计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190820
57(5):912-927,2020

安全持久性内存存储研究综述

杨 帆 李 飞 舒继武
(清华大学计算机科学与技术系 北京 100084)
(yangf17@mails.tsinghua.edu.cn)

Survey on Secure Persistent Memory Storage

Yang Fan, Li Fei, and Shu Jiwu
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract With the rapid development of computer technology, computer security and data privacy protection have always been the focus of academic and industrial. By providing hardware-assisted confidentiality and integrity verification, memory security mechanism helps guarantee the security of application code and data, and prevent them from malicious memory disclosure and modification. The emerging persistent memory delivers a unique combination of affordable large capacity and support for data persistence and provides high-bandwidth and low-latency data access. It can be placed on the memory bus like DRAM and will be accessed via processor loads and stores. However, due to differences in media characteristics, DRAM-oriented memory security mechanisms cannot function efficiently on persistent memory and even have availability issues. Therefore, a secure memory storage system based on persistent memory will bring new opportunities for the secure and efficient memory storage of big data. Firstly, for the write characteristics of persistent memory, the reasons for low-efficiency in applying the security measure against traditional volatile memory to persistent memory are analyzed, and related work is expounded. Secondly, for persistent memory storage, we analyze the problems that need to be considered to ensure the security of persistent memory in its whole life cycle, and introduce research work on guaranteeing the consistency between data and corresponding metadata for security. Finally, we conclude the challenges and compare the related work in building secure memory storage based on persistent memory, and share our views on future research.

Key words persistent memory; memory security; data encryption; integrity verification; crash consistency

摘 要 在计算机技术和互联网技术飞速发展的进程中,计算机安全防护和数据机密性保护一直是学术界和工业界关注的焦点.主流的内存安全机制通过提供硬件辅助的机密性与完整性验证,确保选定的代码在运行时的内存可信,达到数据保护、防止泄漏和遭篡改的目的.新型持久性内存可像 DRAM 一样放置在内存总线上,通过处理器 load 和 store 指令进行访问,此外,持久性内存能够提供大容量和数据持

收稿日期:2019-12-02;**修回日期**:2020-03-09
基金项目:国家重点研发计划项目(2018YFB1003301);国家自然科学基金重点项目(61832011);广东省科技创新战略专项项目(2018B010109002)
This work was supported by the National Key Research and Development Program of China (2018YFB1003301), the Key Program of the National Natural Science Foundation of China (61832011), and the Science and Technology Innovation Strategy Special Project of Guangdong Province (2018B010109002).
通信作者:舒继武(shujw@tsinghua.edu.cn)

久性支持,具有高带宽和低延迟的数据访问特性.然而,由于介质特性上的差异,面向 DRAM 的内存安全机制无法在持久性内存上高效运行,甚至存在可用性问题.因此,构建基于持久性内存的安全内存存储系统将为大数据的安全高效存储带来新的机遇.首先,针对持久性内存的写特性,分析了将面向传统易失内存的安全防护措施应用于持久性内存会引起额外开销的原因,并介绍相关降低开销的研究工作.其次,针对持久性内存的非易失性,分析了为保障持久性内存在其生命周期内的安全性所面临的问题与挑战,并介绍了数据及其安全元数据的一致性管理相关研究工作.最后,总结了构建面向持久性内存的安全存储系统面临的挑战,对相关工作进行综合比较,并提出下一步研究展望.

关键词　持久性内存;内存安全;加密;完整性验证;灾后一致性

中图法分类号　TP303

持久性内存具有大容量、高性能、低价格的特点,是 DRAM 潜在的替代品[1-5].持久性内存的出现改变了传统的内存层级,它可与 DRAM 同处于内存层次,支持字节粒度的访问,同时具备外存的非易失性,使得持久性内存存储成为可能.

持久性内存直接连接到内存总线,会遭受与 DRAM 相同的恶意攻击,如内存数据机密性攻击与完整性攻击.为了避免内存隐私数据被窃取,基于计数器模式的加密方法(counter mode encryption, CME)被广泛使用来防范内存数据机密性攻击[6-8].在 CME 中,每一个内存块都与一个独一无二的计数器相关联.为了防范内存数据完整性攻击,通常采用完整性验证的方法来确保从内存读取的数据与最近一次写入的数据相同[7-13];默克尔树(Merkle tree, MT)被用来确保内存数据的完整性,树状结构的消息认证码(message authenticated code, MAC)构建于整个内存之上,树根存储于安全区域内,无法被篡改.

直接将传统基于 DRAM 的内存安全保障架构移植到持久性内存之上并不能充分发挥原本内存安全保障架构的性能优势,甚至还会影响持久性内存系统的可用性,这主要由 2 个方面导致:

1) 持久性内存的写特性

由于 DRAM 具有很高的耐久性,面向传统 DRAM 的安全架构未考虑安全保障措施造成过多内存磨损的问题.然而,持久性内存的单元寿命有限,与写次数有关,对持久性内存加密与完整性验证会增加写操作的位翻转,加剧单元磨损,从而造成寿命降低[14-16].另外,相比于 DRAM,持久性内存的写带宽更低,而安全元数据(如加密计数器以及完整性验证树节点)的写回会占用数据写回的带宽,从而影响应用程序的执行性能.

2) 持久性内存的非易失性

传统 DRAM 内存数据在断电后快速丢失,面向 DRAM 的安全架构未考虑内存数据的持久性与恢复.持久性内存在断电后仍能保留数据,断电后的安全性也需要得到保障;与此同时,持久性内存支持内存数据的快速恢复,重启之后,对内存数据的机密性与完整性保障需要继续,即需要保障持久性内存在其整个生命周期内的安全性,而现有内存安全保障架构并不支持这一点.在现有的内存安全架构下,为了提高安全验证流程性能,频繁访问的安全元数据会被缓存到内存控制器内的安全元数据缓存或者最后一级缓存(last level cache, LLC)中[13].断电会造成缓存中的数据丢失,从而造成数据与安全元数据之间的不一致,主要分为 2 种:1)持久性内存加密过程中,数据与加密计数器之间出现不一致,异常断电可能会造成重启后解密失败;2)持久性内存完整性验证过程中,数据与完整性验证树之间出现不一致,异常断电可能会造成重启后完整性验证失败.

严格保障数据与安全元数据之间的一致性会带来高昂的运行时开销,这是由于:1)数据的写回会强制相关安全元数据写回,额外的持久性内存写操作占据了内存带宽;2)写操作需要等待安全流程完成之后才能完成,这增加了写操作的完成时间,使得 CPU 写队列更易于被填满,进而影响 CPU 的执行效率.此外,持久性内存应用会频繁使用缓存刷写操作,这些操作将写操作的延迟置于 CPU 执行的关键路径之上,从而严重影响系统的性能.

系统崩溃恢复时,安全元数据的恢复会严重影响系统的可用性.由于持久性内存的容量可达 6 TB[1],而安全元数据的内存存储开销正比于内存容量,对于大容量的持久性内存而言,恢复安全元数据时间可达数小时,严重影响了系统的可用性.

持久性内存在写特性、非易失性等方面呈现出与传统 DRAM 完全不同的特点,直接将现有的内存安全架构部署到持久性内存上并不能完全与持久性内存的特性匹配,也不能充分发挥原本内存安全保障架构的性能优势,甚至影响系统的可用性.为此,需要结合持久性内存特性,优化加密与完整性验证流程,同时针对存储灾后一致性问题重新设计硬件逻辑,以实现高效的安全持久性内存存储.

1 内存安全

在本节中,主要介绍面向 DRAM 的内存安全架构的相关内容.主要包括威胁模型、内存加密的方法、数据结构及流程,内存完整性验证的方法、数据结构及流程,内存安全领域的通用优化等.

1.1 威胁模型

在硬件辅助的内存加密与完整性验证安全系统里,系统资源划分为安全区与危险区.如图 1 所示,安全区称为可信计算基(trusted computing base, TCB),TCB 包括处理器片上资源,主要有 CPU 寄存器和缓存,可信计算基内的数据无法被窃取与篡改.危险区包括 CPU 片外所有资源,主要有内存总线以及内存等.

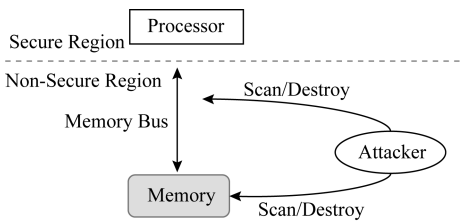

Fig. 1 Threat model[6]

图 1 威胁模型[6]

威胁模型里主要包括 2 种攻击:1)数据机密性攻击,指攻击者可以窃取内存总线及内存里的数据;2)数据完整性攻击,指攻击者可以篡改内存总线以及内存中的数据.完整性攻击分为 3 类:1)数据欺骗攻击(data spoofing attack),直接对数据块进行恶意篡改;2)数据拼接攻击(data splicing attack),交换 2 个有效数据块的内容;3)数据重放攻击(data replaying attack),将数据块重放回之前的旧版本.

1.2 内存加密

内存加密的目的是为了确保数据的机密性.任何从 LLC 替换出去的数据在进入内存总线之前都会被加密.由于内存读操作处于程序执行的关键路径之上,而解密操作处于内存读操作的关键路径之上,内存加密会带来高昂的内存读操作开销.CME 将解密操作从读操作的关键路径上剔除,被广泛应用于内存加密系统中[6-8].图 2 展示了数据加密与解密流程,在加密过程中,CME 将数据块与一次性密码本(one-time pad, OTP)进行异或来产生加密数据.密钥与初始向量(initialization vector, ***IV***)作为加密模块的输入,产生 OTP,其中 ***IV*** 由数据块的地址以及数据块对应的计数器组成.在解密过程中,当数据块从内存中读出时,与加密过程相同的 ***IV*** 会被生成,进而产生相同的 OTP,使用与加密过程相同的 OTP 对数据进行解密.CME 的核心在于将内存读操作与 OTP 的生成并行化,从而隐藏解密延迟.

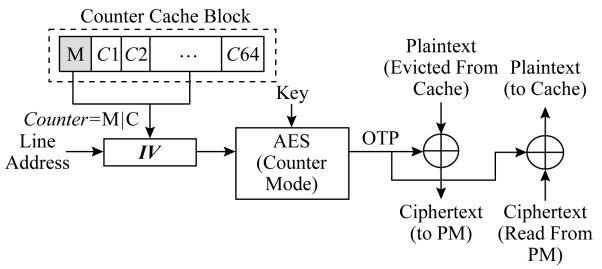

Fig. 2 Counter mode encryption[8]

图 2 计数器模式加密[8]

缓存与内存之间的数据传输通常是以 64 B 的粒度进行,因此 CME 将 64 B 的内存块与一个计数器相关联.目前在学术界广泛使用的一种计数器组织模式是将计数器分为主计数器与从计数器[7-8],从计数器通常是 7 b,与一个 64 B 的特定内存块相关联.主计数器通常是 64 b,由 4 KB 内存页内的所有 64 B 内存块共享.只有当从计数器溢出时,主计数器才会加 1,主计数器覆盖的所有内存块将被取到安全区进行重新加密.

计数器模式加密的安全性基于一个前提,即每个 OTP 都不会被重用,这需要保证 IV 的唯一性,主要包括 2 方面:1)空间上的唯一性,将不同地址的内存块映射到不同的计数器;2)时间上的唯一性,对同一内存块的每次内存写操作,都将该内存块对应的从计数器做加 1 操作.图 2 展示了计数器块的构成,对于一个 64 B 的计数器块(counter cache block),它包含 64 b 的主计数器(M)以及 64 个 7 b 的从计数器($C1$ 到 $C64$).

1.3 内存完整性验证

内存完整性验证确保从内存中读取的值与最近

写入内存的值相同.对于内存写回,计算写回数据的MAC值并一同写入内存,在内存数据读取的过程中验证其对应的MAC值即可检测数据欺骗攻击与数据拼接攻击.然而基于内存数据构建单层的MAC并不能检测重放攻击,攻击者可以将数据及其对应的MAC值重放回旧的一致版本.默克尔树被广泛使用来检测重放攻击[8-10],默克尔树维护以树状结构组织的分层MAC,将数据和计数器作为其叶节点,父MAC节点保护多个子MAC节点.图3展示了一个四叉默克尔树的结构,任意一个树内部节点都包含了4个孩子节点的MAC.当内存读操作发生时,沿默克尔树自底至上读取并检查数据块的MAC值直至树根,来验证所获取的内存块的完整性;当发生内存写操作时,更新树相应分支节点的MAC值;树根包含整个内存的信息,存储在TCB中的寄存器上,无法被攻击者破坏或重放;上述组织方式,保证了内存数据的完整性.

默克尔树片上存储开销较低,它仅需将根保留在处理器芯片内.但是,默克尔树构建于整个内存数据之上,其大小正比于内存容量,因此树节点可能会导致高昂的内存存储开销.目前最先进的内存完整性验证架构是BMT(Bonsai Merkle tree)架构[8],它是基于CME模式构建.如图4所示,它首先使用单层数据消息身份验证代码(data Hash message authenticated code, Data HMAC)来检测欺骗和拼接攻击,每个Data HMAC是通过将加密的数据块、计数器及数据块地址作为输入生成的散列值.其次,BMT架构基于加密计数器构建默克尔树来检测重放攻击,使用散列函数(如基于SHA-1的HMAC)计算出孩子节点的HMAC值,并将其存入父亲节点,HMAC密钥和BMT树根存储在TCB的寄存器里,以防止攻击者将BMT自顶向下重放.与基于整个内存数据构建MT不同的是,在BMT中,加密的数据块不直接受到默克尔树的保护,而计算Data HMAC的输入则包含受默克尔树保护的加密计数器,因此能够检测到重放攻击.BMT只基于加密计数器构建默克尔树,从而减少了树节点的内存存储开销,降低了树的高度,同时缩短了默克尔树的验证时间.

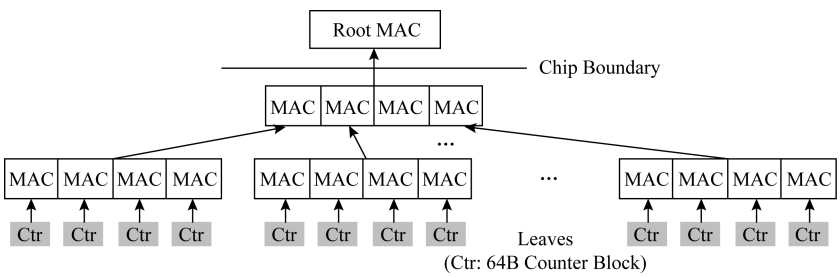

Fig. 3　The structure of Merkle tree[9]

图3　默克尔树[9]

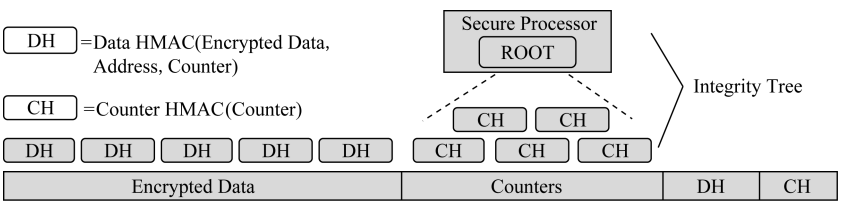

Fig. 4　Overall architecture of BMT[8]

图4　BMT的总体架构[8]

1.4　缓存安全元数据

缓存安全元数据可以提高CME和BMT的运行性能[8-9,11-13].以内存读取和解密为例,如果相应的加密计数器已经被缓存,则可以将OTP的生成和内存访问并行执行,从而隐藏了生成OTP的延迟.对于同一数据页(4 KB)中不同数据块(64 B),所有对应的加密计数器均被缓存到同一高速缓存行(64 B)中[8,13],而数据页中连续数据块的访问会命中到相同的计数器缓存行.因此,对于大多数工作负载而言,加密计数器能够表现出较高的缓存命中率.

默克尔树中经常访问和验证的树节点被缓存在芯片上[8-9,11-13].一旦在片上高速缓存中找到所需的树节点,即可完成数据块的完整性验证.由于缓存的树节点已经通过验证,并且可以确保其在TCB上的

安全性,因此可以像树根一样信任它.较高层中的树节点覆盖更大范围的内存数据,并具有较高的局部性,树根保护整个内存.

1.5 安全元数据存储开销

安全元数据存储开销正比于受保护的内存容量大小.重新设计安全元数据的组织结构[11-12],不仅有利于减少安全元数据的存储开销,而且能够提升安全元数据的缓存命中率,从而加速内存加密与完整性验证流程.

VAULT[11]通过在每个缓存行大小的树条目中容纳更多计数器来减少完整性验证树的深度,从而减少完整性验证树的存储开销.Morphable Counters[12]探索了加密计数器的访问模式,增加了每个缓存行大小的节点中容纳的计数器数目,根据加密计数器的使用模式动态更改计数器组织方式,降低由于计数器溢出导致的重新加密开销.

2 持久性内存安全

本节主要介绍由于持久性内存与 DRAM 不同的写特性,将传统内存安全架构构建于持久性内存在内存加密、完整性验证等方面所面临的挑战以及相关的研究工作.

2.1 持久性内存加密与完整性验证

如何高效处理写操作是设计持久性内存系统的关键挑战之一.与持久性内存读相比,持久性内存写操作不仅延迟更高,且能耗更大,存在耐久性的局限[2-3].内存写操作往往只有少量的位会被修改[15],持久性内存系统会利用这种特性来优化每次写操作中写到持久性内存的位数量.例如持久性内存系统采用一种称为数据比较写(data com-parison write, DCW)的技术[4],仅将高速缓存行中的修改位写入持久性内存介质中.如果超过一半的位被修改,则通过翻转数据位,进一步减少对持久性内存介质的位写操作,这种方式被称为 FNW[5](flip-n-write). FNW 将每次写操作的比特翻转位数限制为缓存行位数的一半.通过这些优化,每次写入存储器的平均位数减少到仅 10%~15%.鉴于持久性内存低于 DRAM 的写性能和有限的耐久性,这些减少比特翻转的方法对于实现持久性内存高性能和高耐久性至关重要.

由于持久性内存耐久性及其写操作的特性,针对持久性内存进行加密与完整性验证会带来一些新的挑战,主要分为 2 个方面:

1) 持久性内存加密带来的挑战

理想的加密算法都遵循雪崩效应[14],即使输入数据中只有 1 b 的改变也会导致加密数据中一半位被改变.尽管雪崩特性对于提供安全性至关重要,但它导致每次写入持久性内存中的位数大致为缓存行位数的 50%.如图 5 所示,其中写操作仅修改1 b,当进行加密操作时,2 个加密值之间的位差异可能是缓存行位数的一半.因此,加密将导致写入持久性内存的位数过多,从而使诸如 DCW 和 FNW 之类的优化失效,比特翻转的增多也会进一步造成更高的能耗.同时,基于持久性内存的应用会有频繁的刷写缓存行操作,这类操作会加速从加密计数器的溢出,当从计数器溢出时,需要重新将总量为 4 KB 的数据读进安全区,重新解密后加密再写回内存,造成运行时停滞.因此,需要针对持久性内存的写特性设计实现低写开销的加密模式.

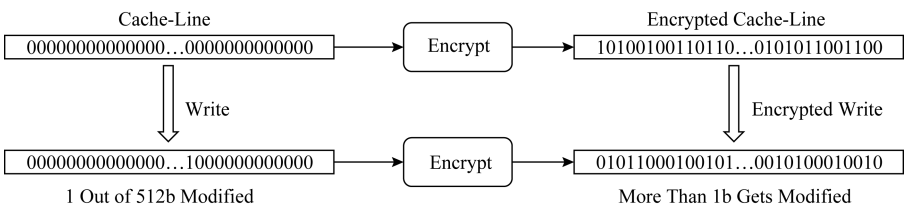

Fig. 5 Encryption incurs an increase in the number of bit flips for write operations

图 5 加密造成写操作的位翻转增多[15]

2) 持久性内存完整性验证带来的挑战

完整性验证树会增加额外的内存写操作,主要包含数据散列消息认证码和计数器散列消息认证码.其次,数据细微位变化可能造成其生成的散列消息认证码的大量位修改,从而增加散列值的比特翻转.最后,持久性内存容量过大,将导致完整性验证树层级过高,加速内存带宽消耗,而持久性内存带宽远低于内存,从而出现带宽资源瓶颈.因此,需要针对持久性内存的写特性设计实现高性能、低写开销的完整性验证模式.

2.2　相关工作

主要介绍3类相关工作:1)减少持久性内存加密场景下的写开销;2)减少持久性内存完整性验证过程的写操作、能耗、内存访问等开销;3)降低加密流程中计数器的溢出频率.

1) 减少加密场景下的写开销

DEUCE[15]是佐治亚理工学院于2015年提出的针对持久性内存的加密模式.对于一个缓存行内的大部分字(word)而言,写操作并不会修改它们,对此未修改部分不需要重新加密,只需要对修改部分的word重加密即可.基于这个想法,DEUCE提出细粒度的加密模式,针对缓存行里修改的word,DEUCE使用更新的加密计数器对其重新加密.解密阶段,对修改的word使用更新的加密计数器进行解密,对于未修改的word使用原本的加密计数器进行解密.

DEUCE加密模式采用了双计数器的加密方式.首先以word为粒度,对每一个缓存行内的word记录一个修改标志位;其次定义2个虚拟加密计数器LCTR(leading counter)和TCTR(trailing counter),其中LCTR值始终保持最新,通过屏蔽LCTR的几个最低有效比特(least significant bits, LSBs)来获取TCTR的值.若屏蔽2个LSB,则每4次写操作将更新TCTR使之与LCTR相同,这段期间称为期间间隔(epoch interval).在一段期间间隔内,TCTR值保持不变,LCTR值则正常更新.

对于写操作,在一个期间间隔内至少修改了一次的word,使用LCTR进行加密,未修改的word则使用TCTR进行加密.因此,对于未修改的word而言,其加密后的数据仍然保持不变.当TCTR与LCTR值相同时,重置缓存行中的所有修改标志位,并对整个缓存行重新加密.

对于读操作,如图6所示,DEUCE会维护2个OTP,分别由LCTR与TCTR生成,修改标志位(modified bit)决定解密word采用哪一个OTP,只有在期间间隔内至少修改一次的word才会使用OTP-LCTR进行解密,否则使用OTP-TCTR进行解密.

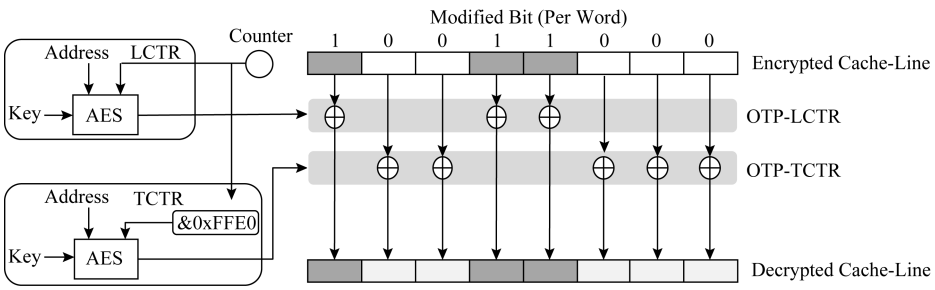

Fig. 6　Decryption of DEUCE[15]

图6　DEUCE解密操作[15]

SECRET[16]是匹兹堡大学于2016年提出的针对持久性内存的细粒度、低功耗的加密模式.与DEUCE思想类似,SECRET只对修改的word进行重新加密,此外,针对word的全0写,SECRET保持其最近的加密状态,避免对全0写操作的重加密开销,进一步减少了写操作比特翻转数量.

SECRET加密模式提出word粒度的加密模式,通过为每一个缓存行提供一个全局加密计数器,同时为缓存行中每一个word提供一个本地计数器(local counter, LC).当LC溢出时,将该缓存行对应的所有LC置0,同时将对应的全局加密计数器加1,对此缓存行重新加密.SECRET加密模式为每一个word提供1 b的zero-flag,用来记录word的状态,当word为全0写时,将对应的zero-flag置1,同时在写回过程中,将word以其先前的加密状态保留在持久性内存中.如图7所示,$Word_1$与$Word_3$被修改,因此其对应的LC自增1,$Word_2$未被修改,因此其对应的密文保持不变.对于$Word_3$而言,由于其为全0写,因此将对应的zero-flag置零,保持其密文不变.

SECRET加密模式还进一步研究了由于加密造成的能耗开销.对于持久性内存单元而言,不同模式的翻转造成的能耗不同(比如单元从01到10的能耗与00到01的能耗不同[17]).针对每一个加密写操作,SECRET加密模式将其与3种掩码进行异或,计算3种异或之后写操作的能耗,并选择能耗最低的作为最终数据写入持久性内存,从而降低写操作的能耗.

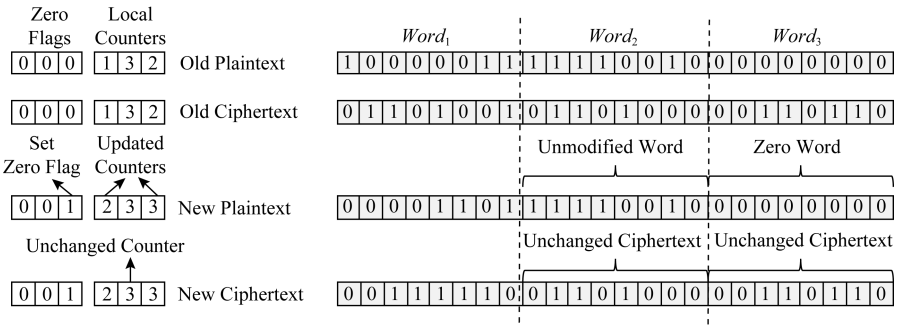

Fig. 7 Encryption of SECRET[16]

图 7 SECRET 加密操作[16]

上述工作提出的技术为位级别的持久性内存写减少技术,旨在减少内存加密中,写操作造成的位翻转数目.下述工作将内存加密应用于特定场景,如 data shredding,或与其他存储技术相结合,如 data deduplication,旨在减少持久性内存写次数.

Silent Shredder[18]是惠普实验室于 2016 年提出的持久性内存写减少技术.为了避免进程之间的数据泄漏,在将进程的内存页分配给另一个进程之前将其清零(data shredding).Silent shredder 技术利用计数器模式加密中使用的初始向量 ***IV***,在进行数据清除时修改对应内存页面的 ***IV***,从而避免了对持久性内存页面的写操作.

DeWrite[19]是华中科技大学于 2018 年提出的针对持久性内存加密的写减少技术.DeWrite 将加密和高效的重复数据删除(data deduplication)集成到持久性内存系统中,对重复数据删除元数据和加密元数据进行协同设计,减少了加密计数器的空间开销;此外,将加密与重复数据删除操作并行执行,进一步提升性能.

2) 减少完整性验证过程的开销

ASSURE[20]是匹兹堡大学于 2017 年提出的针对持久性内存的低写开销的完整性验证模式.与细粒度的持久性内存加密模式思想类似,ASSURE 完整性验证模式采用基于细粒度的加密,采取 word 粒度的完整性验证,只对修改的部分做散列,未修改部分填充 0,并将得到的结果合并.这种方法减少由于散列扩散(少量数据位修改造成其对应散列值的大量位变化)造成的额外写操作比特翻转.

ASSURE 完整性验证模式将原始的加密缓存行拆分为 2 个中间消息(intermediate message, IM).IM 具有与缓存行相同的长度和分区边界.第 1 个(第 2 个)IM,IM_1(IM_2)由修改(未修改)的 word 构成,未修改(修改)的 word 清 0.然后将 IM_1(IM_2)作为输入提供给散列函数,生成中间 HMAC IH_1

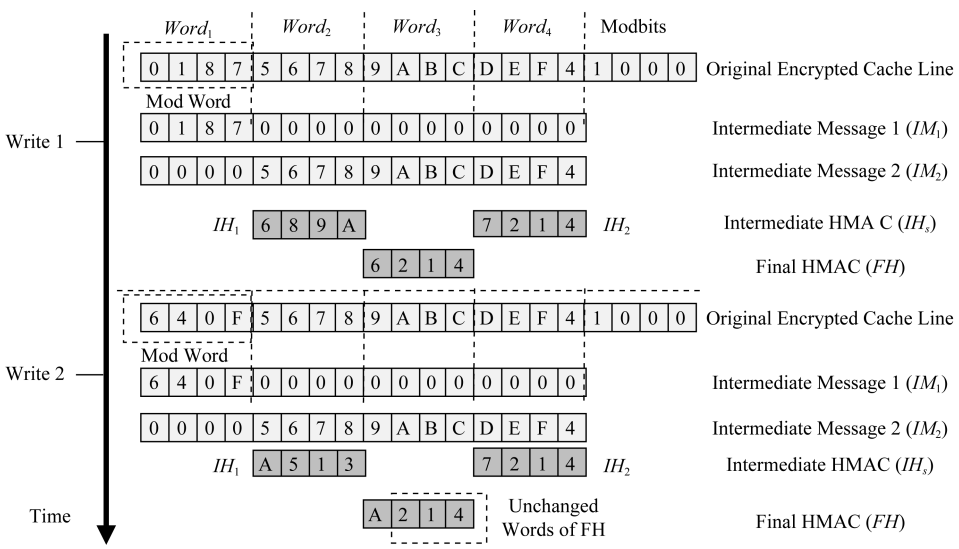

Fig. 8 The MAC computation of ASSURE[20]

图 8 ASSURE 中 MAC 值的计算操作[20]

(IH_2).与 IM 相似,IH 也按 word 粒度进行分区.最终的 HMAC(final HMAC, FH)由 IH_1 和 IH_2 对应的 word 构成.如图 8 所示,对于 2 次写操作,只有 $Word_1$ 被修改,首先设置修改标志位,未修改部分置 0,生成 IM_1,类似地将修改部分置 0,生成 IM_2,分别将 IM_1 和 IM_2 作为散列函数输入,得到各自的 IH 后,整合得到 FH.由于 2 次写操作只修改了 $Word_1$,因此 2 次写操作对应的 FH 只修改了 $Word_1$ 对应的区域,从而减少了散列函数造成的比特翻转.

ASSURE 完整性验证模式还进一步研究如何减少完整性验证的内存访问开销,提出了双根默克尔树,基于数据的冷热程度构造完整性验证树,通过统计各个 MT 组的访问数量,决定下一个 CPU 执行期间的冷热分组,从而将 MT 分成更少树层级的热组与更多树层级的冷组.由于基于热数据构建的完整性验证树的层级远小于基于整个持久性内存构建的完整性验证树,从而降低热数据的完整性验证开销.

3) 降低加密计数器的溢出频率

ACME[21]是匹兹堡大学于 2018 年提出的持久性内存加密模式,利用持久性内存磨损均衡算法来降低加密计数器的溢出频率.ACME 加密模式采用了 region-based start-gap 磨损均衡算法.如图 9 所示,首先将加密计数器与其对应的物理地址一一绑定,当对某一个逻辑地址的更新达到指定次数时,触发磨损均衡算法,从而将该逻辑地址映射到其他物理地址上,如图 9 所示,逻辑地址 D 对应的物理地址从 40 重新映射至 50.触发磨损均衡后,逻辑地址对应的加密计数器也将发生改变,此方法将部分加密计数器的频繁更新均摊到整个区域内的计数器上,减少了加密计数器的溢出频率.

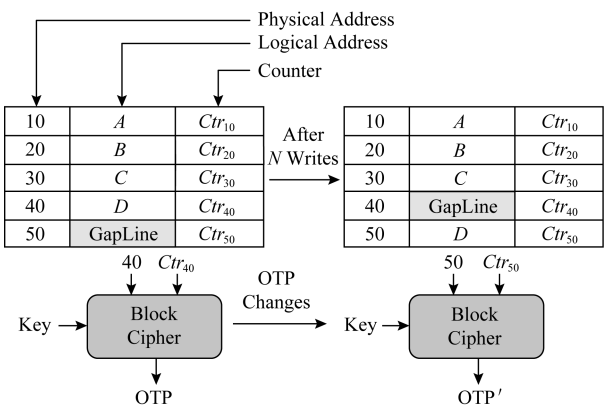

Fig. 9　The impact of wear-leveling on counter mode encryption in ACME[21]

图 9　ACME 中磨损均衡对计数器模式加密的影响[21]

3　安全持久性内存一致性

第 2 节主要介绍了构建安全持久性内存时,减少写操作开销的一系列工作,这些工作未考虑持久性内存的非易失性.本节主要介绍由于持久性内存的非易失性,将传统内存安全架构构建于持久性内存之上所面临的挑战,主要包括:安全持久性内存灾后一致性的起因、持久性内存加密中的灾后一致性问题及相关工作、持久性内存完整性验证中的灾后一致性问题及其相关工作、安全持久性内存灾后恢复方法.

3.1　安全持久性内存灾后一致性起因

传统 DRAM 具有易失性,断电后内存数据会丢失.持久性内存具有非易失性,断电后数据仍然保留,因此攻击者能够直接获取断电后内存里的数据.除此之外,要想在系统重启之后能够持续正确地进行安全保障,就需要保证安全元数据与数据之间的一致性,包括 2 个方面:1)数据与加密元数据之间的一致性,若不一致会导致解密失败;2)数据与完整性验证元数据之间的一致性,若数据与其散列消息认证码不一致,则数据完整性验证不通过.因此,要保障持久性内存在其生命周期内的安全性,必须保障安全持久性内存的灾后一致性.

3.2　持久性内存加密中的灾后一致性

图 10 展示了系统故障可能导致的数据和加密计数器不一致的情况.对持久性内存的每次写访问均包含 2 个单独的写请求,一个写请求作用于加密数据,另一个写作用于加密计数器.如果在数据写到达持久性内存之后且在计数器写入之前发生系统故障,则在系统重启时将观察到滞后的加密计数器值,从而导致原始数据丢失,如图 10(a)所示.如果加密计数器到达持久性内存后发生故障,但数据尚未持久化,也会发生类似的不一致情况,如图 10(b)所示.

要保障持久性内存加密的灾后一致性,主要面临了 2 个挑战:1)如何保障数据与其对应加密计数器的原子写回;2)如何减少由于保障原子性带来的开销,主要包括 2 个方面:1)写放大,数据的写回会造成额外的加密计数器写回;2)性能开销,原子性保障会增加写操作的延迟,而写操作延迟增加会导致

CPU 写队列阻塞频率上升,进而影响 CPU 执行时间.此外,基于持久性内存的应用会使用缓存刷写指令(如 clwb,mfence 等),该指令将写操作置于 CPU 执行关键路径之上,如图 11 所示,在非加密情况下,写回操作到内存控制器即可完成,而在加密情况下,写回操作必须等待加密灾后一致性保障完成之后才能返回,这增长了写操作的关键路径,从而影响了系统性能[23].

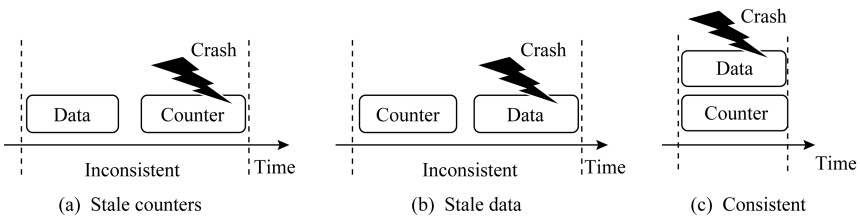

Fig. 10 Crash consistency in encrypted persistent memory[22]

图 10 持久性内存加密中的灾后一致性[22]

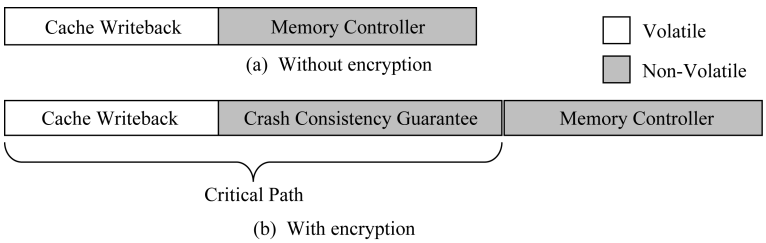

Fig. 11 Write latency with and without encryption

图 11 加密与非加密情况下的写延迟

SCA[22](selective counter-atomicity)是弗吉尼亚大学在 2018 年提出的针对持久性内存加密的一致性保障机制.SCA 机制在内存控制器中增加额外的写队列来存储加密计数器,数据发往内存控制器中的数据写队列,计数器被发往计数器写队列.Intel 提供的硬件异步 DRAM 自刷新(asynchronous DRAM refresh, ADR)机制可确保在电源故障的情况下,使用备用电源将缓存在内存控制器写队列中的所有写请求写回到持久性内存.通过扩展 ADR 机制,使其支持数据写队列和计数器写队列,并确保在电源故障时,只有在写队列中同时包含数据和相关计数器的条目才能写回持久性内存.为了跟踪数据及其计数器,SCA 机制向每个数据写队列和计数器写队列条目添加了一个额外的就绪位.仅当相应的写队列已接受数据和计数器写时,才设置 2 个写队列中的就绪位.为避免系统故障影响设置 2 个写队列的就绪位的原子性,该操作还受到 ADR 机制的保护.

SCA 机制提出的原子写入包括 3 个步骤:1)内存控制器将加密的数据发送到数据写队列,同时,加密引擎将关联的计数器缓存行发送到计数器写队列.2)当数据写到达数据写队列时,内存控制器检查计数器写队列是否具有关联的计数器条目.如果是,则将 2 个条目中的就绪位都设置为 1.否则,就绪位保持为 0.当计数器到达计数器写队列时,内存控制器执行相同的步骤.3)2 个写队列仅放行设置了就绪位的条目,所有未就绪的条目将保持阻塞状态,直到其就绪位被置位.在系统故障期间,2 个写队列仅将就绪条目写入持久性内存中.

图 12 展示了 counter-atomicity 写一个物理地址 0X100 的流程.步骤①处理器对地址 0X100 发出 counter-atomicity 的写请求,持久性内存协调器(步骤②)以及加密引擎(步骤③)会收到这个请求.步骤④若 0X100 地址对应的计数器(地址 0X200)缓存命中,则加密引擎更新相应计数器,生成 OTP,同时将最新的计数器发送至计数器写队列.步骤⑤持久性内存协调器将数据与 OTP 异或,将生成的密文发送至数据写队列.步骤⑥数据写队列接收到数据后检查相应的计数器写队列,由于在计数器写队列中未找到对应的计数器条目,因此其就绪位置 0.步骤⑦计数器写队列接收到计数器条目,同时检查数据写队列中是否有相应的数据条目.步骤⑧由于此时数据和相应计数器都处于对应的写队列中,内存控制器将条目设置为就绪,写操作完成.

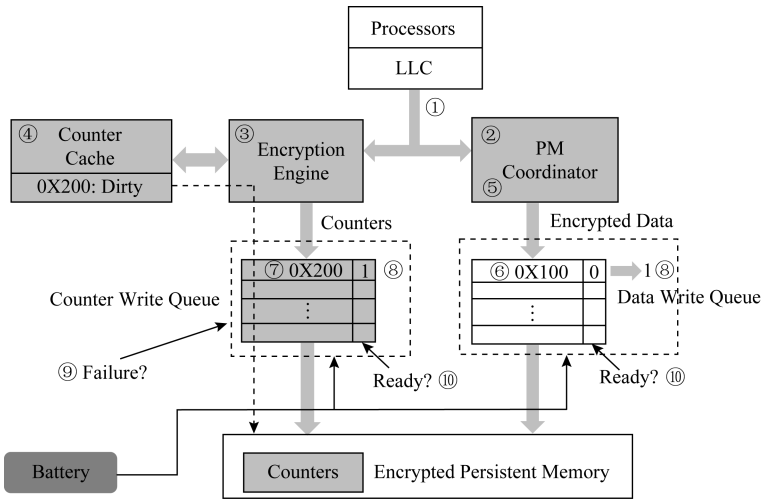

Fig. 12　Counter-atomicity write process in SCA[22]

图 12　SCA 中 counter-atomicity 写操作流程[22]

在系统崩溃时,步骤⑨触发 ADR 机制,计数器与数据写队列开始写回滞留在队列中的条目.步骤⑩写队列检查就绪位,并只写回就绪位置 1 的条目,确保数据与相应计数器在内存中的一致性.

SCA 机制还充分利用事务语义,结合事务语义降低 counter-atomicity 导致的开销.基于持久性内存的应用程序通常采用事务接口来保障数据的灾后一致性.例如使用撤消日志记录或者重做日志记录,日志记录保证了即使在更新期间出现故障,也可以恢复数据的一致状态[24-26].所有这些机制都通过维护 2 个版本的数据来确保灾后一致性.例如,日志记录机制在日志中维护一个版本,在原始数据结构中维护另一个版本,并确保在任意时间点仅修改其中一个版本.在修改其中一个版本时,如果发生任何崩溃,则利用另一个未修改版本恢复一致状态.由于已经修改的数据版本在恢复中不起作用,因此不需要针对这个版本严格保障数据与其加密计数器之间的一致性.以基于重做日志的事务为例,执行分为 3 个阶段:1)准备阶段,创建日志项,对数据进行备份;2)修改阶段,对数据进行就地修改,由于备份中已经存在数据一致性的版本,因此对数据的修改并不会影响数据的可恢复性;3)提交阶段,数据修改完成后,使在准备阶段创建的备份日志条目无效,并将新的修改状态标记为当前一致状态.

表 1 展示了事务执行不同阶段是否需要严格保障数据与其对应加密计数器的一致性.在准备阶段,通过修改日志来创建数据的备份,因此日志不能用于一致恢复,而原始数据未经修改,处于一致性的状态.在准备阶段,修改日志并不会影响数据的可恢复性,因此不需要严格执行 counter-atomicity.同样,在修改阶段,日志中的备份副本是一致的,可以用于将数据恢复至一致的版本.因此,在修改阶段,对数据的写操作不会影响数据的可恢复性,也不需要严格执行 counter-atomicity.另一方面,在提交阶段的写入会使备份日志条目无效.提交阶段将一致状态从日志切换到已修改的数据,由于此阶段的写操作会标记在事务恢复过程中应该使用日志还是数据,因此会影响崩溃后数据的一致性状态,所以此阶段的写操作必须严格执行 counter-atomicity,否则在恢复过程中可能使用错误的版本.需要注意的是,在从一个阶段转换到另一个阶段之前,需要保证数据与其计数器之间的一致性,将相关的计数器从缓存刷写回持久性内存.SCA 机制通过结合事务特性,在不影响灾后一致性的前提下,仅对一部分写操作强制执行 counter-atomicity,提高了事务系统的性能.

Table 1　The Consistency States Affecting Counter-atomicity in Different Stages of a Transaction with Undo-logging[22]

表 1　重做日志事务中不同阶段的一致性状态对 counter-atomicity 的影响[22]

Stage	Backup	Data	Counter-atomicity
Prepare	Inconsistent	Consistent	Unnecessary
Mutate	Consistent	Inconsistent	Unnecessary
Commit	Unknown	Unknown	Necessary

Osiris[27]是中佛罗里达大学在 2018 年提出的针对持久性内存加密的一致性保障机制.它使用纠错码(error correction code, ECC)来检测滞后的加密计数器,在系统恢复期间将加密计数器恢复至与

数据一致的版本,从而放松系统运行时数据与加密计数器的原子性保障.当将 ECC 应用于明文并且将得到的 ECC 位与数据一起加密时,ECC 位可以为加密计数器提供完整性检查,若计数器值与数据不匹配,则解密后明文数据会与其 ECC 位不匹配.

具体而言,在计数器模式的加解密中,使用方法解密数据:$\{X,Z\}=E_{key}(V_X)\oplus Y$,其中 Y 是密文,V_X 是 X 对应的 IV,$E_{key}(V_X)$对 V_X 利用 key 进行加密,Z 则是由 $F_{ECC}(X)$计算得来.在常规系统中,若 $F_{ECC}(X)!=Z$,可认为 X 或 Z 发生错误;在计数器模式的加解密中,还可能是解密过程使用了滞后的计数器.

在运行过程中,Osiris 架构部署了止损机制,当一个计数器缓存行更新 N 次时,将其写回持久性内存.因此在恢复过程中,可尝试使用所有可能的 N 种 IV 解密数据,与数据匹配的 IV 将在$[V_X+1, V_X+N]$范围中,其中 V_X 是存储在持久性内存中的滞后 IV.在 V_X 成功解密该块时停止,即所得的 ECC 值与预期的 ECC 匹配($F_{ECC}(X)=Z$),恢复出一致的计数器数据.

SuperMem[28]是华中科技大学在 2019 年提出的针对持久性内存加密的一致性保障机制,采用了 write-through 的加密计数器缓存,每次更新计数器的同时,将计数器写回内存控制器的写队列中,还修改了缓存刷新指令,增加额外的寄存器,以保障数据写与计数器写之间的原子性.如图 13 所示,在 CPU 发出刷新指令 Flu(A)时,内存控制器首先从缓存中读取地址 A 对应的计数器(Read(Ac)),然后增加计数器值(Ac++),使用更新后的计数器来加密地址 A 对应的数据(Enc(A)),同时将相关的计数器存储到寄存器中(Sto(Ac));加密完数据之后,将密文添加到寄存器中(Sto(A)),最后将密文与对应计数器添加到写队列中(App($Ac+A$));由于 ADR 机制的支持,写队列中的条目最终会写回持久性内存,因此始终保障了持久性内存中数据及其对应的计数器之间的一致性.

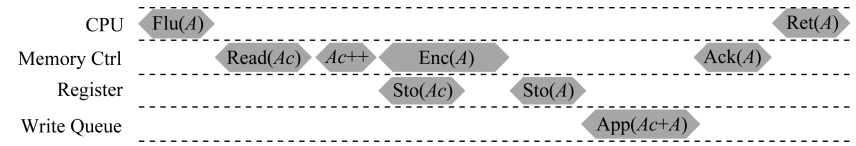

Fig. 13 The sequence that the memory controller deals with a cache line flush in SuperMem[28]

图 13 SuperMem 里缓存刷新指令的流程[28]

由于每次数据写操作会导致额外的计数器写,SuperMem 机制提出计数器写合并优化.在基于计数器的加密模式中,相邻 64 个 64 B 的内存块数据对应同一个 64 B 的计数器缓存行.基于此,当从计数器缓存中逐出的计数器缓存行到达内存控制器写队列时,检查写队列中的写条目是否具有与新到达缓存行相同的物理地址.如果是,则将这些缓存行合并为一次写操作,减少计数器写次数.进一步地,SuperMem 机制还探索了计数器在 bank 级别的并行性,通过将数据及其计数器分散到不同的内存 bank 中,加速数据与计数器的写入,从而提升系统性能.

cc-NVM[29]是清华大学在 2019 年提出的面向安全持久性内存的一致性保障架构.针对加密计数器的一致性保障问题,cc-NVM 架构在运行时并不对每次写操作都严格保障数据与其对应加密计数器的原子性,相反地,cc-NVM 架构采用 write-back 的安全元数据缓存写回策略,充分利用了安全元数据缓存.

cc-NVM 架构利用 BMT 中现有的数据散列消息认证码来检测数据与其加密计数器之间的一致性,在异常崩溃后,通过数据散列消息认证码使断电造成的不一致的数据与加密计数器恢复一致,从而减少了运行时加密计数器写开销与保障持久性内存加密一致性的性能开销.

数据散列消息认证码的计算为 Data HMAC=Hash(address,data,counter),在恢复过程中,首先计算崩溃后的数据与其加密计数器的散列值,将散列值与内存中存在的散列值比较,若相同,则数据与计数器一致;若不同,则计数器增 1 后继续比较,流程如图 14 所示:

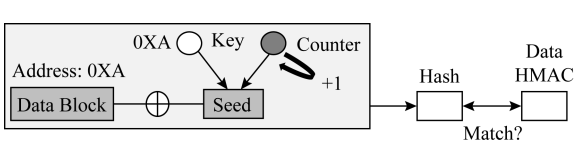

Fig. 14 The process of counter recovery in cc-NVM[29]

图 14 cc-NVM 架构中恢复计数器的流程[29]

3.3　持久性内存完整性验证中的灾后一致性

相比于持久性内存加密而言,持久性内存完整性验证树的一致性保障更为复杂.为了保障异常断电重启后,完整性验证能够正确执行,运行时完整性验证树的更新需要更新至树根,主要流程为:数据写回会修改其对应的加密计数器(BMT 叶子节点),加密计数器的修改会导致重新计算其散列值并存入父亲节点,依次计算散列值直至更新至根节点,其中根节点使用持久性寄存器存储于 TCB 区域.因此,每一次数据内存写都包含一个数据写及一系列直至根节点的安全元数据写操作.

图 15 展示了系统故障可能导致的数据和完整性验证树的不一致.若分支上的所有树节点未能同时抵达持久性内存,任意一个树节点出现滞后,都会造成树本身的不一致,即崩溃重启后无法正确进行完整性验证,如图 15(a)所示.另外,由于树根节点处于安全区域,而树内部节点和叶子节点处于非安全区.TCB 区的根与持久性内存中的树节点未能原子更新也会造成不一致现象,如图 15(b)所示.

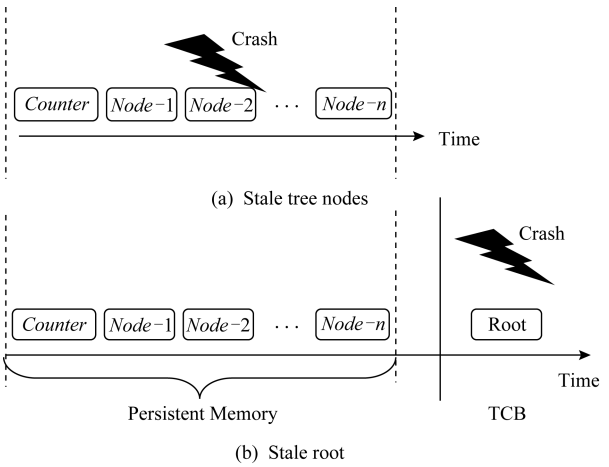

Fig. 15　Crash consistency in authenticated persistent memory

图 15　持久性内存完整性验证中的灾后一致性

保障持久性内存完整性验证的灾后一致性,主要面临 2 个挑战:

1) 如何保障数据及其对应的一系列树节点的原子写回.由于树根存储于安全区,而其他树节点存储于非安全区,因此该一致性保障主要包括 2 个方面:①数据与树相应分支上的所有非根节点之间的一致性;②树根与其他树节点跨区域之间的一致性.

2) 如何减少由于保障原子性带来的开销,主要包括 2 个方面:①写放大,数据的写回会造成额外的一系列树节点的写回;②性能开销,原子性保障会增加写操作的延迟,由于树节点的更新是自底向上依次进行的,其造成的写延迟正比于树层级,而写操作延迟增加会使得 CPU 写队列阻塞频率增加,进而影响 CPU 执行时间.不仅如此,基于持久性内存的应用会使用缓存刷写指令(如 clwb,mfence 等),该指令将写操作置于 CPU 执行关键路径之上,写操作延迟的增加进一步降低了系统性能.如图 16(a)所示,在没有完整性验证情况下,写回操作到内存控制器即可完成.然而,在完整性验证情况下,图 16(b)所示,写回操作必须等待一致性保障完成之后,即树根更新之后才能返回,这极大增长了写操作的关键路径,影响系统性能[23].

严格更新策略被广泛应用于持久性内存完整性验证的灾后一致性保障中[27,30-32].对于严格更新策略,任意数据写回内存,都会从完整性验证树的叶子节点更新至树根.通过利用持久性寄存器,保障数据、叶子节点和树根的原子性更新,而不考虑树的内部节点.如图 17 所示,首先将数据(步骤①)、叶子节点(步骤②)以及树根(步骤④)的更新值全部保留在持久性寄存器中,然后将寄存器中的值依次拷贝到内存控制器的写队列(步骤⑥⑦).引入 DONE_BIT 作为原子性更新流程中的分界线,仅在数据、计数器以及树根的更新写入持久性寄存器后设置 DONE_BIT(步骤⑤),并在所有持久性寄存器值复制到内

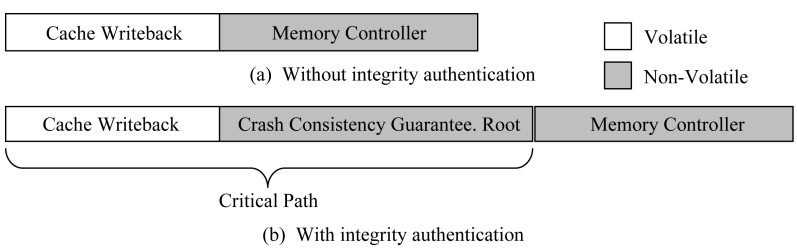

Fig. 16　Write latency with and without integrity authentication

图 16　完整性验证与没有完整性验证情况下的写延迟

存控制器的写队列,以及更新完 TCB 中的树根之后清除 DONE_BIT(步骤⑨).这种方案在本质上类似于重做日志记录,通过使用片上持久性寄存器来存储数据的一致性的版本.

在崩溃后恢复时,利用叶子节点恢复树根,然后将其与安全区内存储的持久性树根做比较,若相同则恢复成功.这种完整性验证的一致性保障主要存在 2 个缺陷:1)每次内存写操作,必须等待完整性验证树更新至树根节点,而父节点必须等待其子节点的散列值计算完成才能继续向上计算,因此会造成极高的写延迟;2)由于一致性保障未考虑内部节点,一旦恢复失败,由于无法定位出错数据块,任意数据块的损坏都会造成整个持久性内存数据丢弃,引起单点故障问题.中佛罗里达大学 2019 年在 Triad-NVM[31] 中提出增加持久性寄存器来存储树分支上的所有树节点来避免单点故障问题(改变图 17 的步骤③⑧),然而这种方式仍会引起极高的写放大,并且未解决写延迟高的问题.

清华大学在 2019 年提出的 cc-NVM 架构[29] 利用内存控制器中的持久性区域(写队列)进行原子性保障,并采用懒惰更新的完整性验证树更新策略,降低运行时的完整性验证一致性保障开销.

cc-NVM 架构采用延迟蔓延的 BMT 更新策略,通过引入记录一段时期写回次数的持久性寄存器 N_{wb},使得每次更新只更新至缓存的树节点,而不更新至树根,极大减小了写操作延迟.如图 18 所示,正常数据写回流程为步骤①～④.为了保障完整性验证树的一致性更新,cc-NVM 架构引入基于 Epoch 的安全元数据更新方式,通过利用内存控制器内的

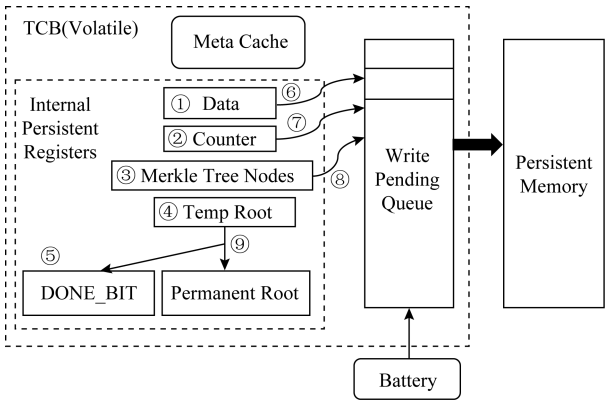

Fig. 17 Process of consistency guarantee based on strict updating strategy in BMT[32]

图 17 BMT 中基于严格更新策略的一致性保障流程[32]

持久性区域保障一个 Epoch 内所有脏安全元数据的原子性转换,在转换开始时内存控制器写队列阻塞所有安全元数据(步骤⑤),在原子性转换过程中将更新从叶子节点蔓延至树根 $ROOT_{new}$(步骤⑥),当所有脏元数据抵达写队列后,写队列放行安全元数据写操作(步骤⑦).由于 ADR 机制的保障,写队列中的条目在步骤⑦之后一定能够写入持久性内存,因此保障了系列树节点的原子性更新.同时,cc-NVM 架构采用双根的方式保障 2 个区域(可信计算基与持久性内存)数据的一致性,在原子性转换的最后,将 $ROOT_{new}$ 赋值给 $ROOT_{old}$(步骤⑧),从而始终保证至少有一个树根与持久性内存中的完整性验证树一致.由于 cc-NVM 架构考虑了树内部节点的一致性,因此避免了出现单点故障,但破坏原子转换期间修改的数据,仍会造成部分内存数据的丢失.

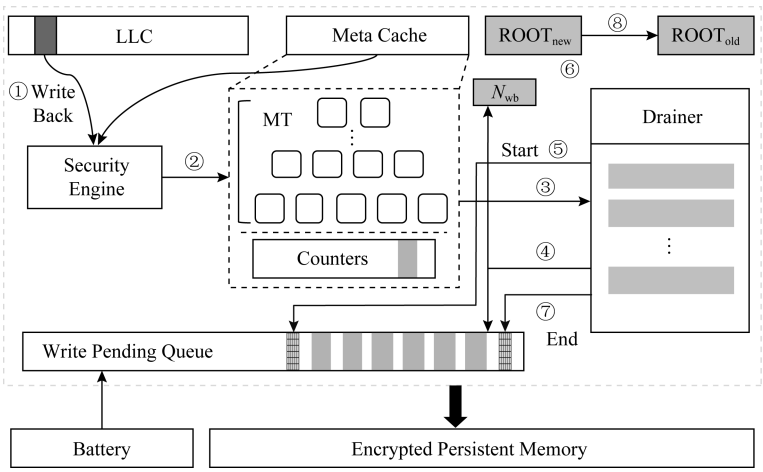

Fig. 18 Overall architecture of cc-NVM[29]

图 18 cc-NVM 架构的总体结构[29]

3.4　安全持久性内存灾后恢复

第 3.2 节和第 3.3 节工作主要是安全持久性内存的运行时优化,本节主要介绍安全持久性内存的恢复机制.持久性内存安全存储系统的恢复过程主要分 2 个步骤[27,32]:1)扫描整个内存数据,恢复加密计数器至最新值;2)扫描整个加密计数器,恢复完整性验证树结构.由于持久性内存容量可达到 TB 级[1],且安全元数据量极大,1TB 的持久性数据对应了 16 GB 的加密计数器,以及基于 16 GB 计数器构建的完整性验证树,恢复安全元数据(加密计数器,完整性验证树)时长可达数小时[32],从而导致扫描内存数据、加/解密与散列计算开销极高.

Triad-NVM 机制[31]通过严格持久化默克尔树的低 N 层来减少恢复时长同时减少写开销,其目标是在恢复时间和性能之间进行权衡.

Anubis[32]是中佛罗里达大学于 2019 年提出的针对安全持久性内存存储系统的快速恢复机制.Anubis 机制通过在持久性内存中记录脏安全元数据的地址以实现灾后快速恢复.针对 BMT 提出相应的快速恢复机制 AGIT(Anubis for general integrity tree),在运行时,对任何数据写,需要同时将其对应安全元数据的地址写回持久性内存中.针对 Intel SGX(Intel software guard extensions)integrity tree,提出 ASIT(Anubis for SGX integrity tree),对任何数据写,需要将其对应安全元数据的地址以及脏元数据写回持久性内存中.在恢复过程只恢复记录的地址对应的数据即可,避免了全盘扫描,从而减少了恢复时长.但是,额外的记录会造成写放大,并且对于记录的地址,需要额外实现安全保护措施,增加了运行时开销.

4　总结与展望

本文从持久性内存特性出发,分别针对持久性内存的写特性与非易失性,展开介绍了基于持久性内存构建安全内存存储系统面临的挑战.针对写特性,由于持久性内存具有有限的耐久性,相关工作[15-16,18-20]主要目的在于减少由于安全措施导致的额外比特翻转,以及减少持久性内存的写次数,并未考虑非易失性.

针对非易失性,基于持久性内存构建安全存储系统需要保障其在生命周期内的安全性,因此需要考虑数据与安全元数据之间的灾后一致性.本文分别从内存加密与内存完整性验证 2 方面详细阐述了一致性保障带来的挑战,同时介绍了相关工作,表 2 对比了不同安全持久性内存存储系统架构.第 1 类工作 SCA[22]和 SuperMem 架构[28]主要解决了内存加密的一致性保障,并未考虑完整性验证的一致性保障.第 2 类工作解决了完整性验证的一致性保障,其中 Osiris[27],Triad-NVM[31],AGIT 架构[32]采取了严格更新策略,而 cc-NVM 架构[29]采取了懒惰更新的策略,其性能开销最低;第 3 类工作 Triad-NVM[31]和 Anubis 架构[32]减少了安全持久性内存恢复时长,提高了系统的可用性.然而,上述工作并未完全解决单点故障问题,如何解决恢复后的完整性验证导致的单点故障问题需要进一步探索.

Table 2　Comparison Between Different Secure Persistent Memory Storage Systems

表 2　不同安全持久性内存存储系统对比

Secure Persistent Memory System	Overhead of Crash Consistency with Encryption	Write Amplification of CrashConsistency with Encryption	Overhead of Crash Consistency with Integrity Verification	Write Amplification of Crash Consistency with Integrity Verification	Recovery Time	Single Point of Failure
SCA[22]	Low	Low			Low	
SuperMem[28]	Low	Low			Low	
Osiris[27]	Low	Low	High	Low	High	Exist
Triad-NVM[31]	Low	Low	High	Medium	High	Exist
AGIT[32]	Low	Medium	High	Medium	Low	Exist
cc-NVM[29]	Low	Low	Low	Medium	High	Exist

此外,现有大部分安全持久性内存研究主要针对 BMT 风格的安全架构,当完整性验证树结构转变为 Intel SGX 风格时,将会给安全持久性内存研究带来新的机遇与挑战.Anubis 架构[32]针对 Intel SGX 完整性树设计了恢复策略 ASIT,然而此策略引入了较高的性能开销.Intel SGX 风格的完整性验

证树具有可并行更新的特性,如何针对此特性设计高效的完整性验证树持久化方法以及恢复策略需要进一步探索.

参 考 文 献

[1] Beeler B. Intel Optane DC persistent memory module (PMM)[EB]. (2019-04-02)[2019-12-12]. https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm

[2] Kultursay E, Kandemir M, Sivasubramaniam A, et al. Evaluating STT-RAM as an energy-efficient main memory alternative [C] //Proc of IEEE Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2013: 256-267

[3] Lee B C, Ipek E, Mutlu O, et al. Architecting phase change memory as a scalable DRAM alternative [C] //Proc of the 36th Annual Int Symp on Computer Architecture. New York: ACM, 2009: 2-13

[4] Zhou Ping, Zhao Bo, Yang Jun, et al. A durable and energy efficient main memory using phase change memory technology [C] //Proc of the 36th Annual Int Symp on Computer Architecture. New York: ACM, 2009: 14-23

[5] Cho S, Lee H. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance [C] //Proc of the 42nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2009: 347-357

[6] Suh G E, Clarke D, Gassend B, et al. Efficient memory integrity verification and encryption for secure processors [C] //Proc of the 36th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2003: 339-350

[7] Yan Chenyu, Englender D, Prvulovic M, et al. Improving cost, performance, and security of memory encryption and authentication [C] //Proc of the 33rd Int Symp on Computer Architecture. New York: ACM, 2006: 179-190

[8] Rogers B, Chhabra S, Prvulovic M, et al. Using address independent seed encryption and bonsai merkle trees to make secure processors OS- and performance-friendly [C] //Proc of the 40th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2007: 183-196

[9] Gassend B, Suh G E, Clarke D, et al. Caches and Hash trees for efficient memory integrity verification [C] //Proc of the 9th Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2003: 295-306

[10] Suh G E, Clarke D, Gassend B, et al. AEGIS: Architecture for tamper-evident and tamper-resistant processing [C] //Proc of the Int Conf on Supercomputing. New York: ACM, 2003: 160-171

[11] Taassori M, Shafiee A, Balasubramonian R. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures [C] //Proc of the 23rd Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2018: 665-678

[12] Saileshwar G, Nair P J, Ramrakhyani P, et al. Morphable counters: Enabling compact integrity trees for low-overhead secure memories [C] //Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2018: 416-427

[13] Lee J, Kim T, Huh J. Reducing the memory bandwidth overheads of hardware security support for multi-core processors [J]. IEEE Transactions on Computer, 2016, 65(11): 3384-3397

[14] Webster A, Tavares S. On the design of s-boxes [G] //LCNS 218: Proc of the 2nd Advances in Cryptology—CRYPTO 85. Berlin: Springer, 1985: 523-534

[15] Young V, Nair P J, Qureshi M K. DEUCE: Write-efficient encryption for non-volatile memories [C] //Proc of the 20th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2015: 33-44

[16] Swami S, Rakshit J, Mohanram K. SECRET: Smartly encrypted energy efficient non-volatile memories [C] //Proc of the 53rd Annual Design Automation Conf. New York: ACM, 2016: Article 166

[17] Xu Cong, Niu Dimin, Muralimanohar N, et al. Understanding the trade-offs in multi-level cell ReRAM memory design [C] //Proc of the 50th Annual Design Automation Conf. New York: ACM, 2013: Article 108

[18] Awad A, Manadhata P, Haber S, et al. Silent Shredder: Zero-cost shredding for secure non-volatile main memory controllers [C] //Proc of the 21st Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2016: 263-276

[19] Zuo Pengfei, Hua Yu, Zhao Ming, et al. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes [C] //Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2018: 442-454

[20] Rakshit J, Mohanram K. ASSURE: Authentication scheme for secure energy efficient non-volatile memories [C] //Proc of the 54th Annual Design Automation Conf. New York: ACM, 2017: Article 11

[21] Swami S, Mohanram K. ACME: Advanced counter mode encryption for secure non-volatile memories [C] //Proc of the 55th Annual Design Automation Conf. New York: ACM, 2018: Article 86

[22] Liu Sihang, Kolli A, Ren Jinglei, et al. Crash consistency in encrypted non-volatile main memory systems [C] //Proc of the 24th IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2018: 310-323

[23] Liu Sihang, Seemakhupt K, Pekhimenko G, et al. Janus: Optimizing memory and storage support for non-volatile memory systems [C] //Proc of the 46th Int Symp on Computer Architecture. New York: ACM, 2019: 143-156

[24] Lu Youyou, Sun Long, Onur M. Loose-ordering consistency for persistent memory [C] //Proc of the 32nd IEEE Int Conf on Computer Design. Piscataway, NJ: IEEE, 2014: 216-223

[25] Lu Youyou, Shu Jiwu, Sun Long. Blurred persistence in transactional persistent memory [C] //Proc of the 31st Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2015: 1-13

[26] Xiao Renzhi, Feng Dan, Hu Yuchong, et al. A survey of data consistency research for non-volatile memory [J]. Journal of Computer Research and Development, 2020, 57(1): 85-101 (in Chinese)
(肖仁智, 冯丹, 胡燏翀, 等. 面向非易失内存的数据一致性研究综述[J]. 计算机研究与发展, 2020, 57(1): 85-101)

[27] Ye Mao, Hughes C, Awad A. Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories [C] // Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2018: 403-415

[28] Zuo Pengfei, Hua Yu, Xie Yuan. SuperMem: Enabling application-transparent secure persistent memory with low overheads [C] //Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2019: 479-492

[29] Yang Fan, Lu Youyou, Chen Youmin, et al. No compromises: Secure NVM with crash consistency, write-efficiency and high-performance [C] //Proc of the 56th Annual Design Automation Conf. New York: ACM, 2019: Article 31

[30] Swami S, Mohanram K. ARSENAL: Architecture for secure non-volatile memories [J]. IEEE Computer Architecture Letters, 2018, 17(2): 192-196

[31] Awad A, Ye Mao, Solihin Y, et al. Triad-NVM: Persistency for integrity-protected and encrypted non-volatile memories [C] //Proc of the 46th Int Symp on Computer Architecture. New York: ACM, 2019: 104-115

[32] Zubair K A, Awad A. Anubis: Ultra-low overhead and recovery time for secure non-volatile memories [C] //Proc of the 46th Int Symp on Computer Architecture. New York: ACM, 2019: 157-168

Yang Fan, born in 1995. PhD candidate. His main research interests include computer architecture, systems software, specifically in system and architecture for the imminent persistent memory (PM) technologies.

Li Fei, born in 1993. Master candidate. His main research interests include flash-based storage system, system software, and system security.

Shu Jiwu, born in 1968. Professor and PhD supervisor. Fellow of CCF. His main research interests include non-volatile memory systems and technologies, network storage system, storage security and reliability, and parallel and distributed computing.