“THEME ARTICLE”, “FEATURE ARTICLE”, or “COLUMN” goes here: The theme topic or column/department name goes after the colon.

Write Deduplication and Hash Mode Encryption for Secure Non-volatile Main Memory

Pengfei Zuo, Yu Hua
Huazhong University of Science and Technology

(Corresponding Author: Yu Hua)

Ming Zhao

Arizona State University

Wen Zhou, Yuncheng Guo

Huazhong University of Science and Technology

For non-volatile main memory (NVMM), both security and endurance are important. However, the diffusion property of memory encryption renders existing bit-level write reduction techniques ineffective. This paper proposes DeWrite, a line-level write reduction technique to enhance the endurance and performance of secure NVMM. Moreover, DeWrite leverages a new encryption technique, i.e., hash mode encryption, and exploits opportunistic parallelism between encryption and deduplication operations to efficiently synergize deduplication and encryption for space and time savings.

Non-volatile memory (NVM) technologies are considered as promising candidates of the next-generation main memory[8]. NVM has high scalability and low energy consumption but suffers from the problems of low endurance and security.

First, NVM is vulnerable to physical access based attacks, including stolen DIMM and bus snooping attacks[3,4,5,6], due to still retaining data after systems are powered down. For example, an attacker stealing the NVM DIMM or acting as a machine repairman can directly stream out all the data from the DIMM. Hence, memory encryption becomes important to ensure the data security in NVM. Second, NVMs typically have limited write endurance, e.g., $10^7 - 10^8$ writes for PCM. Writes on NVM also cause higher latency (i.e., $3 - 8\times$) and energy overhead than reads. The bit-level write reduction techniques, such as Data Comparison Write (DCW)[1] and Flip-N-Write (FNW)[2], are able to significantly reduce the number of bits written to NVM, based on the observation that only a small number of bits are modified for a write.

However, memory encryption renders existing bit-level write reduction techniques ineffective for encrypted non-volatile main memory (NVMM), due to the diffusion property of encrpytion[3]. Because of this property, the change of a single bit in the plaintext leads to the change of about half of the bits in the ciphertext. Hence, existing techniques like DCW[1] and FNW[2] cannot achieve significant data reduction for encrypted NVMM.

We observe that abundant data duplication at the line level exist which can be exploited to reduce the number of writes to encrypted NVMM. These observations motivate us to propose DeWrite[7], a novel solution for enhancing both the lifetime and performance of encrypted NVMM with new write deduplication and encryption techniques and their opportunistic parallelism. Specifically, DeWrite makes the following contributions:

- **Write deduplication.** To reduce the number of NVM writes, DeWrite introduces write deduplication, which eliminates a duplicate write at the cost of a read latency, improving the system performance due to the intrinsic read/write asymmetry of NVM.
- **Hash mode encryption.** To reduce the metadata overhead, DeWrite leverages a space-efficient encryption model, i.e., hash mode encryption, by replacing the per-line counters used in the counter mode encryption with per-line hashes and reusing the same hash store employed by deduplication.
- **Prediction-based parallelism.** To achieve good performance in deduplication and NVM encryption, DeWrite opportunistically performs these two operations in parallel based on a simple yet effective duplication prediction scheme.

WRITE DEDUPLICATION

To explore how many cache lines written to main memory are duplicate, we have examined 20 applications from SPEC CPU2006 and PARSEC 2.1 benchmark suites. We observe that the percentage of duplicate lines vary from 18.6% to 98.4% across the 20 applications[7]. Eliminating these duplicate lines via line-level deduplication would result in much high write reductions. Hence, in DeWrite, we study the use of deduplication for enhancing the endurance and performance of secure NVMM.

To perform in-line deduplication, DeWrite leverages the asymmetric property of NVMs, in which the write latency is much higher than read latency (i.e., 3~8 times). To detect duplicate cache lines, DeWrite computes a light-weight hash to summarize the contents of cache lines, rather than the cryptographic hash with high computation latency. If the hash of the cache line matches that of an existing line in NVMM, DeWrite reads the line and compares the corresponding data byte by byte. Thus DeWrite eliminates a duplicate write at the cost of a read.

TABLE I: Traditional Deduplication v.s. DeWrite.

(a) The comparisons of hash computation latency and sizes.

Hash Functions	SHA-1	MD5	CRC-32
Latency	321 ns	312 ns	15 ns
Size	160 bits	128 bits	32 bits

(b) The comparisons of duplication detection latency.

Methods	Traditional Deduplication	DeWrite
A duplicate line	$\geqslant$312 ns + t_Q	91 ns + t_Q'
A non-duplicate line	$\geqslant$312 ns + t_Q	15 ns + t_Q'

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2886176, IEEE Micro

We compare the latency of duplication detection between traditional deduplication and DeWrite in Table I. Traditional deduplication includes existing main memory deduplication and external storage deduplication, which use a cryptographic hash function (e.g., SHA-1 and MD5) to compute the fingerprints of data and assume no hash collisions. Thus the data are considered to be duplicate if their fingerprints are identical. The detection latency using the cryptographic hash function, regardless of the duplication of the cache line, is more than 312 ns + t_Q, where t_Q denotes the latency of querying the fingerprint store, as shown in Table I(a). The duplication detection latency is even higher than the NVM write latency (300 ns). Hence, traditional deduplication is not cost-effective to eliminate cache-line-level duplication.

Instead, DeWrite uses the light-weight hash, i.e., CRC-32, where hash collisions are practically unavoidable. Thus, if the hash of the cache line matches that of an existing line in NVMM, DeWrite reads the line and compares the corresponding data. Only when the data are identical, the cache line is considered to be duplicate. The latency of computing a CRC-32 hash is 15 ns and the latency of reading a line is 75 ns[3,4]. The comparison of two lines can be implemented in hardware logic with low latency[1], i.e., 1 cycle. Hence, if a cache line is duplicate, the latency to detect the duplicate cache line in DeWrite is about 91 (=15+75+1) ns + t_Q', where t_Q' denotes the latency of querying the hash store. If a cache line is non-duplicate, its hash value will not be found in the hash store and the read can be saved. Thus the latency to determine a non-duplicate cache line is 15 ns + t_Q'. Moreover, when a cache is employed to accelerate accesses to the hash store, the cache can store more CRC-32 hashes than SHA-1/MD5 hashes, because the former is much smaller. Hence, given the same size of cache, using CRC-32 hashes leads to a higher cache hit rate and correspondingly a lower access latency, i.e., $t_Q' < t_Q$. In summary, DeWrite significantly reduces the duplication detection latency by leveraging the read/write asymmetric property of NVM and light-weight hashing.

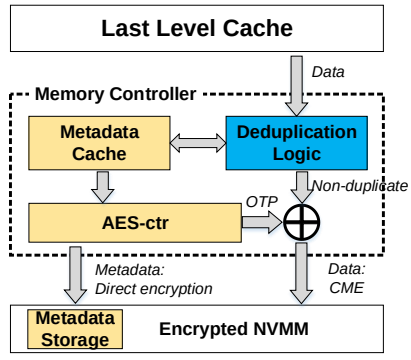


Figure 1: The hardware architecture of DeWrite. (We use the counter storage and counter cache in existing secure NVMMs as the metadata storage and metadata cache, respectively. The deduplication logic is the new component.)

The hardware architecture of DeWrite for supporting cache-line-level deduplication in the secure NVMM is shown in Figure 1. In order to reduce the metadata accesses to NVMM, existing works[3,4,5] on counter mode encryption extend the memory controller to include a write-back metadata cache used for buffering the counters. DeWrite uses this metadata cache to buffer the deduplication-related metadata. A memory region in the encrypted NVMM is used to store the counters. We use this region to store the metadata of deduplication. Hence, compared with existing works[3,4,5], the only one new component in DeWrite is the deduplication logic that is used to determine whether a cache line is duplicate. The data are encrypted using hash mode encryption (HME). To avoid storing the hashes of the metadata, the metadata are encrypted using the direct encryption scheme, i.e., directly encrypting the metadata using AES with a global key. In this case, the latency of loading the metadata into the metadata cache increases since the decryption cannot be executed with memory access in parallel. However, it does not significantly affect the performance since the miss ratio in the metadata cache is very low. We consider the 256B of deduplication granularity to reduce the metadata overheads. Thus the sizes of a memory line in the NVMM and a cache line in the last level cache are 256B. This cache line size has been widely used in the existing work on

NVMM[9]. Moreover, the commercial processors, e.g., IBM z systems processors[10], also use the 256B cache line size for all CPU caches.

To support in-line deduplication, DeWrite uses four data structures[7] for duplication detection and data management, including address mapping table, hash table, inverted hash table, and free space management table. These data structures are stored in the encrypted NVMM and the hot entries are maintained in the metadata cache.

The metadata storage includes four tables, i.e., the address mapping table, inverted hash table, hash table, and free-space management table. In the inverted hash and address mapping tables, the storage overhead is 4B+1bit/line, due to the 4B real address/hash and 1bit flag in each entry. The 4B real address can address up to 1TB of NVMM with 256B line size, which is sufficiently large for the 16GB NVMM in our configurations. In the hash table, each entry is 9B and the number of entries is related to the deduplication ratio. Since the deduplication ratios range from 18.6% to 98.4% across the 20 applications[7], the storage overhead in the hash table is less than 9*(1-18.6%) < 8B/line. The storage overhead of FSM table is 1 bit/line. Hence, the total storage overhead of metadata is (4B+4B+8B+3bit)/256B $\approx$ 6.25% of the NVM capacity.

HASH MODE ENCRYPTION

Space-efficient Encryption Scheme

In general, there are two CPU-side memory encryption models, including direct encryption and counter mode encryption, to encrypt the data in the memory. In direct encryption, each cache line is encrypted by employing a block cipher algorithm, e.g., AES, when written back to the memory from the last level cache, and decrypted after being read from the memory. However, direct encryption incurs high decryption latency in the critical path of memory reads, thus significantly impacting the system performance. In comparison, in counter mode encryption, data decryption can be executed in parallel with memory read, thus reducing the decryption latency. Counter mode encryption generates a one-time pad (OTP) using a counter and encrypts/decrypts data by XORing the plaintext/ciphertext with the OTP. However, counter mode encryption[3,4] has to use the per-line counters to generate the one-time pads (OTP), incurring high metadata overhead. The per-line counter is 28 bits for each line[3].

To reduce the space overhead of metadata, we propose a hash mode encryption by replacing the counters in the counter mode encryption with the hash values of cache lines. Hash mode encryption is inspired by the convergent encryption[11], a traditional encrypted deduplication method. Convergent encryption uses the hash of the data as the key to encrypt the data in order to allow deduplication to be performed on encrypted data, while ensuring the data security. Different from convergent encryption, hash mode encryption uses a secret key, the line address and the per-line hash to generate the OTP. By reusing the hash store (storing the mappings of physical addresses to hash values) for deduplication, hash mode encryption does not need extra space to store the counters, and avoids the problem of counter overflow in existing counter mode encryption. Specifically, in counter mode encryption, when a counter of a memory line overflows, the entire memory needs to be re-encrypted using a new global key. The re-encryption of entire memory significantly degrades the system performance. In comparison, hash mode encryption does not need to re-encrypt the entire memory.

Security Analysis

To analyze the security of hash mode encryption, we need to consider two cases: 1) whether the data in different addresses are encrypted by different OTPs; 2) whether the data rewritten in the same address are re-encrypted by different OTPs.

For the first case, due to using the line addresses to generate OTPs, the hash mode encryption always encrypts the data in different addresses by different OTPs. For the second case, we first discuss the case of the different data rewritten in the same address, in which the hash value of data

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2886176, IEEE Micro

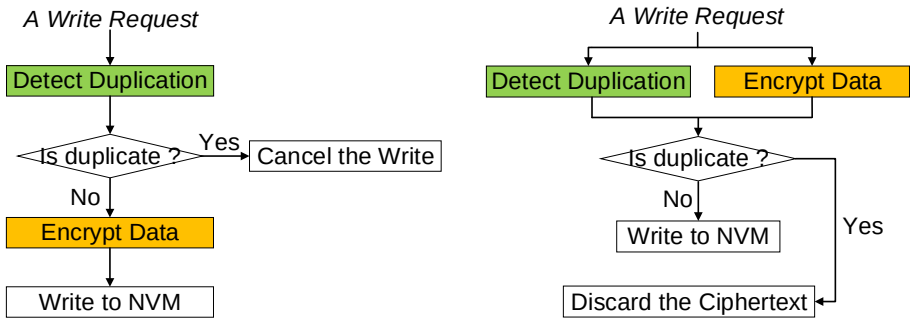
changes with the modified data in the same address. Hence, hash mode encryption generates different OTPs for the different data rewritten in the same address. Nevertheless, for the case of the same data rewritten in the same address, hash mode encryption will use the same OTP to encrypt the same data value since the same data have the same hashes, like the convergent encryption. Hence, hash mode encryption achieves the same security level as convergent encryption.

Metadata Overhead Comparison

As presented in previous section, DeWrite has the 6.25% metadata storage overhead. Due to employing the hash mode encryption, DeWrite achieves lower metadata overhead than the related work DEUCE[3]. DEUCE has two kinds of metadata. First, DEUCE needs a 1-bit flag per word in each line to indicate whether the word is modified. Each word is 16 bits in DEUCE and hence the storage overhead of the flags is 1 bit/ 16 bits = 6.25%. Second, DEUCE uses the counter mode encryption in which the storage overhead of the per-line counter is 28 bits/line. DeWrite only has the 6.25% metadata storage overhead from deduplication without metadata storage overhead from memory encryption since the hash mode encryption reuses the metadata from deduplication, to save the metadata space.

PREDICTION-BASED PARALLELISM

In the traditional secure NVMM solutions[3,4,5], the cache lines are first encrypted and then written to NVMM. DeWrite leverages deduplication to reduce the number of writes to the secure NVMM by eliminating the writes of duplicate cache lines. The direct way to perform deduplication on the encrypted NVMM is shown in Figure 2(a). For a cache line to be written to the NVMM, DeWrite first detects duplication. If the duplication exists in the NVMM, it cancels the write of the cache line and stores the address mapping relationship between the eliminated cache line and its duplicate in the NVMM into an address mapping table. Otherwise, the cache line is encrypted and written to the NVMM. The former reduces the write latency by canceling the write. The latter however increases write latency since the duplication detection and data encryption are executed serially in the critical path of the memory write. Thus the direct way is inefficient for applications where most lines are non-duplicate, such as bzip2 and vips.



(a) The direct way (b) The parallel way

Figure 2: Integrating deduplication and encryption.

To avoid the serialization of detecting duplication and encrypting data, we consider a parallel way to perform deduplication on encrypted NVMM. As shown in Figure 2(b), for a cache line to be written to the NVMM, cache line encryption and duplication detection are carried out in parallel. If no duplicates exist in the NVMM, the encrypted cache line is directly written to the NVMM. Otherwise, the encrypted cache line is discarded. However, for applications where most cache lines are duplicate, such as cactusADM and lbm, the parallel way becomes inefficient since the encryption is unnecessary and causes extra computation (and energy consumption) overhead from the AES circuit.

In summary, the direct way is efficient for the duplicate cache lines but inefficient for the non-duplicate cache lines. The parallel way has the opposite effects. Intuitively, the best solution is to use the direct way for duplicate cache lines and the parallel way for non-duplicate cache lines, respectively.

In order to identify whether a cache line is duplicate beforehand, we propose a simple yet effective prediction scheme by exploiting the duplication states of the most recent memory writes recorded by using a history window. Specifically, DeWrite maintains a history window for the whole main memory. The history window records the duplication states of the most recent cache lines written into main memory. If the most recent memory writes are mostly duplicate (or non-duplicate), the next memory write is predicted to be duplicate (or non-duplicate). The rationale comes from our observation that the duplication states of cache lines written to main memory have temporal locality, i.e., duplicate (or non-duplicate) memory writes are usually consecutive. Based on our evaluation, the duplication states of average 92% memory writes are the same as those of their previous ones. This observation can be interpreted as if a cache line written to main memory is duplicate (or non-duplicate), its next cache line written to main memory is also duplicate (or non-duplicate) with 92% probability. The history window records the duplication state of only the previous one memory write, achieving about 92% prediction accuracy. In order to improve prediction accuracy, we record the duplication states of the three most recent memory writes in the history window.

In summary, DeWrite maintains a history window for the whole main memory to predict whether a cache line to be written into NVMM is duplicate. If a cache line is predicted to be non-duplicate, DeWrite encrypts data in parallel with duplication detection to reduce write latency. Otherwise, DeWrite detects duplication without encrypting data to reduce computation overhead (and energy consumption) from encryption.

EVALUATION

Endurance

One of main objectives of DeWrite is to reduce the number of writes to NVMM and enhance its endurance by identifying and eliminating the writes of duplicate cache lines. As shown in Figure 5, we observe that DeWrite reduces on average 54% of whole-line memory writes across all applications. For some applications which contain a large number of duplicate writes, e.g, cactusADM, libquantum, lbm, and blackscholes, DeWrite even reduces more than 80% of whole-line memory writes. Nevertheless, the memory writes reduced by DeWrite (on average 54%) is less than the total number of duplicate lines existing in these applications (on average 58%), as shown in Figure 3(a). About 4% of write reduction is missed due to two reasons. First, due to the limited range of the hash table references, DeWrite fails to detect a small number of duplicate lines, i.e., on average 1.5%. Moreover, the dirty data evicted from the metadata cache incur on average 2.6% extra writes. The amount of metadata writes to the NVMM is small due to the high hit rate (over 98%) achieved by the metadata cache.

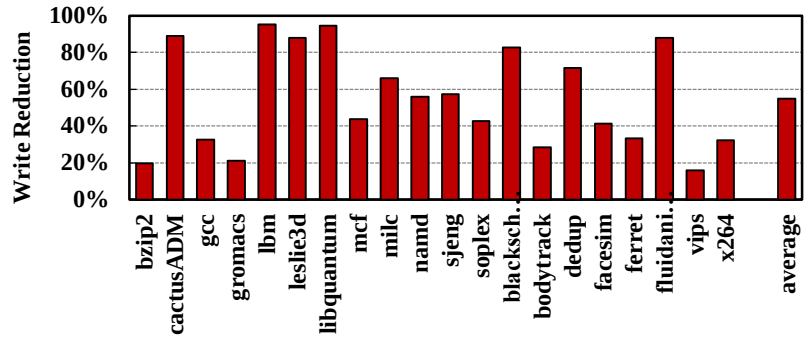
Performance

DeWrite reduces the write latency from two aspects. First, DeWrite removes the write latency of duplicate writes. Second, DeWrite eliminates duplicate writes, which also reduces the write latency of other non-duplicate writes to the same bank via decreasing their wait time. Moreover, DeWrite reduces the wait time of read requests to improve read performance by eliminating duplicate write requests. But to handle a single read request, DeWrite needs to first query the address mapping table and then read the actual data, which slightly increases the read latency. Nevertheless, the increased latency of accessing address mapping table is negligible, compared with the reduced read latency from eliminating duplicate writes.

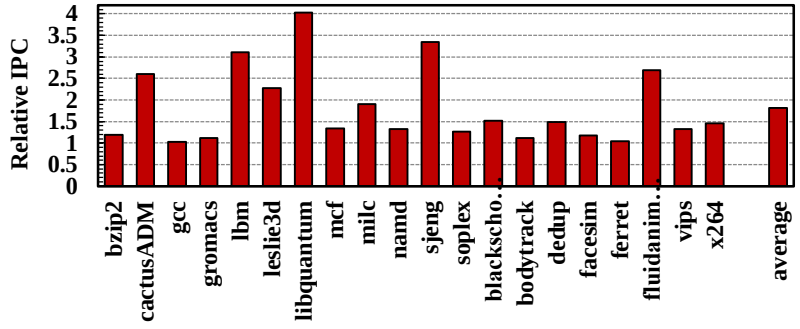
Due to the elimination of duplicate writes, the write and read latencies are reduced, which improves the overall performance of the system, i.e., instructions per cycle (IPC). Figure 3(b) shows

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MM.2018.2886176, IEEE Micro

the relative IPC of DeWrite normalized to the traditional secure NVM system. We observe that DeWrite achieves on average 82% IPC improvement across all applications.



(a) Write reduction



(b) Relative IPC

Figure 3: The experimental results on (a) savings on writes to NVMM and (b) The relative IPC compared with the traditional secure NVMM.

RELATED WORK

As both write endurance and security are important problems for NVMM, many schemes have been proposed to reduce writes in the encrypted NVMM. i-NVMM[12] proposed that the hot data are kept in the unencrypted form in the memory for improving the system performance and encrypted only when the system is powered down. However, i-NVMM fails to protect the memory against the bus snooping attack, since hot data are unencrypted through the bus. Unlike i-NVMM, DeWrite encrypts both hot and cold data written to NVMM and defends against both the stolen DIMM and bus snooping attacks. DEUCE[3] reduces the bits written to secure NVMM by only re-encrypting modified words (i.e., 2 bytes) in a cache line and keeping unmodified words in the last encrypted state. Based on DEUCE, SECRET[5] focuses on MLC NVMs and further reduces the re-encryption of full-zero words in a cache line. DeWrite is a line-level write reduction scheme, which is orthogonal to the subline-level (word-level) ones, like DEUCE and SECRET. Based on these subline-level write reduction techniques, DeWrite can further reduce over half of bit flips. Awad et al.[4] proposed Silent Shredder, which eliminates the writes of full-zero cache lines from data shredding. Unlike Silent Shredder, DeWrite eliminates all duplicate lines besides full-zero lines.

CONCLUSION

In this paper, we present a line-level write reduction scheme, called DeWrite, to enhance lifetime and performance of secure NVMM, along with a new encryption scheme, i.e., hash mode encryption. DeWrite addresses the challenges of performing in-line deduplication on secure NVMM and

integrating deduplication and NVM encryption to deliver high performance. Our experimental results demonstrate that DeWrite eliminates 54% of writes to secure NVMM, and speeds up memory writes and reads by 4.2 times and 3.1 times on average.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61772212.

REFERENCES

1. P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in Proceedings of the Annual International Symposium on Computer Architecture (ISCA), 2009.
2. S. Cho and H. Lee, "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance," in Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2009.
3. V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-efficient encryption for non-volatile memories," in Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2015.
4. A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent Shredder: Zero-cost shredding for secure non-volatile main memory controllers," in Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2016.
5. S. Swami, J. Rakshit, and K. Mohanram, "SECRET: smartly encrypted energy efficient non-volatile memories," in Proceedings of the 53rd Annual Design Automation Conference (DAC), 2016.
6. P. Zuo, Y. Hua, "SecPM: a Secure and Persistent Memory System for Non-volatile Memory", in Proceedings of 10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage), 2018.
7. P. Zuo, Y. Hua, M. Zhao, W. Zhou, AND Y. Guo, "Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes," In Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (2018).
8. X. Hu, M. Ogleari, J. Zhao, S. Li, A. Basak, Y. Xie, "Persistence Parallelism Optimization: A Holistic Approach from Memory Bus to RDMA Network", In Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (2018).
9. M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in Proceedings of the Annual International Symposium on Computer Architecture (ISCA), 2009.
10. "IBM z systems microprocessor optimization primer," https://ibm.co/1qeGrpc, 2018.
11. J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS), 2002.
12. S. Chhabra and Y. Solihin, "i-NVMM: a secure non-volatile main memory system with incremental encryption," in Proceedings of the Annual International Symposium on Computer Architecture (ISCA), 2011.

ABOUT THE AUTHORS

Pengfei Zuo is a PhD student in Huazhong University of Science and Technology. His research interests include non-volatile memory architecture and system, data deduplication system, key-value store and storage security. Contact him at pfzuo@hust.edu.cn.

Yu Hua is a professor in Huazhong University of Science and Technology. His research interests include cloud storage systems, non-volatile memory, big data analytics, artificial intelligence hardware and software infrastructure. Contact him at csyhua@hust.edu.cn.

Ming Zhao is an associate professor in Arizona State University. His research interests include cloud, big-data, and high-performance computing systems as well as operating systems and storage systems. Contact him at mingzhao@asu.edu.

Wen Zhou obtained his PhD degree from Huazhong University of Science and Technology. His research interests include non-volatile memory and storage security. Contact him at zhouwen@hust.edu.cn.

Yuncheng Guo obtained his Master degree from Huazhong University of Science and Technology. His research interests include non-volatile memory and data compression. Contact him at ycguo@hust.edu.cn.