Secure Computing Using Registers and Caches:

The Problem, Challenges, and Solutions

Jingqiang Lin | Chinese Academy of Sciences
Bo Luo | University of Kansas
Le Guan | Pennsylvania State University
Jiwu Jing | Chinese Academy of Sciences

Caches' unique features have enabled researchers to build secure computing environments, effectively preventing various physical and software memory attacks. Existing solutions provide confidentiality and integrity in certain applications and services. Identifying various solutions' advantages and limitations can guide future research in hardware-aided security.

In cryptography, Kerckhoffs's principle states that a cryptosystem's security must reside entirely with the secrecy of the key. In other words, protecting cryptographic keys is essential in any cryptosystem or cryptographic application. In cryptographic computing, keys are loaded as plaintext into the main memory and are therefore subject to various memory disclosure attacks. New physical attacks, such as cold-boot[1] and direct memory access (DMA) attacks,[2,3] have recently been discovered. They exploit hardware properties or functions to bypass all possible protections at the OS level and directly gain unauthorized access to memory. To defend against these physical attacks (as well as traditional software attacks), innovative mechanisms utilize hardware features, such as registers, caches, and hardware transactional memory (HTM), to protect sensitive data.

In a modern commodity computer, the computing core and memory are the necessary components: the computing core picks binary codes from memory and accesses data in memory. Memory is typically implemented as DRAM chips physically, accessible by CPU or GPU cores and peripheral devices. All data and binary codes are stored in DRAM chips, unless the cores directly access data from peripheral devices, occasionally.

To accelerate memory access, high-speed but small-sized memory components, called *caches*, are designed on the processor to store data and binaries temporarily. On-processor caches are located between the cores and the DRAM chips, and the most recently used data and instructions are kept in caches for faster future access. In this way, memory access is performed efficiently in the processor, not in DRAM chips. Synchronization between caches and DRAM chips takes place only when cache replacement happens according to cache policies and configurations or when the core explicitly executes specific cache control instructions.

Because the access in caches exhibits very different characteristics from that in RAM, attacks such as cache-based side channels and cache poisoning seek to exploit these characteristics. Exploiting the fact that accessing cached data is approximately two orders of magnitude faster than data in DRAM chips, various side channels on cryptographic engines are proposed to detect cache hits and misses during the execution and then deduce the cryptographic keys. Because cache access occurs in the processor, the cache-poisoning attack configures the

1540-7993/16/$33.00 © 2016 IEEE Copublished by the IEEE Computer and Reliability Societies

system management RAM (SMRAM) address space as cacheable; therefore, it can bypass the chipset's protection on DRAM chips and overwrite SMRAM in caches from outside the system management mode (SMM) mode. Defensive protection mechanisms also utilize the characteristics of on-chip caches. In particular, due to the very different physical properties and their location in the hardware architecture, on-chip caches naturally resist several RAM vulnerabilities. However, using such properties for secure computing is challenging because caches aren't designed for these security purposes, and hardware manufacturers provide only very limited controls over caches.

In this article, we survey recently proposed cache-based secure computing solutions, including CARMA, Copker, Mimosa, PixelVault, PRIME, Sentry, and TRESOR, and compare their security, flexibility, performance, and limitations.

Security Solutions on Top of Caches

Both caches and DRAM chips are volatile RAM units, but caches are very distinct in the following aspects:

- *On chip*. Caches are implemented as static RAM (SRAM) cells on the processor chip with the computing cores, while DRAM chips are separated out of the processor physically. This feature implies different security guarantees against physical attacks or malicious hardware manufacturers.
- *Data inconsistent*. For the cacheable memory area, the data contents in caches won't always be consistent with those in DRAM chips after modification. Therefore, the caches compose an alternative storage for the memory data to be protected, especially against attacks on DRAM chips.
- *Controlled by cores*. Although data can be transmitted from caches to DRAM chips or from DRAM chips to caches, transmission is controlled only by the CPU or GPU cores located on the processor chip. On the other hand, the data control interface of DRAM chips is exported. This enables cache-based solutions to prevent attacks from outside the processor.
- *Dedicated to each core*. The DRAM chips and peripheral devices are usually accessible to all cores, but each core has its own L1 cache, in addition to shared L2 and L3 caches. This separated and dedicated L1 cache offers an option for isolated computing environments while malicious tasks are running concurrently on other cores.

Building the Minimal Trusted Computing Base Hardware

The size of caches allows cores to perform certain functions without any RAM or peripheral devices. For example, ARM's Cortex-A9 has caches of up to 8 Mbytes, and Intel Core i7-2720QM has 6 Mbytes. LinuxBIOS uses caches as RAM to support stack and eliminate out-of-register errors before DRAM chips are initialized. It then implements portable initialization codes in C.

Using caches as RAM, CARMA establishes a trusted computing base (TCB) with the minimum-required hardware components.[4] CARMA releases the trust assumption on DRAM chips and other peripheral devices, and the only trusted hardware of the computing system is the necessary processor chip with cores and caches. To establish the TCB in CARMA, a secure execution environment loader (SEEL) is first loaded into the processor along with the secure executable code (SEC). Then, the SEEL initializes a computing environment for the SEC in the processor, employing the cache-as-RAM mechanism. However, because the SEEL and the SEC are loaded across untrusted devices such as buses and DRAM chips, they need to be attested to an external trusted device (TD). The attestation is performed by the software-only root-of-trust mechanism,[5] so there's no need for additional trusted hardware. The attestation results in a shared secret between the SEEL and the TD, and all following I/O data transmitted across untrusted hardware is protected by this secret.

Computing without DRAM Chips

Sensitive data in DRAM chips faces the practical threat of cold-boot attacks.[1] Such attacks result from the semiconductor devices' remanence effect; that is, DRAM chip contents survive to some extent without power, even at room temperature, and retention time is increased by cooling. Although the remanence effect was found in the 1970s, practical cold-boot attacks were first demonstrated in 2008.[1] Cold-boot attackers retrieve the DRAM contents of a running computer by rebooting the machine from malicious removable media or placing the DRAM chips into another machine that they control.

Cold-boot attacks don't disclose the data in caches. Caches are usually implemented with high-speed SRAM cells, and SRAM cells show even stronger remanence effects than DRAM chips. However, as on-chip memory units, caches don't export data interfaces for direct access from outside; that is, cache data is controlled and accessed only by the cores. Intel processors automatically invalidate internal caches after power-up or reset. Even if attackers reboot the machine and load malicious code into the core (when the caches aren't cleared), read operations would fetch data from DRAM chips and thereby overwrite the cache data.

The features we described can potentially defeat cold-boot attacks. To protect against cold-boot

attacks on the Advanced Encryption Standard (AES) key of full disk encryption, FrozenCache stores the AES key (as well as its round keys) in caches when users explicitly activate the special frozen state.[6] At the same time, the keys in DRAM are cleared. This "freezes" the AES keys in caches by letting all cores enter the no-fill cache mode to prevent sensitive data from being flushed into DRAM chips. However, it takes a while for the system to recover from the frozen state.[6]

Unlike FrozenCache, which uses caches only for storing sensitive data, Copker implements public-key cryptographic computations entirely in caches (as well as registers).[7] In the typical write-back cache mode, the cached data contents usually differ from those of the same address in DRAM chips, as caches' original design goal was to perform as many operations in high-speed caches as possible.

> **We provide a qualitative comparison of non-RAM security solutions. In particular, we discuss the secure computing environments created in these solutions.**

Copker employs the write-back mode to enable memory-intensive public-key cryptographic computations in caches by solving the following challenges: first, locating enough cache lines to hold the whole computation, and second, ensuring that the cache data isn't being flushed into DRAM chips during the computation. By replacing heap variables with static arrays and by switching the stack pointers, Copker stores all used data in a space of continuous physical addresses, so this data doesn't conflict on the same cache set in Intel 8-way set associative caches. At the same time, other cores sharing caches with Copker tasks are forced into the no-fill cache mode to avoid interference from concurrent tasks (for example, intensive memory access resulting in cache replacement).

Copker employs register-based AES cryptographic engines that are also immune to cold-boot attacks.[8,9] Copker adopts the common key-encryption-key structure, and the RSA private keys are encrypted in memory by an AES master key when it's in idle. To protect the AES master key, both TRESOR and Copker constrain the key only in privileged registers, because the AES memory requirement is much less than public-key cryptographic algorithms. Similar to the LinuxBIOS cache-as-RAM mechanism, Copker breaks the limited register capacity to implement complicated public-key cryptographic functions without DRAM chips. However, Copker has to handle cache access by the tasks concurrent with the protected computations, which don't exist in LinuxBIOS.

Locking All Data of Sensitive Applications

Many smartphone applications (for example, Twitter and Google Maps) contain sensitive data and could leak users' private information under various memory disclosure attacks. While a cryptographic service needs to protect only the key and ensure that its inputs and outputs don't leak any information about the key, an application usually directly processes users' privacy data. When the application is running, all its data appears in memory as plaintext. Therefore, this sensitive information might suffer direct memory access (DMA)[2] as well as cold-boot attacks. DMA attacks are launched from malicious DMA-capable peripheral devices, which issue DMA requests to directly read memory data. The DMA feature allows a peripheral device to access memory without any cooperation from the processor or the OS. These attacks have been observed via Firewire, PCMCIA, PCI Express, Thunderbolt, and other physical interfaces.

The great size of caches allows Sentry to protect all data of a specific application against cold-boot attacks.[10] In Sentry, all pages of the data segments are first encrypted in DRAM chips by an on-chip AES engine that's also implemented entirely in caches. Once the application attempts to access its page, Sentry traps the page fault to decrypt it in caches and then modify the page table entry to point to the plaintext copy. Then, Sentry supports running unmodified applications, but with encrypted data in DRAM chips, effectively preventing cold-boot attacks.

Sentry configures caches to prevent DMA access by (malicious) external peripheral devices. ARM platforms support cache locking to improve computation performance, and Sentry employs this feature to lock the plaintext data of sensitive applications in caches on smartphones and tablets, which are typically ARM devices. This strong cache-locking mechanism is ARM specific. On Intel platforms, similar functions are finished by combining the write-back and no-fill cache modes,[6,7] but DMA attacks would still succeed in obtaining the targeted data. Upon DMA requests from outside the ARM processor, this locked content isn't evicted from caches to DRAM chips. Then, on a Sentry-protected mobile device, both DMA attacks and cold-boot attacks return only the encrypted versions of data.

Protecting the Integrity of Executable Binaries

Software binaries in memory can suffer various code injection or modification attacks, such as buffer overflow

and return-to-libc. In addition to reading unauthorized memory data, physical attacks (especially advanced DMA attacks[3]) could also inject executable codes into the memory space of the OS kernel.

The on-chip instruction caches are controlled only by the internal computing cores located on the same chip, so the binaries running in instruction caches can be protected against external code injection attacks (assuming that the internal cores are trusted). PixelVault[11] loads all the binaries into a commodity GPU's instruction caches (at least 32 Kbytes in size) by exercising all execution paths during the initialization. These GPU binaries run indefinitely in nonpreemptive execution mode and never fetch any new instructions from off-chip memory. Afterward, code injection or modification attacks don't affect the autonomous GPU execution, even when the OS that's running on CPUs has been completely compromised. The attackers from CPUs can terminate the GPU execution but can't modify it.

PixelVault cryptographic services are implemented in the protected GPU binaries, and the secret keys are always in GPU registers, which are also inaccessible from outside. First, the trusted and unmodified GPU binaries never actively disclose any information on the cryptographic keys. Second, any information about the keys is carefully erased after they're loaded into GPU registers during the initialization. Finally, even if attackers launch their own malicious GPU binaries immediately after terminating the PixelVault services, the registers are reset to zero automatically by GPU hardware.

Isolating Sensitive Data from Co-resident Tasks

Memory disclosure attacks can be launched from outside the processor chip (for example, cold-boot attacks and DMA attacks), as mentioned earlier, or from vulnerable or malicious software tasks co-resident on the chip (that is, concurrently running on different cores from the protected target). For example, malicious unprivileged processes can exploit vulnerabilities to access kernel memory space, and the OpenSSL HeartBleed attack copies unauthorized memory data to remote users without any privileges. One way to mitigate this unauthorized disclosure of sensitive data is to limit the copy number and the lifetime of such plaintext data in memory. For example, solutions of different levels are enforced to ensure that only one copy of private keys appears in memory and to reduce the sensitive data in unallocated memory. However, these approaches don't eliminate the ultimate attack opportunity, where the concurrently running malicious tasks access the only copy of sensitive data when it's being processed.

The L1 cache dedicated to a core is an option to locate sensitive runtime data against the malicious tasks co-resident on the processor. During runtime, the sensitive data must be plaintext in memory for meaningful processing. Another choice is fully homomorphic encryption (FHE), in which data is processed in ciphertext; however, FHE is still impractical due to its low efficiency. Here, the memory disclosure attacks during runtime mean unauthorized access while the target process is actively running. Otherwise, if the sensitive application has been launched but is suspended, the data can be encrypted and protected as described above.

The L1 cache, which is physically independent and separated, allows for an isolated computing environment in a symmetric multiprocessing (SMP) system. Mimosa protects cryptographic keys against these runtime attacks by using HTM to keep the keys on L1 caches.[12] Transactional memory is a memory access mechanism of CPUs, originally designed to accomplish fine-grained locking with coarse-grained programming locks. This mechanism is usually implemented on top of caches, for example, on Intel TSX, IBM System Z, and Blue Gene/Q. In a transactional execution, the Mimosa private key is decrypted (or written) into the L1 data cache and then used for cryptographic computations. Similar to Copker and PRIME, which use CPU registers to store active cryptographic keys,[13] when the Mimosa service is idle, the private key is kept encrypted by an AES master key in registers. During the transactional execution of cryptographic computations, any attack attempt to access the plaintext private key or intermediate states (that is, the updated but uncommitted data) results in an unmaskable abort to ensure strong atomicity—that is, the hardware immediately clears all modified data, including the plaintext private key.

In Mimosa, the L1 data cache provides storage for sensitive data, and HTM prevents access attempts from outside the core. DMA attacks that access this sensitive memory data also result in transaction aborts and return cleared data. Moreover, the Mimosa cryptographic computation is performed entirely in L1 caches, so it's immune to cold-boot attacks.

Secure Computing Environments in Caches

Here, we provide a qualitative comparison of non-RAM security solutions. In particular, we discuss the secure computing environments created in these solutions. Although emphasizing cache-based methods, we also briefly discuss register-based solutions.

Cache Features and Security Goals

As we described earlier, these solutions are designed for different goals, most of which relate to cryptographic computing. With the exceptions of PRIME and Copker, all these systems offer performance comparable to commodity cryptographic computing approaches.

Table 1. Comparison of secure computing solutions that don't use RAM.

Solution	Hardware feature	Security goal	Cryptography capability	Performance
TRESOR*[8]	Register	Key confidentiality	Advanced Encryption Standard (AES)	Comparable to native AES
PRIME[13]	Register	Key confidentiality	RSA-2048	10 times degradation
CARMA[4]	Cache (on chip)	Execution isolation	N/A	N/A
Copker[7]	Cache (on chip, data inconsistent)	Key confidentiality	RSA-4096	Comparable to native RSA, on only one core
Sentry[10]	Cache (on chip, data inconsistent, controlled by cores)	Data confidentiality	N/A	Comparable to unprotected applications
PixelVault[11]	Cache (on chip, data inconsistent, controlled by cores)	Key confidentiality, binary integrity	RSA-1024, AES	Comparable to native versions
Mimosa[12]	Cache (on chip, data inconsistent, controlled by cores, dedicated to each core)	Key confidentiality	RSA-4096	Comparable to native RSA

** The analysis of TRESOR is also applicable to Amnesia, another register-based AES engine.*[9]

TRESOR and PRIME employ registers to provide key confidentiality, whereas cache-based solutions use caches' distinct features to provide security guarantees. As shown in Table 1, all cache-based approaches need on-chip caches to minimize the required hardware components or to circumvent physical attacks on RAM. Data inconsistency allows the plaintext data to be stored in caches, while the encrypted versions are stored in DRAM chips, or distinguishes the running binaries on instruction caches from the unprotected programs in RAM. To withstand DMA attacks, Sentry, PixelVault, and Mimosa store the cached data and binaries such that they're controlled only by the cores and not impacted by external DMA requests. Finally, the L1 data cache dedicated to each core provides Mimosa an isolated place against concurrent (malicious) processes from other cores.

Security Borders and Attacks

Each solution summarized in this article establishes a secure computing environment against attacks from outside its security border; components in the security border are assumed to be trustworthy. The security borders of register-based solutions, such as TRESOR and PRIME, contain the registers only. Figure 1 compares the different security borders of these solutions on top of caches.

First, the Mimosa computing environment is composed of a core and its L1 cache, and only this core can access the private keys protected in the L1 data cache. Second, CARMA, Sentry, and PixelVault security borders are the same as the processor's border. CARMA substitutes on-chip caches for DRAM chips, whereas Sentry encrypts applications in DRAM chips and decrypts them in caches. Hence, in CARMA and Sentry, both data and instructions are kept in the caches.

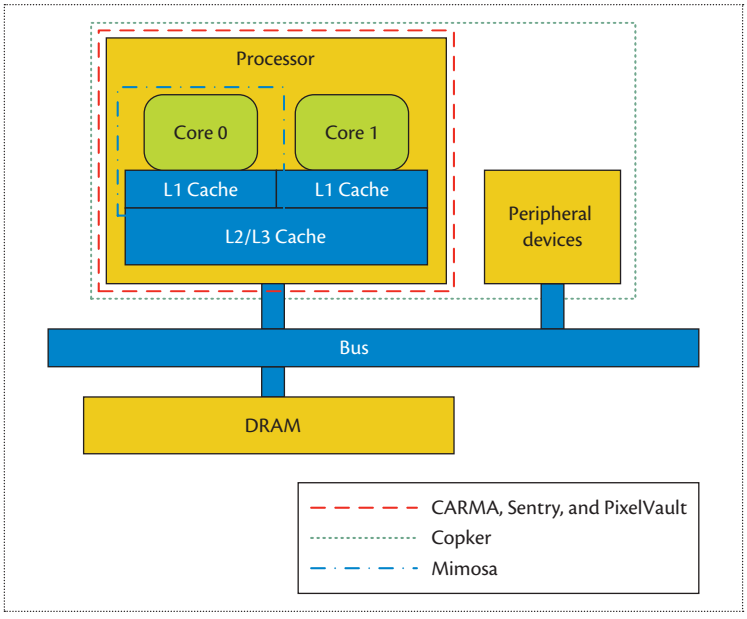

Figure 1. The security borders of solutions on top of caches. Mimosa requires the smallest secure computing environment, which includes only its computing core and corresponding L1 cache, which is used to implement transactional memory. CARMA, Sentry, and PixelVault need the CPU to be trustworthy, while Copker also requires peripheral devices to be trustworthy.

On the other hand, PixelVault uses the instruction caches of GPUs to protect the integrity of cryptographic services, and all cryptographic keys and sensitive intermediate states are stored in GPU registers. Note that

Table 2. Attack analysis of secure computing solutions that don't use RAM.

Solution	Assumptions	Prevented attacks	Remaining attacks
TRESOR[8]	Trusted OS kernel	Cold-boot attack, bus monitoring, direct memory access (DMA)-I attack,* software attack	Nonmaskable interrupt (NMI), DMA-II†
PRIME[13]	Trusted OS kernel	Cold-boot attack, bus monitoring, DMA-I attack,* software attack	NMI, DMA-II†
CARMA[4]	CPU	Malware, bus monitoring	N/A
Copker[7]	Trusted OS kernel	Cold-boot attack, bus monitoring	Software attack, DMA attack
Mimosa[12]	Trusted OS kernel	Cold-boot attack, bus monitoring, software attack, DMA attack	Denial of service (DoS)
PixelVault[11]	Trusted bootstrap	Cold-boot attack, bus monitoring, software attack, DMA attack	DoS
Sentry[10]	Trusted OS kernel, no sensitive data on pages shared with untrusted applications	Cold-boot attack, bus monitoring, DMA attack	Software attack

** DMA-I attacks read memory data by issuing DMA requests.*[2]

† DMA-II attacks inject malicious binaries into memory that access data.[3]

the GPU data caches are accessible from the untrusted CPU in PixelVault; otherwise, data caches could be used to improve performance. Finally, because Copker can't defeat malicious DMA requests, its security border includes both the processor and peripheral devices. In Copker, cryptographic computations are securely performed in caches, but keys and other sensitive data might be evicted by DMA attacks from peripheral devices.

All the non-RAM secure computing solutions make assumptions about an ultimately trusted anchor. As shown in Table 2, CARMA trusts only CPU hardware, whereas other solutions require a trustworthy OS kernel all the time; the exception is that PixelVault trusts the OS kernel only during bootstrapping. Table 2 also summarizes the attacks prevented by each of these solutions as well as the remaining attack surface against each solution. Note that because all these mechanisms assume correctly implemented CPU hardware (that is, CPU is always assumed to be trusted), sophisticated physical attacks through side channels and IEEE 1149.1 (JTAG) test access ports (TAP) might be effective but aren't discussed. Last but not least, the cache-based timing side channels are mostly mitigated, since the cryptographic computations exclusively occupy caches.[7,11,12]

Limitations

As typical hardware-aided solutions, cache-based security systems are usually platform specific. Hence, approaches on one platform can't be directly applied to other platforms. The prototype systems are implemented on different platforms, including Intel CPU,[6,7,12] AMD CPU,[4] NVIDIA CUDA GPU,[11] and ARM CPU.[10] Developing a general solution can be very challenging because these platforms have different cache mechanisms and hierarchies.

Table 3 summarizes the different cache characteristics available in major hardware platforms. We briefly explain how these mechanisms influence the design of the aforementioned cache-based security solutions. First, caches are automatically invalidated and left in a disabled state after the CPU is powered on or during a hard reset in Intel x86 and PowerPC platforms; therefore, cold-boot attacks aren't likely to be effective on caches of these platforms. However, this isn't true for ARM chips, where data in caches can leak if caches are enabled without being explicitly invalidated first. Next, although all the major chips support a modified Harvard cache architecture, where data and instructions are separately addressed in L1 caches and backed by a unified L2/L3 cache, Intel x86 hides the visibility of this feature by automatically synchronizing data and instruction caches. Therefore, the method in PixelVault fails to protect binaries in Intel x86 chips.[11] Third, the cache-locking feature that Sentry depends on is only available in ARM and PowerPC platforms, so it will be difficult to migrate it to other platforms.[10] Finally, Mimosa can't be applied to GPUs and ARM chips because transactional memory hasn't been integrated in these platforms yet.[12]

Because caches are volatile memory units, the computing environment in caches requires a secure initialization phase or a root of trust for dynamic establishment of the environment. Copker, Sentry, PixelVault, and Mimosa need a secure initialization phase to establish necessary data and parameters for the secure computing environment. During this short period, the

Table 3. Characteristics of caches in major hardware platforms.

Platform	Cache status on reset	Visibility of separated data and instruction caches	Locking	Hardware transactional memory	Typical cache hierarchy
Intel x86	Invalidated	No	No	Yes	32 Kbytes L1, several Mbytes of unified L2/L3
ARM	Unpredicted	Yes	Yes	No	32 Kbytes L1, several hundred Kbytes of unified L2
PowerPC	Invalidated	Undefined	Yes	Yes	32 Kbytes L1, several hundred Kbytes of unified L2
Nvidia GPU	N/A	Yes	No	No	64 Kbytes L1, several hundred Kbytes of unified L2

whole computer system is assumed to be trustworthy. Moreover, solutions that dynamically build the secure computing environments in caches (Copker, Sentry, and Mimosa) need an AES master key as the root of trust. This trusted master key always exists as plaintext in the system. Hence, Sentry locks the key in caches to defend against physical memory attacks,[10] whereas Copker and Mimosa store it in privileged registers and patch the OS to defeat both physical and software attacks on the master key.[7–9,12] As a result, these solutions assume an unmodified kernel to prevent the AES master key from being stolen. Finally, the root of trust in CARMA is an external device, to which the computing base attests itself after it's established.

Availability (or performance) is another issue in these cache-based security solutions. Although the cache space is sufficient to support very strong cryptographic algorithms[7–9,12] or store user data[10] (as shown in Table 1), the supported security functionality is ultimately restricted by its physical capacity. Moreover, because caches weren't originally designed for exclusive use or security, reserving caches for a secure computing environment introduces new issues. In particular, it affects concurrent tasks' performance[7] or might monopolize computing resources.[11] In multiprocessor systems, when certain caches are intended to be shared by multiple cores, secure computing solutions need to invoke cache control instructions to prevent other cores from accessing the caches to avoid unexpected cache replacement or flush triggered by the cache-sharing cores. As a result, concurrent computing tasks on other CPU cores suffer reduced performance due to the (partial) unavailability of CPU caches. To mitigate the impact on concurrent tasks, some solutions exclusively occupy parts of caches when the secure service is active and release the cache resources when the services are idle.[7,10,12] Meanwhile, Mimosa has almost no performance impact on concurrent tasks;[12] however, its protected service suffers the risk of denial-of-service (DoS) attacks when frequent memory access by concurrent tasks happens. On the other hand, the performance of register-based cryptographic solutions is influenced by the limited storage capacity,[13] but such solutions don't affect the performance of the concurrent tasks on other CPU cores. Finally, the switch between the cache-protected state and the normal running state is somewhat time consuming.[6,10]

Cache-based side-channel attacks on the implementations of cryptographic algorithms have been studied.[14,15] In particular, Taesoo Kim and his colleagues give a comprehensive overview of cache-based side-channel attacks.[15] In theory, these attacks can be launched against the non-RAM secure computing mechanism that utilizes CPU cache as alternative storage. However, mechanisms such as Copker and PixelVault are immune to cache-based side-channel attacks because they exclusively occupy caches, so no other process could share the cache and spy on the secure computing mechanisms.

Finally, although multiple non-RAM secure computing solutions have been proposed, the use of cache-based security solutions is still in its infant stage. Most of the prototypes haven't been adopted in the commercial or open source software industry. To the best of our knowledge, the most influential commercially available solution is Coreboot, an open source BIOS program that takes advantage of the cache-as-RAM feature to initialize the chip before the RAM is ready to use. Coreboot is replacing the compatibility support module as the fallback mode in the unified extensible firmware interface (UEFI) boards.

Through this article, we hope to attract more attention to hardware-aided security and further stimulate new ideas and research discoveries in this direction. In particular, we would expect more research activities on hardware vulnerabilities caused by physical properties; hardware attacks exploiting such vulnerabilities; defense mechanisms, hardware, or software against hardware attacks; and the use of hardware features to support trustworthy computing. ■

Acknowledgments

Jingqiang Lin and Jiwu Jing were partially supported by National 973 Program of China awards 2013CB338001 and 2014CB340603 and the Strategy Pilot Project of the Chinese Academy of Sciences award XDA06010702. Bo Luo was partially supported by National Science Foundation (NSF) CNS-1422206, NSF IIS-1513324, NSF DGE-1565570, and a subgrant of NSF OIA-1308762. Le Guan was partially supported by ARO W911NF-13-1-0421 (Multidisciplinary University Research Initiatives).

References

1. J. Halderman et al., "Lest We Remember: Cold Boot Attacks on Encryption Keys," *Proc. 17th USENIX Security Symp.* (USENIX Security 08), 2008, pp. 45–60.
2. P. Stewin and I. Bystrov, "Understanding DMA Malware," *Proc. 9th Conf. Detection of Intrusions and Malware and Vulnerability Assessment* (DIMVA 13), 2013, pp. 21–41.
3. E.-O. Blass and W. Robertson, "TRESOR-HUNT: Attacking CPU-Bound Encryption," *Proc. 28th Ann. Computer Security Applications Conf.* (ACSAC 12), 2012, pp. 71–78.
4. A. Vasudevan et al., "CARMA: A Hardware Tamper-Resistant Isolated Execution Environment on Commodity x86 Platforms," *Proc. 7th ACM Symp. Information, Computer and Communications Security* (AsiaCCS 12), 2012, pp. 48–52.
5. A. Seshadri et al., "Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms," *Proc. 20th ACM Symp. Operating Systems Principles* (SOSP 05), 2005, pp. 1–16.
6. J. Pabel, "FrozenCache: Mitigating Cold-Boot Attacks for Full-Disk-Encryption Software," *Proc. 27th Chaos Communication Congress* (27C3 10), 2010; https://events.ccc.de/congress/2010/Fahrplan/events/4018.en.html.
7. L. Guan et al., "Copker: Computing with Private Keys without RAM," *Proc. 21st ISOC Network and Distributed System Security Symp.* (NDSS 14), 2014; www.internetsociety.org/sites/default/files/07_1_1.pdf.
8. T. Müller, F. Freiling, and A. Dewald, "TRESOR Runs Encryption Securely outside RAM," *Proc. 20th USENIX Security Symp.* (USENIX Security 11), 2011, pp. 17–32.
9. P. Simmons, "Security through Amnesia: A Software-Based Solution to the Cold Boot Attack on Disk Encryption," *Proc. 27th Ann. Computer Security Applications Conf.* (ACSAC 11), 2011, pp. 73–82.
10. P. Colp et al., "Protecting Data on Smartphones and Tablets from Memory Attacks," *Proc. 20th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 15), 2015, pp. 177–189.
11. G. Vasiliadis et al., "PixelVault: Using GPUs for Securing Cryptographic Operations," *Proc. 21st ACM Conf. Computer and Communications Security* (CCS 14), 2014, pp. 1131–1142.
12. L. Guan et al., "Protecting Private Keys against Memory Disclosure Attacks Using Hardware Transactional Memory," *Proc. 36th IEEE Symp. Security and Privacy* (S&P 15), 2015, pp. 3–19.
13. B. Garmany and T. Müller, "PRIME: Private RSA Infrastructure for Memory-Less Encryption," *Proc. 29th Ann. Computer Security Applications Conf.* (ACSAC 13), 2013, pp. 149–158.
14. D. Page, "Defending against Cache-Based Side-Channel Attacks," *Information Security Technical Report*, vol. 8, no. 1, 2003, pp. 30–44.
15. T. Kim, M. Peinado, and G. Mainar-Ruiz, "StealthMem: System-Level Protection against Cache-Based Side Channel Attacks in the Cloud," *Proc. 21st USENIX Security Symp.* (USENIX 14), 2014, pp. 189–204.

Jingqiang Lin is a professor at the Data Assurance and Communication Security Research Center and State Key Laboratory of Information Security in the Institute of Information Engineering at the Chinese Academy of Sciences. His research interests include system security and applied cryptography. Lin received a PhD in information security from the Graduate University of Chinese Academy of Sciences. Contact him at linjingqiang@iie.ac.cn.

Bo Luo is an associate professor in the Department of Electrical Engineering and Computer Science at the University of Kansas. His research interests include data science, privacy, and security. Luo received a PhD in information sciences and technology from the Pennsylvania State University. Contact him at bluo@ku.edu.

Le Guan is a postdoctoral researcher in information sciences and technology at the Pennsylvania State University. His research interests include system security and mobile security. Guan received a PhD in information security from the Chinese Academy of Sciences. Contact him at lug14@psu.edu.

Jiwu Jing is a professor at the Data Assurance and Communication Security Research Center and State Key Laboratory of Information Security in the Institute of Information Engineering at the Chinese Academy of Sciences. His research interests include system security and applied cryptography. Jing received a PhD in information security from the Graduate University of Chinese Academy of Sciences. Contact him at jingjiwu@iie.ac.cn.