A Content-Based Ransomware Detection and Backup Solid-State Drive for Ransomware Defense

Donghyun Min, Yungwoo Ko, Ryan Walker, Junghee Lee, and Youngjae Kim

***Abstract*—Ransomware is a growing concern in business and government because it causes immediate financial damages or loss of important data. There is a way to detect and block ransomware in advance, but evolved ransomware can still attack while avoiding detection. Another alternative is to back up the original data. However, existing backup solutions can be under the control of ransomware and backup copies can be destroyed by ransomware. Moreover, backup methods incur storage and performance overhead. In this article, we propose AMOEBA, a device-level backup solution that does not require additional storage for backup. AMOEBA is armed with: 1) a hardware accelerator to run content-based detection algorithms for ransomware detection at high speed and 2) a fine-grained backup control mechanism to minimize space overhead for data backup. For evaluations, we not only implemented AMOEBA using the Microsoft solid-state drive (SSD) simulator but also prototyped it on the OpenSSD-platform. Our extensive evaluations with real ransomware workloads show that AMOEBA has high ransomware detection accuracy with negligible performance overhead.**

***Index Terms*—Ransomware attack, solid-state drive (SSD), storage security, storage system.**

I. INTRODUCTION

RANSOMWARE is a type of malware that takes the user's data files as hostage by encrypting them until the victim user pays a ransom. Ransomware is one of the most concerning cybersecurity threats not only for individuals but also enterprise environments [1], [2]. Existing techniques detect and prevent ransomware by identifying known behaviors of ransomware such as frequent access to cryptographic libraries and receiving encryption keys from a remote server [3]–[6]. When ransomware accesses a cryptographic library, the user's encryption key can be hooked, and it can be used for decryption [7]. Ransomware can also be identified by static analysis of binary code if a specific cryptographic library is used [8]. However, it is difficult to detect ransomware when its behavior has not been observed. Furthermore, malware can use packing techniques to hide themselves and encryption keys [3]. One of the guaranteed protections against ransomware is frequent data backup on external backup storage devices [9], [10]. However, typical software backup solutions require additional backup storage devices. Also, a copy of the data on the backup storage device can be destroyed by ransomware again [11].

There have been studies that performed backups inside devices, called *device-level backup* solutions. They do not require additional backup devices [11]–[13]. This solution utilizes the solid-state drive (SSD)'s NAND flash memory characteristics. The SSD performs out-of-place writes for writes, invalidating original pages and writing them to new pages [12], [14]–[16]. More importantly, invalidated pages are the previous version of the original page and they remain on the SSD until they are erased by garbage collection (GC). That is, the out-of-place write property by the NAND flash memory-based SSD allows the opportunity to automatically create backup data inside the SSD. If these invalidated pages are carefully managed, the SSD itself performs backup internally.

FlashGuard [11] and SSD-Insider [12] are the state-of-the-art device-level backup SSDs. They detect ransomware attacks within SSDs by observing the ransomware's write I/O pattern in real time. Typical ransomware often shows overwrite patterns as it chooses victim files and reads, encrypts, and rewrites them. Specifically, once ransomware invades the system, it encrypts many files in a short time [11], [12], showing a bursty write-after-read (WAR) I/O pattern. Based on this observation, FLASHGUARD [11] and SSD-INSIDER [12] detect ransomware attacks by observing WAR I/O patterns and their burstiness. However, recent intelligent ransomware, such as Sleeper ransomware (SR), performs a delayed encryption attack [17]–[19], which does not exhibit a bursty WAR I/O pattern.

FLASHGUARD can detect delayed encryption attacks if it continuously keeps track of whether the page has been read in the past for a long period of time. This is because FLASHGUARD can perform backup whenever it observes the WAR I/O pattern. However, it does not precisely detect ransomware attacks. Thus, it performs backups in a conservative manner, producing a lot of unnecessary backups. On the contrary, SSD-INSIDER performs backup by considering the

Manuscript received January 7, 2021; revised July 8, 2021; accepted July 18, 2021. Date of publication July 26, 2021; date of current version June 20, 2022. This work was supported in part by the Samsung Semiconductor Research Grant; in part by the Korea University Grant; and in part by the Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korea Government (MSIT) under Grant 2021-0-00528 (Development of Hardware-Centric TrustedComputing Base and Standard Protocol for Distributed SecureData Box). This article was recommended by Associate Editor Y. Jin. *(Corresponding authors: Junghee Lee; Youngjae Kim.)*

Donghyun Min and Youngjae Kim are with the Department of Computer Science and Engineering, Sogang University, Seoul 04107, South Korea (e-mail: mdh38112@sogang.ac.kr; youkim@sogang.ac.kr).

Yungwoo Ko is with TmaxSoft, Seoul, South Korea.

Ryan Walker is with the Global Defense Group, Booz Allen Hamilton, McLean, VA 22102 USA (e-mail: ryan.walker@my.utsa.edu).

Junghee Lee is with the School of Cybersecurity, Korea University, Seoul 02841, South Korea (e-mail: j_lee@korea.ac.kr).

Digital Object Identifier 10.1109/TCAD.2021.3099084

WAR I/O pattern together with its burstness. However, SSD-INSIDER fails to detect I/O access patterns that have not been trained, such as nonbursty I/O patterns of ransomware that are typical patterns of delayed ransomware attacks.

In this article, we propose AMOEBA, a device-level backup SSD that more accurately detects the delayed attack of ransomware and performs a space-efficient backup. AMOEBA is armed with 1) a DMA engine performing inline content-based ransomware detection and 2) a fine-grained backup copy mechanism. AMOEBA makes the following technical contributions.

1) Content-based ransomware detection has high ransomware detection accuracy [3], [20]. AMOEBA performs content-based ransomware detection along with bursty WAR I/O pattern-based detection. However, executing content-based detection inline while processing I/O can slow down the I/O processing speed. Thus, AMOEBA employs a new DMA hardware that performs an inline content-based ransomware detection at high speed. The DMA engine performs similarity and entropy computations at high speed while performing direct memory access of the host's memory for writes. It accurately detects ransomware attacks by combining several attack detection factors (similarity, entropy, and burstiness of I/O requests), called a Ransomware Attack Risk Indicator (RARI).
2) AMOEBA implements a fine-grained backup copy management mechanism where each page is assigned a RARI value, and the RARI value determines whether or not to back up the previous page. AMOEBA keeps only one backup page per page, saving backup space. Since AMOEBA maintains only necessary backup copies, GC runtime overhead and backup space overhead are minimized. Also, the time to recover after ransomware infection is shortened. AMOEBA further optimizes the GC process by placing overwritten pages by ransomware and backup pages in different NAND blocks. This segregation helps the SSD reduce the GC overhead caused by valid page copy operations during the GC process.
3) In order to demonstrate the effectiveness of AMOEBA, we performed both simulation-based experiments and real prototype implementation-based experiments. For the simulation study, we implemented AMOEBA using the Microsoft Research's SSD simulator based on DiskSim [21], [22] and evaluated it using a mix of real ransomware (e.g., Erebus [23]) and normal application workloads. Extensive simulation experiments show that AMOEBA is highly comparable to the state-of-the-art device-level backup SSD systems in terms of performance. We also confirmed that the best and worst false detection rates of the AMOEBA system are just 0.14% and 5.41%, respectively. For the real implementation study, we have prototyped the AMOEBA system by modifying the flash translation layer (FTL) of the firmware on the Cosmos+ OpenSSD development platform [24]. From the experiments, we observed that there exists a high tradeoff between SSD performance and ransomware detection accuracy when content-based detection algorithms are implemented in firmware.

II. Background and Motivation

A. Solid-State Drive

NAND flash memory-based SSDs require read, write, and erase operations. Erase operations are performed at the granularity of a block, which is composed of multiple pages. A page is the granularity at which reads and writes are performed. Each page on flash can be in one of three different states: 1) valid; 2) invalid; and 3) free/erased. When no data has been written to a page, it is in the erased state. A write operation can be done only to an erased page, changing its state to valid. Erase operations (on average 1–2 ms) are significantly slower than reads or writes. Therefore, out-of-place writes (as opposed to in-place writes in HDDs) are performed to existing free pages, along with marking the page storing the previous version invalid. Later, invalid pages are collected after the erase operation during GC.

B. Device-Level Backup SSDs

An invalid page can be used as a backup page in NAND flash memory-based SSDs. The invalid page is the data of the previous version of the valid page. The invalid page remains until the GC is triggered. Therefore, when ransomware writes to the SSD to encrypt some pages, if invalidated pages are well kept in the SSD, they can be used as backup pages. FLASHGUARD [11] and SSD-INSIDER [12] use this property of the invalid page of the SSD for device-level backup SSDs. For example, FLASHGUARD keeps all invalid pages per a valid page as backup pages. However, keeping all invalid pages as backup pages increases the space overhead. It is necessary to make only backup pages of pages encrypted by ransomware. To do this, SSD-INSIDER runs algorithms that detect ransomware attacks according to I/O patterns inside the SSD. This ransomware I/O pattern-based detection approach is called behavior pattern-based detection [3], [25].

Typical ransomware encrypts user data. For data encryption, ransomware performs I/Os following a pattern that reads data and overwrites it and repeats these two steps, called a WAR pattern. In addition, the I/Os of ransomware attacks tend to exhibit a burst pattern, which means that many I/Os arrive in a short time. Specifically, FLASHGUARD monitors the WAR pattern of I/O requests for ransomware detection. SSD-INSIDER monitors the number of overwrite requests, i.e., the average length of continuously overwritten blocks for ransomware detection. However, this behavior pattern-based detection often fails to detect the evolved ransomware that does not follow the existing ransomware I/O pattern. For instance, recent ransomware performs encryption intermittently, so it does not follow the typical ransomware's I/O pattern [17]–[19]. Also, normal I/Os that similarly follow the typical ransomware pattern may be falsely detected as a ransomware attack.

C. Motivation

Content-based ransomware detection is more powerful than the aforementioned behavior pattern-based detection [3], [20]. Scaife *et al.* [3] stated that the most important features of ransomware could be the change of content. Specifically, there are two representative indicators of content-based ransomware detection: 1) similarity, or the change rate of data content after

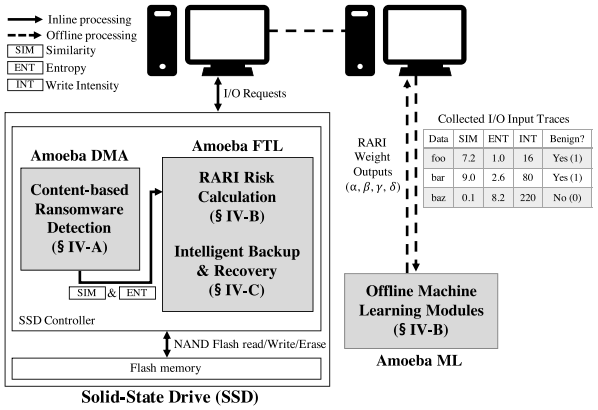

Fig. 1. Depiction of AMOEBA's modules (DMA, FTL, and ML modules) and their interaction.

a write operation and 2) entropy, or the randomness of the data. For example, a file encrypted by ransomware is very dissimilar to its original data. Also, the entropy value of an encrypted file is much higher than that of a normal file. Ransomware uses a cryptographic library to encrypt data, which increases the uncertainty of data. If these two values are high, it can be determined that there is a ransomware attack.

In this article, we design and develop AMOEBA, a content-based ransomware detection and data backup SSD system that meets the following goals: 1) it guarantees a high ransomware detection rate with little computational delay and 2) minimizes the number of backup copies by keeping only necessary backup copies.

III. OVERVIEW OF AMOEBA

The AMOEBA system consists of three major components: 1) AMOEBA DMA; 2) AMOEBA ML; and 3) the AMOEBA FTL. AMOEBA DMA and the AMOEBA FTL are embedded inside an SSD and operate at runtime, while the AMOEBA ML is an offline module that runs outside of the SSD. The AMOEBA ML runs offline to learn the ransomware's I/O patterns and data content on separate host machines. Fig. 1 depicts how the components of the AMOEBA system interact.

1) The AMOEBA DMA is a DMA hardware for performing content-based ransomware detection at high speed. AMOEBA DMA enhances the existing SSD's DMA hardware to perform content-based detection during I/O operations. Specifically, the DMA engine executes content-based detection algorithms, such as similarity and entropy computations [3] while transferring data between the host's main memory and the device's NAND flash memory.
2) The AMOEBA FTL is a software module for backup and recovery that is implemented by extending the existing FTL module. The AMOEBA FTL runs a ransomware detection algorithm for which we defined a RARI that considers three indicators: 1) similarity; 2) entropy; and 3) write arrival rate (intensity) of I/O requests. Each indicator of the RARI metric has significance.
3) The AMOEBA ML module sets the significance of each indicator of the RARI metric. Specifically, the AMOEBA ML performs supervised machine learning to find the optimal significance values of each significance in the RARI used in the AMOEBA FTL. The AMOEBA ML runs on a host offline where it uses collected block-level I/O traces of ransomware workloads as an input and finds the optimal each significance (weight) values for the RARI. A high ransomware detection rate by the AMOEBA ML module enables the AMOEBA FTL to only maintain one necessary backup page per logical page, thus minimizing the SSD space overhead for backup. Moreover, the recovery process is simple, as the AMOEBA FTL can simply recover original data pages by changing the state of each backup page to the valid state.

The AMOEBA ML may be vulnerable to privileged malware because it runs on a host system, which can be compromised. For example, an adversarial attack is possible [26]. To attack data integrity, malware could perform a poisoning attack that intentionally injects malicious learning data and destroys the machine learning model. There is also an evasion attack that deceives machine learning with input data tampering. However, this vulnerability issue of machine learning is not in the scope of our research. Our goal is to defend against ransomware attacks that read and encrypt data stored on SSDs.

IV. DESIGN AND IMPLEMENTATION

A. DMA Design for Content-Based Ransomware Detection

AMOEBA DMA executes inline byte-level similarity and entropy algorithms during data transfer. The AMOEBA DMA calculates similarity and entropy for every page write request. When a traditional SSD receives a write request from a host, the DMA engine fetches the data of the write request from the host and temporarily stores it in the DRAM region of the SSD. Then, the data is transferred to the NAND flash memory by an internal DMA controller. On the other hand, the AMOEBA DMA can calculate similarity and entropy inline while the data is being transferred from a host to the DRAM region of the SSD.

Similarity: What is actually calculated by the DMA is the difference between the old and new data. When a write request is issued, the DMA controller reads data of the old page and calculates the difference between the old and new data page. The difference is counted in bytes. In other words, the number of bytes of the new data that are different from the old data is accumulated. For example, suppose new data `0x00112233` overwrites old data `0x00AABBCC`. Only the first byte is the same (`0x00`), and the others are different (`11` versus `AA`, `22` versus `BB`, and `33` versus `CC`). Thus, the difference is calculated as three.

Entropy: The Shannon entropy is used as an indicator of randomness in the data [20]. The entropy is calculated per page. The formula is given in the following equation:

$$e = \sum_{i=0}^{255} P_{B_i} \log \frac{1}{P_{B_i}}. \tag{1}$$

P_{B_i} means the probability of finding byte i in a page. As an example, suppose a page has only eight bytes, and they are `00, 01, 02, 03, 00, 01, 04, 00`. Byte `00` occurs three times out of eight bytes, which results in $P_{B_0} = 3/8$. In a similar vein, bytes `01, 02, 03, 04` occur twice, once, once, and once, respectively. Thus, $P_{B_1} = 2/8$, $P_{B_2} = 1/8$,

$P_{B_3} = 1/8$, and $P_{B_4} = 1/8$. All other P_{B_i} where $i > 4$ are zero.

Thus, the probability P_{B_i} can be rewritten by C_{B_i}/S where C_{B_i} is the number of occurrences of byte i and S is the page size. Typically, the page size is in 2^N form. For example, N=12 if the page size is 4 kB. If we denote S as 2^N, (1) is rewritten as follows:

$$e = \sum_{i=0}^{255} \frac{C_{B_i}}{2^N} \log \frac{2^N}{C_{B_i}}. \tag{2}$$

The DMA controller maintains a counter for each byte and increments it whenever the corresponding byte is found while transferring data. After finishing data transfer, it calculates the entropy from the counter values. Since the entropy is calculated per page, if the request size is smaller than a page, the remaining bytes need to be read before calculating the entropy.

If we use (2) as it is, it will incur excessive overhead to implement in hardware because the formula requires real number computation. To reduce hardware cost, we only use integer computation by approximating the equation.

The purpose of calculating the entropy is not to find the exact value but to compare the relative values of different write requests. Thus, the AMOEBA DMA controller calculates $2^N \times e$ to avoid expensive division operations

$$2^N e = \sum_{i=0}^{255} C_{B_i} \log \frac{2^N}{C_{B_i}}. \tag{3}$$

The log term can be approximated as the count of leading zeros if we cut off all decimal points

$$2^N e = \sum_{i=0}^{255} C_{B_i} \left\{ \mathrm{CLZ}\left(C_{B_i}\right) + 1 \right\} \tag{4}$$

where CLZ means the count of leading zeros.

In a binary number, the count of leading zero (CLZ) is the number of "0" digits in the most significant positions of data, up to the position in which the first "1" is present. For example, if a binary number is 00001001, the CLZ of this number is 4. Suppose C_{B_i} has k leading zeros out of N bits [i.e., $k = \mathrm{CLZ}(C_{B_i})$]. Then, $2^{N-k-1} \leq C_{B_i} < 2^{N-k} - 1$. Note that the log computation is required only if $C_{B_i} > 0$, which means $k < N$. If we cut off all decimal points, $\log C_{B_i} = N - k - 1$, which means $N - \log C_{B_i} = k + 1$ and $\log(2^N/C_{B_i}) = k + 1$. Therefore, $\log(2^N/C_{B_i})$ is approximated by $k + 1$, which is $\mathrm{CLZ}(C_{B_i}) + 1$.

DMA Design: DMA controllers may have advanced features, such as burst, pipelined, and scatter-gather transfer. To focus on the proposed features of DMA, we assume a basic DMA controller without such advanced features, as shown in Fig. 2(a). The baseline DMA is a simple DMA that has three states. While it is in the *Idle* state, if the firmware triggers its operation, it moves to the *Read* state to read data from the source location. For a write request, the source location is in the main memory of an SSD. After reading data, it moves to the *Write* state to write the data to the destination, which is a NAND flash memory. It repeats reading and writing until all of the requested data are transferred.

The AMOEBA DMA has two more states, as shown in Fig. 2(b). In the *Read* state, it reads data from both the source and destination to calculate their difference. In the *Write* state, it actually calculates the difference and counts the occurrences of bytes while writing data. In the *Page* state, it reads the remaining data in the page, which are not read in the *Read* state, to calculate entropy. If there is no remaining data, this state is skipped. In the *Calc* state, it calculates entropy by going through counters of occurrences of bytes.

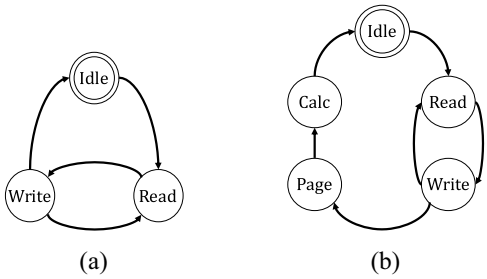

Fig. 2. State diagram of (a) baseline DMA and (b) AMOEBA DMA controllers.

If the write request is aligned with pages, most of the calculation delay is hidden by parallelizing calculation with data transfer. The only perceivable delay is the final entropy calculation that takes place in the *Calc* state. Compared to memory access delay, the final entropy calculation delay is negligible.

B. Ransomware Risk Calculation

RARI considers similarity, entropy, and write intensity. Then, the AMOEBA FTL computes RARI values by using the following equation:

$$\rho = \frac{1}{1 + \exp(-\zeta)}, \quad \text{where}$$
$$\zeta = \alpha * SIM + \beta * \mathrm{ENT} + \gamma * INT + \delta \tag{5}$$

Equation (5) shows the RARI calculation. Specifically, *SIM*, *ENT*, and *INT* refer to similarity, entropy, and write intensity, respectively. The weights α, β, and γ are prelearned weights of similarity, entropy, and write intensity and δ is a bias of the neural network.

The AMOEBA ML determines what portion of each indicator is reflected in the RARI calculation. The AMOEBA ML employs logistic regression, a representative statistical algorithm used to predict the classification of each instance [27]. In (5), a ζ is a linear combination of each indicator and its weights. This linear form is a general way to put them on a computation unit (neuron) in a neural network [28].

ρ is the actual RARI value of the write request page, which means the possibility of being ransomware, and is computed using (5). In order to compute the probability (ρ) of whether the write request with ζ is by ransomware, we employ the softmax sigmoid function, which is often used as a hypothesis function in the classification problem [28]. This function always generates expected results between 0 and 1. We map normal write requests to 0 and ransomware attack write requests to 1. By default, the AMOEBA FTL determines that it is a ransomware attack if $\rho \geq 0.5$.

The AMOEBA FTL receives similarity and entropy values from the AMOEBA DMA for all write requests and receives weights of the indicators in advance (e.g., firmware update).

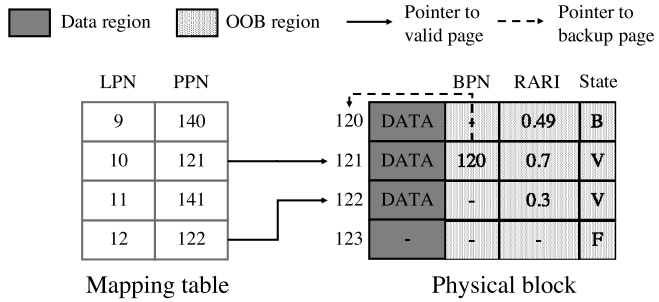

Fig. 3. Description of OOB metadata area for supporting the backup mechanism. LPN: logical page number, PPN: physical page number, BPN: backup page number, V: valid state, B: backup state, and F: free state.

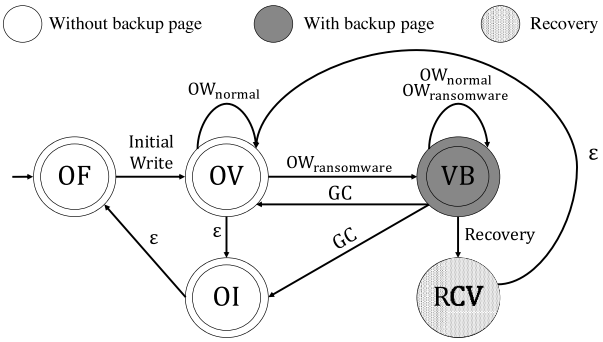

Fig. 4. NFA for a logical page in AMOEBA.

The write intensity can be obtained by counting the number of write requests during a given time window, which can be implemented by the firmware FTL at a negligible cost. Therefore, based on (5), the AMOEBA FTL finally computes the RARI value for each page write request and determines ransomware attacks.

C. Backup Process

The AMOEBA FTL uses the out-of-band (OOB) region of a physical page as a data structure to implement the backup and recovery mechanism, as depicted in Fig. 3. The OOB region of a page contains the backup page number (BPN) and the RARI value of the page. In Fig. 3, when a ransomware write I/O is requested at physical address 121, the AMOEBA FTL keeps its previous physical address 120 in the BPN of the OOB region.

Fig. 4 describes the state transition behavior of AMOEBA logical pages using nondeterministic finite automata (NFA). The graphical representation of the NFA consists of states (node) and inputs for state transition (edge). Table I presents the description of the state and input of the AMOEBA NFA. When the first page write occurs to a free page, its state changes from OF to OV through the state transition. From this point, if an overwrite request arrives, AMOEBA checks whether the request is safe.

The AMOEBA backup mechanism checks whether the RARI value of the current write request is larger than the defined threshold value. If this condition is satisfied, AMOEBA will regard it as the write request of a ransomware attack ($OW_{\text{ransomware}}$). If either of the two conditions is not satisfied, it will be regarded as a normal write request (OW_{normal}).

TABLE I
DESCRIPTION OF THE STATE AND INPUT OF THE NFA. (A) EACH STATE DESCRIPTION IN THE NFA. (B) EACH STATE TRANSITION DESCRIPTION IN THE NFA

(a)

Q (State)	Description
OF (Only Free)	Data is not yet written, so it is a free page.
OV (Only Valid)	Data is written and only a valid page exists.
OI (Only Invalid)	Data has been invalidated.
VB (Valid & Backup)	Both valid and backup pages exist.
RCV (Recovery)	Valid data will be replaced by the backup page.

(b)

Σ_ε (Input)	Description
Initial Write	Write data to free page
$OW_{ransomware}$	Ransomware overwrite
OW_{normal}	Normal overwrite
Recovery	Recovering a valid page using the backup page
GC	Valid page or backup page collected by GC

The state transitions of the ransomware overwrite and normal overwrite depend on the current state of the page (OV or VB). For instance, let $LPN_{(x,\text{ov})}$ denote logical page x in the OV state and $PPN_{(y,\text{valid})}$ denote physical page y in the valid state.

1) $OW_{\text{ransomware}}$ or OW_{normal} Upon OV State: Suppose that a logical page $LPN_{(a,\text{ov})}$ has been mapped to a physical page $PPN_{(b,\text{valid})}$.
   a) When $OW_{\text{ransomware}}$ is written to $LPN_{(a,\text{ov})}$, the physical page $PPN_{(b,\text{valid})}$ in the valid state becomes $PPN_{(b,\text{backup})}$ in the backup state and a new physical page $PPN_{(c,\text{free})}$ becomes $PPN_{(c,\text{valid})}$ in the valid state. Therefore, $LPN_{(a,\text{ov})}$ will have both the valid page $PPN_{(c,\text{valid})}$ and the backup page $PPN_{(b,\text{backup})}$. Then, $LPN_{(a,\text{ov})}$ changes to $LPN_{(a,vb)}$.
   b) If a normal write request OW_{normal} occurs to $LPN_{(a,\text{ov})}$, a page overwrite request is immediately executed without creating a backup page, and remains in the OV state. In other words, the existing mapping $PPN_{(b,\text{valid})}$ of the $LPN_{(a,\text{ov})}$ becomes $PPN_{(b,\text{invalid})}$, and the new $PPN_{(c,\text{free})}$ becomes the $PPN_{(c,\text{valid})}$ being mapped to $LPN_{(a,\text{ov})}$.
2) $OW_{\text{ransomware}}$ or OW_{normal} Upon VB State: Suppose that a logical page $LPN_{(a,vb)}$ is mapped to a physical page $PPN_{(c,\text{valid})}$ and has a backup page $PPN_{(b,\text{backup})}$. In other words, a logical page in the VB state has both a physical page in the valid state and a physical page in the backup state. The backup state is maintained since one or more ransomware requests have been received before. When an overwrite request is received in the VB state, a new physical page $PPN_{(d,\text{free})}$ is allocated and ready for writing. Since AMOEBA defines all write requests ($OW_{\text{ransomware}}$, OW_{normal}) to ransomware attacks, $PPN_{(b,\text{backup})}$ is still maintained as a backup page, $PPN_{(c,\text{valid})}$ becomes $PPN_{(c,\text{invalid})}$ and $PPN_{(d,\text{valid})}$, and $LPN_{(a,vb)}$ has $PPN_{(d,\text{valid})}$ and $PPN_{(b,\text{backup})}$.

When a recovery operation is requested by the user, the AMOEBA FTL changes SSD status to *read-only*. Pages in the

VB state also change to the RCV state. The valid page is just substituted by the current backup page during the recovery process by referencing the OOB metadata region. This process does not require user intervention multiple times compared with previous works [11], [12]. This is because a logical page has only one backup page at most. Therefore, it can be restored with a single user's recovery instruction.

D. Backup Data Management in AMOEBA

AMOEBA has backup state pages that are also considered a valid state. Until the recovery operation is called, AMOEBA must basically keep backup pages that have been created. But according to prior work [29] and our experimental results, the more backup pages created and the more SSD space occupied, the more GC calls and page copies that are needed. This is a side effect of maintaining a backup page in SSD. Meanwhile, after the recovery operation is called, AMOEBA changes the infected page to an invalid state and the backup page to a valid state. Then, the invalid page is later reclaimed by GC. However, if valid pages and invalid pages are mixed in a victim block, a valid page copy is necessarily required during GC. This is one of the factors that decreases GC efficiency. To address this side effect and increase GC efficiency, we propose the following backup page management policies: 1) *backup page leveling* and 2) *segregation between ransomware-infected data and backup data*.

1) Backup Page Leveling: When a block is chosen to be erased by GC, the existing SSD uses a greedy algorithm that selects a block with the smallest number of valid pages as a victim block. The GC with this greedy algorithm aims at reducing the number of valid page copies.

In GC of AMOEBA, backup pages are valid pages. In other words, unless they are forced to be deleted, they are regarded as valid pages that are not reclaimed during GC. If the number of backup pages increases, even though the total utilization of the SSD is not 100%, the SSD may have no free pages. In this situation, AMOEBA can have three design options: 1) to ignore incoming write request; 2) to stop backup allowing write requests (e.g., standard SSDs with no backup feature); and 3) to notify the user that the SSD is full and ask if backup pages are to be deleted. However, in the last option, if the user tells AMOEBA to delete backup pages, it may delete some backup pages that should not be deleted. Every backup page has a different probability of ransomware infection. Therefore, AMOEBA provides an algorithm that selects the backup pages to delete according to the probability of ransomware infection. We assume each backup state has a backup level defined by the following equation:

$$\text{Backup Level} = \left\lceil \frac{(\rho - \text{MIN})L}{\text{MAX} - \text{MIN}} \right\rceil \tag{6}$$

where ρ is the RARI value of the corresponding page in the valid state, L is the fixed number of possible backup levels, and MIN and MAX denote possible smallest and largest RARI values of an infected page, respectively. These are experimental parameters. In our experimental setting, we set MIN and MAX to 0.5 and 1.0, respectively, to evenly divide the range of suspicious RARI values into L regions.

As the backup level is lower, the importance of the backup becomes lower since its corresponding valid page has less possibility of being infected by the ransomware. If the backup page space occupies too much of the SSD, AMOEBA can delete backup pages with the lowest level, which have the lowest probability of ransomware infection. In our implementation, AMOEBA begins to delete backup pages when the SSD occupancy ratio reaches 90%. AMOEBA reduces GC overhead and consequently stabilizes SSD performance even when SSD's page occupancy is extremely high.

2) Segregation of Infected Data: We see that the lifetimes of all infected pages by ransomware are the same until the recovery is triggered. This is because when AMOEBA executes a recovery operation, the state of all infected pages changes to the invalid state and that of all backup pages changes to the valid state. Later, invalid pages created after recovery are initialized as free pages through the GC process.

If the victim block selected during GC consists of invalid pages and valid pages, it is inevitably needed to copy valid pages to a free block. Note that this valid page copy operation is expensive during GC. To decrease the number of pages copied during GC, AMOEBA provides an algorithm to place backup pages and infected pages in different blocks. If we prevent backup pages from being mixed with infected pages in the same block in advance, only invalid pages remain in the block that stores only the infected page after recovery. Thus, a copy for valid pages is not required for this block. Therefore, we can also reduce the number of page copy operations during GC and minimize the SSD performance overhead [30]. We call this scheme AMOEBA-Div hereinafter.

Fig. 5 illustrates how AMOEBA and AMOEBA-Div perform incoming I/O requests, respectively. In Fig. 5(a), AMOEBA, which does not have the segregation technique, scatters valid, backup, and infected pages by ransomware over several blocks. When a user recognizes a ransomware attack and triggers the recovery process, infected pages become invalid and the backup pages become valid again. After the recovery process, invalid pages are mixed with valid pages. As a result, page copy operations are required during GC. In particular, if these blocks are selected as a victim block later, two page copy operations are required.

On the other hand, in Fig. 5(b), AMOEBA-Div segregates infected pages from other types of pages in advance by assigning different types of blocks. A type 1 block is used to collect both valid and backup pages in AMOEBA-Div. A type 2 block is used to keep only infected pages that have a RARI value of 0.5 or greater. After recovery, only invalidated pages exist in the type 2 block and only valid pages exist in the type 1 block. Therefore, in AMOEBA-Div, an erase operation does not require a page copy operation during GC. In particular, if block 2 is selected as a victim block, no page copy operation is required.

E. Recovery Process

AMOEBA has only one backup page per logical page. Since AMOEBA performs content-based ransomware detection, a page attacked by ransomware is detected accurately. As a result, it makes it possible for AMOEBA to maintain only one backup page for each page necessary for recovery and not to keep other pages. Therefore, the recovery process

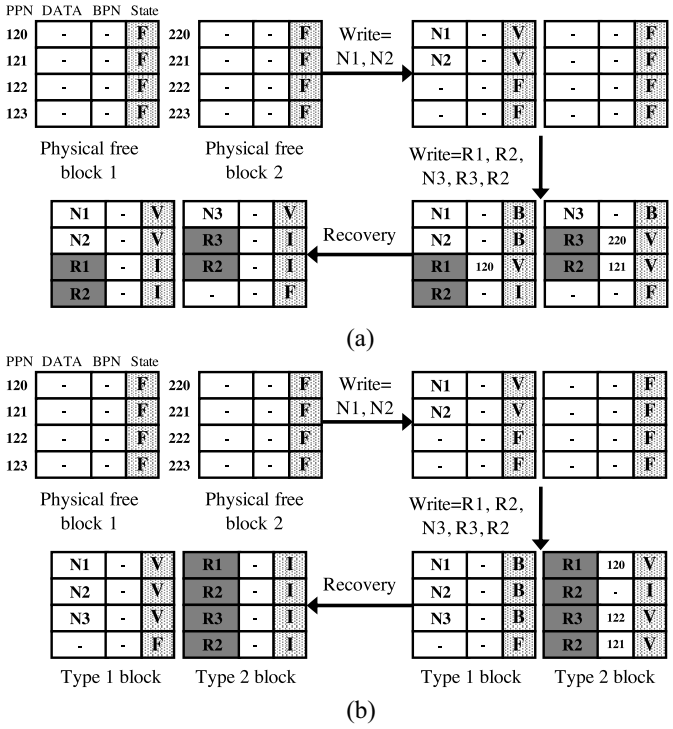

Fig. 5. Illustration of (a) AMOEBA without block segregation and (b) AMOEBA with block segregation for two write sequences and a recovery call. N = user's valid or backup data (Normal), R = ransomware data. F, V, I, and B means free, valid, invalid, and backup state. Each flash block consists of four pages.

is simple because it only needs to switch to backup pages during recovery. Hence, AMOEBA can be restored to the correct backup page, which is before ransomware infection, due to accurate ransomware detection. On the other hand, the recovery procedures of FLASHGUARD and SSD-INSIDER are complicated and can be recovered as incorrect backup pages. FLASHGUARD unconditionally creates backup pages for all WAR I/O patterns. FLASHGUARD creates backup pages for overwritten pages of normal applications as well as overwritten pages of ransomware. FLASHGUARD starts the recovery process only when the user sends the recovery operation (passive policy). During recovery, the user must individually scan all backup pages to find the correct pages to recover. SSD-INSIDER runs ransomware detection based on an overwrite pattern every time slice (1 s). If SSD-INSIDER detects ransomware in a time slice, it creates backup pages for all overwrite requests for the time slice. SSD-INSIDER keeps backup pages for up to 10 s because it follows the belief that ransomware can be detected within 10 s. Unlike FLASHGUARD, when SSD-INSIDER detects consecutive ransomware attacks for 3 s (tunable parameter), the SSD blocks write requests, automatically reports the detection results to the user and waits for the user's response to begin recovery (active policy). However, if the attack is not detected within 10 s due to false detection, SSD-INSIDER does not perform recovery and removes necessary backups. This can make recovery incorrect. AMOEBA can be equipped with either the passive recovery policy or the active policy. We will compare the recovery efficacy of AMOEBA (passive and active policies) with FLASHGUARD and SSD-INSIDER in Section VI.

TABLE II
SSD SIMULATOR CONFIGURATIONS

SSD Parameters		Machine Learning Parameters	
Page Size	8KB	Learning rate	0.1
Pages per Block	128	Regularization factor	0.001
Blocks per Plane	2048	**RARI Weights**	
Planes per Package	8	α	-3.9659
Cleaning Policy	Greedy	β	5.7759
Page Read Latency	0.025ms	γ	2.8070
Page Write Latency	0.2ms	δ	-4.1219
Block Erase Latency	1.5ms	**Backup Deletion Parameters**	
Request Size for DMA	8KB	Backup Level	2
GC Threshold	5%	Backup Deletion Threshold	90%

V. EXPERIMENTAL SETUP

AMOEBA *Simulation:* In order to evaluate the efficacy of AMOEBA, we implemented AMOEBA, FLASHGUARD [11], and SSD-INSIDER [12] using the SSD simulator developed by Microsoft Research [21], [22]. FLASHGUARD performs backups whenever WAR is visible. SSD-INSIDER implements an ID3 decision with tree-based machine learning. We use six ransomware's behavioral traits that are described in [12] as inputs of the decision tree. If SSD-INSIDER detects ransomware at specific time slice (1-s interval), it keeps backups of all overwrites requested during this time interval. For AMOEBA, we implemented a RARI value computation module and a mechanism that maintains only one backup page per page. Each page has a backup page level that indicates the likelihood of the page being infected by ransomware. For experiments, we used a two-level (high and low) backup page policy. Whenever GC is triggered, AMOEBA checks whether 90% of the flash chip is in use or not. If the page occupancy reaches 90%, the backup pages with low levels are not copied into the free block and are deleted during GC. We also implemented AMOEBA-Div to see the efficacy of segregating infected pages from other types of pages. When AMOEBA serves a write I/O request or valid and backup pages are copied during GC, the user's data and infected pages are stored separately on different blocks. All detailed configuration parameters used in the simulator are described in Table II.

AMOEBA *DMA Simulation:* AMOEBA DMA has the following additional overheads: 1) an additional page read for similarity computation and 2) extra clock cycles to compute entropy after the data transfer. Other computations (e.g., counting the number of occurrences of bytes) are hidden by parallelizing them with data transfers. The AMOEBA DMA is modeled in Verilog. To take into account the performance overhead of the AMOEBA DMA, we estimated the number of clock cycles consumed by baseline DMA and AMOEBA DMA using the Xilinx ISE simulator [31]. Note that baseline DMA is only responsible for data transfer, while the AMOEBA DMA not only performs data transfer but also calculates similarity and entropy. All these overheads are accounted as an additional latency in the AMOEBA DiskSim simulation.

Content-Based I/O Trace Collection: We collected block-level I/O traces including contents using blkCtrace [32] while running real ransomware samples and normal workloads. blkCtrace extracts block contents of every I/O request at the generic block I/O layer in Linux. Each I/O request's content is required to calculate the entropy and similarity of each block.

TABLE III
TESTBED CONFIGURATIONS FOR AMOEBA EMULATION

Host System	**Configuration**
CPU	Intel(R) Core(TM) i5-2400 CPU @ 3.10Ghz
Core Number	4
Memory	8GB
Kernel Version	Linux-Kernel-4.4.0-31 Generic
Cosmos plus OpenSSD	**Configuration**
CPU	Dual 1GHZ ARM Cortex-A9 Core
Memory	1GB
Capacity	1TB
SSD Controller	HYU Tiger4 (FPGA)
Host Interface	PCIe Gen2 8-lane (NVMe)

In the case of a write request, blkCtrace extracts I/O contents when the request is inserted into the device queue. In the case of a read request, blkCtrace extracts I/O contents when I/O is complete, since data must be read from the device to the main memory. Based on the fact that ransomware changes the extensions of encrypted files, we labeled block I/O performed on files with changed extensions as ransomware I/O.

Prototype: In addition to simulation, we also prototyped AMOEBA on a real SSD platform, Cosmos+ OpenSSD [24] in a Linux environment. We implemented content-based ransomware detection algorithms for RARI computation and backup mechanisms on FTL in firmware. When an overwrite I/O is requested on the SSD, FTL performs algorithms (similarity and entropy) using the CPU's computational capabilities and checks whether each I/O is ransomware I/O or user I/O. Therefore, the input values of algorithms (the previous data and the new data) should reside in device DRAM memory. For the previous data, the FTL issues the internal I/O request to the flash controller to fetch the data. For the new data, it is transferred from the host memory to device memory via the external DMA. By polling manner, the FTL confirms that the both of them reside in DRAM memory and performs the algorithm. According to (5), the FTL calculates the actual RARI value with the RARI weights that have already completed learning. If the RARI value is higher than 0.5, the FTL determines that the I/O was sent from ransomware.

If an infection is confirmed, FTL keeps the previous page as a backup page. In our firmware, GC does not erase physical pages where previous data exists. The physical address for the backup page can be accessed from an extra area of the current DRAM. The configurations used for OpenSSD are summarized in Table III. Our proposed AMOEBA system is independent of the host's operating system.

Workloads: To make files to be infected by ransomware, we generate a total of 3544 files in Table IV, which is 3.34 GB in size. We infect these files with ransomware, such as Erebus [23] (ER) and SR. The ER is capable of infecting files without delaying an encryption attack. Unlike ER, the SR performs a delayed encryption attack by slow encryption, which is intended to avoid existing detection systems. We extended an open-source ransomware from Github [33] to mimic the fundamental behavior of SR. We denote the SR with 0, 2–3, and 8–9 s of delay as SR(z), SR(s), and SR(l), respectively. We added each delay whenever ransomware encrypts individual files. Our set of ransomwares uses cryptographic algorithms, such as AES, DES, which are often used by well-known ransomware [34].

TABLE IV
FILE WORKLOAD CHARACTERISTICS TO BE INFECTED BY RANSOMWARES

Type	**Number**		**Size**		
	Num	%	Avg (KB)	Total (MB)	%
pdf	1114	31.43	926.58	1008.02	30.14
html	307	8.66	51.70	15.50	0.46
image files	617	17.41	107.05	64.50	1.93
xls	318	8.97	363.07	112.75	3.37
ppt	125	3.53	1483.33	181.07	5.41
doc	74	2.09	428.70	30.98	0.93
zip files	189	5.33	2699.51	498.25	14.90
txt	309	8.72	9.64	2.91	0.09
others	491	13.85	2983.93	1430.77	42.78
Total	3544	100	966.4288939	3344.75	100

TABLE V
WORKLOADS OF RANSOMWARES AND NORMAL APPLICATIONS. THE AVERAGE REQUEST SIZE OF EACH WORKLOAD IS 8 KB

	Name	**Description**	**Characteristics**	
			# of Read	# of Write
Only Ransomware	ER	Erebus	1,139,084	684,878
	SR(z)	Sleeper Ransomware (ZERO)	700,499	653,470
	SR(s)	Sleeper Ransomware (SMALL)	700,499	653,470
	SR(l)	Sleeper Ransomware (LARGE)	700,499	653,470
Mixed Workload	NER	Normal program & ER	1,214,107	1,751,681
	NSR(z)	Normal program & SR(z)	1,624,440	1,706,910
	NSR(s)	Normal program & SR(s)	1,624,440	1,706,910
	NSR(l)	Normal program & SR(l)	1,624,440	1,706,910
	CWER	Compression and Data Wiping & ER	1,972,664	2,256,046
	CWSR(z)	Compression and Data Wiping & SR(z)	1,970,286	2,258,424
	CWSR(s)	Compression and Data Wiping & SR(s)	1,970,286	2,258,424
	CWSR(l)	Compression and Data Wiping & SR(l)	1,970,286	2,258,424

For normal workloads (not ransomware), we used normal applications such as compression and data wiping programs. This wiping program overwrites a set of files with 0 and does the same with 1. The compression and data wiping programs follow similar I/O patterns as ransomware as their I/Os show a WAR pattern. In particular, compression and data wiping can dilute the content-based ransomware detection effect. Compression increases data randomness because compressed data requires fewer bits to store the same information. Data wiping decreases the similarity of files before and after wiping by completely replacing data with 0 or 1. Moreover, to consider a variety of I/O patterns, we developed a synthetic workload generator (called a normal I/O program hereinafter) that iterates read and overwrite I/Os to a file at a random offset. We also perform these normal applications to files.

We also generated mixed workloads by mixing ransomware and normal workloads. Table V shows two groups of ransomware. One group consists of a set of only ransomware workloads. The other group consists of mixed workloads. Specifically, we run normal applications on the files and then run ransomware. For example, for CWER in Table V, we compressed half of the files and wiped the rest simultaneously, then ran the Erebus ransomware on all files.

Parameter Setting From Machine Learning: In order to obtain the significance values of each indicator of the RARI equation, we used a logistic regression of supervised machine

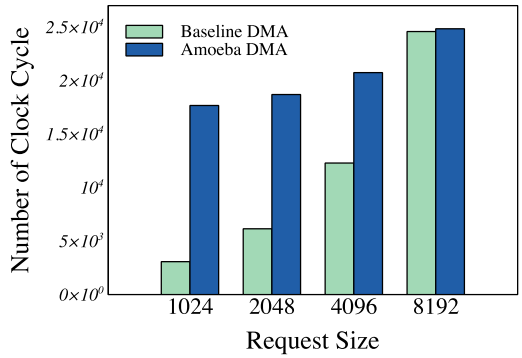

Fig. 6. Performance comparison between baseline DMA and AMOEBA DMA for overwrite requests of different size.

TABLE VI
RESULTS FOR GATE COUNT ESTIMATION

	Baseline DMA	**AMOEBADMA**
Gate Count	948	37,481

learning by training all types of normal programs and realistic ransomware workloads.

Table II shows the machine learning parameter values (α, β, and γ) set for the RARI in AMOEBA.

VI. EVALUATION RESULTS

A. DMA Performance and Hardware Cost

We analyze the performance overhead of the AMOEBA DMA. Fig. 6 compares clock cycles according to different request sizes. A request can be bigger than a page size in an SSD. If it is bigger than the page size, it should be divided into subrequests with the page size. If the size of a subrequest is aligned with the page size in SSD, the additional clock cycles consumed by the AMOEBA DMA are negligible compared with baseline DMA. In Fig. 6, we see there is only a 2% difference in clock cycles at an 8-kB request size (page size in an SSD). On the other hand, if the size of a subrequest is not aligned with the page size in SSD, the AMOEBA DMA generates up to four times more clock cycle overhead because it needs to read an entire page regardless of the request size to calculate entropy. However, we expect that many studies [14], [35] that reduce the number of subrequests can be integrated with AMOEBA and lead to fully reducing this clock cycle overhead as a result.

Next, we analyze hardware cost in terms of gate count. For this, we used the Synopsys Design Compiler with a 45-nm Nangate OpenCell Library [36] to estimate the gate count of each DMA approach. The results are shown in Table VI. Compared to the baseline DMA, the AMOEBA DMA seemingly incurs much more overhead. This is attributed mainly to two facts. First, the baseline DMA is quite simple because it only implements basic data transfer. Second, to calculate entropy, the AMOEBA DMA employs 256 counters, which are the major components (in terms of area) of the DMA. However, compared to the area of an entire SSD controller, a 37 K gate count is not a significant overhead. For example, hardware accelerators are often employed for low-density parity check (LDPC) in SSD controllers. One of the LDPC decoders is reported to incur 36.6 K to 158.6 K gates [37]. Thus, we believe that 37 K gates are acceptable in modern SSD controllers.

B. Performance Analysis

We evaluate the performance overhead of AMOEBA. Fig. 7 shows a performance comparison of AMOEBA and a baseline SSD that does not perform backup for Workload-NER. The average I/O response time was measured while varying the SSD's initial occupancy ratio. All I/O average response times are normalized to that of the baseline.

In Fig. 7(a), we observe that the response times of AMOEBA are similar to the baseline. AMOEBA has about 4% performance degradation when the SSD occupancy ratio is 0%. Even when the SSD occupancy ratio is 80%, AMOEBA has only 7.5% overhead. In Fig. 7(b) and (c), we can attribute this performance degradation to the increased number of page copies and GC calls. The more backup pages an SSD keeps, the more frequently GC is invoked, which results in migrating more backup pages.

Next, we evaluate performance between AMOEBA and AMOEBA-Div by using Workload-NER. To see the effects of block segregation, we performed recovery after finishing a ransomware attack. The average I/O response times were measured. The average response times are normalized to that of AMOEBA. In Fig. 8(a), we observe that AMOEBA-Div reduces the response time compared with AMOEBA. We see that AMOEBA-Div's response time is up to 13% lower than baseline AMOEBA. Because AMOEBA-Div segregates infected pages from other valid pages in advance, the GC process is faster than AMOEBA. In Fig. 8(b) and (c), the reduced response times of AMOEBA-Div are attributed to the reduced number of page copies and GC calls. Overall, AMOEBA-Div reduces the GC overhead up to 15% compared with AMOEBA.

C. Detection Accuracy Comparison

We compare the accuracy for ransomware detection techniques of AMOEBA, FLASHGUARD, and SSD-INSIDER with all workloads in Table V by counting false positive (FP) and false negative (FN) for each experiment. FP means normal I/O is mistakenly determined as ransomware I/O, and it unnecessarily makes backup pages. On the other hand, FN means ransomware I/O is mistakenly determined as normal I/O, and it does not make backup pages that should have been made. For machine learning of AMOEBA and SSD-INSIDER, we trained them with both normal and ransomware workloads. For example, normal program, ER, and SR(z) are trained workloads. Then, we evaluated AMOEBA and SSD-INSIDER for both trained workloads and untrained workloads, such as SR(s) and SR(l). In particular, the weight values of each RARI indicator in AMOEBA were determined through the training process, and the values used are shown in Table II.

Table VII shows FP and FN values for each workload. Workload-ER, SR(z), SR(s), and SR(l) contain only ransomware I/Os. For these workloads, we observe that AMOEBA has the highest ransomware detection accuracy. Specifically, AMOEBA only has a 0.26% of total false ratio on average, while FLASHGUARD and SSD-INSIDER have 1.06%

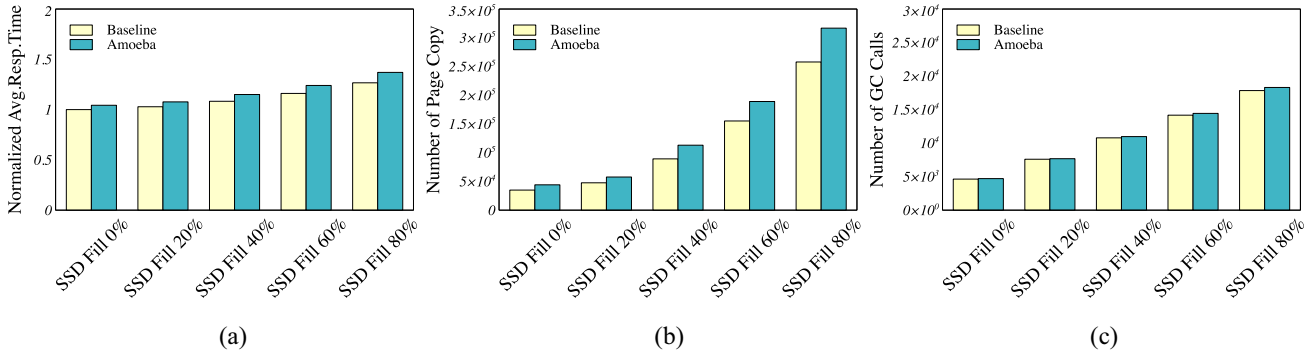

Fig. 7. Comparison of (a) response time, (b) number of page copies, and (c) number of GC calls between AMOEBA and the baseline system for different SSD occupancy ratios. The average I/O response time for baseline is 0.487 ms when SSD occupancy is 0%. Workload-NER is used.

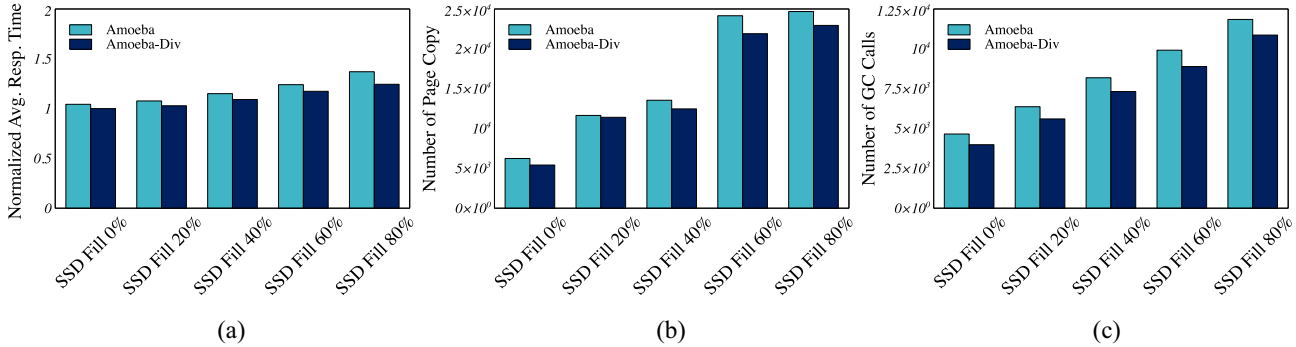

Fig. 8. Comparison of (a) response time, (b) number of page copies, and (c) number of GC calls between AMOEBA and AMOEBA-Div for different SSD occupancy ratios. The average I/O response time for AMOEBA is 0.503 ms when SSD occupancy is 0%. Workload-NER is used.

and 7.87% of total false ratios on average, respectively. For detection technique, FLASHGUARD only uses a WAR pattern. SSD-INSIDER completely relies on the overwrite I/O pattern. On the other hand, AMOEBA detects ransomware more accurately because it uses both I/O pattern and content-based detection.

We also observe that AMOEBA and SSD-INSIDER tend to miss the ransomware I/Os if ransomware workloads have a large delayed I/O pattern. For Workload-SR(z), AMOEBA and SSD-INSIDER only show 2624 FNs and 49 273 FNs. However, for Workload-SR(l), AMOEBA and SSD-INSIDER show 3160 FNs and 113 558 FNs. Since AMOEBA and SSD-INSIDER both take write intensity as a ransomware indicator, FN tends to increase if ransomware performs a delayed encryption attack. In particular, SSD-INSIDER's FN increases more than AMOEBA's FN because SSD-INSIDER only relies on the overwrite I/O pattern. In FLASHGUARD, detection is irrelevant to the delayed attack because only a pattern of a WAR is used for detection.

For Workload-NER, NSR(z), NSR(s), and NSR(l), normal I/O programs and ransomware workloads are mixed. In these workloads, normal I/O can be misjudged as ransomware I/O. For these workloads, we also see that AMOEBA has the highest accuracy. In FLASHGUARD, the best-case and worst-case accuracy are 17.12% and 17.24%, respectively. In SSD-INSIDER, the best-case and worst-case accuracy show 13.77% and 18.25%, respectively. On the other hand, in AMOEBA, the best-case and worst-case accuracy are 0.15% and 0.2%, respectively. Unlike FLASHGUARD and SSD-INSIDER, the number of misjudgments is low since AMOEBA uses content-based detection. For these workloads, we also observe that FN tends to increase if ransomware performs a delayed encryption attack. For Workload-NSR(z), AMOEBA shows 2574 of FN and SSD-INSIDER shows 72 281 of FN. However, for Workload-NSR(s) and NSR(l), AMOEBA shows 2578 and 2659 and SSD-INSIDER shows 90 192 and 103 740, respectively. Unlike SSD-INSIDER, AMOEBA maintains a low FN even if normal I/Os are included. This is because content-based detection significantly affects AMOEBA's detection accuracy.

For Workload-CWER, CWSR(z), CWSR(s), and CWSR(l), data compression and wiping applications are included. These workloads include many normal I/O patterns. In compression and wiping programs, both WAR I/O and overwrite I/O patterns appear. In these workloads, AMOEBA has a 5.18% total false ratio on average. This false ratio is higher than the average false ratios of other workloads. This is because the effect of similarity and entropy indicators is diminished due to compression and data wiping applications.

However, AMOEBA still has higher accurate detection ratios than FLASHGUARD and SSD-INSIDER for these workloads. FLASHGUARD only considers the WAR pattern, which is also often found in normal applications. Ransomware, which uses intermittent encryption, such as delayed attacks, can avoid SSD-INSIDER's detection mechanism because SSD-INSIDER considers only I/O behavioral features of known ransomware. Typical malicious code uses anti-debugging, such as timing checks [38], to avoid being detected by debuggers. Along with SR, other malware can also attempt a delayed encryption attack to avoid detection after identifying the existence

TABLE VII
RESULTS FOR RANSOMWARE DETECTION ACCURACY. (A) RANSOMWARE DETECTION ACCURACY OF AMOEBA. (B) RANSOMWARE DETECTION ACCURACY OF FLASHGUARD. (C) RANSOMWARE DETECTION ACCURACY OF SSD-INSIDER

(a)

Workload	AMOEBA		
	FP	FN	Total (ratio)
Workload-ER	0	2,874	2,874 (0.43%)
Workload-SR(z)	0	2,624	2,624 (0.19%)
Workload-SR(s)	0	2,624	2,624 (0.19%)
Workload-SR(l)	0	3,160	3,160 (0.23%)
Workload-NER	726	2,868	3,594 (0.2%)
Workload-NSR(z)	45	2,574	2,619 (0.15%)
Workload-NSR(s)	45	2,578	2,623 (0.15%)
Workload-NSR(l)	45	2,659	2,704 (0.15%)
Workload-CWER	97,965	3,504	101,469 (4.49%)
Workload-CWSR(z)	100,893	21,383	122,276 (5.41%)
Workload-CWSR(s)	100,892	21,384	122,276 (5.41%)
Workload-CWSR(l)	100,892	21,386	122,278 (5.41%)

(b)

Workload	FLASHGUARD		
	FP	FN	Total (ratio)
Workload-ER	0	13,803	13,803 (2.01%)
Workload-SR(z)	0	9,961	9,961 (0.75%)
Workload-SR(s)	0	9,961	9,961 (0.75%)
Workload-SR(l)	0	9,961	9,961 (0.75%)
Workload-NER	291,419	8,575	299,994 (17.12%)
Workload-NSR(z)	287,424	6,928	294,352 (17.24%)
Workload-NSR(s)	287,424	6,928	294,352 (17.24%)
Workload-NSR(l)	287,424	6,928	294,352 (17.24%)
Workload-CWER	365,325	8,873	374,198 (14.65%)
Workload-CWSR(z)	441,296	65,704	507,000 (22.44%)
Workload-CWSR(s)	441,296	65,704	507,000 (22.44%)
Workload-CWSR(l)	441,296	65,704	507,000 (22.44%)

(c)

Workload	SSD-INSIDER		
	FP	FN	Total (ratio)
Workload-ER	0	92,829	92,829 (13.55%)
Workload-SR(z)	0	49,273	49,273 (3.7%)
Workload-SR(s)	0	74,117	74,117 (5.62%)
Workload-SR(l)	0	113,558	113,558 (8.6%)
Workload-NER	226,991	92,697	319,688 (18.25%)
Workload-NSR(z)	169,025	72,281	241,306 (13.77%)
Workload-NSR(s)	168,502	90,192	258,694 (14.76%)
Workload-NSR(l)	137,023	103,740	240,763 (14.10%)
Workload-CWER	226,991	92,697	319,688 (18.25%)
Workload-CWSR(z)	153,496	85,094	238,590 (10.56%)
Workload-CWSR(s)	177,177	110,560	287,737 (12.74%)
Workload-CWSR(l)	190,951	165,585	356,536 (15.78%)

of the debugger. Unlike FLASHGUARD and SSD-INSIDER, AMOEBA considers not only the I/O access patterns but also I/O contents for detection. Therefore, AMOEBA can more accurately detect ransomware even if it uses anti-debugging and performs delayed encryption attacks.

D. Recovery Comparison

We compare the recovery performance of each system. We measure how much infected data cannot be recovered after a ransomware attack. Note that FLASHGUARD has a passive recovery policy and SSD-INSIDER has an active recovery policy. For a fair comparison, we compare AMOEBA that has a passive recovery policy with FLASHGUARD. Also, we compare AMOEBA that has an active recovery policy with SSD-INSIDER. We used workload-NSR(s) with a total of 37 min of simulation time.

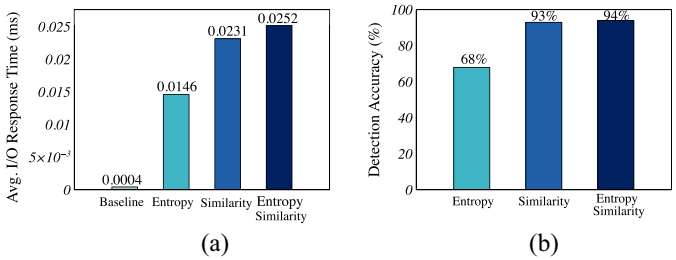

Fig. 9. Tradeoff between (a) SSD performance and (b) detection accuracy in AMOEBA-emulated SSD. Workload-NER is used.

To compare with FLASHGUARD, we perform a recovery after all write requests are processed. In FLASHGUARD, if a recovery is requested, it has to navigate all backup pages per page to find the correct backup page. The high FPs of FLASHGUARD lead to longer backup page search times. Also, even if all of the backup pages have been searched, it may not find the correct backup page. In our experiment, among total infected pages, 1.18% are not recovered. This stems from FN. On the other hand, AMOEBA, which has a passive recovery policy, is fast in searching for the backup page because it keeps one backup page per page. In addition, AMOEBA can recover more pages than FLASHGUARD. In AMOEBA, only 0.308% of total infected pages are not recovered, which is much less than that of FLASHGUARD. This is because AMOEBA's FN is lower than the FN of FLASHGUARD.

To compare with SSD-INSIDER, we can vary the time window that triggers recovery from 1 to 10 s. For example, if AMOEBA or SSD-INSIDER has a 2-s time window, they trigger the recovery process after detecting ransomware consecutively for 2 s and block all next incoming I/Os. In most cases, AMOEBA triggers recovery faster than SSD-INSIDER because SSD-INSIDER's FN is higher than AMOEBA's FN. Since SSD-INSIDER regards some ransomware I/Os as normal I/Os, it may result in late recovery execution. In SSD-INSIDER, if the recovery runs late, some backup pages disappear. Therefore, the recovery process cannot operate correctly. On the other hand, AMOEBA performs an early recovery, which helps prevent the loss of backup pages. In our experiment, 7.7% of infected pages are not recovered in SSD-INSIDER in the worst case. However, in AMOEBA only 0.207% of infected pages are not recovered in the worst case, which is much less than that of SSD-INSIDER.

E. Prototyping AMOEBA on Real SSD Platform

We prototyped AMOEBA on the Cosmos+ OpenSSD platform [24]. We implemented ransomware detection modules in firmware on the SSD platform and studied the tradeoff between ransomware detection accuracy and performance overhead. For this, we compared three cases of ransomware detection: 1) when only entropy is considered; 2) when only similarity is computed; and 3) when both entropy and similarity are considered. Fig. 9 shows the tradeoff results. In Fig. 9(a), we observe that considering either entropy or similarity shows a much higher I/O response time than when neither of them is

TABLE VIII
COMPARISONS OF AMOEBA, FLASHGUARD, AND SSD-INSIDER AGAINST VARIOUS I/O PATTERNS OF APPLICATIONS. ◯ DENOTES THAT SYSTEM DETECTS RANSOMWARE. X DENOTES SYSTEM CANNOT DETECT RANSOMWARE WELL, CAUSING DATA LOSS. ◉ DENOTES SYSTEM MISJUDGES USER I/O AS A RANSOMWARE I/O, CAUSING UNNECESSARY BACKUP COPIES

	Case 1	Case 2	Case 3	Case 4
FLASHGUARD [11]	◯	◯	◉	◉
SSD-INSIDER [12]	◯	X	◉	◯
AMOEBA	◯	◯	◉	◯ (or ◉)

considered. This overhead is attributed to data comparison and calculation byte by byte. In addition, we observe that using the only similarity is 1.5 times as slow as using only entropy. This is because both new data and existing data should be loaded from the NAND flash memory to DRAM for access byte by byte in order to perform the similarity calculation. Entropy, on the other hand, only needs to count the number of bytes present for the new data.

Fig. 9(b) shows the probability of detection accuracy depending on which ransomware detection algorithm is used. We observe that considering both similarity and entropy has the lowest false detection rate. However, considering the only similarity shows similar detection results to consider both. On the other hand, considering only entropy has a high false detection rate compared to others. Specifically, considering both similarity and entropy exhibits a total false detection rate of 6%, while considering only entropy exhibits a total false detection rate of 32%. This result is caused by a significant increase in the number of FPs due to data files that originally have high entropy (e.g., PDFs).

In summary, we see that there is a tradeoff between system performance and ransomware detection accuracy. According to our experiments, we observed that the similarity indicator has high detection efficiency compared to computational overhead. Moreover, the need for the AMOEBA DMA is further emphasized because it not only reduces the performance overhead from RARI computation but also detects ransomware accurately.

F. Discussion for Various Ransomware Attack Scenarios

In this section, we discuss the ransomware detection accuracy, recovery, and backup efficiency of various ransomware defense SSDs for the following application scenarios.

- Case 1: Ransomware's encryption attack with bursty I/O pattern.
- Case 2: Ransomware's encryption attack without bursty I/O pattern.
- Case 3: Legitimate encryption with bursty I/O pattern.
- Case 4: Legitimate encryption without bursty I/O pattern.

Table VIII summarizes the aforementioned comparison and analysis of ransomware detection for AMOEBA with FLASHGUARD and SSD-INSIDER.

- Case 1: FLASHGUARD simply checks the WAR I/O pattern and performs backup accordingly. SSD-INSIDER can detect ransomware correctly because it considers the bursty I/O access pattern for detection. AMOEBA is also able to detect correctly because it can observe a high write I/O arrival rate and changed data content.
- Case 2: Ransomware that executes a delayed encryption attack belongs to case 2. In this case, SSD-INSIDER fails to detect ransomware because it has learned only ransomware with a bursty I/O access pattern. Therefore, it misjudges delayed encryption attacks by ransomware as a user's I/O. Unlike SSD-INSIDER, AMOEBA correctly detects delayed attacks of ransomware because it further considers the data content as well as the bursty I/O intensity.
- Case 3: All of three systems misjudge user I/O as a ransomware I/O. FLASHGUARD and SSD-INSIDER perform the backup because data encryption by the user also shows a WAR pattern and bursty I/O access pattern. AMOEBA also performs backup. This is because the entropy of encrypted data is high, and the content is dissimilar to previous content. AMOEBA also checks a bursty I/O access pattern.
- Case 4: SSD-INSIDER does not misjudge user I/O as a ransomware I/O. On the other hand, AMOEBA is likely to misjudge it as a ransomware I/O according to entropy and similarity indicators. This is because the data is encrypted. However, it is not always misjudged, as AMOEBA also takes into account the low I/O arrival rate. AMOEBA exhibits these two behavioral patterns.

Overall, AMOEBA is superior to other ransomware defense SSDs in terms of detection accuracy, recovery, and backup efficiency. But, AMOEBA does not guarantee data recovery if the user legitimately and intentionally encrypts data multiple times in a burst pattern and ransomware encrypts the data. In the case of a user's intentional encryption, AMOEBA can misjudge user I/O as ransomware I/O. Therefore, in AMOEBA data before encryption is created as backup data. Immediately afterward, AMOEBA is actually attacked by ransomware, but it does not create a backup page. Therefore, it is impossible to recover data encrypted by the user just before a ransomware attack. However, this case will happen very rarely.

VII. RELATED WORK

FLASHGUARD [11] proposes a mechanism to perform data backup inside SSD rather than OS. If any page inside an SSD shows a WAR pattern it keeps invalidated pages as backup pages. FLASHGUARD adds a *backup* state between valid and invalid states, considering the feature of an SSD device in which the valid state page is not deleted immediately but rather kept in the invalid state for a certain period of time. If a WAR pattern that is suspected to be an access pattern of ransomware is found, FLASHGUARD switches the valid state to the *backup* state instead of *invalid* for all overwrites. Thus, FLASHGUARD does not require additional backup disks.

SSD-INSIDER [12] proposes a new approach to increase the detection accuracy. SSD-INSIDER observes that the ransomware's overwrite patterns are distinguishable from a benign application's patterns. For example, the number of overwrites of ransomware is greater than that of a normal application. There is also a difference in that the average

number of blocks that are consecutively overwritten is less in ransomware writes. SSD-INSIDER detects ransomware by calculating these features every time slice (1 s) with negligible overhead. In particular, SSD-INSIDER uses the ID3 decision tree for high detection accuracy. SSD-INSIDER also uses the invalid page for a backup. However, unlike FLASHGUARD, a fixed-size data structure, called a *recovery queue* is required to maintain the backup information (previous physical address). When ransomware is detected three times in a row, SSD-INSIDER automatically triggers the recovery process. Then, SSD-INSIDER will wait for the user's response and get ready to roll-back the FTL's mapping information to 10 s ago by using the recovery queue.

RansomBlocker [34] leverages existing hardware accelerators to calculate entropy and to apply convolutional neural networks (CNNs) to SSD to prevent ransomware. RansomBlocker employs a time-out backup policy to prevent ransomware attacks that delete original data after writing encrypted data. This policy only temporarily backs up all the pages that are regarded unnecessary by the host in a conservative manner. The CNN incurs significant overhead in terms of latency and hardware cost. Even though a high-performance FPGA-based CNN accelerator is employed, its latency is usually at least 1 ms [39], [40]. Considering the response time of modern SSDs, it is difficult to apply CNN prediction in real time for ransomware detection. Furthermore, the CNN hardware accelerator requires considerable hardware cost. The number of gates that the AMOEBA DMA requires is only 37 K. This gate count can be converted to 2.7 K logic cells [41]. On the other hand, the CNN accelerator is implemented using 80% of ZCU102's FPGA logic cells, which means it uses 480 K logic cells [42]. It is 192 times higher than the AMOEBA DMA.

VIII. CONCLUSION

A ransomware attack encrypts data and requires a ransom fee, causing financial damages to companies, financial institutions and organizations. To defend ransomware attacks, device-level backup SSDs have been proposed. They detect ransomware only based on I/O access patterns. However, they often fail to detect attacks that do not exhibit traditional I/O patterns of ransomware. In this article, we proposed AMOEBA, a content-based ransomware detection SSD framework. AMOEBA consists of three modules. First, the AMOEBA DMA is a hardware module that performs ransomware detection at high speed. This DMA module not only causes negligible performance overhead but also uses content-based detection to enable accurate detection. Second, the AMOEBA ML is a supervised machine learning module that operates on a host machine offline. With this module, the weight values of the RARI can be calculated. Accordingly, AMOEBA further minimizes the ransomware detection error rate. Third, the AMOEBA FTL has a backup mechanism with one backup page per logical page. Since it manages only necessary backup pages, it has a low GC overhead. We conducted extensive evaluations with mixed traces of normal workloads and real ransomware workloads and observed there is little performance overhead caused by detection and backup management in AMOEBA. We also saw that the performance of AMOEBA is almost similar to that of an SSD without backup. From the perspective of ransomware detection rate, AMOEBA has significantly lower false detection rates than FLASHGUARD and SSD-INSIDER.

ACKNOWLEDGMENT

The authors thank Prof. Sungyong Park and Jinwoo Ahn for their constructive comments that help improve this article.

REFERENCES

[1] V. Wilson. (2019). *24 Recent Ransomware Attacks in 2019*. [Online]. Available: https://spinbackup.com/blog/24-biggest-ransomware-attacks-in-2019

[2] (2019). *McAfee Labs Threats Report*. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf

[3] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop it): Stopping ransomware attacks on user data," in *Proc. 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 303–312.

[4] C. Moore, "Detecting ransomware with honeypot techniques," in *Proc. Cybersecurity Cyberforensics Conf. (CCC)*, Amman, Jordan, Aug. 2016, pp. 77–81.

[5] M. M. Ahmadian and H. R. Shahriari, "2entFOX: A framework for high survivable ransomwares detection," in *Proc. Int. ISC Conf. Inf. Security Cryptol. (ISCISC)*, Sep. 2016, pp. 79–84.

[6] K. Cabaj and W. Mazurczyk, "Using software-defined networking for ransomware mitigation: The case of CryptoWall," *IEEE Netw.*, vol. 30, no. 6, pp. 14–20, Nov./Dec. 2016.

[7] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, "PayBreak: Defense against cryptographic ransomware," in *Proc. ACM Asia Conf. Comput. Commun. Security (ASIACCS)*, Apr. 2017, pp. 599–611.

[8] D. Maiorca, F. Mercaldo, G. Giacinto, C. A. Visaggio, and F. Martinelli, "R-PackDroid: API package-based characterization and detection of mobile ransomware," in *Proc. Symp. Appl. Comput. (SAC)*, 2017, pp. 1718–1723.

[9] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Trans. Comput. Syst.*, vol. 10, no. 1, pp. 26–52, 1992.

[10] M. Vrable, S. Savage, and G. M. Voelker, "BlueSky: A cloud-backed file system for the enterprise," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2012, p. 19.

[11] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, "FlashGuard: Leveraging intrinsic flash properties to defend against encryption ransomware," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, 2017, pp. 2231–2244.

[12] S. Baek, Y. Jung, A. Mohaisen, S. Lee, and D. Nyang, "SSD-insider: Internal defense of solid-state drive against ransomware with perfect data recovery," in *Proc. Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Vienna, Austria, Jul. 2018, pp. 875–884.

[13] D. Min *et al.*, "Amoeba: An autonomous backup and recovery SSD for ransomware attack defense," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 245–248, Jul.–Dec. 2018.

[14] M. Kang, W. Lee, and S. Kim, "Subpage-based flash translation layer for solid state drivers," in *Proc. KAIST Open Access Self Archiving Syst. (KOASAS)*, 2016, pp. 1–30.

[15] S. Kim, H. Oh, C. Park, S. Cho, and S.-W. Lee, "Fast, energy efficient scan inside flash memory SSDs," in *Proc. Int. Workshop Accelerating Data Manage. Syst. (ADMS)*, Sep. 2011 pp. 1–8.

[16] S. Kim and J.-S. Yang, "Optimized I/O determinism for emerging NVM-based NVMe SSD in an enterprise system," in *Proc. 55th Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2018, pp. 1–6.

[17] L. Fernández Maimó, A. H. Celdrán, Á. L. P. Gómez, F. J. G. Clemente, J. Weimer, and I. Lee, "Intelligent and dynamic ransomware spread detection and mitigation in integrated clinical environments," *Sensors*, vol. 19, no. 5, p. 1114, 2019.

[18] (2019). *Cyemptive Launches Protection Against Sleeper Ransomware*. [Online]. Available: www.intelligencecommunitynews.com/cyemptive-launches-protection-against-sleeper-ransomware/

[19] (2020). *How Will I Recover From Ransomware?*. [Online]. Available: https://www.davosnetworks.com/how-will-i-recover-from-ransomware

[20] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security Privacy*, vol. 5, no. 2, pp. 40–45, Mar./Apr. 2007.

[21] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Techn. Conf. (ATC)*, Jun. 2008, pp. 57–70.
[22] J. S. Bucy *et al.*, "The DiskSim simulation environment version 4.0 reference manual," Dept. Parallel Data Lab., Carnegie Mellon Univ., Pittsburgh, PA, USA, Rep. CMU-PDL-08-101, May 2008.
[23] Trend Micro. (2017). *Erebus Linux Ransomware: Impact to Servers and Countermeasures*. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/erebus-linux-ransomware-impact-to-servers-and-countermeasures
[24] *Cosmos+ OpenSSD Platform*. Accessed: Aug. 2016. [Online]. Available: http://www.openssd.io/
[25] A. Continella *et al.*, "ShieldFS: A self-healing, ransomware-aware filesystem," in *Proc. 32nd Annu. Conf. Comput. Security Appl. (ACSAC)*, Dec. 2016, pp. 336–347.
[26] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognit.*, vol. 84, pp. 317–331, Dec. 2018.
[27] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic Regression*. New York, NY, USA: Springer, 2002.
[28] C. C. Aggarwal *et al.*, *Neural Networks and Deep Learning*, vol. 10. Cham, Switzerland: Springer, 2018, pp. 978–983.
[29] OCZ Technology. (2012). *PCI Express OCZ Technology*. [Online]. Available: http://www.ocztechnology.com/products/solid_state_drives/pci-e_solid_state_drives
[30] E. Rho *et al.*, "FStream: Managing flash streams in the file system," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, Feb. 2018, pp. 257–263.
[31] Xilinx. *ISE Simulaltor (ISim)*. Accessed: May 2012. [Online]. Available: https://www.xilinx.com/products/design-tools/isim.html
[32] D. Park, Y. Kim, S. Park, and J. Lee, "blkCtrace : Lightweight block I/O layer content tracing on Linux," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, Feb. 2018.
[33] *Virtual Gangster*. Accessed: Jun. 2017. [Online]. Available: https://github.com/roothaxor/Ransom/
[34] J. Park, Y. Jung, J. Won, M. Kang, S. Lee, and J. Kim, "RansomBlocker: A low-overhead ransomware-proof SSD," in *Proc. Annu. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.
[35] H. Lv *et al.*, "Exploiting minipage-level mapping to improve write efficiency of NAND flash," in *Proc. Int. Conf. Netw. Archit. Storage (NAS)*, Chongqing, China, Oct. 2018, pp. 1–10.
[36] J. Knudsen, "Nangate 45nm open cell library," in *Proc. CDNLive EMEA Conf.*, 2008.
[37] N. Miladinovic, "LDPC compiler for NAND flash and SSD controllers," presented at the Flash Memory Summit, Santa Clara, CA, USA, 2012.
[38] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Anchorage, AK, USA, Jun. 2008, pp. 177–186.
[39] X. Wei *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proc. Annu. Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.
[40] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL™ deep learning accelerator on Arria 10," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2017, pp. 55–64.
[41] "XA Zynq-7000 SoC data sheet: Overview," Data Sheet DS188 (v1.3.2), Xilinx, San Jose, CA, USA, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds188-XA-Zynq-7000-Overview.pdf
[42] Xilinx. (2019). *ZCU102 Evaluation Board User Manual*. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf

Donghyun Min received the B.S. and M.S. degrees in computer science and engineering from Sogang University, Seoul, South Korea, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree in computer science and engineering.

His research interests include high-performance I/O and emerging storage technologies, operating system, and system security.

Yungwoo Ko received the B.S. degree with a double major in English literature and computer engineering and the M.S. degree in computer science and engineering from Sogang University, Seoul, South Korea, in 2019 and 2021, respectively.

He is currently employed with TmaxSoft, Chicago, IL, USA, as a Software Engineer. His research interests include operating system, storage, and system security.

Ryan Walker received the B.S. degree in chemistry from the University of Texas, Austin, TX, USA, in 2014, and the B.S. and M.S. degrees in computer engineering from the University of Texas at San Antonio, San Antonio, TX, USA, in 2018 and 2020, respectively.

In 2019, he worked as an Intern with NXP Semiconductors, Eindhoven, The Netherlands, on the functional verification of a new hardware design for the Design for Test Group. He is currently employed with Booz Allen Hamilton, McLean, VA, USA, as a Design and Verification Engineer. His areas of interest include chip design and system-on-chip verification.

Junghee Lee received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 2000 and 2003, respectively, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2013.

He has been with the School of Cybersecurity, Korea University, Seoul, since 2019. From 2003 to 2008, he worked with Samsung Electronics, Suwon, South Korea, on electronic system level design of mobile system on chip. From 2014 to 2019, he was with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX, USA, as an Assistant Professor. His research interests include secure design of processor, nonvolatile memory, storage, and dedicated hardware.

Youngjae Kim received the B.S. degree in computer science from Sogang University, Seoul, South Korea, in 2001, the M.S. degree in computer science from KAIST, Daejeon, South Korea, in 2003, and the Ph.D. degree in computer science and engineering from Pennsylvania State University, State College, PA, USA, in 2009.

He was a Staff Scientist with the Oak Ridge National Laboratory, U.S. Department of Energy, Oak Ridge, TN, USA, from 2009 to 2015 and an Assistant Professor with Ajou University, Suwon, South Korea, from 2015 to 2016. He is currently an Associate Professor with the Department of Computer Science and Engineering, Sogang University. His research interests include operating system, file and storage system, database system, system security, and distributed system.