EPiC Series in Computing
Volume 82, 2022, Pages 122–130
Proceedings of 37th International Conference on Computers and Their Applications

Performance Study of Software-based Encrypting Data at Rest

Luka Daoud[*], and Hingkwan Huen[†]

Samsung Semiconductor, Inc., San Jose, California, USA
Data Center Device Solutions, Memory Solutions Lab
Luka.Daoud@ieee.org, Kwan.Huen@samsung.com

Abstract

Data security is an increasing concern, not only in cloud storage data centers but also in personal computing and memory devices. It is important to maintain the confidentiality of data at rest against ransom and theft attacks. However, securing data, in runtime, into the drives is associated with performance penalties. In this paper, a study of the performance impact for software-based self-encrypting solid-state drives is presented. This performance evaluation is conducted on the NVMe subsystem which supports encryption and decryption of the user data on an I/O command basis. Additionally, this paper demonstrates the potential of encryption and decryption acceleration for data storage in self-encrypting drives.

1 Introduction

Encrypting data stored on servers has become a common practice in modern cloud and data centers. Encryption hides plaintext information from unauthorized users and makes it inaccessible to unapproved requests. Unprotected data is vulnerable to the theft of sensitive information, which leads to damaging consequences. Encryption protects private information and sensitive data, nevertheless, there is an implication of performance penalty caused by the encryption overhead. Many organizations seek a high level of data security with the balance of performance and power to protect data in the storage system, such as database-level encryption [7].

Most modern operating systems (OS) provide the capability to encrypt their disks in their entirety, where a key or passphrase has to be entered for the encrypted disk to boot up the system. Unlike full-disk encryption, file systems can protect data at rest. The OS divides physical disks into one or more file systems, which typically allows administrators to encrypt selected file systems, or even selected folders within the file system. Software-based and hardware-based disk encryption are two types of data encryption at rest, providing security to the system and confidentiality to the data. However, the additional encryption process does bring performance and power overhead. Disk encryption decreases the system performance since encryption or

[*]IEEE Member, PhD in Electrical and Computer Engineering.
[†]Principal Engineer at Samsung Semiconductor, Inc.

decryption is needed every time data is written to or read from the disk. As a result, encrypted systems consume more power and system resources than non-encrypted ones. It can potentially impact data bandwidth seen by the application as well.

There are several techniques to implement data encryption at rest on modern operating systems. These schemes are tightly associated with the OS storage stack, where encryption can be implemented at any layer of the storage stack. For encrypting data in the storage, the encryption method is applied either at the block or at the file level. Implementing the encryption on the upper stack level, such as the application layer, provides more flexibility in the encryption scheme. However, this requires application designers the knowledge of specific cryptographic algorithms. Additionally, it does not leverage the benefit of OS-level caching. For example, in the case when an application consumes data, it has to decrypt the data every time, adding load to the CPU.

In this paper, we investigate the performance of the most common software-based disk encryption using Linux dm-crypt [6]. Additionally, we explore the potential of hardware-assisted inline crypto processing embedded in the drive. The rest of this paper is organized as follows: Section 2 presents the encryption scheme applied to storage systems. Section 3 demonstrates data encryption at rest. Section 4 provides the performance evaluation of software-based self-encrypting drives. Section 5 discusses the performance study of the software-based self-encrypting and introduces suggestions for future work. Finally, Section 6 concludes the presented evaluation.

2 Data Encryption Scheme

Considering cryptography and securing data techniques, there are several encryption algorithms [8], such as AES [10], DES [9], and RSA [11]. Among the existing security protocols, AES provides better performance without sacrificing the system security [2], and with different key sizes [5]. It is fast compared to other algorithms, such as DES. AES represents a fundamental building block of many security protocols to ensure data confidentiality in various applications ranging from data servers to low-power embedded systems. The AES algorithm consists of multiple rounds of encryption. Each round has three main layers of ciphering techniques to provide data confusion and diffusion through nonlinear transformation.

There are different modes of the encryption operation, which describe how a cipher's single-block operation repeatedly is applied for data larger than a block. These modes of operation vary for the security features to include or combine confidentially and authenticity, such as ECB, CBC, CTR, and XTS modes. Self-encrypting drives support a full-security process for end-to-end data protection using the XTS-AES mode. As the work presented in this paper focuses on storage-level data encryption, we briefly present the XTS-AES encryption mode.

NIST released a special publication on block cipher modes of operation recommending the XTS-AES mode for confidentiality on storage devices [3]. XTS stands for XEX Tweakable Block Ciphertext Stealing. It can encrypt and decrypt sequences of arbitrary length of data blocks. By the IEEE Standard, XTS operates on two cipherkeys: the encryption key, which is used for performing the AES block encryption, and the tweakable key, which is used for encrypting what is known as a "Tweak Value". Figure 1 shows the block diagram of the XTS-AES encryption mode for data storage. As we can see from the figure, the tweak value is encrypted by an AES encryption engine with *key_1* and additionally modified with a Galois polynomial function (GF). The output is then XORed with both the plaintext and the ciphertext of each block that is encrypted with *key_2*. This process provides further diffusion and ensures that blocks of identical data will not produce identical cipher text. This achieves the goal of each block producing unique cipher text given identical plain text. On the other hand, decryption of the data is accomplished by reversing the encryption process, as described in Figure 2.

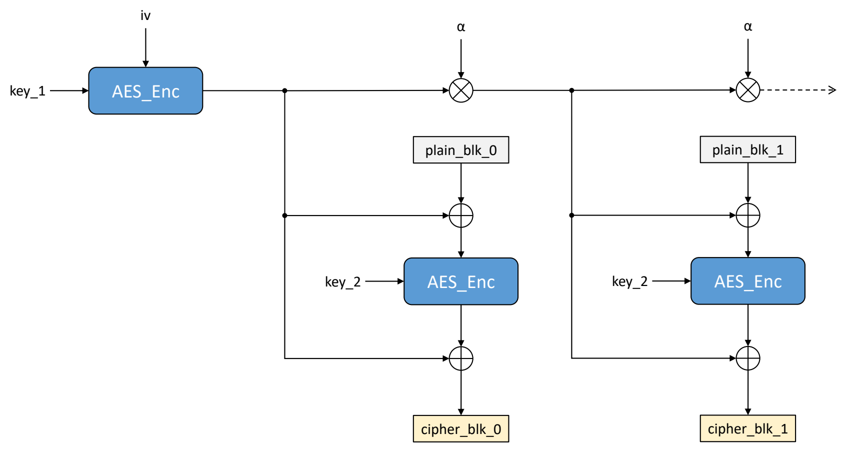

Figure 1: Encryption diagram of the XTS-AES mode operation.

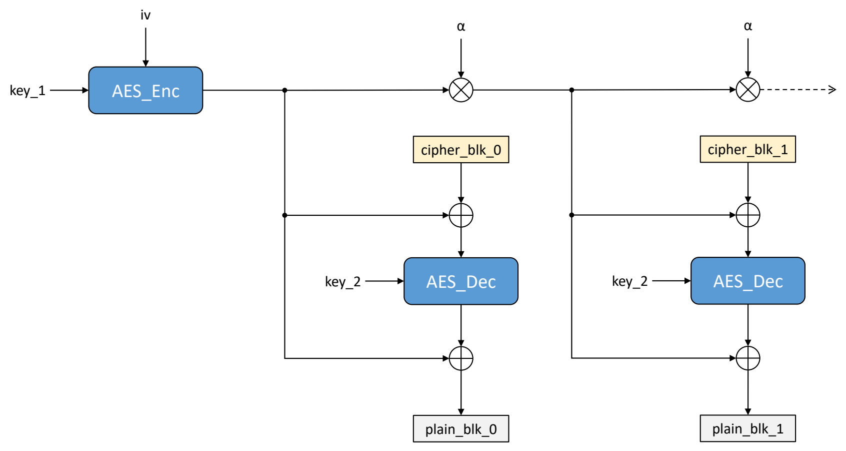

Figure 2: Decryption diagram of the XTS-AES mode operation.

3 Data at Rest Encryption

Storage encryption enables enterprises to cipher data at the storage subsystem. The encryption may be applied on the file level or at the block level to encrypt files, directories, and storage blocks. Protecting data at reset can occur at the software or hardware level before being stored in the disk. Software-based encryption is typically cheap to implement without additional hardware. However, it shares the processing resources of the computer, which potentially slows down the entire system as data is encrypted or decrypted. For hardware-based encryption, a separate processor is dedicated to the encryption process. Therefore, hardware-based encryption is safer than software-based one. However, hardware-based encrypted storage is much more expensive compared to software-based one.

Software disk encryption in the Linux kernel is open and has been examined by many security professionals across the world. Linux implements transparent disk encryption via dm-crypt [6], which is a subsystem that is part of the device-mapper framework in the Linux kernel. The device-mapper allows the mapping of physical block devices onto higher-level virtual block devices in the system. In this way, many disk-level features can be implemented in software within the device mapper. Logical volume manager (LVM), software RAIDs, dm-crypt, and offers additional features such as file system snapshots are common usage of device-mapper, along with dm-crypt serving the purpose of transparent disk encryption. The device-mapper allows pre/post-process IO requests as they travel between the file system and the underlying block device. The dm-crypt encrypts the ”write” IO request before sending them further down the stack to the physical block devices and decrypts ”read” IO requests before sending them up to the file system.

In traditional self-encrypting drives, media encryption keys are generated or held by the storage device and the encryption scope is based on contiguous LBA ranges per namespace basis [4]. This is executed at the interface speeds using a small number of keys held in NVM by the storage device. However, NVMe and TCG are demonstrating a Key Per IO (KPIO) [1] proposal to define a new KPIO security subsystem class. KPIO will allow large numbers of encryption keys to be managed and securely downloaded into the NVM subsystem. Encryption of user data occurs on a per-command basis, where each command may use a different key.

This provides a finer granularity of data encryption that enables a granular encryption scheme. The KPIO proposal supports secure data erasure when data is spread over disks or mixed with other data needing to be preserved. It allows encryption keys to be managed and securely downloaded by the host to the NVM subsystem. Therefore, in Section 4, we evaluate the impact of software-based encryption and decryption on the system performance for securing data in the hard drive. The non-encrypted case presented is expected to be a good representation of KPIO enabled device capable of encrypting and decrypting data at interface speed.

4 Performance Evaluation

This section exhibits the performance study of the software-based self-encrypting drives, excluding the associated overhead of the storage system.

4.1 Simulation Environment

To precisely evaluate the write and read bandwidth of secured and non-secured drives associated with software-based encryption, we need to eliminate the overhead introduced by the storage hardware. In this simulation environment, we created two partitions on the NVMe drive: one encrypted; the other non-encrypted. The encrypted partition is mapped with dm-crypt with 512-bit AES-XTS. The non-encrypted partition is used as-is. For the encrypted partition, data will be encrypted and decrypted on the way by dm-crypt. The data is stored in the ciphertext in case of a ”write” request and decrypted in case of a ”read” request. Meanwhile, for the non-encrypted partition, no additional data processing is needed aside from normal disk IO. We compare the performance of encrypted vs non-encrypted partition by measuring the bandwidth, CPU utilization, and IO latency using the Flexible I/O (FIO) benchmark tool. The simulation was evaluated with different IO queue depths, block sizes, and the number of jobs that are best representing the IO characteristics of the drive under test, which is a Samsung PM1733 NVMe SSD. The measurements were collected on a Dell R740xd 2U server with Intel Xeon 6152 CPU running Ubuntu 18.04 LTS with v5.4.0 kernel.

4.2 Evaluation Results

In the first experiment, we ran the FIO test for random read, write, and random 70% read & 30% write. The experimental run was executed on the non-encrypted and encrypted partitions for IO depth of 64 and block size of 4 KB. The experiment was repeated for IO depth of 8 and block size of 128 KB for sequential read, sequential write, and sequential read and write. Figure 3 summarizes the throughput result of reading and writing to the drive. As expected, the security process on the secured partition has a significant impact on overall throughput. The throughput of the secured partition is 33% lower in the case of random read & write ("rand_rw"), and 50% lower as in the case of random write ("rand_write"). Similarly in case of sequential I/O operations, "seq_read", "seq_write", and "seq_rw".

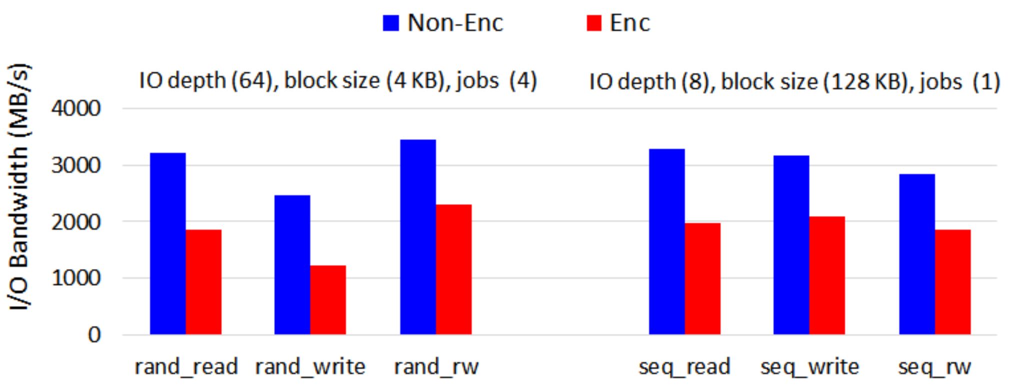

Figure 3: I/O read, write bandwidth for different queue depth and block sizes.

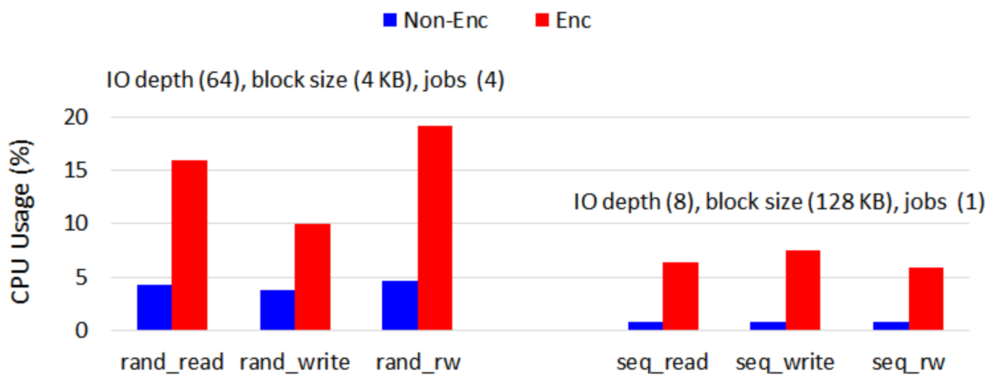

Figure 4: CPU usage for different queue depth and block sizes.

In the case of non-encrypted partition, the I/O bandwidth of "read", "write", and "read & write", for sequentially and randomly cases, nearly reaches the maximum speed value of the I/O system. However, for the case of encrypted partitions, the system is occupied with encryption and decryption processing before writing and reading operations, respectively, on the drive.

Throughout the experiments, the CPU usage of the system is also collected. Figure 4 presents the CPU usage while running the same FIO tests. In the encrypted case, it is noticeable that more CPU resources are used in order to process the encryption and decryption of data stored in the drive. The figure shows writing to or reading from the encrypted drive consumes up to 9.5x higher CPU usage. For the non-encrypted partition, the system consumes little CPU cycles to set up the DMA and service the IO interrupt. It stays mostly idle for the rest of the time. However, for the encrypted partition, the difference in CPU utilization is caused by data encryption decryption processes.

The second experiment was developed to measure the amount of time it takes to read or write a single block. In this assessment, the average latency of the FIO test was measured with a single job, single IO depth, and 4 KB block size. Table 1 presents the IO latency between the submission to the kernel and completing the request. The IO latency for the encrypted partition reaches up to 3.29x higher than the non-encrypted one.

Table 1: IO latency of the FIO test on encrypted and non-encrypted drives for a single job and IO depth

FIO Test	**Non-Encrypted Drive (us)**	**Encrypted Drive (us)**	**Latency ratio**
rand_read	87	110	1.26
rand_write	21	69	3.29
rand_rw	75	94	1.25

For further investigation, multiple experiments were conducted to maximize the IO bandwidth and explore the resources usage of the system at that moment. The FIO tests were run for random and sequential read, write, and read & write for IO depth of 64 and block size of 128 KB. The tests were repeated for multiple job numbers. Figures 5 and 6 show the sequential read and sequential write IO bandwidth, and the CPU usage for different job numbers. Because of increasing the IO depth to 64 and the block size to 128 KB, the IO bandwidth reaches its peak value. As a result of multiple IO requests, cryptographic processing of decryption and encryption is performed by the kernel crypto device-mapper, which keeps the system CPU fully utilized to serve as many I/O requests. In the case of encrypted partition tests, the CPU usage increases with minimum ∼35% to ∼95% for the number of jobs 1 and 64, respectively, whereas negligible CPU usages, 1∼2.5%, is consumed for the non-encrypted partition tests.

Similarly, Figures 7 and 8 express alike behavior, where the CPU usage for the FIO tests on encrypted partition increases with a ratio 30∼40x compared to the the non-encrypted ones.

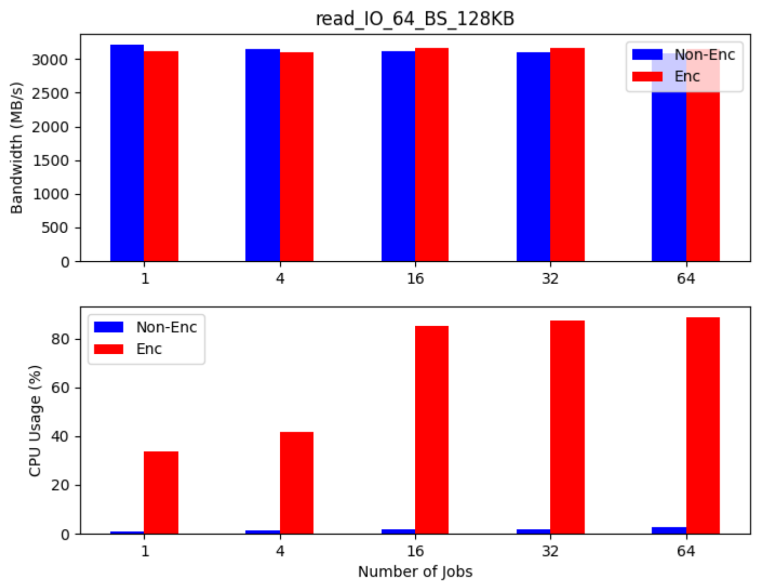

Figure 5: Sequential read IO Bandwidth and CPU usage for various job-numbers.

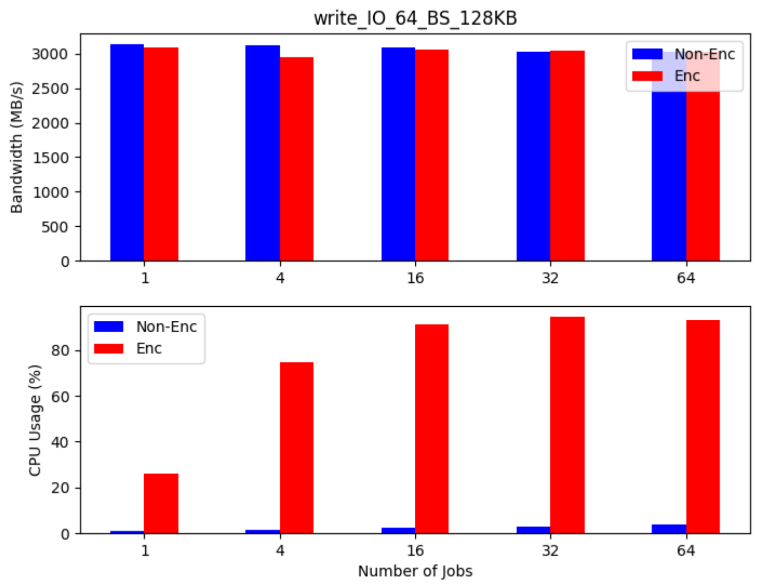

Figure 6: Sequential write IO Bandwidth and CPU usage for various job-numbers.

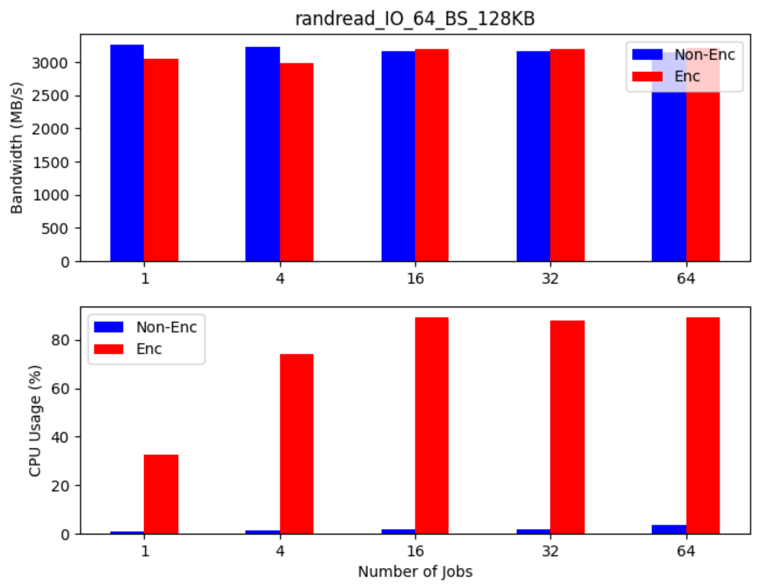

Figure 7: Random read IO Bandwidth and CPU usage for various job-numbers.

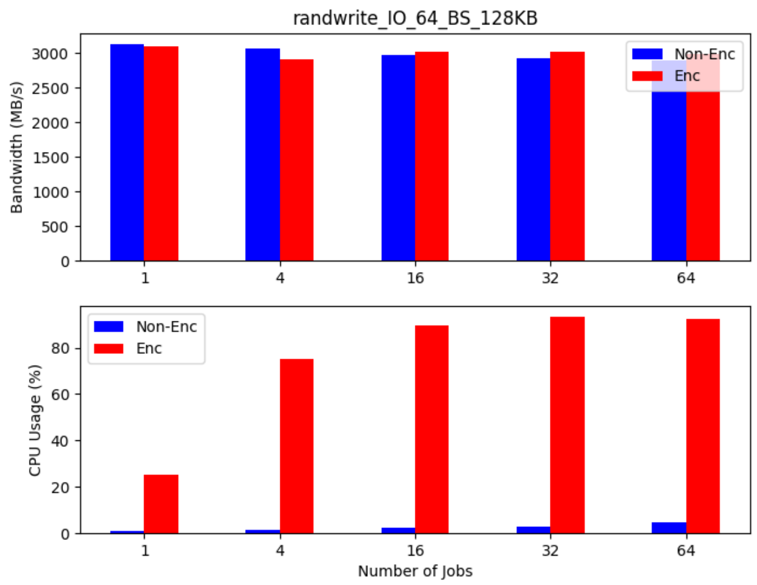

Figure 8: Random write IO Bandwidth and CPU usage for various job-numbers.

5 Discussion and Future Work

The conducted experiments showed the impact of the software-based encryption and decryption processes on the system performance and CPU usages for securing data at rest. The non-encrypted case can be considered as a KPIO enabled device capable of inline encryption and decryption. This demonstrates the potential of hardware acceleration of the security processes on the device. Therefore, the maximum bandwidth performance can be achieved with minimum CPU usage for hardware-based KPIO devices. The experiments were executed for one key per encrypted partition on a single SSD drive with 4x PCIe v3.0, which saturates the IO bandwidth to 3∼4 GB/s. As future work, the system performance is evaluated on PCIe v4.0 for multiple SSDs in parallel to maximize the disk IO with multiple key and dm-crypt partitions.

6 Conclusion

Encryption can provide strong security for data at rest, but it accompanies computing processing which may result in performance degradation to the whole system. This paper studied the impact of software-based data encryption and decryption on system performance. The read and write IO bandwidth on encrypted and non-encrypted devices was measured to demonstrate the impact of the encryption process to securely store data on the drive. The maximum I/O bandwidth is restricted to the encryption rate and the system CPU usage was increased to process the encryption and decryption before writing to or reading from the encrypted drive. In this performance evaluation, for multiple I/O requests and multiple job numbers, the I/O bandwidth reaches its peak, 3∼4 GB/s with up to 95% CPU usage.

References

[1] Sridhar Balasubramanian and Frederick Knight. ”Key Per IO Security Subsystem Class for NVM Express Storage Devices, 2020.

[2] Luka Daoud, Fady Hussein, and Nader Rafla. Optimization of advanced encryption standard (aes) using vivado high level synthesis (hls). In *Proceedings of the 34th International Conference on Computers and Their Applications*, volume 58 of *EPiC Series in Computing*, pages 36–44. EasyChair, 2019.

[3] Morris J Dworkin et al. Recommendation for block cipher modes of operation: The xts-aes mode for confidentiality on storage devices. 2010.

[4] Trusted Computing Group. ”TCG Storage, Opal, and NVMe”, August 2015.

[5] L. Daoud, F. Hussein, and N. Rafla. High-level synthesis optimization of aes-128/192/256 encryption algorithms. *International Journal of Computers and Their Applications*, 29:129–136, 2019.

[6] Linux man page. ”cryptsetup(8) - linux man page”, 2021.

[7] Ulf T Mattsson. Database encryption-how to balance security with performance. *Available at SSRN 670561*, 2005.

[8] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[9] NIST FIPS Pub. Data Encryption Standard (DES). *Federal Information Processing Standards Publication*, 112, 1999.

[10] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal information processing standards publication*, 197(441):0311, 2001.

[11] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.