February 16, 2017

Improved Security for Non-Volatile Main Memory

Stuart Haber
Hewlett Packard Enterprise

Pratyusa K Manadhata

Improved Security for Non-Volatile Main Memory

Abstract: A technique that improves security for non-volatile main memory in computer systems is disclosed. Some prior approaches that secure data between OS processes in such systems reduce the number of NVM write cycles by using encryption instead of "shredding" (zeroing out) physical memory pages between processes. However, in some circumstances, this solution can be less secure. The disclosed technique uses a pseudorandom function to change how the major counter is updated for a page that is to be shredded in order to increase security.

This disclosure relates to the field of non-volatile memories (NVMs).

NVMs suffer from the problem of limited write endurance: phase change memory can only tolerate 10 million to 100 million write cycles. NVMs also suffer from data remanence vulnerability: they retain data for a long time after a system is powered off, as compared to DRAM which loses data quickly, and thus are susceptible to unauthorized access. One solution to this vulnerability is to encrypt the data stored in NVMs. However, while effective in dealing with the vulnerability, memory encryption worsens the write endurance problem of NVMs. This is because a good encryption scheme has the property of diffusion: a change in one bit in the original data should change many bits in the encrypted data. Techniques to reduce the number of writes (for unencrypted data) in NVMs, such as Data Comparison Write (DCW) and Flip-N-Write (FNW), lose effectiveness due to diffusion, because these techniques are inspired by the observation that few bits will have their values changed after successive cache line writes.

Where main memory is NVM-based, an operating system (OS) may perform data shredding to avoid inter-process data leakage that occurs where a process accidentally or intentionally reads the old data of another process. Data shredding undesirably generates a large number of writes to NVM as it zeroes out each physical memory page before mapping it to a new process. As a result, one encryption-based technique reduces the number of writes by up to 40% by making a data page unintelligible when the page is allocated to another process, rather than zeroing it. When data in a newly allocated page is read, for software compatibility, the controller should skip the actual reading from NVM and instead should supply a zero-filled block to the cache. This may be achieved, as understood with reference to the Figure, by repurposing the initialization vectors (IVs) 20 used in a symmetric memory encryption mechanism 10. The IVs are manipulated to render data pages unintelligible and to encode the shredded state of a page. The IV consists of 128 total bits allocated in three parts: a cache line address (64 bits), a page specific major counter (56 bits) 30, and a minor counter for every cache line (8 bits) 40.

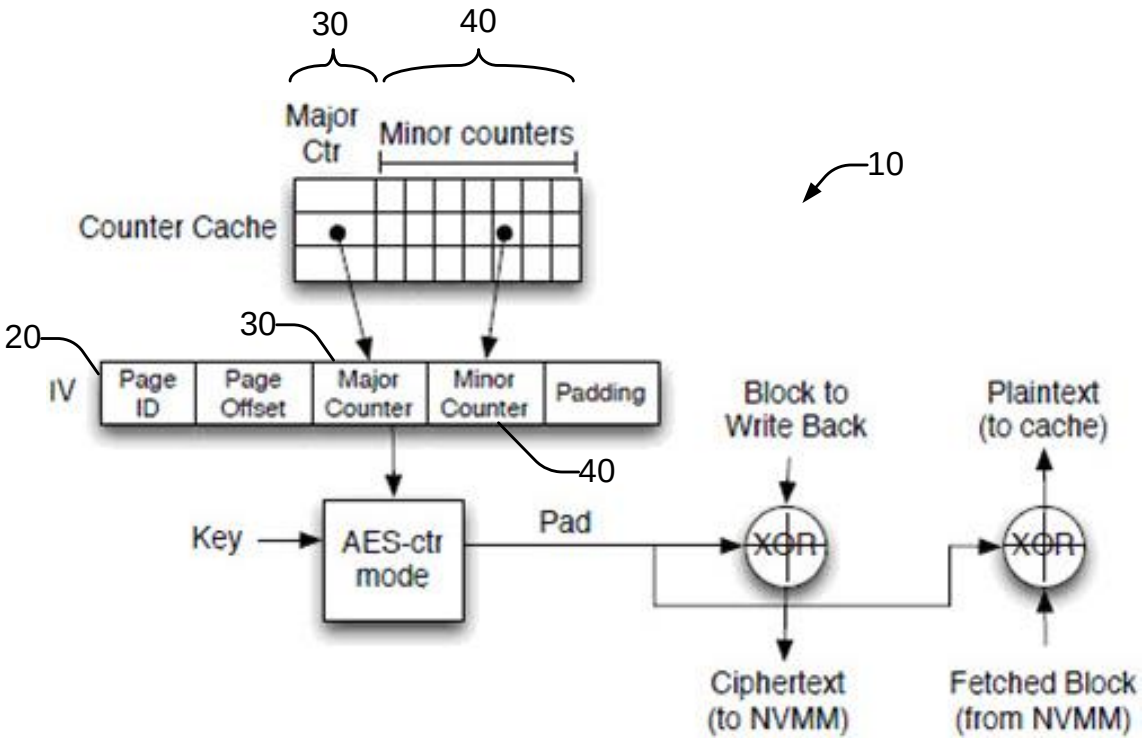
In one such solution, when a page is to be shredded, instead of explicitly writing zeros to the page, the major counter for the page is incremented by 1 and the minor counters are reset to 0. Because the IV used to encrypt the data is now lost, the data in the shredded page can't be decrypted. However, under certain circumstances, this approach can be less secure than the scheme of explicitly writing zeros to a memory page. In particular, if an attacker manages to obtain the encryption key, all that remains is to generate the IV in

order to reconstruct the shredded data. The attacker can get the encryption major counter by reducing the current major counter by 1. Then the attacker can "guess" the minor counter by considering all possible 8 bit minor counter values (128 attempts) and get the data of the older process.

The disclosed security improvement technique makes it computationally infeasible to guess the previous value of the major counter, and/or makes it considerably more difficult to try all possible values of the minor counter. This technique uses a family of pseudorandom functions, as can be specified using a MAC (message authentication code) scheme, a one-way hash function, or a block cipher to change how the major counter is updated for a page that is to be shredded. If the major counters are B bits long, a pseudorandom function F is fixed on B-bit strings with B-bit outputs. When a particular page is to be shredded, instead of simply incrementing its major counter, its value v is replaced by the value $F(v)$.

One implementation makes this single change, taking $B = 56$, and leaves all other operations unchanged. Another implementation increases the robustness by reducing the length of the cache-line address to 48 bits, freeing up 16 bits, and reallocating these bits to the counters. For example, all 16 extra bits can be allocated to the major counter, increasing its length to 80 bits, taking $B = 80$. Another choice is to allocate 8 extra bits each to the major and minor counters, increasing the major counter to $B = 64$ bits and the minor counter to 16 bits.

With the disclosed implementations, instead of merely decrementing the major counter for a recently shredded page in order to obtain its previous value, an attacker must now invert F on the current value of the major counter, which is a computationally difficult task. In addition, lengthening the minor counter increases the attacker's cost in trying all possible values as part of the attack. The disclosed technique can be implemented with any block cipher in any encryption mode that uses IVs.



Disclosed by Stuart Haber and Pratyusa Manadhata, Hewlett Packard Enterprise