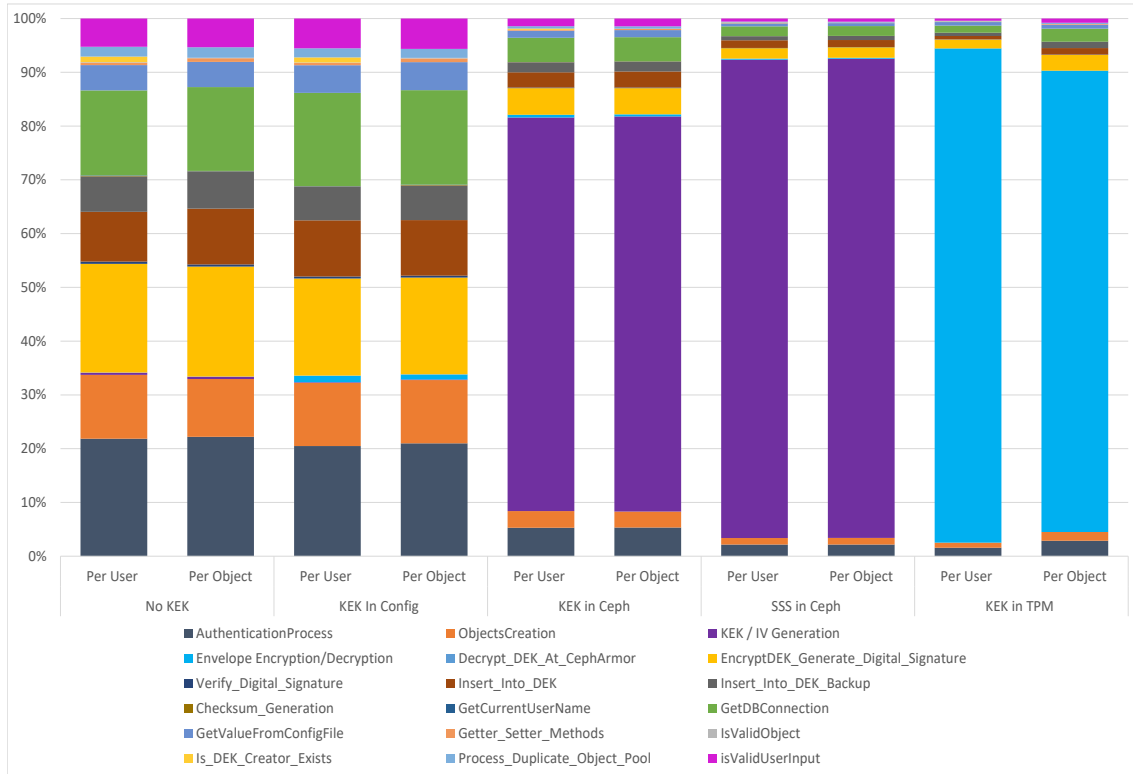
sis focuses on various *EVPs*, including AES-CTR, AES-GCM, and AES-CBC (NI), within the realm of software-based cryptography. In hardware-based cryptography, *CephVault* exclusively employs RSA as the sole *EVP*, and its performance is detailed in Section 5.4.3. This examination aims to assist administrators in making informed decisions regarding the choice of *EVP* when opting for software-based cryptography. In encryption, AES-CBC (NI) outperformed AES-GCM by 2% and AES-CTR by 1%. In decryption, AES-CBC (NI) outperformed AES-GCM by 1% and exhibited comparable performance to AES-CTR. Since AES-CBC (NI) leveraged AES instruction sets tailored for AES encryption and decryption operation on the supported processor, it showed improved performance over the other two modes. AES-GCM uses AAD, additional data provided to the encryption operation to compute the tag, a cryptography checksum that ensures the integrity of the ciphertext and AAD. During envelope encryption operations, AES-GCM consumed additional time for AAD and tag generation. Similarly, during decryption operations, extra time was taken for the tag verification. AES-CTR, without utilizing any hardware-accelerated instruction sets, displayed a 1% decrease in performance during encryption operations. However, it demonstrated performance nearly comparable to that of AES-CBC (NI) during decryption operations.

In summary, when the AES new instruction is enabled—provided the processor supports it—opting for AES-CBC (NI) is advisable as it offers better performance than the other two modes. Next, we will delve into *CephVault*'s internal events to identify the factors that notably impact the overall performance.

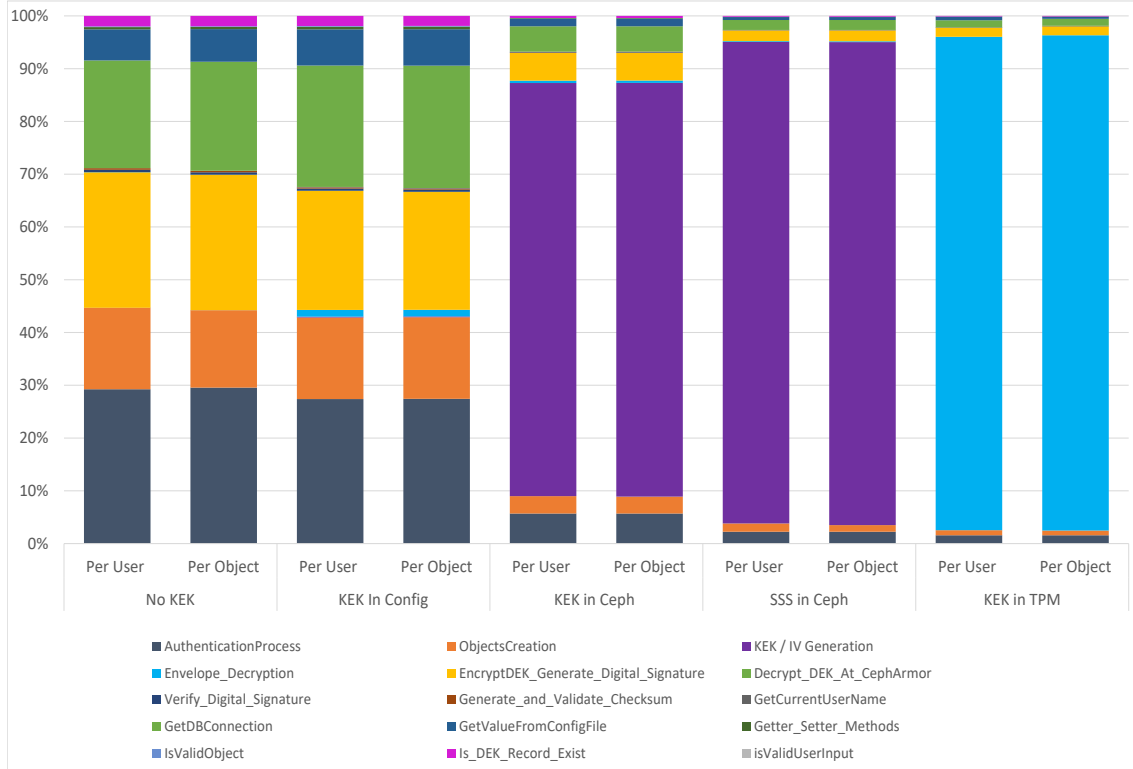
5.4.5 Dependency on CephVault's Internal Events

When an encryption or decryption request is received by *CephVault*, it processes the request through a sequence of internal events. We captured the time for each individ-

ual event to identify any resource-intensive step that could be optimized or safely disabled based on the organization’s requirement. Figure 5.4 depicts the time captured for *CephVault*’s events while processing encryption-decryption requests in per-user and per-object modes. In *CephVault*, the generation of a checksum on the DEK and IV is optional. Similarly, administrators can enable or disable the encryption of the DEK with a pre-shared symmetric key and the generation of a digital signature on the encrypted DEK. For our evaluation, we enabled both options to calculate the respective times. An event *Checksum_Generation* represents the time taken for checksum calculation of the DEK and IV. Events *EncryptDEK_Generate_Digital_Signature*, *Decrypt_DEK_At_CephArmor*, and *Verify_Digital_Signature* describe the time taken for performing the digital signature calculation on the encrypted DEK (Figure 5.4a). Similarly, *Generate and Validate Checksum* represents the time taken for checksum calculation on the DEK and IV, retrieval of the old checksum from the database, and comparison of both results (Figure 5.4b). The time consumption analysis reveals that, on average, the processing of *Checksum_Generation* and *Generate and Validate Checksum* took merely 0.05 and 0.14 milliseconds, respectively. These durations are deemed insignificant when considering the overall execution time of *CephVault*. However, in contrast, the events associated with digital signature generation and verification on the encrypted DEK exhibited a more substantial impact, consuming an average of 10.2 milliseconds. This digital signature-related processing constituted 22%, 20%, 5%, 2%, and 2% of the total execution time during encryption, and 26%, 24%, 5%, 2%, and 1% of the total execution time during decryption under the scenarios of *No EVP*, *KEK in Config*, *KEK in Ceph*, *SSS in Ceph*, and *KEK in TPM*, respectively. With this analysis, administrators can turn off digital signatures to gain performance. For example, suppose *No EVP* is chosen for the encryption operation; disabling digital signatures can increase performance by 22%. However, only 2% performance is gained when *KEK in TPM* is selected. If the organization prioritizes



(a) Encryption Operations



(b) Decryption Operations

Figure 5.4: CephVault Events During Cryptography Operations

security over performance, enabling digital signatures provides the integrity of the DEK during key distribution from *CephVault* to *CephArmor*.

In addition to assisting administrators in determining the enablement or disablement of various configurable options, internal events within *CephVault* also provide valuable insights into the optimal location for the KEK. Administrators can make informed decisions based on the percentage of time consumed by each location, thereby enhancing the efficiency of *CephVault* operations. In Figure 5.4, *No KEK* did not employ an envelope encryption scheme, so it did not consume any time for the generation of the KEK. In *KEK in Config*, the generation of the KEK took 0.13 milliseconds, which was insignificant with respect to the total execution time. However, a significant time consumption was observed for *KEK in Ceph* and *SSS in Ceph*. The generation of the KEK took 148.7 and 444.0 milliseconds, which were 73%, and 89% of the total execution time during encryption, and 77%, and 91% of the total execution time during decryption, when *KEK in Ceph*, and *SSS in Ceph* were selected, respectively. For *KEK in TPM*, since the generation of the KEK is internal to TPM and envelope encryption and decryption operations are performed inside the TPM, we measured those time in the *Envelope Encryption/Decryption* event. For *KEK in TPM*, while serving encryption request, the envelope encryption event consumed 639.0 (per-user) and 323.7 (per-object) milliseconds, which were 92% and 86% of the total execution time, respectively (Figure 5.4a). While serving the decryption request, the envelope decryption took 639.2 milliseconds (per-user) and 640.5 milliseconds (per-object), which were 93% of the total execution time, for both (Figure 5.4b). This analysis will help administrators select the appropriate option for the location of the KEK based on the organization's security and performance requirements. For example, if security is not a priority, *No KEK* will be the most performant option. On the contrary, storing the KEK in TPM will be the most

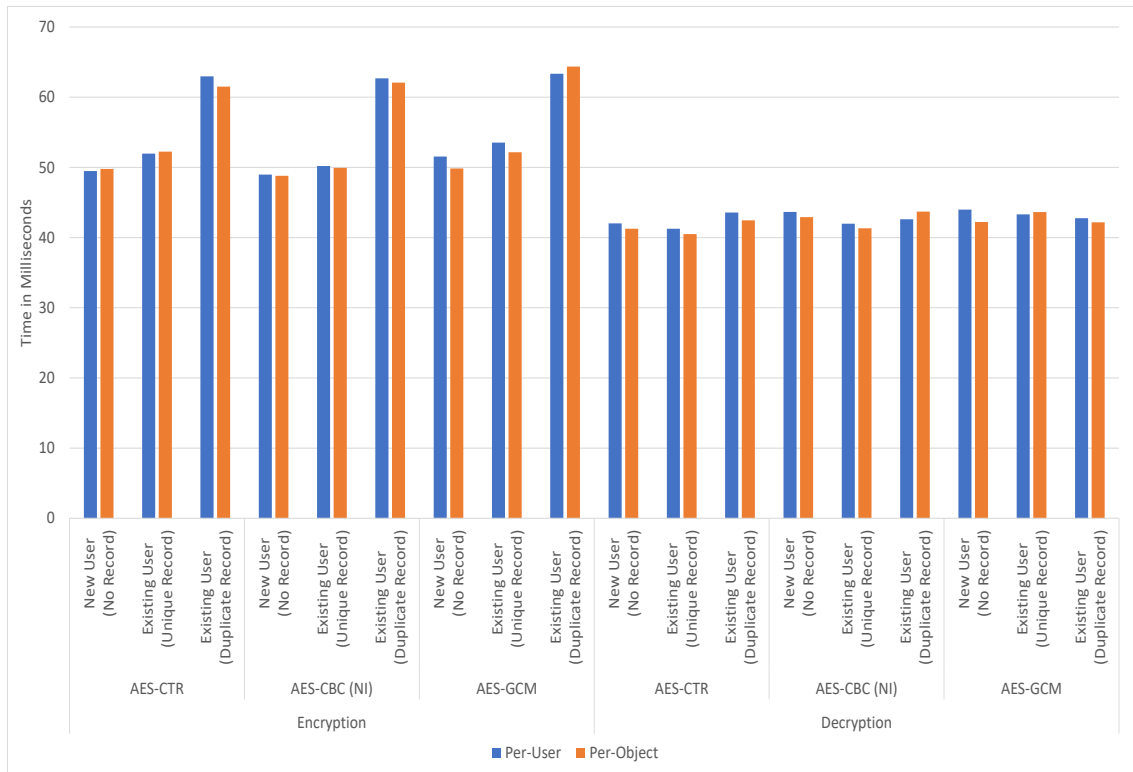
secure option; however, in terms of performance, it will be the least efficient option available to the organization.

In summary, disabling the checksum option will not improve the total execution time significantly; rather, it will compromise the integrity of the DEK and IV. However, disabling the digital signature option will improve performance significantly based on the choice of cryptography operation (encryption or decryption) and the location of the KEK. Although disabling the digital signature will improve performance, it will undermine the integrity of the DEK during key distribution. Similarly, the location of the KEK notably contributes to the overall performance and security. *No KEK* provides improved performance with compromised security. On the contrary, *KEK in TPM* offers improved security with inferior performance.

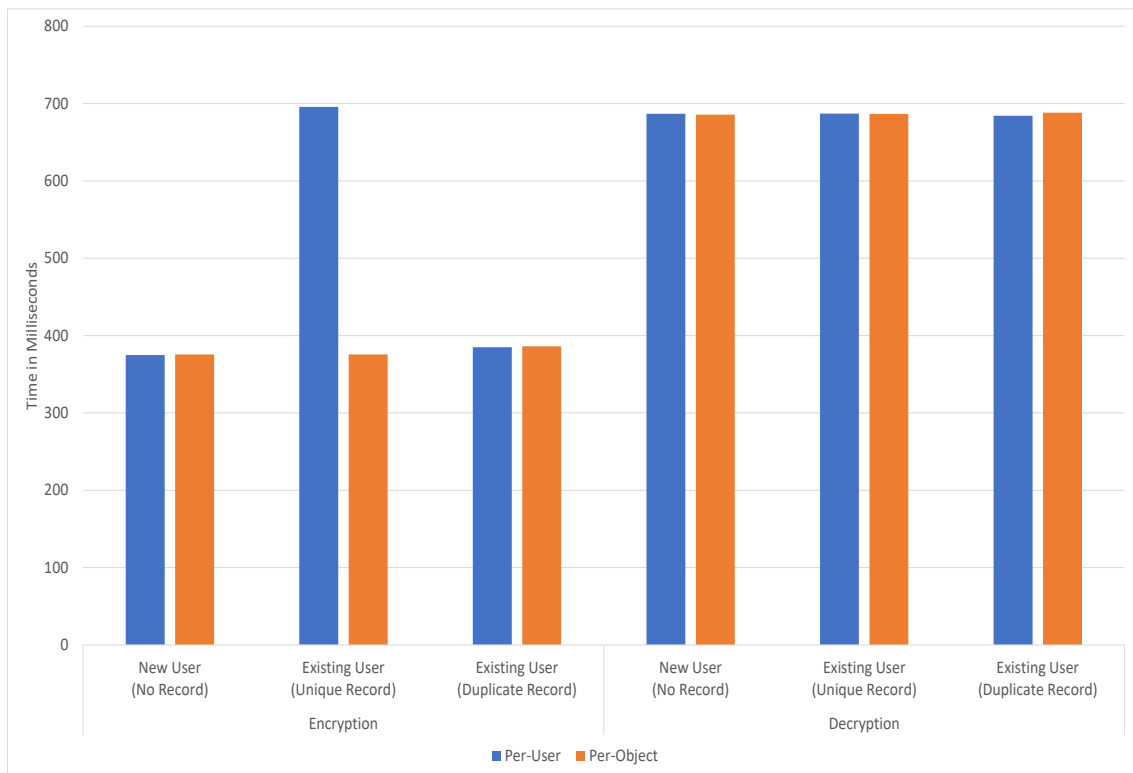
5.4.6 Dependency on Use Cases

CephVault handles various use cases during cryptography operations. Experiments in this section focus on finding out how execution time in *CephVault* is dependent on (a) when there is no record existing for the user in the database, (b) when the user already exists in the database and the request (combination of object, pool, namespace) is unique, or (c) when the user already exists in the database, however, the request (combination of object, pool, namespace) is a duplicate one, meaning the same record already exists in the database. For the evaluation, we captured *CephVault*'s execution time while processing cryptography operations on 1 MB of data with various modes of operation and *EVPs*. Since we had already established that data size did not impact the overall execution time, we did not consider data sizes other than 1 MB. For software-based cryptography, we opted for *KEK in Config* since the execution time is faster than other locations of the KEK. However, we observed similar results for other locations of the KEK (results are not included

to reduce verbosity). For hardware-based cryptography, we selected *KEK in TPM*. Figure 5.5a depicts encryption and decryption requests in software-based cryptography. During encryption, the new user performed better than the existing user with unique records, and the existing user with unique records showed better performance than the existing user with a duplicate record. For a new user, no existing DEK details were retrieved from the database; however, for the existing user without any duplicate records, the database retrieval contributed some extra time. When a duplicate record was found, the record was deleted from the primary and the backup tables. Moreover, the deleted records were stored in the deleted table temporarily, contributing more time than the other two use cases. However, during decryption, the use cases did not impact the time since the requests were processed via the same steps, providing almost the same result. Figure 5.5b illustrates encryption and decryption requests in hardware-based cryptography. During encryption, the new user consumed about the same time in per-user and per-object mode, respectively. For a new user, envelope encryption was performed on the DEK for both modes of operation. In contrast, for the existing user (unique record), both modes of operation performed differently. In the per-user mode, the encrypted DEK was retrieved from the database, and envelope decryption was performed to get the plaintext DEK. However, in per-object mode, the new DEK was encrypted with an envelope encryption scheme. Since, in TPM, envelope decryption consumed 639.0 milliseconds (per-user), and envelope encryption consumed 323.7 milliseconds (per-object), per-object mode outperformed per-user mode by 97.39%. For the existing user (duplicate record), once the duplicate record was removed from the database, the request was treated as if it was a new user (no record). So, for both modes of operation, envelope encryption was performed. In decryption, for both modes of operation, envelope decryption was performed, showing a similar trend. The Y-axes of Figure 5.5a and Figure 5.5b are different.



(a) Cryptography Operations (KEK in Config)



(b) Cryptography Operations (KEK in TPM)

Figure 5.5: Dependency on Use Cases

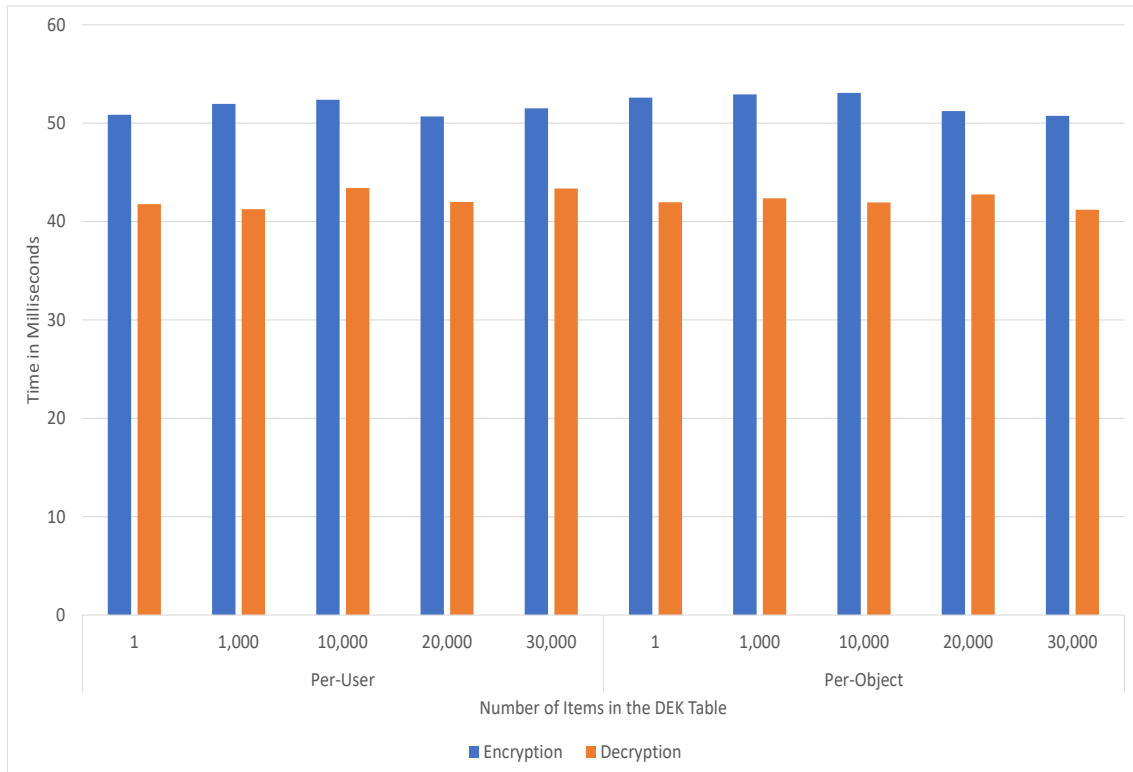
In summary, execution time during encryption operations was impacted by the presence of a duplicate record. However, it had an insignificant influence on the execution time during decryption operations.

5.4.7 Dependency on Number of Records in the Database

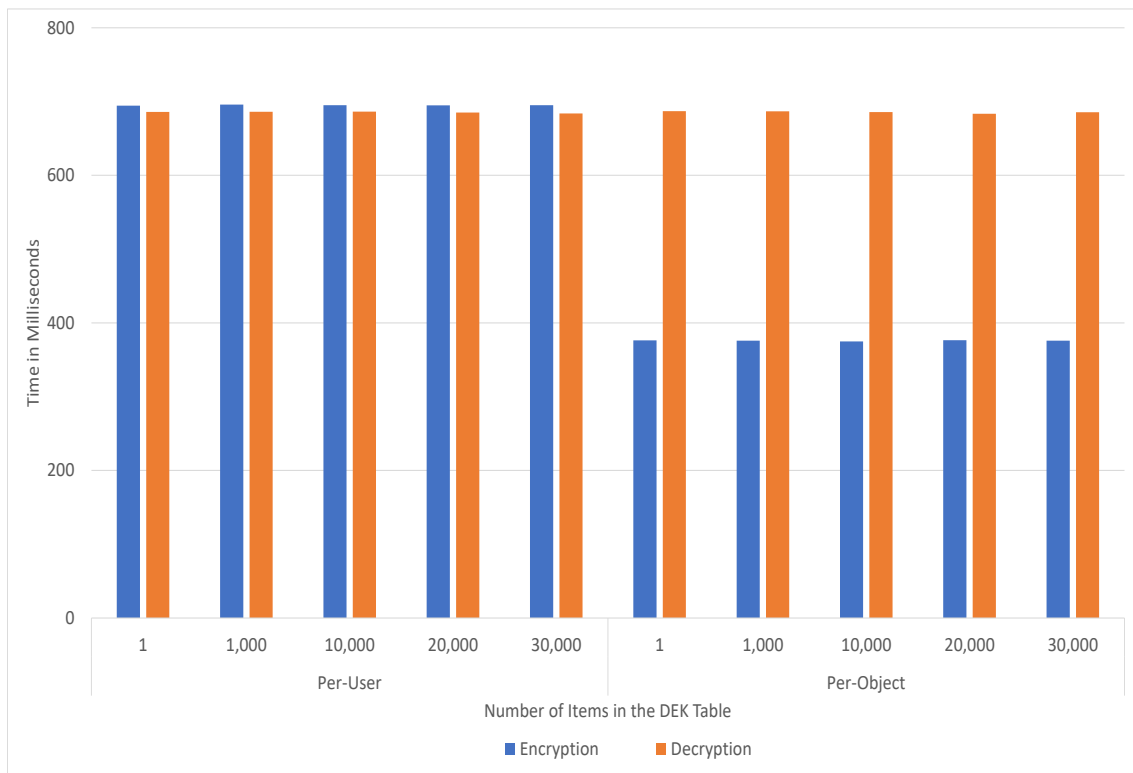
This experiment aimed to determine whether the number of records in the database can impact the total execution time during cryptography operations. We tested with up to 30,000 records in the DEK table. In software-based cryptography (Figure 5.6a), we captured the performance during encryption-decryption operations, selecting *KEK in Config* (the location of the KEK) and AES-CTR, AES-CBC (NI), and AES-GCM (*EVP*). Similarly, in hardware-based cryptography (Figure 5.6b), we captured the performance of *KEK in TPM* (location of the KEK) and RSA (*EVP*). We found that execution time did not vary significantly with varying numbers of records in the database. We observed similar results for other KEK locations and *EVP*s. We concluded that the number of records in the database did not significantly impact the overall execution time of *CephVault* (up to 30,000 records). The Y-axes of Figure 5.6a and Figure 5.6b are different.

5.5 Summary

In this chapter, we evaluated *CephVault*'s performance with respect to various factors, such as dependency on data sizes, locations of the KEK, various *EVP*s, and use cases. Next, we will discuss storage requirements for *CephVault*.



(a) Cryptography Operations (KEK in Config)



(b) Cryptography Operations (KEK in TPM)

Figure 5.6: Dependency on Number of Records

Chapter 6

Storage Analysis

One aspect of our research is to identify the storage requirements when we employ *CephVault* to store KMS data, such as the encrypted DEK, IV, etc. So, we used different modes of operation or envelope encryption schemes. For each unique combination of object, pool, and namespace, a new row was created in the primary table in the database, and a backup was taken for the same. So, the storage requirements for an encryption request was calculated by the bytes occupied in the primary and the backup tables, as depicted in Figure 6.1.

The primary table comprises current records required for cryptography operations, whereas the backup table supports versions up to two levels comprising current records and previous records. The backup table requires more storage than the primary table to support additional versions. However, the backup table is designed to reduce the storage requirements by storing common metadata, such as object, pool, and namespace, only once for both versions. Storage requirements changed based on the types of *EVPs*. A total of 665 bytes/row was required to store data related to a single encryption request when we opted for the *No EVP*, where the primary and the backup tables occupied 273 bytes/row and 392 bytes/row, respectively. Similarly,

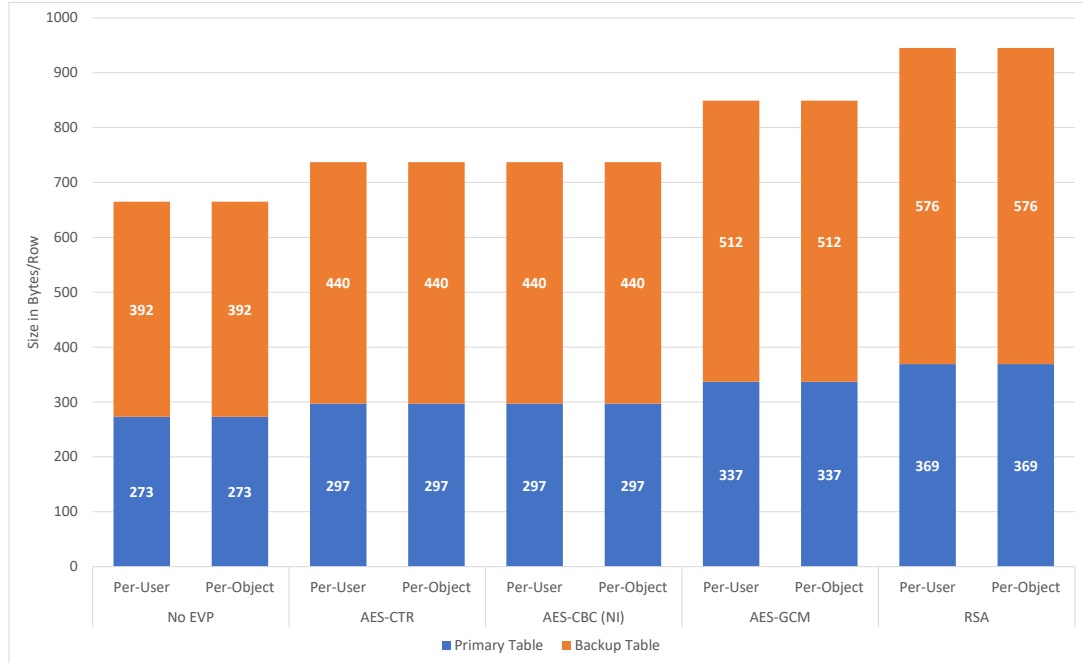


Figure 6.1: Storage Requirements for CephVault

AES-CTR or AES CBC (NI), AES-GCM, and RSA occupied 737 bytes/row, 849 bytes/row, and 945 bytes/row, respectively. The storage requirements did not differ when per-user or per-object mode was chosen. For the current storage analysis, we assumed users would prefer a maximum of 25 characters for an object, pool, or namespace. However, the storage requirements might vary for a larger or smaller name for an object, pool, or namespace.

In summary, the storage was least occupied by *No EVP* and the most by RSA. The additional storage requirements with respect to *No EVP* were 11%, 11%, 28%, and 42% for AES-CTR, AES-CBC (NI), AES-GCM, and RSA, respectively. Lastly, the mode of operation (per user or per object) or the location of the KEK did not contribute to the storage requirements. In the next chapter, we will analyze the security of *CephVault*.

Chapter 7

Security Analysis

We evaluated *CephVault* against various information security principles, secure design principles, adherence to FIPS 140-2 (Levels 1, 2, 3, and 4), and countermeasures to various known attacks.

7.1 Information Security Principles

The primary goal of information security principles is to provide guidelines for organizations to follow and protect sensitive data (Section 2.2). *CephVault* adheres to various information security principles, discussed in Table 7.1 and Table 7.2.

7.2 Security Design Principles

It is nearly impossible to develop techniques that can ensure the complete security of the developed system or prevent all types of attacks [44]. However, widely accepted design principles (Section 2.3) can mitigate various security vulnerabilities in the system. *CephVault* is developed with adherence to the security design principles depicted in Table 7.3, and Table 7.4.

Table 7.1: Adherence to Information Security Principles

Security Principle	CephVault Features Adhered to Information Security Principles
Confidentiality	Envelope encryption is employed to encrypt the DEK, proving confidentiality while the DEK is stored in the database. The DEK is also encrypted during key distribution. However, employing <i>No EVP</i> compromises confidentiality. Also, since the selection of encrypting the DEK during key distribution is optional, not opting for encrypting the DEK undermines security.
Integrity	<i>CephVault</i> employs SHA-256 to provide integrity of the DEK and DEK_IV. In encryption, a checksum is calculated on the DEK and DEK_IV, and stored in the database. In decryption, the checksum is retrieved from the database and compared with the newly generated checksum on the DEK and DEK_IV, ensuring the DEK and IV are not altered in the database. Also, during key distribution, an RSA digital signature is employed to provide integrity to the DEK. However, calculating the checksum on the DEK and DEK_IV and generating digital signatures on the encrypted DEK are optional, making them vulnerable to undermining integrity.
Availability	Backups are taken for the DEK, KEK, and checksum of the <i>CephVault's</i> binaries. These are stored in different databases, ensuring recovery of data in case data is corrupted in the primary database. However, availability can be compromised if backups are deleted by unauthorized persons.
Authentication	Users are authenticated for each encryption and decryption request. In encryption, only authenticated users are able to receive the DEK for cryptography operations. In decryption, only the user who encrypts the data can decrypt the corresponding data. However, malicious users who compromise user credentials can undermine authentication.
Accountability	<i>CephVault</i> is reinforced by audit logs and the real-time notification system. Each method is secured with exception handling, and administrators receive notification for any exception or unusual behavior.

Table 7.2: Adherence to Information Security Principles (Continued)

Security Principle	CephVault Features Adhered to Information Security Principles
Non-Repudiation	During key distribution, the DEK is encrypted with a separate key, which is pre-shared between the requester and <i>CephVault</i> . An RSA digital signature is employed on the encrypted DEK, which provides non-repudiation. The user, who leverages <i>CephArmor</i> API to encrypt the data, can be certain that the encrypted DEK is distributed by the <i>CephVault</i> , since <i>CephVault</i> signs the encrypted DEK and the <i>CephArmor</i> verifies it prior to use it for cryptography operations. However, disabling generating the digital signature on the encrypted DEK can compromise non-repudiation.

7.3 Defense Against Various Attacks

Various security vulnerabilities and attacks are described in Section 2.8, and *CephVault* is designed to mitigate those attacks. Countermeasures for these vulnerabilities and attacks are discussed below. In Table 7.5 and Table 7.6, some countermeasures are discussed pertaining to those vulnerabilities and attacks.

7.4 Comparative Analysis of Popular KMSs

We compared some of the existing KMS solutions described in Section 3.2 and 3.3 in a table format, where column names, represented as unique upper-case letters (A–N), depict various attributes that are the basis for the evaluation. Table 7.7 describes the column names and their respective representations. The column value **Y** denotes **Yes** and **N** denotes **No**. Table 7.8 describes various features of *CephVault*, which was evaluated against other KMSs. For this evaluation, we considered the best possible features for both *CephVault* and other KMSs.

Table 7.3: Adherence to Fundamental Security Design Principles

Design Principle	CephVault Features Adhered To Security Design Principles
The economy of mechanism	The architecture of <i>CephVault</i> is simple. It is designed with loosely coupled modules, with reusable and smaller methods. Each functionality can be individually tested and measured.
Open design	<i>CephVault</i> follows a well-established design pattern (mediator pattern) and security best practice while designing the KMS. The design of <i>CephVault</i> is made available in this thesis so that security experts can identify any potential vulnerability. Also, comprehensive future research directions are provided in Sections 8.1 and 8.2, identified by our research team, to further enhance the application.
Separation of privilege	Each module in <i>CephVault</i> is responsible for performing dedicated tasks. Methods belonging to one module do not have the privilege to access resources from other modules unless it is necessary.
Least privilege	Users in <i>CephVault</i> have the least privilege to perform cryptography operations; however, they are restricted to perform other tasks that need higher privilege.
Least common mechanism	<i>CephVault</i> 's modular design adheres to the least common mechanism. <i>CephVault</i> 's public module only exposes two methods for <i>CephArmor</i> to communicate with the <i>CephVault</i> . <i>CephVault</i> 's global module comprises the least number of methods to be used by both <i>CephVault</i> and <i>CephArmor</i> . Access specifiers control the number of shared functions between modules.
Psychological acceptability	Users can perform cryptography operations without knowing any underlying details of <i>CephVault</i> . It facilitates various command line options to interact with the application to perform operations, such as key rotation, key backup, key restore, and key distribution, without users exposing the technical details. If there is any issue, custom messages are provided to the users to help them understand the next steps; at the same time, admins are notified so that the issue can be addressed quickly.

Table 7.4: Adherence to Fundamental Security Design Principles (Continued)

Design Principle	CephVault Features Adhered To Security Design Principles
Isolation	<i>CephVault</i> employs the TPM, which provides a physically isolated cryptography environment, making it harder to extract the KEK. The mediator design pattern provides code isolation. Various databases are created to provide additional isolation, where respective tables are stored in databases instead of having only one database with all tables. However, since employing the TPM is optional in <i>CephVault</i> , not opting for the TPM can undermine isolation.
Encapsulation	<i>CephVault</i> leverages the Object-Oriented Programming (OOP) concept, where a clear separation of data and methods is implemented. Each module is developed in such a way that they are not tightly coupled with other modules. Most of the methods are kept private in each module, and only a few methods remain public to access data.
Modularity	<i>CephVault</i> is developed with the modular approach, where a total of 27 modules are developed, each with different functionalities, such as key generation, key usage, and key distribution. Also, there are modules, such as <i>CephVault Global Module</i> , which contain various reusable codes used by other modules.
Layering	Despite Ceph users being authenticated when they log in to Ceph servers, <i>CephVault</i> again authenticates for the cryptography operations, which provides an additional layer of security. No DEK is kept in the database or anywhere in the plaintext (except <i>No KEK</i>); the DEK is protected by the KEK using <i>EVP</i> (except <i>No KEK</i>). Smart pointers are employed to provide memory safety, such as avoidance of dangling pointers, interoperability, and exception safety, which provides additional code safety. Checksum of the KEK and <i>CephVault</i> binaries are taken and periodically checked to ensure the integrity of the KEK and the deployed binaries.
Least astonishment	The <i>CephVault</i> command line interface is designed to work as expected. For example, suppose a security goal is to rotate the DEK. In that case, it is mapped to a command that the user can invoke, and the command only performs key rotation without surprising the user by performing other tasks.

Table 7.5: Countermeasure Against Various Attacks

Attacks	Countermeasures
Buffer Overflow Attacks	<i>CephVault</i> validates the input before using it; for example, the user-provided key size is validated to identify whether it is within the supported encryption key sizes or whether the buffer size is the same for the accepted data.
SQL Injection Attacks	Extensive input validation is in place to identify any malformed data that are supposed to be stored in the database; for example, any string provided by the user or accepted from the configuration file is validated with REGEX patterns for potential attacks. Each function has a return type, and based on the proper output, the next action is taken. All SQL queries leverage parameterized queries, secured with exception handling to reduce the attack surface.
Directory Traversal Attacks	Code is developed to mitigate directory traversal attacks while retrieving a pre-shared symmetric key from the user's home directory. Also, <i>CephVault</i> does not store any secrets in any environmental variable, which is vulnerable to attacks.
Side Channel Attacks	AES-NI mode [68] of envelope encryption is employed by <i>CephVault</i> to countermeasure side-channel attacks.
Physical Attacks	<i>CephVault</i> employs a TPM for storing the KEK and performing cryptography operations. The TPM provides a tamper-resistant hardware module that resists physical attacks. However, not opting for the TPM will not provide tamper resistance against physical attacks.
Insider Attacks	Only the user who encrypts the data can decrypt the corresponding ciphertext; unauthorized persons cannot access other's data using <i>CephArmor</i> 's API. Generation of the KEK using Shamir's secret shares ensures a compromised party alone cannot reconstruct the original secret. Intentional or unintentional modification of the KEK can be captured using a scheduled job that compares the old KEK checksum with the new one. However, not opting for Shamir's secret shares as the preferred option for the KEK can undermine protection against insider attacks.

Table 7.6: Countermeasure Against Various Attacks (Continued)

Attacks	Countermeasures
Man-in-the-middle Attacks	<i>CephVault</i> 's digital signature ensures that the DEK and DEK_IV are sent from the authenticated source; if the digital signature fails, users are not provided with the DEK and DEK_IV. However, not opting for the digital signature on the encrypted DEK can be vulnerable to man-in-the-middle attacks.
Brute Force Attacks	During use authentication, users can provide a maximum of three incorrect passwords before the application exits, which helps to mitigate brute-force attacks. Also, FIPS 140-2 validated CSPRNG and TRNG are used, which are difficult to compromise using brute force attacks.
Rainbow Attacks	<i>CephVault</i> does not employ password-based key derivation for key generation; rather, a FIPS 140-2 validated random number generator is used for the same, which eliminates the need to store password hashes. So, <i>CephVault</i> does not have a vulnerability for rainbow attacks.
Chosen Plaintext Attacks	<i>CephVault</i> 's per-object mode ensures that the encrypted DEK is different than the existing ones in the database. For each request, a new DEK and DEK_IV are generated, and a new KEK_IV is used for envelope encryption. However, not opting for per-object can lead to the chosen plaintext attacks.
Padding Oracle Attacks	<i>CephVault</i> employs envelope encryption using AES-CTR and AES-GCM mode, which provide countermeasures to padding oracle attacks. However, not opting for AES-GCM or AES-CTR can lead to padding oracle attacks.

Table 7.7: The Column Names and their Respective Representations

Column	Description
A	Can be used for Ceph
B	Implemented Shamir's secret share as the KEK
C	Implemented envelope encryption
D	Implemented AES-NI (hardware accelerated AES intrinsic code for envelope encryption)
E	Implemented 12 phases of KMS life cycle
F	Implemented a secure design (clear separation of data and methods, private-public methods, defensive coding)
G	Implemented digital signature (or certification) for key distribution
H	Implemented support for a TPM or HSM
I	Implemented checksum (for the integrity of secrets)
J	Implemented user-specific encryption key
K	Implemented object-specific encryption key
L	Implemented audit logs
M	Implemented real-time monitoring
N	Reference

Table 7.8: Comparative Analysis of KMSs with CephVault

KMS	A	B	C	D	E	F	G	H	I	J	K	L	M	N
CephVault	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	NA
Barbican	Y	N	Y	N	N	N	N	Y	N	Y	N	Y	Y	[116]
Hashicorp Vault	Y	Y	Y	N	N	N	Y	Y	N	Y	N	Y	Y	[122]
Flix-Keeto	N	N	N	N	N	N	Y	N	N	Y	N	Y	N	[129]
Ansible Vault	N	N	N	N	N	N	N	N	N	Y	N	N	N	[124]
Cloudflare Red October	N	N	Y	N	N	N	N	N	N	Y	N	N	N	[125]
Codahale Sneaker	N	N	Y	N	N	N	N	N	N	N	N	N	N	[126]
XOR Data Exchange Crypt	N	N	N	N	N	N	N	N	N	Y	N	N	N	[128]
Square Keywhiz	N	N	Y	N	N	Y	Y	Y	N	Y	N	Y	N	[127]
Olejade Trousseau	N	N	N	N	N	N	N	N	N	Y	N	N	N	[131]
Cyberark Conjur	N	N	N	N	N	N	N	N	N	Y	N	Y	N	[132]

7.5 Evaluation of CephVault’s Cryptography

Module Adhering FIPS 140-2

In this section, we evaluated *CephVault*’s cryptography module to identify its adherence to FIPS 140-2 Level 2. Table 2.3 describes the summary of the security standards for a cryptography module. Table 7.9 and Table 7.10 describe *CephVault*’s adherence to the standards.

7.6 Evaluation of CephVault as a KMS Adhering FIPS 140-2

Section 2.11.2 describes the security requirements for a KMS adhering to FIPS standards. We evaluated *CephVault* against the standards as below.

- **Documentation Requirements:** This thesis provides information pertaining to the keys used in *CephVault* and details regarding software-based and hardware-based cryptography modules that adhere to the FIPS 140-2.
- **Random Number Generators (RNGs):** *CephVault* used FIPS 140-2 approved RNG *rand_bytes()* for deterministic RNG and TPM 2.0 for non-deterministic RNG.
- **Key Generation:** *CephVault* used FIPS 140-2 validated OpenSSL for key generation. Also, a TPM is used to generate keys. *Key Storage:* In software-based cryptography, the KEK is stored in a secure location in the Ceph or configuration file, whereas in hardware-based cryptography, the KEK is stored in the TPM. In both approaches, the DEK is always kept encrypted in the database (except *No KEK*).
- **Labeling of Cryptographic Information:** Each unique DEK and KEK are labeled with unique UUIDs.
- **Electronic Key Distribution:** Encrypting the DEK with a key and creating a

Table 7.9: Evaluation of Cryptography Module in CephVault for FIPS 140-2

Attributes	CephVault Features
Roles, Services, and Authentication	<i>CephVault</i> supports identity-based authentication. Unauthenticated users are not permitted to perform cryptography operations.
Cryptographic Module Specification	We employed FIPS 140-2 approved algorithms, and we provided in detail the cryptography module (hardware and software components) used in the <i>CephVault</i> . In this thesis, we described all logical interfaces that are responsible for data input and output. We documented the modes of operation, languages used, diagrams depicting all modules, and key storage. A detailed security analysis is also provided.
Cryptographic Module Ports and Interfaces	User access to the cryptographic modules is restricted by authentication. Also, data flow to the cryptographic modules is restricted by input validation, and modular design, where the public interface is exposed to the outside world and other modules are kept private to <i>CephVault</i> . The TPM is integrated with the servers and powered by the predefined power port.
Physical Security	We employ the TPM, which is a tamper-resistant cryptography module.
Operational Environment	Ubuntu OS provides an operating environment for <i>CephVault</i> , where all system processes are controlled. The security of the operating environment is out of the scope of this thesis. Organizations who would employ <i>CephVault</i> as a KMS for Ceph could secure the operating environment with firewalls, disk encryption, anti-malware detection software, ransomware detectors, Intrusion Detection Systems (IDSs), Intrusion Prevention Systems (IPSs), and so on.
Cryptographic Key Management	This thesis specified all cryptographic keys, such as DEKs and KEKs, and other Critical Security Parameters (CSPs), such as IVs, shared secrets in detail. Also, we developed and described the KMS life cycle employed in <i>CephVault</i> . A comprehensive description is provided in Section 7.6.
Mitigation of Other Attack	We specified common attacks and their countermeasures employed in <i>CephVault</i> .
Self-Tests	We performed conditional tests as per FIPS 140-2 guidelines: we tested a plaintext DEK and the ciphertext generated after encrypting the plaintext DEK, and both were different. We verified digital signatures and terminated the process in case the digital signature failed.

Table 7.10: Evaluation of Cryptography Module in CephVault for FIPS 140-2 (Continued)

Attributes	CephVault Features
Design Assurance	We followed security best practices to design the cryptography module. We tested the module thoroughly and documented the results in detail. The TPM is installed in the 45Drives Storinator by the authorized personnel as per the <i>Delivery and Operation</i> guideline of FIPS 140-2.

digital signature on the encrypted DEK safeguards the DEK during key distribution.

- **Key Usage:** *CephVault* employs different keys for various purposes; a DEK is used to encrypt the data, a KEK is used to encrypt the DEK, a pre-shared key is used for encrypting the DEK during key distribution, and a key pair is used for digital signature during key distribution.

7.7 Security Analysis of Location of Keys

Key storage is an important aspect of a KMS's overall security. A compromised location can reveal the keys, hence the data. In this section, we will discuss the security implications of storing keys in various locations.

7.7.1 User's Home Directory

One of the most challenging tasks for a KMS is to store secrets securely. *CephVault* employs the user's home directory to store a pre-shared symmetric key and a private-public key pair used for digital signature. These keys are different than the ones that are used for data encryption (performed by a DEK) and envelope encryption (performed by a KEK). The pre-shared symmetric key is used for encrypting the DEK during key distribution, and the key pair of digital signature is used for signing the encrypted DEK and verifying the signature during key distribution. The reasons for leveraging the user's home directory to store these keys rather than storing them in

the SQLite table are manifold.

- **Key Private to the User:** These keys are generated for each authenticated *CephVault* user. Opting to utilize something other than the home directory necessitates the implementation of an additional KMS, adding a layer of complexity, since we have a KMS for serving data encryption decryption requests.
- **Permissions:** User's home directory, which is commonly at the location `~/home-username,` is secured with the permission `700 (rwx---`), signifying only the user, who is the owner of the directory, can access the directory.
- **Similar to .ssh Directory:** We have created a `.cephvault` directory inside the user's home directory, which has the same permissions as the `.ssh` directory. In the `.ssh` directory, key pairs are generally stored for a user, and we adopted the same methods to safeguard these secrets.
- **Faster Access:** Since keys are kept in plaintext, no envelope encryption is employed; when these keys are required to be used, an unwrapping process is not required, making the key retrieval faster.
- **Flexibility in Key Rotation:** Since users maintain or own their respective home directories, they can change the keys multiple times at their convenience without affecting the performance of the *CephVault*, making the KMS secure and performant.
- **Data Protection:** Even if the user's home directory is compromised, the previously encrypted data cannot be decrypted by an attacker since these keys are not used for data encryption. However, newly encrypted data, which are encrypted after the directory is compromised, are vulnerable to data leaks. Also, with a compromised home directory, *CephArmor* will be unable to authenticate the source of secrets.

7.7.2 Configuration File

In software-based cryptography, a configuration file is used to store the KEK. By default, the configuration file is stored in the `/etc/ceph` directory, which is the same

```
Shamir Secret Split:
Please note down these shares securely. You will need these shares for the key recovery.:
Share 1: 7cbd372f0b683cc36bb4dde2a9848e3c7e02d87422d90372f8ee4e272325eb1230
Share 2: 5df31ccf8196275d90ced90c546bee5404a119c40d3999eb302adc097360f71e1b
Share 3: ea0a65ee98664e2aaa01301507e892cc3a05894f1a6550d459af161b78d83a5e34
Share 4: ee7e981653b3e96d1502bc2c47cd6f23ebbe1eb4e213075a492930c13640e23b3d
Share 5: 226f10a435c4080669890336ff9f7e188a873d768f979683520776d1d5e9ca4691
*** DEVELOPER MODE: setting PATH, PYTHONPATH and LD_LIBRARY_PATH ***
pool '26d216b76b472e6ffa4e6c443ef896be427b780b192fe36a0cecc9062cf67287' created
*** DEVELOPER MODE: setting PATH, PYTHONPATH and LD_LIBRARY_PATH ***
added key for client.cephvault.shamir
Shamir Secret Split is securely stored.
```

Figure 7.1: Shamir’s Secret Shares

directory where Ceph’s other configuration files and authentication lists are stored. Only root users can access the KEK in the configuration file. However, root users can change the location of the file.

7.7.3 Ceph

In software-based cryptography, *CephVault* leverages Ceph to store the KEK, which can be a 32-bit random number or Shamir secret shares based on the option selected by the administrator. In this section, the security implications of storing Shamir’s secret shares in Ceph are discussed. When the root user creates Shamir’s secret shares (depicted in Figure 7.1), a new pool and objects (equal to the number of secret shares) are created in Ceph. *CephVault* creates the pool name as a 32-bit hexadecimal number, which is randomly chosen. Similarly, object names are created as 32-bit random numbers. Ceph’s default behavior allows overwriting an object inside a pool if the object name matches the existing one in that pool. Randomizing the names of the pool and objects minimizes the accidental or intentional overwriting of objects. Once the pool and objects are created, the names of the pool and objects are stored in the database, and a backup is taken at the same time. In the database, only the names of the pool and objects are stored, not the actual data containing the actual Shamir’s secret shares. Actual secret shares are stored inside objects in Ceph. The pool that contains objects is encapsulated with a unique namespace that

```

client.cephvault.masterkey
  key: AQC8yY11NLR5LRAAJ9o6S2RVE1BUTE1xHhL+JQ==
  caps: [mon] allow r
  caps: [osd] allow rwx pool=0078d8c092cfb52a3b003a75b0b78e0eb2e07a49bff102
7e58ddc2157503d589 namespace=cephvault.masterkey
client.cephvault.shamir
  key: AQDLhJ11hb6mORAAm1bRHWctau4P2G/EVghz7w==
  caps: [mon] allow r
  caps: [osd] allow rwx pool=305cc74ebfd24b2287d66a228376e34186da3a7f3caac5
42c3aba37be620ecd6 namespace=cephvault.shamir.shares

```

Figure 7.2: Security of Pool Contains Shamir Secret Shares

provides a higher level of security. To overwrite an object, the user needs to provide the same namespace, pool, and object name. An entry is created in the Ceph authentication list to grant access to the Ceph storage cluster. Then permissions and capabilities are added to the namespace to prevent unauthorized access to the namespace, pool, and objects, as depicted in Figure 7.2. Once the pool and objects are recorded in the tables, they can not be overwritten, so accidentally running the command (`cephvault -createkek`) will not result in the creation of a new KEK. This helps to prevent data loss, as DEKs, which were previously encrypted with the existing KEK, would still be valid for data decryption. In the enterprise settings, the disk that contains the objects is encrypted, so compromising the disk will not reveal the contents inside the objects. An additional security layer is provided by Ceph’s erasure coding [164], which converts an object into multiple data chunks, making it difficult to extract a secret share from a single object, let alone retrieve all the shares from all objects. When a single KEK is stored in Ceph, it follows a process similar to that of Shamir’s secret shares. The primary difference is to store a single KEK, only one pool and one object are created.

7.7.4 TPM

Storing encryption keys inside the TPM is the most secure option offered by *Ceph-Vault*. Its tamper-resistant coating provides security from the malicious software [165].

In *CephVault*, an RSA key is employed as the KEK, created under the primary key, and the private part of the KEK never leaves the TPM in plaintext. Comprising keys reside in the TPM requires physical access to the TPM [166][167], making it an ideal choice for a KMS [168].

7.8 Threat Analysis

We follow STRIDE model (Section 2.12) for threat analysis.

- **Spoofing:** If a user’s credentials are spoofed, it will expose the user’s data. An adversary who is able to spoof the user will have access to the victim’s home directory, exposing the pre-shared encryption key pair used for the DEK distribution. However, spoofing a user’s credentials will not reveal other user’s data.

- **Tampering:** If the KEK is tampered with, the data will be lost permanently. To countermeasure tampering of the KEK, it is kept in a secure location in Ceph (Section 7.7). *CephVault* periodically validates the KEK checksum to identify any possible tampering. Also, if the KEK is stored in the TPM, tampering with the TPM requires physical access to the server. Moreover, the tamper-resistant property of the TPM makes it difficult for an adversary to alter the KEK stored in the TPM. Similarly, tampering with the configuration file will have unintentional consequences. The altered configuration file can lead to data unavailability to legitimate users. Keeping the configuration file in a secure location, which requires root privilege, safeguards unauthorized tampering.

- **Repudiation:** If logs are altered, application events will not provide accurate information regarding the key usage or a user’s activities, leading to repudiation. To countermeasure log tampering, logs are securely stored in the directory to which only the root user has access.

- **Information Disclosure:** Compromising logs can reveal data flow, method

names, and timestamps for running particular functions. An adversary can reveal encryption keys by analyzing the time a method performs cryptography operations [83]. *CephVault* stores log files in a secure location where only the root user can access them. Moreover, *CephVault* does not log any sensitive information such as encryption keys, user credentials, and IVs. *CephVault* leverages Object-Oriented Programming (OOP) concepts, where data and methods are kept separately to minimize data exposure. Similarly, databases, which contain encrypted DEKs and associated metadata, can disclose sensitive information. Only the root user can access the location where databases are stored. Also, compromising the database will not reveal the actual data, as DEKs are encrypted with the KEK. Administrators have the flexibility to choose a preferred location of the database, such as a different server.

- **Denial of Service (DoS):** If the KEK stored in Ceph is removed from the Ceph cluster, encrypted data will not be decrypted, leading to Denial of Service (DoS). Ceph ensures data availability through *erasure coding* [142]. If a copy of data containing the KEK is removed by the adversary, Ceph automatically detects the missing replication and replaces the missing data. Ceph's self-healing features minimize the threat of DoS attacks that replace the data containing the KEK. Further, *CephVault* accepts objects, pools, and namespaces as strings. A malicious user can provide malformed strings to initiate SQL inject attacks, exhaust resources, query locking, and eventually DoS attacks. *CephVault* validates user input, identifies malformed strings for possible SQL injection attacks, and terminates the process once the malformed string is identified.

- **Elevation of Privilege:** A compromised root user will have full access to the entire cluster, comprising Ceph, *CephArmor*, *CephVault*, databases, configuration files, and logs. Unauthorized elevation of privilege will expose the KEK from the configuration file, Ceph, or the TPM. An adversary can alter the KEK, configuration file, and database. *CephVault* does not have any mechanism to identify the unauthorized

elevation of privilege. However, if the KEK is altered after its creation, it can be detected by periodically checking the checksum of the KEKs. Moreover, *CephVault* can identify any modification of the *CephVault*'s binary files and notify the target recipients, minimizing the compromise duration.

7.9 CephVault Risk Assessment

Risk assessment is employed by organizations to identify assets, assess the overall security risks and vulnerabilities, and prioritize risks to address the risks. Table 7.11, and Table 7.12 illustrate the risk assessment in *CephVault*. We categorize risk levels as high, medium, and low based on the impact on the confidentiality, integrity, or availability of *CephVault*. The high risk describes a major impact, the medium risk describes a moderate impact, and the low risk illustrates a minor impact.

Table 7.11: CephVault Risk Assessment

Asset	Threat/ Vulnerability	Existing Controls	Level of Risk
User's credentials	Compromised credentials could lead to unauthorized access to encrypted data.	The user is authenticated, and only authenticated users can perform cryptography operations.	High
Integrity of the Encrypted DEK	Corrupted Encrypted DEK could lead to modification of the DEK. Encrypted data cannot be decrypted.	A checksum is employed for internal checksum verification and for key distribution.	High
The user's home directory	Compromised home directory could lead to the DEK being encrypted with the attacker's created key during the DEK distribution. <i>CephArmor</i> might fail to verify whether the DEK generated from <i>CephVault</i> .	The directory is owned by the user and reinforced with access control. Moreover, the user's ability to rotate the keys stored in the user's home directory without impacting <i>CephVault</i> 's performance enhances security.	Medium
KEK in Config	Compromised configuration could reveal the KEK.	The configuration file is secured with authentication and access control. Only root users can have access to the configuration file.	High
KEK in TPM	Compromised TPM could reveal the KEK.	TPM are secured with tamper-resistant enclosures. Physical access to the server is needed to access TPM.	High

Table 7.12: CephVault Risk Assessment (Continued)

Asset	Threat/ Vulnerability	Existing Controls	Level of Risk
KEK is Ceph	Compromised Ceph could reveal the KEK.	Ceph is secured with access control and authentication. Even if Ceph is compromised, the KEK retrieval is not straightforward; the KEK needs to be retrieved from objects, which can be further divided by erasure coding.	High
Configuration File	Since the configuration file contains many parameters that influence the default behavior of the <i>CephVault</i> , altering the configuration file could lead to the wanted behavior.	Configuration file is stored in a secure location, protected by authentication and access control.	High
Database	Compromised database could reveal the encrypted DEK and DEK_IV.	Database stored in a secure location, protected by authentication and access control. Employed <i>EVP</i> to safeguard DEK in the database and checksum to identify the DEK and DEK_IV alteration.	High
Memory region containing sensitive data	Operating system swapped the memory (pages) containing sensitive data into a disk, and the compromised disk could reveal the data.	Employed smart pointer that destroys objects no longer used, reducing Use-After-Free (UAF) vulnerability [169]. Memory lock is employed to prevent memory (pages) containing sensitive data, such as the DEK swapping to disk. Disk encryption is employed in the enterprise Ceph servers.	High

7.10 Summary

In this chapter, we analyzed the security aspect of *CephVault* and evaluated against information security principles, secure design principles, ability to countermeasure various attacks, and adherence to FIPS 140-2 security guidelines. We also discussed the threat model and defensive coding principles that *CephVault* followed. In the next chapter, we will discuss future research directions.

Chapter 8

Future Work

Future research directions can be targeted to provide enhanced security and better performance. *CephVault* could be extended in many ways, including developing new functionalities and evaluating against different use cases. Some of the future research directions are explored in the sections that follow.

8.1 Future Directions For Development

Future research directions involve the development of new functionalities for *CephVault*. Some of the research directions are as follows:

- **Implement Key Rotation Schemes with Better Security:** *CephVault* employs a traditional key rotation scheme, where encrypted data is decrypted with the old encryption key before being encrypted with a new one. Security is a major concern with this design since the data will be unprotected for a brief period during the decryption operation. A future research direction would be to update the encryption key and the ciphertext periodically. Updatable Encryption (UE) is one such encryption scheme where encrypted data is never decrypted [170]. Proxy re-encryption [171] could be used if *CephArmor* supported asymmetric encryption schemes. Since *CephVault* and *CephArmor* were developed separately, and *Ceph-*

Vault uses *CephArmor*'s API for key rotation, the traditional key rotation scheme is the only option. However, to implement UE or proxy re-encryption, both *CephArmor* and *CephVault* must be modified and tested thoroughly.

- **Support Various Cryptographic Processors:** *CephVault* supports a Trusted Platform Module (TPM) as a cryptographic processor. In the future, a Hardware Security Module (HSM) [172] could be employed as an alternative. Field Programmable Gate Arrays (FPGAs) could be another alternative for cryptographic processors, where an FPGA could be programmed to speed up AES encryption [173]. For envelope encryption of DEK, a volatile FPGA could be sufficient; however, to store encryption keys, a non-volatile FPGA would be required. Prior to research in this direction, a feasibility study would be required as TPMs are relatively more affordable than HSMs, design and implementation would be required for FPGA-based cryptographic processors, and FPGAs might provide a lower level of tamper-evidence protection for the secrets than HSM and TPM. PUF could also be employed for a key generation; however, PUF could be compromised by a side-channel attack [174].

- **Support Various Encryption Algorithms In TPM:** Currently, RSA is the only supported encryption algorithm in *CephVault*, when hardware-based cryptography is selected. However, other encryption algorithms supported by the TPM can be explored to identify any performance and security improvement.

- **Implement Group Key Management System:** *CephVault* was developed to manage encryption keys for users who leveraged Ceph for secure data storage. *CephVault* facilitates a secure KMS solution that eliminates any significant user intervention for managing the KMS life cycle. For security reasons, only the user who encrypted the data leveraging *CephArmor* can decrypt the respective data; even the root user is be unable to decrypt other user's data. A group key management system would be required for sharing data among other users. Research could be performed to identify which group key management would be suitable for *CephVault*.

• **Implement Database Encryption:** *CephVault* uses SQLite as a backend database to store encrypted DEKs and associated metadata. Although no encryption key is stored in the database in plaintext (except *No KEK*), database encryption would provide additional security. Currently, SQLite provides SQLite Encryption Extension (SEE) that provides a plethora of features, such as seamless integration with the existing SQLite databases, file-level encryption that protects the databases, cross-platform support, and secure management of the database encryption keys. However, SEE is a commercial product, and the customers are responsible for the implementation [175]. If we do not opt for purchasing SEE, we can develop similar functionalities to secure the *CephVault* databases.

• **Use Multi-Threading For Performance Gain:** *CephVault* was developed in C++, which supports multi-threading. In the future, we could explore the possibility of using multi-threading to improve the overall performance of the *CephVault*. However, multi-threading of *CephVault* would require judicious design, consideration, and implementation since *CephVault* leverages OpenSSL for cryptography operations, such as encrypting the DEK using envelope encryption, and OpenSSL is not thread-safe by default [176]. Though OpenSSL 1.1.0 or higher versions support multi-threading [177] for the random number generator `Rand.Bytes()`, OpenSSL is not entirely thread-safe [178].

• **Use Error Correction Code:** We developed a module for error correction code using Hamming code; however, we did not employ Hamming code for *CephVault*. Hamming code can detect single-bit errors and correct them, or Hamming code can detect a double-bit error without correcting the error. Hamming is also more computationally complex and time-consuming than checksum, so *CephVault* employs SHA-512 for data integrity. Various other error correction codes, such as Reed-Solomon codes [179] or Bose-Chaudhuri-Hocquenghem code [180], could be employed to identify their applicability.

- **Support Various Encryption Schemes:** *CephVault* was developed with future extensibility in mind. Various other encryption schemes, such as lightweight encryption [46] [181], or post-quantum encryption [182], could easily be incorporated to support envelope encryption of the DEK. Research could be done to identify any performance improvement and security enhancement of the KMS.
- **Support Various Secrets:** *CephVault* manages the life cycle of cryptographic keys. In the future, *CephVault* could be extended to support other secrets such as certificates, API keys, session keys, one-time passwords (OTPs), and database passwords.
- **Support Various Platforms:** Ceph supports various flavors of Linux as well as Windows [183]. *CephVault* was tested in Ubuntu 20.04.6 LTS, and in the future, *CephVault* could be extended to support a Windows operating systems. Since *CephVault* was written in C++, which is generally platform-independent, some of the platform-specific libraries, such as authentication modules used in *CephVault*, would require updates with the Windows operating system-specific libraries.
- **Support Various Databases:** *CephVault* leverages SQLite for storing encrypted keys as it was open source and did not incur additional costs. However, various enterprise SQL databases, such as Oracle Database, MySQL Enterprise Edition, or Microsoft SQL Server, could provide additional benefits over SQLite: enterprise databases support various clustering and replication options and are tailored to support massive amounts of data. On the other hand, SQLite lacks concurrent operations, which can support multiple read operations, but does not support simultaneous writes [184], making it a relatively less performant choice for *CephVault*. Also, the SQLite database mandates keeping data in the same physical machine where the application resides, making it less secure than the enterprise SQL databases that follow client-server architecture [185]. NoSQL databases, such as MongoDB, could be explored as an alternative, where MongoDB provides better scalability than SQLite.

However, like SQLite, the MongoDB community edition does not provide encryption of the data. Also, MongoDB does not support referential integrity [186], which is essential for *CephVault* to establish a relation between DEK and KEK.

- **Key Management Interoperability Protocol (KMIP):** The Organization for the Advancement of Structured Information Standards (OASIS), a non-profit consortium aimed to develop various security standards, introduced the Key Management Interoperability Protocol (KMIP) to integrate multiple cryptographically enabled applications seamlessly. KMIP facilitates users to send a request to a KMS server that encrypts and decrypts data without the need for direct access to the encryption key. *CephVault* provides similar functionalities where the key management life cycle is delegated to the server, and users do not need to manage the encryption keys. However, *CephVault* only supports key management for Ceph. Enabling the KMIP standard would allow *CephVault* to be interoperable with various encryption systems across the organization, supporting new and legacy enterprise applications [187].

- **User Interface for Administrators:** *CephVault* facilitates command line options for administrators and users to interact with the KMS and perform various operations. A web-based user interface could be developed to provide a better user experience to the administrators. *CephVault* employs various configuration settings that could change the behavior of the KMS. Having options to modify the setting through a user interface would have been a better user experience.

8.2 Future Direction for Evaluation

CephVault could be further evaluated after thorough testing with various platforms, scenarios, and parameters and research based on the outcome.

- **Evaluate CephVault After Penetration Testing:** *CephVault*, which was developed with basic security principles such as confidentiality, integrity, availability,

access control, accountability, and non-repudiation, is capable of preventing various attacks such as Buffer Overflow attacks, SQL injection attacks, directory traversal attacks, and Side Channel attacks. However, proper penetration testing could be performed to evaluate the results. Some of the popular penetration tools are Nmap [188], Astra [189], Metasploit [190], and Burp Suit [191]. Penetration testing would require expertise on various security implications on the network, applications, and systems.

- **Evaluate CephVault In Multiple Servers:** *CephVault* was developed as a centralized KMS, where one primary server assumed responsibility for serving requests for DEK and IV, and secondary servers could have replication of the application running on the primary server. In case the primary server was inaccessible or inoperable, one secondary server could assume the responsibility of the primary server. Hashicorp Vault operates in the same principle for the disaster recovery of their enterprise KMS [192]. *CephVault* was tested in a single server, and in the future, *CephVault* could be tested with multiple servers and with multiple users to identify performance impact.

- **Evaluate CephVault With Various Monitoring Tools:** *CephVault* was supported by a custom monitoring tool tailored to monitor *CephVault*. Currently, the monitoring tool developed for *CephVault* has some limited functionalities, but in the future, more popular monitoring tools could be employed to capture various data pertaining to the system, network, server, the *CephVault* application, and databases. Some of the open-source monitoring tools are Prometheus [193], Grafana [194], Nagios [195], and Zabbix [196]. However, unlike the in-house monitoring tool for *CephVault*, these open-source monitoring tools require separate processes or servers to function properly and put additional overhead on the server that runs Ceph, which is a resource-intensive application. For example, the Prometheus server needs to be up and running to scrape data at regular intervals. Also, enterprise monitoring

tools, such as AppDynamics [197], IBM Tivoli Monitoring [198], and Splunk [199] could be employed for monitoring. However, cost and performance analysis would be required for these monitoring tools. Research could also be performed to leverage Ceph monitor [200], whose primary responsibility is to monitor Ceph’s health, to monitor application-specific logs of *CephVault*.

- **Evaluate CephVault With Various Audit Loggers:** *CephVault* employs a custom logger leveraging spdlog, a faster, cross-platform, and multi-threaded logger. Since logging is a resource-intensive process, *CephVault* could be evaluated with various other loggers, such as Graylog [201], rsyslog [202], and log4j [203] to identify any performance improvement. Research would be required to identify whether the selected logger suffers known vulnerabilities. For example, log4j has some known vulnerabilities [204], so log4j would need to be used with caution. Security-Enhanced Linux (SELinux) log could also be integrated with *CephVault*. Unlike spdlog, SELinux is platform-dependent. Also, SELinux logs are primarily focused on system-wide access control and security policy, whereas custom loggers capture application-specific events, which is necessary to understand the behavior of the application.

- **Evaluate CephVault With Benchmark Tools:** Ceph provides a benchmarking tool, namely RADOS Bench, that helps to measure the performance of the OSDs. RADOS Bench primarily measures the capability of the cluster during the read and write operation to Ceph. Various sequential, random reads and writes can be simulated and measured using RADOS Bench. Since *CephVault* neither writes encrypted data to Ceph nor reads encrypted data from Ceph, *CephVault* was evaluated with microbenchmarks. In the future, various other benchmark tools could be explored to capture the performance of *CephVault*.

- **Evaluate CephVault With Various Network Connections:** *CephVault* was evaluated as the user provided a command to Ceph to store encrypted data, which

was intercepted by Ceph and forwarded to *CephVault* through *CephArmor*. Once *CephVault* generates the DEK and IV, and *CephArmor* verifies the origin of the secrets, the timing was captured. However, time to encrypt and decrypt user-provided data was not in the scope of this thesis; it was the responsibility of *CephArmor*. In this scenario, the user resides on the same LAN as the Ceph server resides. If there is a latency in the process between the time the user provides the command and *CephArmor* verifies the secrets, our performance evaluation would be different. So, in the future, more evaluation could be performed to identify the impact of the network latency in the overall performance of the *CephVault*. However, we also captured various activities, such as the time to generate a key, store, and backup the keys, and the generation and verification time of digital signature. Since these activities are performed in the Ceph server, the time for these functions is expected to be the same, even if there is a latency in the network.

8.3 Summary

In this chapter, we discussed various future research directions in terms of new development and evaluation that could reinforce *CephVault*. Research directions were provided with relevant references and details, which would help to embark on the proper direction. In the next chapter, we will conclude our research.

Chapter 9

Conclusion

CephVault is an amalgamation of various information security principles, secure design principles, defensive coding practice, and adherence to FIPS 140-2 guidelines, where some of the recommended security primitives such as AES encryption, digital signature, checksum, and envelope encryption are used. *CephVault*'s modular design, clear separation between data and method, data-hiding, input validations, use of smart pointers, real-time notification, extensive exception handling, compilation with secure flags, evaluation against static code analyzers to countermeasure known vulnerabilities, and mitigation of some known attacks make it a safer key management solution. The primary focus of this solution was performance as well as security, with various configuration options as per the requirements.

The flexibility to leverage both hardware and software to perform cryptography operations and secure key storage, options to choose different envelope encryption types, and operate either per-user or per-object basis makes *CephVault* a competitive key management solution and a preferable choice to many of the existing KMS solutions in the market.

Bibliography

- [1] F. Pub, “Security requirements for cryptographic modules,” *FIPS PUB*, vol. 140, pp. 140–2, 1994.
- [2] C. Cohen. Cryptographic keys. (Accessed: Jan. 15, 2024). [Online]. Available: https://google.github.io/tpm-js/#pg_keys
- [3] K. Masuch, M. Greve, S. Trang, and L. M. Kolbe, “Apologize or justify? examining the impact of data breach response actions on stock value of affected companies?” *Computers & Security*, vol. 112, p. 102502, 2022.
- [4] E. Barker, E. Barker, W. Burr, W. Polk, M. Smid *et al.*, *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2006.
- [5] E. Barker and W. Barker, “Recommendation for key management, part 2: best practices for key management organization,” National Institute of Standards and Technology, Tech. Rep., 2018.
- [6] L. Scott and D. E. Denning, “A location based encryption technique and some of its applications,” in *Proceedings of the 2003 National Technical Meeting of The Institute of Navigation*, 2003, pp. 734–740.
- [7] M. Ebrahim, S. Khan, and U. B. Khalid, “Symmetric algorithm survey: a comparative analysis,” *arXiv preprint arXiv:1405.0398*, 2014.

- [8] statista. Data growth worldwide 2010-2025. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.statista.com/statistics/871513/worldwide-data-created>
- [9] S. A. Weil, “Ceph: reliable, scalable, and high-performance distributed storage,” Ph.D. dissertation, University of California, Santa Cruz, 2007.
- [10] C. authors and contributors. Data security and hardening guide red hat ceph storage 5 — red hat customer portal. (Accessed: Jan. 15, 2024). [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/5/html-single/data_security_and_hardening_guide/index
- [11] Ceph. Encryption. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/quincy/radosgw/encryption>
- [12] F. K. Parast, B. Kelly, S. Hakak, Y. Wang, and K. B. Kent, “Cepharmor: A lightweight cryptographic interface for secure high-performance ceph storage systems,” *IEEE Access*, vol. 10, pp. 127 911–127 927, 2022.
- [13] R. Moulds. Key management for dummies. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.ictsecurity.com.au/wp-content/uploads/2017/12/key-management-for-dummies.pdf>
- [14] Thales. What is the encryption key management lifecycle? — thales. (Accessed: Jan. 15, 2024). [Online]. Available: <https://cpl.thalesgroup.com/faq/key-secrets-management/what-encryption-key-management-lifecycle>
- [15] B. L. Tomhave, “Key management: The key to encryption,” *EDPACS: The EDP Audit, Control, and Security Newsletter*, vol. 38, no. 4, pp. 12–19, 2008.
- [16] J. Li, J. Zheng, and P. Whitlock, “Efficient deterministic and non-deterministic pseudorandom number generation,” *Mathematics and Computers in Simulation*, vol. 143, pp. 114–124, 2018.

- [17] S. Chandra, S. Paul, B. Saha, and S. Mitra, “Generate an encryption key by using biometric cryptosystems to secure transferring of data over a network,” *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 12, no. 1, pp. 16–22, 2013.
- [18] S. M. Matyas, “Public key registration,” in *Advances in Cryptology—CRYPTO’86: Proceedings 6*. Springer, 1987, pp. 451–458.
- [19] C. Schleiffer, M. Wolf, A. Weimerskirch, and L. Wolleschensky, “Secure key management-a key feature for modern vehicle electronics,” *SAE Technical Paper, SAE International*, pp. 01–1418, 2013.
- [20] W. Staff. 5 key elements of data encryption best practices. (Accessed: Jan. 15, 2024). [Online]. Available: <https://wickr.com/5-key-elements-of-data-encryption-best-practices/>
- [21] Google. Envelope encryption. (Accessed: Jan. 15, 2024). [Online]. Available: <https://cloud.google.com/kms/docs/envelope-encryption>
- [22] A. Penrig, D. Song, and D. Tygar, “Elk, a new protocol for efficient large-group key distribution,” in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 2000, pp. 247–262.
- [23] M. Steiner, G. Tsudik, and M. Waidner, “Diffie-hellman key distribution extended to group communication,” in *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, 1996, pp. 31–37.
- [24] L. Keuninckx, M. C. Soriano, I. Fischer, C. R. Mirasso, R. M. Nguimdo, and G. Van der Sande, “Encryption key distribution via chaos synchronization,” *Scientific reports*, vol. 7, no. 1, pp. 1–14, 2017.
- [25] H. Harney and C. Muckenhirn, “Rfc2094: Group key management protocol (gkmp) architecture,” 1997.

- [26] Y. Cheng and D. P. Agrawal, “Efficient pairwise key establishment and management in static wireless sensor networks,” in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005*. IEEE, 2005, pp. 7–pp.
- [27] V. Cholvi, “Quantum byzantine agreement for any number of dishonest parties,” *Quantum Information Processing*, vol. 21, no. 4, pp. 1–11, 2022.
- [28] M. Khorrampanah and M. Houshmand, “Effectively combined multi-party quantum secret sharing and secure direct communication,” *Optical and Quantum Electronics*, vol. 54, no. 4, pp. 1–11, 2022.
- [29] S. Myers and A. Shull, “Efficient hybrid proxy re-encryption for practical revocation and key rotation,” *Cryptology ePrint Archive*, 2017.
- [30] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, “Key homomorphic prfs and their applications,” in *Annual Cryptology Conference*. Springer, 2013, pp. 410–428.
- [31] A. Everspaugh, K. Paterson, T. Ristenpart, and S. Scott, “Key rotation for authenticated encryption,” in *Annual International Cryptology Conference*. Springer, 2017, pp. 98–129.
- [32] A. Lehmann and B. Tackmann, “Updatable encryption with post-compromise security,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 685–716.
- [33] K. Lee, K. Yim, and J. T. Seo, “Ransomware prevention technique using key backup,” *Concurrency and Computation: Practice and Experience*, vol. 30, no. 3, p. e4337, 2018.

- [34] J. T. Soma, “Encryption, key recovery, and commercial trade secret assets: A proposed legislative model,” *Rutgers Computer & Tech. LJ*, vol. 25, p. 97, 1999.
- [35] D. E. Denning and W. E. Baugh Jr, “Key escrow encryption policies and technologies,” 1996.
- [36] B. T. E. Commerce, “Private key escrow system,” in *SPA/AEA Cryptography Policy Workshop*, vol. 17, 1995.
- [37] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, P. G. Neumann, R. L. Rivest, J. I. Schiller *et al.*, “The risks of key recovery, key escrow, and trusted third-party encryption,” 1997.
- [38] R. Gennaro, P. Karger, S. Matyas, M. Peyravian, A. Roginsky, D. Safford, M. Willett, and N. Zunic, “Two-phase cryptographic key recovery system,” *computers & Security*, vol. 16, no. 6, pp. 481–506, 1997.
- [39] D. E. Denning and D. K. Branstad, “A taxonomy for key escrow encryption systems,” *Communications of the ACM*, vol. 39, no. 3, pp. 34–40, 1996.
- [40] Y. Wang, B. Ramamurthy, and X. Zou, “Keyrev: An efficient key revocation scheme for wireless sensor networks,” in *2007 IEEE International Conference on Communications*. IEEE, 2007, pp. 1260–1265.
- [41] K. Pradeep and V. Vijayakumar, “Survey on the key management for securing the cloud,” *Procedia Computer Science*, vol. 50, pp. 115–121, 2015.
- [42] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, “Building an encrypted and searchable audit log,” in *NDSS*, vol. 4, 2004, pp. 5–6.
- [43] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, “Nist special publication 800-57,” *NIST Special publication*, vol. 800, no. 57, pp. 1–142, 2007.

- [44] W. Stallings, *Computer security principles and practice*, 2015.
- [45] H. Delfs and H. Knebl, “Symmetric-key encryption,” in *Introduction to Cryptography*. Springer, 2007, pp. 11–31.
- [46] M. Gupta, K. K. Gupta, and P. K. Shukla, “Session key based fast, secure and lightweight image encryption algorithm,” *Multimedia Tools and Applications*, vol. 80, no. 7, pp. 10 391–10 416, 2021.
- [47] G. J. Watson, “Provable security in practice: Analysis of ssh and cbc mode with padding,” Ph.D. dissertation, Citeseer, 2010.
- [48] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [49] U. SenthilKumar and U. Senthilkumaran, “Review of asymmetric key cryptography in wireless sensor networks,” *International Journal of Engineering and Technology*, vol. 8, no. 2, pp. 859–862, 2016.
- [50] Q. Zhang, “An overview and analysis of hybrid encryption: the combination of symmetric encryption and asymmetric encryption,” in *2021 2nd international conference on computing and data science (CDS)*. IEEE, 2021, pp. 616–622.
- [51] W. Ren and Z. Miao, “A hybrid encryption algorithm based on des and rsa in bluetooth communication,” in *2010 Second International Conference on Modeling, Simulation and Visualization Methods*. IEEE, 2010, pp. 221–225.
- [52] Ü. Çavuşoğlu, S. Kaçar, A. Zengin, and I. Pehlivan, “A novel hybrid encryption algorithm based on chaos and s-aes algorithm,” *Nonlinear Dynamics*, vol. 92, no. 4, pp. 1745–1759, 2018.

- [53] T. Landstra, M. Zawodniok, and S. Jagannathan, “Energy-efficient hybrid key management protocol for wireless sensor networks,” in *32nd IEEE conference on local computer networks (LCN 2007)*. IEEE, 2007, pp. 1009–1016.
- [54] K. Sajay, S. S. Babu, and Y. Vijayalakshmi, “Enhancing the security of cloud data using hybrid encryption algorithm,” *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2019.
- [55] T. Nie and T. Zhang, “A study of des and blowfish encryption algorithm,” in *Tencon 2009-2009 IEEE Region 10 Conference*. IEEE, 2009, pp. 1–4.
- [56] G.-l. Guo, Q. Qian, and R. Zhang, “Different implementations of aes cryptographic algorithm,” in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 2015, pp. 1848–1853.
- [57] M. Sharma and R. Garg, “Des: The oldest symmetric block key encryption algorithm,” in *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*. IEEE, 2016, pp. 53–58.
- [58] O. Y. Cheung, K. H. Tsoi, P. H. W. Leong, and M.-P. Leong, “Tradeoffs in parallel and serial implementations of the international data encryption algorithm idea,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2001, pp. 333–347.
- [59] M. Amara and A. Siad, “Elliptic curve cryptography and its applications,” in *International workshop on systems, signal processing and their applications, WOSSPA*. IEEE, 2011, pp. 247–250.

- [60] K. Neela and V. Kavitha, “An improved rsa technique with efficient data integrity verification for outsourcing database in cloud,” *Wireless Personal Communications*, vol. 123, no. 3, pp. 2431–2448, 2022.
- [61] Y. Tsiounis and M. Yung, “On the security of elgamal based encryption,” in *International Workshop on Public Key Cryptography*. Springer, 1998, pp. 117–134.
- [62] N. Fazio, R. Gennaro, T. Jafarikhah, and W. E. Skeith, “Homomorphic secret sharing from paillier encryption,” in *International Conference on Provable Security*. Springer, 2017, pp. 381–399.
- [63] S. Myers and A. Shull, “Practical revocation and key rotation,” in *Cryptographers’ Track at the RSA Conference*. Springer, 2018, pp. 157–178.
- [64] P. Rogaway and T. Shrimpton, “Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance,” in *International workshop on fast software encryption*. Springer, 2004, pp. 371–388.
- [65] M. Harini, K. P. Gowri, C. Pavithra, and M. P. Selvarani, “A novel security mechanism using hybrid cryptography algorithms,” in *2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE)*. IEEE, 2017, pp. 1–4.
- [66] A. A.-R. El-Douh, S. F. Lu, A. Elkony, and A. Ameen, “A systematic literature review: The taxonomy of hybrid cryptography models,” in *Future of Information and Communication Conference*. Springer, 2022, pp. 714–721.
- [67] H. Abroshan, “A hybrid encryption solution to improve cloud computing security using symmetric and asymmetric cryptography algorithms,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, 2021.

- [68] S. Gueron. Intel advanced encryption standard (aes) new instruction set. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.intel.com/content/dam/develop/external/us/en/documents/aes-wp-2012-09-22-v01-165683.pdf>
- [69] Verkkokauppa. Asus trusted platform module tpm-m r2.0 -moduuli. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.verkkokauppa.com/fi/product/387140/Asus-Trusted-Platform-Module-TPM-M-R2-0-moduuli>
- [70] Sefira. Hardware security module. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.sefira.cz/en/hsm-hardware-security-module/>
- [71] J. Obermaier, F. Hauschild, M. Hiller, and G. Sigl, “An embedded key management system for puf-based security enclosures,” in *2018 7th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2018, pp. 1–6.
- [72] Spartan. Xilinx spartan-6 fgg484 fpga board (5v i/o) (xcm-019-lx75). (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.amazon.ca/Xilinx-Spartan-6-FGG484-board-XCM-019-LX75/dp/B00N3LH00Q>
- [73] W. Arthur, D. Challener, and K. Goldman, *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature, 2015.
- [74] C. Böhm and M. Hofer, *Physical unclonable functions in theory and practice*. Springer Science & Business Media, 2012.
- [75] A. Cilardo, A. Mazzeo, L. Romano, and G. P. Saggese, “An fpga-based key-store for improving the dependability of security services,” in *10th IEEE international workshop on object-oriented real-time dependable systems*. IEEE, 2005, pp. 389–396.
- [76] T. E. Güneysu, *FPGAs in Cryptography*. Boston, MA: Springer US, 2011, pp. 499–501.

- [77] B. Libert and M. Yung, “Non-interactive cca-secure threshold cryptosystems with adaptive security: New framework and constructions,” in *Theory of Cryptography Conference*. Springer, 2012, pp. 75–93.
- [78] L. T. Brandao, N. W. Mouha, A. T. Vassilev *et al.*, “Threshold schemes for cryptographic primitives,” 2019.
- [79] S. Adi, “How to share a secret,” *Commun. ACM*, vol. 22, pp. 612–613, 1979.
- [80] A. Sheikh, “Buffer overflow,” in *Certified Ethical Hacker (CEH) Preparation Guide*. Springer, 2021, pp. 165–173.
- [81] M. Jouini, L. B. A. Rabai, and A. B. Aissa, “Classification of security threats in information systems,” *Procedia Computer Science*, vol. 32, pp. 489–496, 2014.
- [82] M. Flanders, “A simple and intuitive algorithm for preventing directory traversal attacks,” *arXiv preprint arXiv:1908.04502*, 2019.
- [83] F.-X. Standaert, “Introduction to side-channel attacks,” *Secure integrated circuits and systems*, pp. 27–42, 2010.
- [84] F. Koeune and F.-X. Standaert, “A tutorial on physical security and side-channel attacks,” *International School on Foundations of Security Analysis and Design*, pp. 78–108, 2004.
- [85] R. Al Nafea and M. A. Almaiah, “Cyber security threats in cloud: Literature review,” in *2021 international conference on information technology (ICIT)*. IEEE, 2021, pp. 779–786.
- [86] F. Callegati, W. Cerroni, and M. Ramilli, “Man-in-the-middle attack to the https protocol,” *IEEE Security & Privacy*, vol. 7, no. 1, pp. 78–81, 2009.
- [87] L. R. Knudsen, M. J. Robshaw, L. R. Knudsen, and M. J. Robshaw, “Brute force attacks,” *The Block Cipher Companion*, pp. 95–108, 2011.

- [88] L. Zhang, C. Tan, and F. Yu, “An improved rainbow table attack for long passwords,” *Procedia Computer Science*, vol. 107, pp. 47–52, 2017.
- [89] Y. Zhong and U. Guin, “Fault-injection based chosen-plaintext attacks on multicyle aes implementations,” in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 443–448.
- [90] J. Rizzo and T. Duong, “Practical padding oracle attacks,” in *4th USENIX Workshop on Offensive Technologies (WOOT 10)*, 2010.
- [91] C. authors and contributors. Architecture. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/latest/architecture/>
- [92] A. D’atri, V. Bhembre, and K. Singh, *Learning Ceph: Unifed, scalable, and reliable open source storage solution*. Packt Publishing Ltd, 2017.
- [93] K. Singh, *Learning Ceph*. Packt Publishing Ltd, 2015.
- [94] C. authors and contributors. Ceph user management. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/latest/rados/operations/user-management>
- [95] S. Wilbur L. Ross, Jr. Security requirements for cryptographic modules. (Accessed: Jan. 15, 2024). [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf>
- [96] OpenSSL. FIPS 140-3 Plans. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.openssl.org/blog/blog/2022/09/30/fips-140-3/>
- [97] A. O. Stanley R. Snouffer. A Comparison of the Security Requirements for Cryptographic Modules in FIPS 140-1

and FIPS 140-2. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.nist.gov/publications/comparison-security-requirements-/cryptographic-modules-fips-140-1-and-fips-140-2>

- [98] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014.
- [99] I. Venafi. The world leader in machine identity management. (Accessed: Jan. 15, 2024). [Online]. Available: <https://venafi.com>
- [100] Thales. Thales - building a future we can all trust. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.thalesgroup.com/en>
- [101] J.-W. Lee, E. Lee, Y.-S. Kim, and J.-S. No, “Hierarchical galois key management systems for privacy preserving aiaas with homomorphic encryption,” *Cryptology ePrint Archive*, 2022.
- [102] Z. Mahmood, H. Ning, and A. Ghafoor, “A polynomial subset-based efficient multi-party key management system for lightweight device networks,” *Sensors*, vol. 17, no. 4, p. 670, 2017.
- [103] Y. W. Law, M. Palaniswami, G. Kounga, and A. Lo, “Wake: Key management scheme for wide-area measurement systems in smart grid,” *IEEE Communications Magazine*, vol. 51, no. 1, pp. 34–41, 2013.
- [104] O. Pal, B. Alam, V. Thakur, and S. Singh, “Key management for blockchain technology,” *ICT express*, vol. 7, no. 1, pp. 76–80, 2021.
- [105] C. K. Wong and S. S. Lam, “Keystone: A group key management service,” in *International Conference on Telecommunications, ICT*, vol. 2000. Citeseer, 2000.

- [106] H. Harney and C. Muckenhirn, “Group key management protocol (gkmp) architecture,” Tech. Rep., 1997.
- [107] L. Veltri, S. Cirani, S. Busanelli, and G. Ferrari, “A novel batch-based group key management protocol applied to the internet of things,” *Ad Hoc Networks*, vol. 11, no. 8, pp. 2724–2737, 2013.
- [108] R. Roman, C. Alcaraz, J. Lopez, and N. Sklavos, “Key management systems for sensor networks in the context of the internet of things,” *Computers & Electrical Engineering*, vol. 37, no. 2, pp. 147–159, 2011.
- [109] M. S. Yousefpoor and H. Barati, “Dskms: A dynamic smart key management system based on fuzzy logic in wireless sensor networks,” *Wireless Networks*, vol. 26, no. 4, pp. 2515–2535, 2020.
- [110] X. Du, Y. Xiao, M. Guizani, and H.-H. Chen, “An effective key management scheme for heterogeneous sensor networks,” *Ad hoc networks*, vol. 5, no. 1, pp. 24–34, 2007.
- [111] W. Fumy and P. Landrock, “Principles of key management,” *IEEE Journal on selected areas in communications*, vol. 11, no. 5, pp. 785–793, 1993.
- [112] W.-B. Lee and C.-D. Lee, “A cryptographic key management solution for hipaa privacy/security regulations,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 12, no. 1, pp. 34–41, 2008.
- [113] M. Dworkin, “Nist special publication 800-38b,” *NIST special publication*, vol. 800, no. 38B, p. 38B, 2005.
- [114] C. authors and contributor. Openstack barbican integration. (Accessed: Jan. 15, 2024). [Online]. Available: <https://mta.openssl.org/pipermail/openssl-users/2020-November/013146.html>

- [115] C. authors. Hashicorp vault integration. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/latest/radosgw/vault>
- [116] OpenStack. Barbican. (Accessed: Jan. 15, 2024). [Online]. Available: <https://wiki.openstack.org/wiki/Barbican>
- [117] O. Stack. Using keystone middleware with barbican. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.openstack.org/barbican/latest/configuration/keystone.html>
- [118] OpenStack. Use cases. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.openstack.org/security-guide/secrets-management/secrets-management-use-cases.html#passwords-in-config-files>
- [119] B. OpenStack. Configure secret store back-end. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.openstack.org/barbican/latest/install/barbican-backend.html>
- [120] O. Foundation. Encrypt your volumes with barbican. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.youtube.com/watch?v=pWu0fDgBbk4>
- [121] R. H. Inc. Developer guide red hat ceph storage 3. (Accessed: Jan. 15, 2024). [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html-single/developer_guide/index#s3-api-encryption
- [122] Hashicorp. Manage secrets and protect sensitive data. (Accessed: Jan. 15, 2024). [Online]. Available: <https://developer.hashicorp.com/vault>
- [123] V. Matsiako. Hashicorp vault pricing — complete guide [2023]. (Accessed: Jan. 15, 2024). [Online]. Available: <https://infisical.com/blog/hashicorp-vault-pricing>

- [124] A. project contributors. Protecting sensitive data with ansible vault. (Accessed: Jan. 15, 2024). [Online]. Available: https://docs.ansible.com/ansible/latest/vault_guide
- [125] N. Sullivan. Red october: Cloudflare's open source implementation of the two-man rule. (Accessed: Jan. 15, 2024). [Online]. Available: <https://blog.cloudflare.com/red-october-cloudflares-open-source-implementation-of-the-two-man-rule/>
- [126] C. Hale. codahale/sneaker: A tool for securely storing secrets on s3. (Accessed: Jan. 15, 2024). [Online]. Available: <https://github.com/codahale/sneaker>
- [127] Square. Keywhiz: A system for distributing and managing secrets. (Accessed: Jan. 15, 2024). [Online]. Available: <https://square.github.io/keywhiz/>
- [128] C. Hale. xordataexchange: crypt. (Accessed: Jan. 15, 2024). [Online]. Available: <https://github.com/xordataexchange/crypt>
- [129] S. Roland. Keeto. (Accessed: Jan. 15, 2024). [Online]. Available: <https://github.com/flix-/keeto>
- [130] Keeto. Docker. (Accessed: Jan. 15, 2024). [Online]. Available: <https://keeto.readthedocs.io/en/0.4.1-beta/docker.html>
- [131] T. Crevon. oleiade/trousseau: File based encrypted key-value store. (Accessed: Jan. 15, 2024). [Online]. Available: <https://github.com/oleiade/trousseau>
- [132] CyberArk. Conjur: Secrets management. (Accessed: Jan. 15, 2024). [Online]. Available: <https://github.com/cyberark/conjur>
- [133] S. Consortium. What is sqlite. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.sqlite.org/index.html>

- [134] Q. Li and Y.-L. Chen, “Data flow diagram,” in *Modeling and Analysis of Enterprise and Information Systems*. Springer, 2009, pp. 85–97.
- [135] C. authors and contributors. Encryption. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/quincy/radosgw/encryption>
- [136] I. Documentation. Ceph on-wire encryption. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.ibm.com/docs/en/storage-ceph/7?topic=components-ceph-wire-encryption>
- [137] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.
- [138] M. Särelä, T. Kyöstilä, T. Kiravuo, and J. Manner, “Evaluating intrusion prevention systems with evasions,” *International Journal of Communication Systems*, vol. 30, no. 16, p. e3339, 2017.
- [139] 45Drives. Snapshield ransomware activated fuse. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.45drives.com/solutions/ransomware/>
- [140] Bitdefender. Bitdefender - global leader in cybersecurity software. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.bitdefender.com/>
- [141] S. Bhatt, P. K. Manadhata, and L. Zomlot, “The operational role of security information and event management systems,” *IEEE security & Privacy*, vol. 12, no. 5, pp. 35–41, 2014.
- [142] C. authors and contributors. Erasure code. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/latest/rados/operations/erasure-code/>
- [143] J. Viega and G. R. McGraw, *Building secure software: How to avoid security problems the right way portable documents*. Pearson Education, 2001.

- [144] M. Olsson and M. Olsson, “Smart pointers,” *C++ 17 Quick Syntax Reference: A Pocket Guide to the Language, APIs and Library*, pp. 157–160, 2018.
- [145] T. Ramananandro, G. Dos Reis, and X. Leroy, “A mechanized semantics for c++ object construction and destruction, with applications to resource management,” *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 521–532, 2012.
- [146] G. Obiltschnig, “Using c++ to create better device software.”
- [147] D. A. Wheeler. Flawfinder. (Accessed: Jan. 15, 2024). [Online]. Available: <https://dwheeler.com/flawfinder>
- [148] T. O. P. Authors. Openssl cryptography and ssl/tls toolkit. (Accessed: Jan. 15, 2024). [Online]. Available: https://www.openssl.org/docs/man1.1.1/man3/RAND_bytes.html
- [149] K. Geisshirt, “Pluggable authentication modules,” *Birmingham: Packt Publishing Ltd*, 2007.
- [150] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn, *A practical guide to trusted computing*. Pearson Education, 2007.
- [151] M. D. James, *A reconfigurable trusted platform module*. Brigham Young University, 2017.
- [152] A. Segall, *Trusted Platform Modules: Why, when and how to use them*. Institution of Engineering and Technology, 2016.
- [153] T. Published. Trusted platform module library part 1: Architecture. (Accessed: Jan. 15, 2024). [Online]. Available: <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf#page=222>

- [154] W. Roberts. TPM2.Encryptdecrypt not supported by my TPM2. (Accessed: Jan. 15, 2024). [Online]. Available: <https://github.com/tpm2-software/tpm2-tools/issues/3180>
- [155] G. Melman. Spdlog. (Accessed: Jan. 15, 2024). [Online]. Available: <https://github.com/gabime/spdlog>
- [156] K. D. Dent, *Postfix: The Definitive Guide: A Secure and Easy-to-Use MTA for UNIX*. " O'Reilly Media, Inc.", 2003.
- [157] J. Viega, M. Messier, and P. Chandra, *Network security with openssl: cryptography for secure communications*. " O'Reilly Media, Inc.", 2002.
- [158] IBM. Size considerations for public and private keys. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.ibm.com/docs/en/zos/2.3.0?topic=certificates-size-considerations-public-private-keys>
- [159] SQLite. Defense against the dark arts. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.sqlite.org/security.html>
- [160] I. Snowflake. Transactions. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.snowflake.com/en/sql-reference/transactions#label-scoped-transactions>
- [161] T. H. Tran, H. L. Pham, and Y. Nakashima, "A high-performance multimem sha-256 accelerator for society 5.0," *IEEE Access*, vol. 9, pp. 39 182–39 192, 2021.
- [162] A. Segall. Introduction to trusted computing. (Accessed: Jan. 15, 2024). [Online]. Available: <https://opensecuritytraining.info/IntroToTrustedComputing.html>

- [163] I. Red Hat. Chapter 6. ceph user management. (Accessed: Jan. 15, 2024). [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/5/html/administration_guide/ceph-user-management
- [164] C. authors and contributors. Erasure coding. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/latest/rados/operations/erasure-code/>
- [165] V. Pamnani. Trusted Platform Module Technology Overview. (Accessed: Jan. 15, 2024). [Online]. Available: <https://learn.microsoft.com/en-us/windows/security/hardware-security/tpm/trusted-platform-module-overview>
- [166] M. B. de Assumpção, M. A. dos Reis, M. R. Marcondes, P. M. da Silva Eleutério, and V. H. Vieira, “Forensic method for decrypting tpm-protected bitlocker volumes using intel dci,” *Forensic Science International: Digital Investigation*, vol. 44, p. 301514, 2023.
- [167] O. Cyberdefense. TPM sniffing. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.orange cyberdefense.com/ch/insights/blog/tpm-sniffing>
- [168] D. Liu, J. Lee, J. Jang, S. Nepal, and J. Zic, “A cloud architecture of virtual trusted platform modules,” in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. IEEE, 2010, pp. 804–811.
- [169] J. Huang, “Ufo: predictive concurrency use-after-free detection,” in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 609–619.
- [170] R. Nishimaki, “The direction of updatable encryption does matter,” in *IACR International Conference on Public-Key Cryptography*. Springer, 2022, pp. 194–224.

- [171] D. Nunez, I. Agudo, and J. Lopez, "Proxy re-encryption: Analysis of constructions and its application to secure access delegation," *Journal of Network and Computer Applications*, vol. 87, pp. 193–209, 2017.
- [172] M. H. Murtaza, H. Tahir, S. Tahir, Z. A. Alizai, Q. Riaz, and M. Hussain, "A portable hardware security module and cryptographic key generator," *Journal of Information Security and Applications*, vol. 70, p. 103332, 2022.
- [173] S. S. S. Priya, P. Karthigaikumar, and N. R. Teja, "Fpga implementation of aes algorithm for high speed applications," *Analog Integrated Circuits and Signal Processing*, pp. 1–11, 2022.
- [174] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-channel analysis of pufs and fuzzy extractors," in *Trust and Trustworthy Computing: 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings 4*. Springer, 2011, pp. 33–47.
- [175] SQLite. Sqlite. (Accessed: Jan. 15, 2024). [Online]. Available: <https://sqlite.org/purchase/see#:~:text=Aperpetualsourcecodelicense,forSEEsourcedeonly>
- [176] O. WiKi. FIPS 140-3, Security Requirements for Cryptographic Modules. (Accessed: Jan. 15, 2024). [Online]. Available: https://wiki.openssl.org/index.php/Libcrypto_API#Thread_Safet
- [177] M. Caswells. Rand_bytes thread safety. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/latest/radosgw/barbican/>
- [178] T. O. P. Authors. OpenSSL Cryptography and SSL/TLS Toolkit. (Accessed: Jan. 15, 2024). [Online]. Available: https://www.openssl.org/docs/man1.0.2/man3/threads.html#:~:text=OpenSSLcangenerallybeused,set,thelocking_functionandthreadid_func

- [179] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.
- [180] R. Chien, “Cyclic decoding procedures for bose-chaudhuri-hocquenghem codes,” *IEEE Transactions on information theory*, vol. 10, no. 4, pp. 357–363, 1964.
- [181] A. Vahi and S. Jafarali Jassbi, “Separ: A new lightweight hybrid encryption algorithm with a novel design approach for iot,” *Wireless Personal Communications*, vol. 114, pp. 2283–2314, 2020.
- [182] D. Joseph, R. Misoczki, M. Manzano, J. Tricot, F. D. Pinuaga, O. Lacombe, S. Leichenauer, J. Hidary, P. Venables, and R. Hansen, “Transitioning organizations to post-quantum cryptography,” *Nature*, vol. 605, no. 7909, pp. 237–243, 2022.
- [183] B. Kelly. Ceph on window. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.45drives.com/community/articles/Ceph-on-Windows>
- [184] L. Carr. Sqlite’s serverless architecture doesn’t serve iot environments well. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.actian.com/blog/data-management/sqlites-serverless-architecture-doesnt-serve-iot-environments-well/#:~:text=SQLitecanhandlesimultaneouslyread,ofwritesfromseveralconnections>
- [185] SQLite. Appropriate uses for sqlite. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.sqlite.org/whentouse.html#:~:text=AnSQLiteDatabaseislimited,tosomethinglessthanthis>
- [186] A. Cloud. Mariadb vs. mongodb – which one should i choose. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.alibabacloud.com>

com/blog/mariadb-vs--mongodb-which-one-should-i-choose_598549#:~: text=MongoDBdoesnotuseforeign,grabdatafromanytable

- [187] T. Cox and C. White, “Oasis key management interoperability protocol (kmip) tc,” *by OASIS. Retrieved from*, 2019.
- [188] Nmap. Nmap: the network mapper free security scanner. (Accessed: Jan. 15, 2024). [Online]. Available: <https://nmap.org>
- [189] I. ASTRA IT. Security conscious companies trust astra for continuous pentests. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.getastra.com>
- [190] Metasploit. Metasploit the world’s most used penetration testing framework. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.metasploit.com/>
- [191] PortSwigger. What do you want to do with burp suite. (Accessed: Jan. 15, 2024). [Online]. Available: <https://portswigger.net/burp>
- [192] HashiCorp. Vault enterprise replication. (Accessed: Jan. 15, 2024). [Online]. Available: <https://developer.hashicorp.com/vault/docs/enterprise/replication>
- [193] P. Authors. Prometheus monitoring system & time series database. (Accessed: Jan. 15, 2024). [Online]. Available: <https://prometheus.io>
- [194] G. Labs. Grafana: The open observability platform — grafana labs. (Accessed: Jan. 15, 2024). [Online]. Available: <https://grafana.com/>
- [195] L. Nagios Enterprises. The Open Source Standard In Monitoring. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.nagios.org/>
- [196] Z. LLC. The enterprise-class open source network monitoring solution. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.zabbix.com/>

- [197] Cisco. Observability Platform: Cloud Monitoring: Free Trial. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.appdynamics.com/>
- [198] I. Corporation. Overview of IBM Tivoli Monitoring. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.ibm.com/docs/en/iad/7.2.1?topic=introduction-overview-tivoli-monitoring>
- [199] S. Inc. Splunk - Get Started With Splunk. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.splunk.com/>
- [200] Ceph. Architecture - ceph documentation. (Accessed: Jan. 15, 2024). [Online]. Available: <https://docs.ceph.com/en/quincy/architecture/#:~:text=ACephMonitormaintainsa,andreportsbacktomonitors>
- [201] I. Graylog. Graylog: Industry Leading Log Management & SIEM. (Accessed: Jan. 15, 2024). [Online]. Available: <https://graylog.org/>
- [202] R. Gerhards. The rocket-fast Syslog Server - rsyslog. (Accessed: Jan. 15, 2024). [Online]. Available: <https://www.rsyslog.com/>
- [203] T. A. S. Foundation. Apache Log4j™ 2. (Accessed: Jan. 15, 2024). [Online]. Available: <https://logging.apache.org/log4j/2.x/>
- [204] H. Gupta, A. Chaudhary, and A. Kumar, “Identification and analysis of log4j vulnerability,” in *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)*. IEEE, 2022, pp. 1580–1583.

Appendix A

Codes

Code snippets used in *CephVault*:

A.1 Integration of CephVault with CephArmor

CephArmor employs *CephVault*'s methods from the *CephVault Public Interface* to send the user's requests and retrieve the DEK and IV. Below are the integration of encryption and decryption methods with *CephArmor*'s `AES_CBC::_encrypt()` and `AES_CBC::_decrypt()`

```
unsigned int AES_CBC::_encrypt(char *dst, const char *src, unsigned int len)
{
    CephVault::CephVaultPublicInterface pi;
    int ivLengthInBytes=16;
    std::unordered_map<CephVault::Encryption_Key_Storage, std::string>
    encData = pi.getVerified_Encryption_Key_IV_From_CephVault
    (
        CephVault::Encryption_Type::_AES,
        CephVault::Encryption_Mode::_AES_CBC,
        CephVault::Key_Length::_AES_256,
        ivLengthInBytes,
        CephArmor::getNamespace(),
        CephArmor::getObjectname(),
        CephArmor::getPoolName()
    );
}
```

```

std::string encKey = encData[CephVault::Encryption_Key_Storage::KEY];
std::string IV = encData[CephVault::Encryption_Key_Storage::IV];

// Convert string to const unsigned char
const unsigned char *key_from_KMS =
reinterpret_cast<const unsigned char *>(encKey.c_str());
const unsigned char *iv_form_KMS =
reinterpret_cast<const unsigned char *>(IV.c_str());
}

```

```

unsigned int AES_CBC::_decrypt(char *dst, const char *src, unsigned int len)
{
    CephVault::CephVaultPublicInterface pi;
    int ivLengthInBytes=16;

    std::unordered_map<CephVault::Encryption_Key_Storage, std::string>
encData =pi.getVerified_Decryption_Key_IV_From_CephVault
(
    CephVault::Encryption_Type::_AES,
    CephVault::Encryption_Mode::_AES_CBC,
    CephVault::Key_Length::_AES_256,
    ivLengthInBytes,
    CephArmor::getNamespace(),
    CephArmor::getObjectName(),
    CephArmor::getPoolName()
);

    std::string encKey = encData[CephVault::Encryption_Key_Storage::KEY];
    std::string IV = encData[CephVault::Encryption_Key_Storage::IV];

    const unsigned char *key_from_KMS =
reinterpret_cast<const unsigned char *>(encKey.c_str());

    const unsigned char *iv_from_KMS =
reinterpret_cast<const unsigned char *>(IV.c_str());
}

```

A.2 Scalability of CephVault

At present, *CephArmor* supports various modes of AES encryption. In the future, if *CephArmor* supports other encryption schemes, *CephVault* can be extended to support future encryption schemes. Developers need to build similar methods for other encryption schemes and provide proper switch case labels for the particular scheme. Encryption names depicted below are for representation purposes; they need to be replaced with the actual scheme.

```
/**
 * @brief: This function processes user requests for encryption
 * @details: This function accepts CephVault objects and user provided
 *           data, initiating the encryption of the DEK.
 * @warning: Malicious users can create a malformed Object, Pool or Namespace
 *           and try to exploit the database.
 * @returns: Boolean flag to indicate whether
 *           the user requests are processed successfully
 **/
bool KeyRequestProcessor::process_UserRequests_For_Encryption
(
const CephVault::Object_Cluster &objects,
const CephVault::User_Provided_Data &data
)
{
    std::string methodName = "[process_UserRequests_For_Encryption]:";
    std::string message = "";
    bool isProcessSuccessful = false;
    try
    {
        switch (data.encryptionType)
        {
            case Encryption_Type::_AES:
                isProcessSuccessful = objects.getRequestProcessor()->
                    processRequest_AES_Encryption(objects, data);
                break;
            // Future Scope for post-quantum cryptography
            case Encryption_Type::_NTRUEncrypt:
                break;
            // Future Scope for post-quantum cryptography
            case Encryption_Type::_Kyber:
```

```

        break;
        // Future Scope for post-quantum cryptography
        case Encryption_Type::_NewHope:
        break;
        default:
        message = fmt::format
        (
            "{0} Encryption mode is not supported.", methodName
        );

        SPDLOG_CRITICAL(message);
        break;
    }
}
catch (const std::exception &e)
{
    isProcessSuccessful = false;
    message = fmt::format
    (
        "{0} Exception occurred while processing requests for encryption.
        Exception details: {1}",
        methodName,
        e.what()
    );

    SPDLOG_ERROR(message);
    objects.getEmailGenerator()->sendEmail
    (
        message,
        "Error",
        objects.getEmailGenerator()->getEmailCreationDate(),
        objects
    );
}

return isProcessSuccessful;
}

```

Appendix B

CephVault Commands

CephVault supported commands are described below.

- **cephvault -v** or **cephvault -version**: Gets the version of *CephVault*
- **cephvault -h** or **cephvault -help**: Shows help.
- **cephvault -status**: Shows the installation status of prerequisite packages.
- **cephvault -trust**: Creates and shares a private-public key pair for the digital signature and a separate pre-shared symmetric key for encrypting the DEK.
- **cephvault -init**: Creates databases and tables inside the respective databases and assigns defensive flags to databases.
- **cephvault -rotatedek -user U1**: Rotates the DEK based on the user name.
- **cephvault -revokedek -user U1**: Revokes the DEK based on the user name.
- **cephvault -createkek**: Creates the KEK.
- **cephvault -recoverkek**: Recovers the KEK.
- **cephvault -protectkek**: Creates checksum of KEKs.
- **cephvault -verifykek**: Verifies checksum of the KEKs.
- **cephvault -protectbinary**: Creates checksum of *CephVault* binaries.
- **cephvault -verifybinary**: Verifies checksum of *CephVault* binaries.
- **cephvault -destroy**: Permanently destroys keys. Need root access to execute.

Vita

Candidate's full name: Subhabrata Rana

Universities attended:

- Maulana Abul Kalam Azad University of Technology, Bachelor of Technology in Computer Science and Engineering, India, 2010.
- University of New Brunswick, Master of Computer Science, Canada, 2024.

Journal Publications:

- **Subhabrata Rana**, Fatemeh Khoda Parast, Brett Kelly, Yang Wang, and Kenneth B. Kent. “**A Comprehensive Survey of Cryptography Key Management Systems.**”. *Journal of Information Security and Applications* 78 (2023): 103607. <https://doi.org/10.1016/j.jisa.2023.103607>
- **Subhabrata Rana**, Prasanna Iyengar, Vishesh Saxena, Eswari Jayakumar, Rabia Asif, Kenneth B. Kent, and Saqib Hakak. “**IoT Applications in Healthcare: Recent Advances Attacks and Mitigation Strategies.**” *Journal of Network and Computer Applications*, 2024 (preparing to submit)
- **Subhabrata Rana**, Brett Kelly, Yang Wang, and Kenneth B. Kent. “**CephVault: A Secure Key Management System for Ceph.**”. *Journal of Information Security and Applications*, 2024 (preparing to submit)

Posters:

- **Subhabrata Rana**, Fatemeh Khoda Parast, Brett Kelly, and Kenneth B. Kent, **CephVault : A Secure Key Management System (KMS) for Ceph**, 17th Annual Research Exposition of the UNB Faculty of Computer Science, Fredericton, Canada April 14, 2023
- **Subhabrata Rana**, Brett Kelly, and Kenneth B. Kent, **CephVault : A Secure Key Management System (KMS) for Ceph**, 20th Annual International Conference on Privacy, Security & Trust (PST2023), Fredericton, Canada October 16, 2023 (**Selected as the top three finalists**)