FeSSD: A Fast Encrypted SSD Employing On-Chip Access-Control Memory

Junghee Lee, Kalidas Ganesh, Hyuk-Jun Lee, and Youngjae Kim

Abstract—Cryptography is one of the most popular methods for protecting data stored in storage devices such as solid-state drives (SSDs). To maintain integrity of data, one of the popular techniques is that all incoming data are encrypted before they are stored, however, in this technique, the encryption overhead is non-negligible and it can increase I/O service time. In order to mitigate the negative performance impact caused by the data encryption, a write buffer can be used to hide the long latency by encryption. Using the write buffer, incoming unencrypted data can be immediately returned as soon as they are written in the buffer. They will get encrypted and synchronized with flash memory. However, if the write buffer itself is not encrypted, unencrypted secret data might leak through this insecure write buffer. On the other hand, if the entire write buffer is fully encrypted, it incurs significant performance overhead. To address this problem, we propose an on-chip access control memory (ACM) and presents a fast encrypted SSD, called FeSSD that implements a secure write buffering mechanism using the ACM. The ACM does not require a memory-level full encryption mechanism, thus not only solving the unencrypted data leaking problem, but also offering relatively fast I/O service. Our simulation results show that the I/O response time of FeSSD can be improved by up to 56 percent over a baseline where encrypted data are stored in the normal write buffer.

Index Terms—Solid-state drive (SSD), security, on-chip memory, encryption

1 Introduction

Cryptography is often employed to protect a solid-state drive (SSD) from data breach. Sensitive and private information stored in mobile devices such as smartphones and laptops can be at risk of being exposed when they are *physically* lost or stolen. User passwords or screen locks cannot prevent direct access to storage medium bypassing the operating system. Cryptography is the strongest countermeasure to protect sensitive data [1]. However, encryption can incur non-negligible overheads on response time. Many existing works have investigated the problems of reducing the data encryption overhead by using hardware accelerators [2] or light-weight algorithms [3], [4], [5], [6]. These techniques can help reduce this performance overhead of encryption to a certain extent, however, as the overhead is incurred during every memory access operation, even small overheads cannot be negligible.

In order to further reduce the encryption overhead, an approach that utilizes a small write buffer can be employed. It allows request operations to be completed before they are encrypted, and data may be lazily encrypted on the buffer using the SSD controller. The encrypted data is then synchronized with flash memory later. This asynchronous write buffering approach could improve the performance of encrypted SSDs. However, it might happen that the data on the write buffer, yet encrypted becomes insecure. The longer the data is present in the write buffer, the more insecure the SSD is. On the other hand, a synchronous encryption approach on the write buffer that allows data to be encrypted as soon as it arrives in the write buffer can guarantee that data is secure in the SSD. However, this approach will significantly increase encryption overhead again. To address this issue, we propose to employ a "secure" on-chip memory as a write buffer.

In this paper, we propose a fast encrypted SSD that employs an *access-control memory* (ACM) as a "secure" and "fast" write buffer, called FeSSD. The ACM is implemented as on-chip non-volatile memory that allows memory access only if an access code is matched, which cannot be acquired by *compromised* firmware. ACM is secure against compromised firmware (details of our threat model are presented in Section 2), and is fast because it does not need encryption to protect data. In FeSSD, all writes will not be encrypted when they are stored in the write buffer. They can be immediately committed after they are written on it. Then later they can be encrypted and transferred to flash memory in background. As a result, FeSSD markedly reduces the encryption overhead without employing expensive hardware accelerators. It can also take full advantage of temporal locality presented in the write buffer because it does not have to encrypt the same data again.

This paper makes the following specific contributions: First, we introduced the access-control memory for fast encrypted SSD. It integrates an access-control mechanism with memory, where the access-control mechanism cannot be bypassed by compromised firmware. Second, to make sure FeSSD is secure, we identified a threat model that defines when it is secure and analyzed potential vulnerabilities and their countermeasures.

2 Threat Model

Fig. 1 shows the architectural design of a typical SSD comprised of multiple components: a micro-controller where firmware is running, an on-chip non-volatile memory inside the micro-controller employed as a write buffer, a read-only memory (ROM) for storing firmware, a working memory (e.g., DRAM), and an array of flash memories to store actual data.

Our threat model assumes the firmware of an SSD can be compromised. An example attack scenario can be that of reassembling components of a stolen SSD. The adversary may detach flash memories from the board of the SSD and attach them to another equipment to read or modify the data. If flash memories are not encrypted, any data stored in them could be revealed.

Under this threat model, any unencrypted on-chip memory is not secure because they can be accessed by compromised firmware. If it is synchronously encrypted, the encryption overhead is unavoidable. The uncompromised firmware may have an access control mechanism to protect the on-chip memory, however if the firmware is compromised, the adversary can bypass the access control mechanism.

In addition, we assume that the power supply to an SSD is disconnected at least once before any attack is committed. In other words, all data in volatile memories and registers is erased before an attack. Since encryption of an SSD usually targets to prevent data breach from a stolen SSD, we assume all volatile data is erased before an attack.

In summary, our threat model assumes the firmware of an SSD can be compromised. However, it is assumed that the host is secure because the security of a host is out of scope of this paper. There are two more assumptions: (1) on-chip memory cannot be accessed without help of firmware and (2) power goes off at least once before an attack is committed. These two assumptions are justified by Section 3.3.

3 Design of FeSSD: A Fast Secure SSD with Access-Control Memory

In this section, the proposed access-control memory is introduced and how it can be used for building the FeSSD is presented.

- *J. Lee and K. Ganesh are with the University of Texas at San Antonio, San Antonio, TX 78249. E-mail: {junghee.lee, dyk567}@my.utsa.edu.*
- *H.-J. Lee and Y. Kim are with Sogang University, Seoul 121-742, South Korea. E-mail: {hyukjun1, youkim}@sogang.ac.kr.*

Manuscript received 19 July 2016; revised 11 Jan. 2017; accepted 4 Feb. 2017. Date of publication 12 Feb. 2017; date of current version 5 Jan. 2018.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2017.2667639

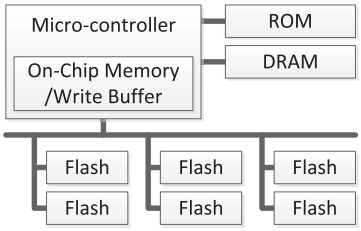


Fig. 1. An architectural overview of a flash-based solid-state drive (SSD) with various hardware components such as controller, on-chip memory/write buffer, and flash memories [7].

3.1 Access-Control Memory

The key idea of the access-control memory is to integrate an access-control mechanism with on-chip memory. If the memory is an access-control memory, the access-control mechanism cannot be bypassed nor disabled even if firmware is compromised because the access-control mechanism is not under control from the firmware. A well-known hardware-based access-control mechanism is memory management unit (MMU) and memory protection unit (MPU) which can protect memory from unauthorized accesses. However, MMU and MPU do not have countermeasures to compromised firmware. The firmware, if compromised, can simply disable them.

The purpose of the access-control memory is to protect sensitive data stored in *non-volatile* memory from compromised firmware. Volatile memory is not the target of the access-control memory because it will be erased when power goes off. Fig. 2 depicts a block diagram of the access-control memory. Any type of on-chip non-volatile memory can be used for the access-control memory as long as it is compatible with an existing CMOS technology. The access-control memory grants access only if the access code is matched. In other words, it returns valid data and accepts data to be written, only when the access code is approved. In fact, this is exactly what encryption techniques intend to do. Encryption techniques allow access only when encryption key is accompanied. Else, it will be never be granted. The access-control memory performs the same function as encryption technique while it does not employ data encryption.

The access code in the access-control memory is specific to an SSD and it is given by the manufacturer. The initial access code can be provided in a user's manual. When installing an SSD, the user should input the access code. Then the access code can be maintained by operating systems on the host. The access-control memory allows a user to update the access code. The access code in the access-control memory is mapped to a predefined address and accessible through the memory interface. It allows only a write operation but not a read operation, so that compromised firmware can never read it. If firmware tries to read the address mapped to the access code, the access-control memory only returns predefined reset values. If firmware writes data to that address, the access-control memory accepts the write request only if the access code matches. Since the code update is granted only if the current access code is matched, any unauthorized update is prohibited.

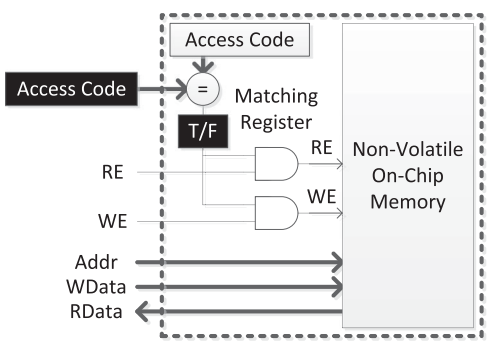


Fig. 2. A block diagram of an access-control memory.

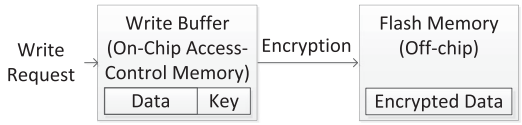


Fig. 3. A data path when an access-control memory is employed as a write buffer.

While booting, the firmware receives an access code from the host and stores it in a *volatile* register. Since it is volatile, it will be erased when power is switched off. When a hardware-based full disk encryption is employed, the key is usually stored in the device (SSD) [1]. However, under our threat model where the firmware can be compromised, the access code cannot be permanently stored in an SSD because it could be revealed by compromised firmware. If the access code is revealed for any reason, the access code can be changed by the aforementioned method.

The access code is sent to the access-control memory. The access-control memory compares it against its internal access code to check if they both match. The matching result is stored in a temporal *volatile* register, which is called *matching register*. Subsequent accesses to the memory refer to this register to check if the access code is matched or not. Recall that we assume all volatile data be erased before an attack is committed as mentioned in Section 2. Therefore, the access code in the volatile register, which is accessible by compromised firmware is erased before an attack. The matching register in the access-control memory is also reset to false when power is off.

The matching register is employed to reduce the critical path delay. If it is not employed and the access code is compared for every access, the comparison logic affects the critical path. Since the access code should be at least 128-bit long to prevent exhaustive search, the comparison logic may incur non-negligible delay to the critical path delay. To minimize the impact on the critical path delay, the matching register is added.

3.2 Employing Access-Control Memory as Write Buffer

The proposed encryption technique for an SSD falls in to the category of full-disk encryption, but it is in-between conventional software and hardware techniques [1]. Conventional hardware techniques are referred to as self-encrypting drives where encryption and key management are handled by an SSD. In contrast, they can be handled by the operating system in the software techniques. In our proposed technique, encryption is handled by an SSD, but the encryption key is managed by software (operating system) on the host that connects to the SSD. The encryption key can be given at booting time or different encryption keys can be given along with data. Note that the encryption key is different from the access code. The former is for data encryption and the latter is for controlling accesses to the access-control memory. Both should be managed by the host operating systems.

A typical data path for FeSSD is shown in Fig. 3 when the access-control memory is employed as a write buffer. When an incoming write request arrives, it is stored in the access-control memory and immediately committed. While the request is stored, the key to encrypt data should also be stored. Note that even though the key is stored in the access-control memory, it is safe under our threat model. In the background, the firmware encrypts the stored data and transfers the encrypted data to a flash memory.

Since the access code is given by the host, if the host is compromised or physically stolen, the access-control memory can be accessed by the adversary. In fact, encryption techniques have the same issue when the host is compromised. Since an SSD cannot determine whether the host is compromised or not, it will allow the compromised host to access data in it. We regard logical and physical security issues of a host as out of the scope in this paper.

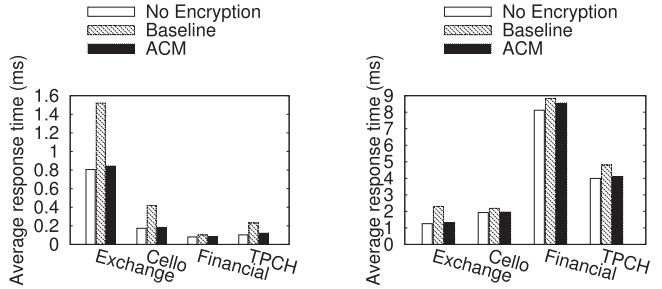
TABLE 1
SSD Model Parameters

SSD configuration	
Total capacity	8 GB
Reserved free blocks	15 %
Minimum free blocks	5 %
Cleaning policy	Greedy
Flash chip packages	4
Planes per package	4
Blocks per plane	512
Pages per block	64
Page size	4 KB

Latency	
Page read	0.025 ms / page
Page write	0.200 ms / page
Block erase	1.5 ms / page
Encryption	1,720 ns / sector

Write Buffer (STT-RAM [11])	
Size	1 MB & 32 KB
Read latency	1,870 ns / sector
Write latency	3,240 ns / sector

(1 sector = 512 bytes).



(a) 1 MB write buffer (b) 32 KB write buffer

Fig. 4. Performance comparisons of response time for different write buffer cache size.

3.3 Security Analysis

The basic functionality of the access-control memory is the same with that of encryption techniques: it allows access of those who know the access code (key) and disallows those who do not. The difference is that encryption techniques can hide contents from an adversary while the access-control memory prevents accesses only at the interface. Thus, it has potential vulnerabilities that are not present in encryption techniques. This section is focused on discussing them.

Penetrative Attack. If an adversary is as capable as chip manufacturers, e.g., having expensive equipment and in-depth knowledge on fabrication, it is possible to extract information from the access-control memory without firmware. An adversary may de-package the chip, remove top metal layers and observe the structure of the memory through an optical microscope. However, the adversary will have to examine thousands or even millions of memory cells to see the stored values. We believe it is practically impossible to assume such highly invasive attacks by the adversary in most cases. Especially, the write buffer is supposed to store a small piece of data for a short period of time. There is no guarantee that sensitive data, which adversaries are looking for, can be found. From adversary's perspective, this means a poor return-on-investment (ROI). This is the rationale behind applying the access-control memory only to a write buffer. If such a strong adversary should be assumed, the access-control memory is not suitable in that environment. However, we may consider employing existing techniques to prevent penetrative attacks [8]. We will explore their applicability in our future work.

Hot Plug Attack. If an SSD is kept alive, an adversary may access the access-control memory because the access code in the volatile register has not been erased. This is possible only if the power cable of an SSD is kept connected to a host while its bus interface (e.g., SATA or SCSI) is reconnected to the adversary's machine. This is a common issue to self-encrypting drives. It is a type of hot plug attacks and can be prevented by introducing sensitivity to the bus interface [1]. When the bus interface is disconnected, an SSD should be locked so that it cannot be reconnected to other machines.

Power Monitoring Attack. By analyzing power consumed by the micro-controller chip, an adversary may acquire the access code. However, since the comparison logic of the access codes is very small compared to the entire chip, the power profile of the logic is within noise margin. Therefore, a power monitoring attack is infeasible for the access-control memory.

Fault Injection Attack. The matching register could be flipped by a fault injection attack. If the matching register is flipped, an adversary may access the access-control memory even if the access code is not matched. However, the matching register is recovered immediately at the following clock cycle as long as the access code remains unmatched. Since faults cannot be injected persistently, the fault in the matching register is unlikely to last for more than one clock cycle. If one wants to eliminate this potential vulnerability, one can employ redundant comparison logic and matching resister [9]. Although repetition is considered the most expensive countermeasure for fault injection attacks [9], it does not incur much overhead for the access-control memory because the comparison logic and matching register are very small.

4 Evaluation

4.1 Experimental Setup

In order to study the performance implications of the access-control memory, we enhanced the DiskSim augmented SSD simulator developed by Microsoft Research [10] to implement FeSSD, an encrypted SSD employing an on-chip access-control memory as write buffer. For evaluation, we assume STT-RAM for the access-control memory whose read and write latencies are 1,870 ns and 3,240 ns per sector, respectively [11]. All the parameters used for SSD simulation are summarized in Table 1. For encryption and decryption, we consider a recent hardware accelerator of advanced encryption standard (AES) [2] where its encrypt/decrypt throughput is 2.33 Gb/s. Namely, it is 1,720 ns per sector for both encryption and decryption.

To demonstrate the performance improvement by the access-control memory, we compared two implementations for secure SSDs.

- *Encrypted Write Buffering:* All data are encrypted before they are stored in the write buffer. As all data in the write buffer are encrypted, they are all secure under our threat model. The encrypted data are transferred to flash memory in background.
- *Access-Control Memory:* An on-chip access-control memory is employed as a write buffer. Since the access-control memory is secure under our threat model even though it does not employ encryption, data stored in the write buffer will not be encrypted. The data store in the write buffer will be lazily encrypted and transferred to flash memory in background.

To study the efficacy of the access-control memory in FeSSD, we used real enterprise-scale I/O workloads; Microsoft (MSR) Exchange Server, Cello, Financial, and TPC-H traces.

4.2 Results

4.2.1 Performance Analysis

Fig. 4 compares the performance of encrypted write buffering of an SSD (Baseline) and the access-control memory of FeSSD. As the write buffer size can affect the performance, we used two different cache sizes, which are big and small. Fig. 4a shows when the write buffer size is big (1 MB). The results show that ACM improves the average response time by 44.35, 56.20, 15.95, and 46.97 percent for Exchange, Cello, Financial, and TPC-H, respectively when compared to the baseline. On average, the performance improvement is 40.87 percent. This result clearly shows that the performance of an encrypted SSD can be significantly

improved by using the access-control memory. This is because even a small encryption delay can be accumulated when the I/O is intensive. When compared to insecure SSD without encryption, the overhead of the proposed implementation is only 8.90 percent on average.

Fig. 4b shows when the write buffer size is small (32 KB). As seen from the results, we observe that if the size of the write buffer decreases, the performance improvement by the access-control memory also decreases because it cannot hide the encryption overhead. A small write buffer is more likely to quickly fill than a large buffer. If a write request arrives when the write buffer is full, it should be pending until the buffer becomes available. To flush requests in the buffer, the data should be encrypted and transferred to flash memory. Thus, the encryption delay cannot be hidden when the buffer is full. However, as shown in Fig. 4b, even when a 32 KB buffer is employed, the performance improvement is still as significant as 17.28 percent on average compared to the baseline.

4.2.2 Hardware Cost Analysis

The hardware cost of the additional logic is analyzed with Cadence Encounter [12]. The additional logic includes the access code comparison logic, matching register, and additional AND gates. Using Nangate 45 nm technology, it was estimated that the total number of additional gates is 413, and the additional power consumption is 98.41 μW. Compared to state-of-the-art AES accelerators which require at least 21,337 gates [2], hardware cost for ACM is negligible.

5 Conclusion

This paper proposes an access-control memory, which allows access only if the access code is matched. It is as secure as an encrypted memory but does not incur encryption overhead. The proposed access-control memory can be an alternative to encryption techniques unless a strong adversary is assumed who is as capable as chip manufacturers. The experimental result demonstrates that the performance of an encrypted SSD can be improved by up to 56 percent by employing the access-control memory as a write buffer.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MISP) (No. 2015R1C1A1A0152105). This research also used resources of The University of Texas at San Antonio, San Antonio, TX. Youngjae Kim is the corresponding author.

References

[1] T. Müller and F. C. Freiling, "A systematic assessment of the security of full disk encryption," *IEEE Trans. Depend. Secure Comput.*, vol. 12, no. 5, pp. 491–503, Sep./Oct. 2015.
[2] X. Zhang, H. Li, S. Yang, and S. Han, "On a high-performance and balanced method of hardware implementation for AES," in *Proc. IEEE 7th Int. Conf. Softw. Secur. Rel. Companion*, 2013, pp. 16–20.
[3] S. Chhabra and Y. Solihin, "i-NVMM: A secure non-volatile main memory system with incremental encryption," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 177–188.
[4] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-efficient encryption for non-volatile memories," in *Proc. 20th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2015, pp. 33–44.
[5] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers," in *Proc. 21st Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2016, pp. 263–276.
[6] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2006, pp. 179–190.
[7] Mtron, "2.5-Inch Mtron SATA solid state drive-MSP 7000," 2008. [Online]. Available: http://www.mtron.net
[8] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Proc. 14th Int. Conf. Cryptographic Hardware Embedded Syst.*, 2012, pp. 23–40.
[9] D. Karaklaji, J. M. Schmidt, and I. Verbauwhede, "Hardware designer's guide to fault attacks," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 12, pp. 2295–2306, Dec. 2013.
[10] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70.
[11] M. R. Jokar, M. Arjomand, and H. Sarbazi-Azad, "Sequoia: A high-endurance NVM-based cache architecture," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 3, pp. 954–967, Mar. 2016.
[12] Cadence, "Encounter," [Online]. Available: www.cadence.com, 2005.