www.ijprems.com
editor@ijprems.com

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

Vol. 03, Issue 04, April 2023, pp : 615-622

e-ISSN : 2583-1062

Impact Factor : 5.725

MITIGATING RAM EXFILTRATION ATTACKS ON CRYPTOGRAPHIC KEYS THROUGH THE USE OF HARDWARE TRANSACTIONAL MEMORY

M. Ajith Kumar[1], Mr. Saravanan Elumalai[2], Dr. S. Geetha[3]

[1]Final Year M.Tech CFIS, Dr. M.G.R Educational And Research Institute, India.

[2]Professor, CSE, Dr. M.G.R Educational And Research Institute, India.

[3]Head of Department, CSE, Dr. M.G.R Educational And Research Institute, India.

DOI: https://www.doi.org/10.58257/IJPREMS30993

ABSTRACT

Cryptography is crucial for network security. In practice, the cryptographic keys are loaded into the reminiscence as plaintext at some stage in cryptographic computations. Therefore, the keys are problem to remains memory disclosure attacks that study unauthorized statistics from RAM. This paper affords protecting RSA private keys in opposition to both software and Hardware based memory disclosure attacks. We uses Hardware Transactional Memory (HTM) [24],to make certain that (a) each time a malicious method attempts to examine the plaintext personal key, The transaction aborts where all touchy records are routinely cleared, Because of the atomicity assure of HTM; (b) all touchy information appear as plaintext most effective inside caches, are never loaded to RAM. To the exceptional of our know-how it is the first to we use (HTM) to defend touchy statistics towards memory attacks. We implemented Mimosa with Intel Transactional Synchronization Extensions (TSX) [19], but the fragility of TSX transactions introduces more cache-clogging denial-of-service (DOS) threats, and attackers could sharply degrade the overall performance. We in addition partition an RSA private key computation into more than one transactional elements, even as intermediate results are blanked throughout transactional components. Experiments display that successfully projects of cryptographic keys against memory disclosure attacks, and introduces a small overhead, even with concurrent cache-clogging workloads.

Keywords—Cold-boot attack, CPU-bound encryption, DMA attack, transactional memory, Software Memory Disclosure Attack

1. INTRODUCTION

Cryptosystems plays an important role in computer and communication security, The cryptographic keys are shall be protected with the higher level of security. In the signing or decryption operation, The private keys are usually loaded into the memory as plaintext, And becomes vulnerability to memory disclosure attack read the unauthorized data into the memory. Such attacks are launched through software exploitations. For instance, the OpenSSL Heartbleed vulnerability allows remote as buffer-overflow guards and attackers to steal sensitive memory data. Un privileged Linux show that 16.2 percent of the vulnerabilities can be exploited to read the unauthorized data from memory space of operating system (OS) kernel or user processes. Such attacks can be launched effectively, even though the integrity of the victim systems binaries is maintained at all times. So present mechanisms such and kernel integrity protections are not effective against these “silent” attacks. Meanwhile, attackers are capable of bypassing all OS protections to directly read a data form the RAM, even if the device is free of the vulnerabilities Mentioned on Cold-boot attacks “freeze” the RAM Chips of the computer. This paper presents uses an (HTM) to protect private keys against both software and physical memory disclosure attacks described above. We use Intel Transactional Synchronization ex tensions (TSX) [19], a commodity HTM solution in the platforms. Transactional Memory remains becomes firstly proposed as a speculative memory access to get right of entry to mechanism to enhance the overall performance of multi-thread applications [9], [10]. An execution with transactional memory with finishes successfully, only within the case of no data conflict happens; Otherwise all operations are Discarded and the execution is rolled back. A data conflict happens when the multiple threads simultaneously access the same memory location and at least one of them is a write operation. The strong atomicity assure provided via HTM is applied to defeat illegal accesses to the memory space that contains a more useful sensitive data. Furthermore, Intel TSX and most HTM are physically carried out into the caches, So the computing is constrained absolutely with CPU, Effectively preventing the cold-boot attacks on the RAM.This paper adopts the Key-encryption-key structure the RSA private keys in the memory remains encrypted by and AES master key, when there is no sign or decryption request. TRESOR , a register-based AES cryptographic engine, is integrated to protect the key-encryption keys constantly in debug registers that are handiest accessible with ring 0 privileges. The AES master key is derived form the password, input while the system boots. When Mimosa is triggered for an request, the RSA private key is decrypted by way of the AES master key and then used as follows.

www.ijprems.com
editor@ijprems.com

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

Vol. 03, Issue 04, April 2023, pp : 615-622

e-ISSN : 2583-1062

Impact Factor : 5.725

In mimosa every private-key computation is done by an atomic transaction. During the stage of transaction, the encrypted private key is first decrypted into plaintext, and used to decrypt or sign messages. If the transaction is interrupted due to any cause (e.g, attack attempt, interrupt, or fault), a hardware-enabled abort handler clears all updated however uncommitted data inside the transaction, which guaranteed that the private key (and intermediate states) cannot be accessed by means of malicious processes. We implemented the Mimosa prototype with Intel TSX, however the design is applicable to other HTM implementations the use of on-chips caches [2], or store buffers [23]. While the private-key computation is done as an HTM transaction and the private key is decrypted (i.e., the data are updated) in the transactional execution, any attack try to access the private key result in data conflicts that abort the transaction. These HTM solutions are CPU-bound, so they also effective against the cold-boot attacks.

2. BACKGROUND AND PRELIMINARIES

2.1 MEMORY DISCLOSURE ATTACKS ON SENSITIVE DATA

These attacks are roughly categorized into two categories:Software-based and hardware (or physical) attacks.Software Memory Disclosure Attack. Software vulnerabilities permits adversaries to read unauthorized data form the memory space of OS kernels or user processes, without editing the binaries. Those vulnerabilities result from unverified inputs, isolation defects, memory dump, memory reuse or cross-use.However, this selection is exploited in another way of physical memory attacks . Read-only DMA attacks read out sensitive memory by means of DMA requests from Firewire or PCI interfaces [6],and the malicious behaviors do no longer want any “cooperation” of OSes. Advanced DMA attacks injecting the malicious binaries into the memory of victim computers by DMA requests, and then the injected codes get right of entry to access data in memory or registers[10].

2.2 CPU-BOUND SOLUTIONS AGAINST COLD- BOOT ATTACKS

While there are numerous answers in opposition protection against cold-boot attacks is to bound the operations in CPUs. CPU-bound solutions avoid loading sensitive data into RAM chips, so the cold-boot attacks fail. Register-based cryptographic engines, implemented the AES algorithm entirely within CPUs. TRESOR stores an AES key in debug.PRIME [25], RegRSA [17] and Copker expand the CPU-bound technique to RSA. The AES key protected by using TRESOR is used as a key-encryption key to encrypt RSA private keys. In PRIME, the private key is decrypted into AVX registers and the RSA computations are carried out in these registers. The performance is reduced to approximately 10 percent of traditional implementations, due to the limited size of registers. RegRSA processed high by using the use of registers and encrypting sensitive intermediate states in memory, so the performance is enhances. Copker employs CPU caches to carried out the RSA computations against cold-boot attacks. It assumes the integrity of OS kernels without any memory disclosure vulnerabilities so Copker is not proof against to software memory disclosure attacks.

3. SYSTEM DESIGN

This section presents the assumptions and protection desires of Mimosa. We then introduce the system architecture, and some important layout design details.

3.1 ASSUMPTIONS AND SECURITY GOALS

Assumptions. We expect the correct hardware implementation of HTM (i.e., Intel TSX within the prototype Different from the existing mechanisms which try to come across detect or prevent software attacks (e.g., buffer-overflow guards [22], Mimosa follows a System or others in the future). We also assume a secure initialization at some stage in the OS boot process technique; this is, the system is clean and now not attacked during this small time window. Attackers are assumed to be view to launch memory disclosure attacks. They can stealthily read memory data in OS kernels by means of exploiting memory disclosure vulnerabilities or launch cold-boot attacks. They can eavesdrop the communication on the CPU and then the and RAM chips. Mimosa attempts to defend against these “silent” memory disclosure attacks that read memory data without breaking the integrity of privileged binaries. We do not recollect the multi-step attacks the attackers first write malicious binaries into the victim system’s kernel, and then access the data via injecting codes. This is, we assume that the integrity of OS kernels is not compromised constantly, while memory disclosure vulnerabilities exist inside the kernel. Kernel integrity can be guaranteed by means of current mechanisms, which include TPM all through initialization, and SBCFI, Lares or kGuard at runtime. Besides, the adversaries may also perform any operations with non-root privileges, e.g., run concurrent memory intensive tasks to compete for resources with Mimosa. TRESOR protects the AES master key in privileged debug registers, so Mimosa inherits its assumptions. TRESOR and similar comparable solutions expect no interface or vulnerability that allows get attackers to access debug registers. The right of entry to do these privileged registers is blocked by patching the ptrace system call (the only interface from user space), disabling loadable kernel modules and getting rid of JTAG ports (accomplished in COTS products). Security Goal. We design Mimosa with the subsequent

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 03, Issue 04, April 2023, pp : 615-622

e-ISSN : 2583-1062

Impact Factor : 5.725

goals. (1) During each private-key operation, no thread aside from the Mimosa task can access the sensitive data in memory, including the AES master key, the plaintext RSA private key and intermediate states. (2) both efficiently completed or by chance interrupted, each Mimosa computing challenge is ensured to without delay immediately clear all sensitive data, so it cannot be suspended to dump these sensitive data. And (3) The sensitive data never appear on the RAM chips.

3.2 THE MIMOSA ARCHITECTURE

Mimosa adopts the common key-encryption-key structure. The AES master key's derived during the OS boot process method and is saved in debug registers since then. The RSA context is dynamically constructed, used and subsequently destroyed within a transactional execution, while mimosa serves the signing/decryption requests. Whilst the mimosa service is in idle, private keys remain encrypted and via by the master key.

The operations of Mimosa encompass of two phases in Fig. 1: an initialization phase and a protected computing section. The initialization is accomplished only once while the system boots. It initializes the AES master key in debug registers.

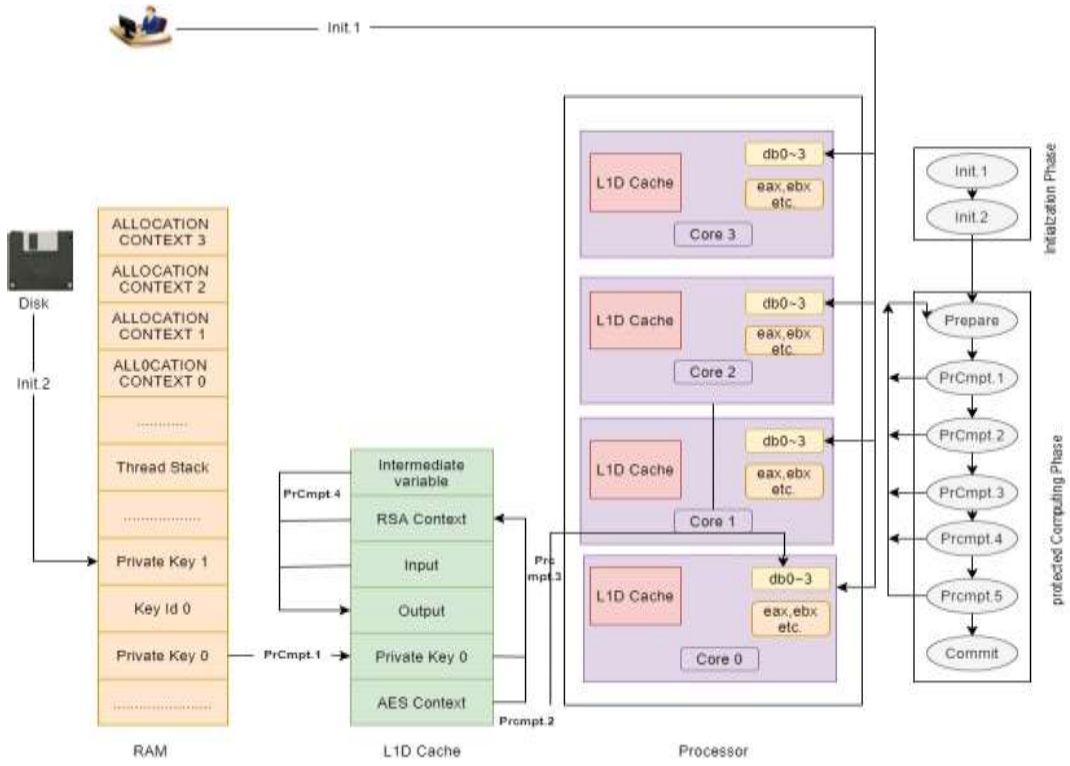


Fig.1. Mimosa Overview

- Prepare: HTM begins to track memory access within the examine read-set and the write-set inside L1D cache.
- PrCmpt.1: The ciphertext private key is loaded form the RAM to the cache.
- PrCmpt.2: The master key loaded from the debug registers to the cache.
- PrCmpt.3: With the master key and the ciphertext private key, the private key by the use context is constructed.
- PrCmpt.4: With the plaintext private key, the requested decryption/signing operation is achieved.
- PrCmpt.5: All the variables in caches and registers are erased, beside the end result.
- Commit: Finish and the end of the transaction and make the result available.
- All memory accesses during in the phase of this section are strictly monitored via hardware.

4. IMPLEMENTATION

We first introduce the RTM interface and a native implementation of Mimosa as an Linux kernel module. We then take a look at the reason of the aborts that significantly reduces overall performance, and optimize the implementation to obtain the overall performance similar to conventional RSA engines.

4.1 RTM PROGRAMMING INTERFACE

RTM we are selected instructions (XBEGIN, XEND and XABORT) to start, commit and abort a TSX transaction. XBEGIN includes a two-byte opcode 0xC70xF8 and an operand. The operand is a relative offset to the EIP registers, to calculate the address of the program-distinctive fallback function. On aborts, the CPU right away immediately

www.ijprems.com
editor@ijprems.com

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

Vol. 03, Issue 04, April 2023, pp : 615-622

e-ISSN : 2583-1062

Impact Factor : 5.725

breaks the transaction and restores micro-architectural states. Then, the execution resumes at fallback function. At the same time, the abort motive is marked in the corresponding bit(s) of the EAX register. The reason code in EAX is used for quick selections at runtime; at a instance example, the third bit suggest indicates data conflicts, and the fourth indicates that shows the cache is full. However, this returned code does not precisely reflect every event [12]. As an example, the aborts due to unfriendly instructions or interrupts do not longer set any bit. In reality, Intel suggests overall performance monitoring for deep analyses when timing-based aspect channels of AES implementations [1], [7], are removed by running in constant time. programming with TSX, earlier than releasing the software. We encapsulate the RTM instructions into C functions in Linux kernel. At the time of our implementation, we did no longer find any guide for RTM in the re dominant Linux kernel branch. Despite the fact that Intel Compiler, Microsoft Visual Studio, and GCC have helps for RTM in user-space programming, they're not ready for kernel programming. We talk over with the Intel manual to implement the RTM intrinsic using inline assembler equivalents. The _xbegin() feature to start the transaction is as follows:

static__attribute__ ((__always_inline__)) inline int_xbegin(void) {

intret = _XBEGIN_STARTED; asmvolatile(''.byte0xC7,0xF8;.long0'': ''+a''(ret)::''memory''); returnret; }

4.2 THE NATIVE IMPLEMENTATION

The AES master key is usually protected in debug registers, and the protected computing we adopted PolarSSL v1.2.10 because the base of our AES and RSA modules.

PolarSSL is an efficient Library with a small memory footprint. A smaller work-set means adequate cache resources assets to complete the transaction. Inside the long-integer module of PolarSSL, a piece of assembly codes make use of the MMX registers. It is marked as unfriendly instructions of Intel TSX [12]. We replaced MMX with XMM. It needs simplest a little modification because both operands are supported in the SSE2 extension. The AES module of PolarSSL is an S-box-based implementation, but we improved with that AES-NI [13].

4.3 PERFORMANCE TUNING

Mimosa thread monopolize its very own allocation context inside the transactional region. We reserve a global array of allocation contexts, and each context is defined for one core. The first member in ALLOCATION_CONTEXT is aligned on a 64-byte boundary (i.e., a cache line), which is the granularity to track the read/write-sets units. This prevents false data sharing among the contexts, which takes place when two threads access their distinct memory locations in the same cache line. Disabling Interrupts and Preemption: SDE does now not simulate interrupts. The private-key computation is time-consuming, so it's far very probably that the transactional execution is interrupted by way of task scheduling on real hardware, which definitely causes aborts. Other interrupts also cause aborts. To give Mimosa enough time to finish computations, interrupts and preemption are temporarily disabled while it's far inside the transactional region. Existing CPU-bound cryptographic engines [25], [17] disable interrupts to make ensure atomicity, at the same time as Mimosa requires it for efficiency because Intel TSX ensures atomicity already Delay after Continuous Aborts.

5. CACHE-CLOGGING DOSAT TACKS AND PARTITIONED PROTECTED COMPUTING

We inspect the aborts in the presence of cache-clogging DoS attacks (or concurrent memory-intensive tasks), and partition the RSA private-key operation into more than one transactional parts to mitigate the impact of such threats.

Algorithm 1. RSA Decryption Partitioned into Three Parts

Input: ciphertext, encpdp

Output: t1cipher

(p,dp) – AESDecrypt(encpdp);

T1 = ciphertext dq mod p;

T1cipher = AESEncrypt(t1)

Input: ciphertext, encqdq

Output: t2cipher

(q,dq) = AESDecrypt (encqdq);

T2 = ciphertext dq mod q;

T2 cipher = AESEncrypt(t2);

Input: ciphertext, T1cipher, T2cipher,

Output: plaintext

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 03, Issue 04, April 2023, pp : 615-622

e-ISSN : 2583-1062

Impact Factor : 5.725

(p,q,qinv) = AEADecrypt(encpqq);

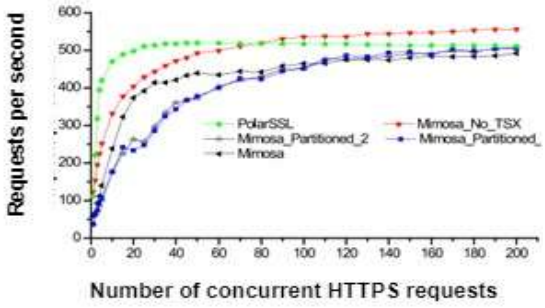
(T1,T2) =AESDecrypt (T1cipher, T2cipher);

Plaintext = (T1 – T2) * qinv mod p;

Plaintext = T2 + plaintext * q;

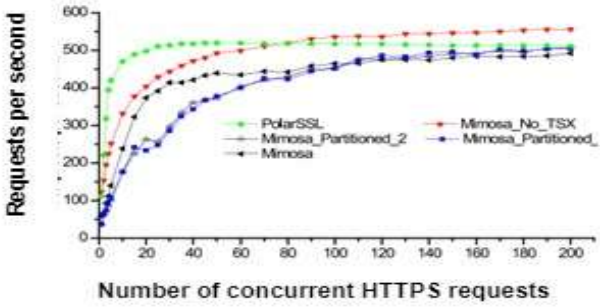
6. HTTPS THROUGHPUT AND LATENCY

The client ran Apache Bench sending requests at distinctive concurrence levels, and the numbers of HTTPS requests dealt with in line with second are proven The most throughput of Mimosa loses 17.6 percent of its local capacity, even it is 17.2 percent for Mimosa_Partitioned_2 and Mimosa_ Partitioned_3 loses 16.5 percent. The numbers of Mimosa_ No_TSX and PolarSSL are 13.5 and 6.5 percentage, respectively. From the results, we are estimate that the first 6.5 percent loss for all process attributed to the unavoidable overhead of HTTP,



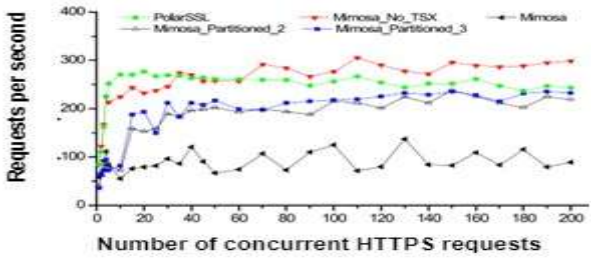
(a) In clean environments

Fig. 2. HTTPS throughput



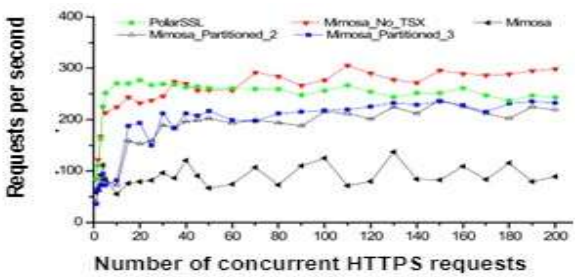
(b)With STREAM workloads

Fig. 3. HTTPS requests



(c)In clean environments

Fig. 4. HTTPS Latency



(d)With STREAM Workloads

fig. 5. HTTPS workloads

www.ijprems.com
editor@ijprems.com

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

Vol. 03, Issue 04, April 2023, pp : 615-622

e-ISSN : 2583-1062

Impact Factor : 5.725

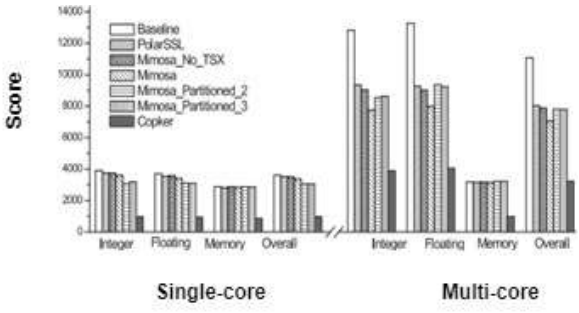


Fig. 6. Geekbench 3 Scores under RSA computations

7. SECURITY ANALYSIS AND DISCUSSION

This section validates that Mimosa achieves the security goals in Section 3.1. Then, the ultimate attack surfaces are mentioned, and we compare Mimosa with existing defenses against cold-boot attacks (and also other attacks) on the RSA private keys. We additionally speak the applicability of Mimosa.

7.1 VALIDATION AND ANALYSIS

Latest studies display that, the memory contents are scrambled on DDR3 RAM chips, so cold-boot attacks cannot at once read the plaintext data but at most 128 bytes of known plaintext are required in a descrambling attack to recover the memory content [4]. The Mimosa prototype works with two 4G-byte DDR3 RAM chips, and we did not no longer re-construct the special descrambling tool to recover the private keys in RAM chips. In the meantime, according to the Intel manual (see when cache eviction in the write-set happens, a transaction aborts immediately, and modified data are discarded inside the L1D caches. At some point of the time during the transactional execution of Mimosa, the plaintext private key is kept in caches but not RAM chips, [12].

7.2 REMAINING ATTACK SURFACE

Attackers might be exploit side channels to compromise the keys. Cache-based side channels [7], do not longer exist in Mimosa, because AES-NI is free of such attacks [13] and the RSA computations are done entirely in the caches. Other aspect channels of timing [3], electromagnetic fields ground electric potential power or acoustic emanations [36], may be prevented by RSA blinding . The random bits in RSA blinding will be additionally protected by the AES master key against memory disclosure attacks.Mimosa assumes the integrity of the OS kernels, so integrity protections (e.g., SecVisor, SBCFI Lares, and kGuard) shall work complementarily. Whilst the kernel integrity solutions protect the Mimosa binaries from being modified, Mimosa defeats memory disclosure attacks not violating the integrity of binaries. Ref. [10] exhibits an advanced DMA attacking injects malicious codes into an OS kernel (i.e., breaks the integrity) after which ca an accesses the AES key in debug registers. Fortunately, the DMA attacks are countered through way of various solutions.

7.3 COMPARISON WITH SOFTWARE CRYPTOGRAPHIC ENGINES AGAINST COLD-BOOT ATTACK

There are RSA implementations on common OS against cold-boot attacks, specifically, PRIME [25], RegRSA [17], Copker and the proposed work. Those solutions undertake the same key-encryption-key structure—an AES master key is kept in privileged registers at same point of the operation of the system, and the RSA private key is decrypted on demand to carry out requested operations. Table summaries four approaches in terms of OS assumption, performance and RSA implementation. Hardware assumptions are not shown in the table, along with Intel TSX, cache-filling modes, CPU privilege rings, etc. With the hardware support from Intel TSX, Mimosa significantly outperforms other solutions. Moreover, because the private key computation is implemented in C language, it is much easier for Mimosa and Copker to support other algorithms which includes DSA and ECDSA.

7.4 APPLICABILITY

Most HTM solutions share the similar programming interface. In other HTM implementations, the counterparts of XBEGIN and XEND are without problems recognized, and the abort processing conforms to the Mimosa design. In the HTM facility of IBM zEC12, transactions are described by the means of TBEGIN and TEND. On aborts, the PC register is restored to the instruction immediately after TBEGIN, and a condition code is set to a non-zero value. A program tests the condition code after TBEGIN to start the transactional execution if CC=0 or branch to the fallback function if not. AMD proposed its HTM extension, called Advanced Synchronization Facility (ASF),however currently products are not ready. ASF provides comparable commands to begin and commit a transactional execution (i.e., SPECULATE and COMMIT) and tracks memory accesses in caches [2]. ASF has a slightly unique function that all to-be-traced memory must be explicitly specified.Finally, maximum HTM implementations use on-chip additives for the transaction execution [2], [24], so they can also work with Mimosa to prevent form cold-boot attacks.

8. RELATED WORK

8.1 ATTACK AND DEFENSE ON SENSITIVE MEMORY DATA

Frozen Cache stores AES keys in caches and configures the cache-filling modes to save form the attack keys from being flushed to RAM chips. The CPU-bound approach is extended to the RSA algorithm. Using the AES key protected via TRESOR as a key-encryption key, PRIME [25] carried out the RSA computation in AVX registers while Copker did it in caches. RegRSA [17] improved PRIME by using extra more registers and encrypting intermediate consequences in memory, so the efficiency is stepped forward. Mimosa implements RSA against. PhiRSA exploits the vector instructions of Intel Xeon Phi to enforce excessive high-performance RSA computations.

There are SGX-based security solutions [5], and [implements cryptographic engines in SGX enclaves. There are vulnerabilities observed in SGX enclaves [20], leaking sensitive data. Flicker built dynamical isolated execution environments, utilizing the hardware characteristic of overdue launch and attestation in Intel Trusted execution Technology (TXT) and AMD Secure Virtual Machine (SVM). The overhead initialize and exit form the SGX enclave or a Flicker piece is heavier than a TSX transaction [12], so Mimosa is greater suitable for often -called kernel modules. These solutions display the same tendency of building security systems of hardware features. Last, Ram Crypt and Hyper Crypt are software-based memory encryption for Linux approaches towards software and physical memory disclosure attacks; however the performance penalty is significant.

8.2 TRANSACTIONAL MEMORY APPLICATION AND EXPLOITATION

Transactional memory boost thread-level parallelism, and is applied in the database systems [19] and game servers [16]. Transactional memory improve the multi-threaded aid in dynamic binary translation to make sure the correct executions of concurrent threads [21]. By retaining shared resources in the read/write-set, TMI enforces authorization rules once this sort of resource is accessed [8], [9]. TMI and Mimosa depend on transactional memory to the access to sensitive resources. TMI enforces authorization policies on every access. Whilst Mimosa guarantees confidentiality via clearing sensitive keys once any illegal read operation occurs. TSX-CFI maps control flow transitions into TSX transactions, and violations of the supposed flow graph will trigger aborts. leverages the strong atomicity of HTM to synchronize virtual machine introspection (VMI) and guest OS execution, so that VMI is carried out more timely and consistently. It monitors the read-set to locate and detect concurrent update operations that cause inconsistence, while Mimosa monitors the write set to detect illegal read operations.

8.3 TRANSACTIONAL MEMORY IMPLEMENTATION

Transactional memory designs are proposed, from the hardware solutions are [24],[12], to software-program-based solutions and hybrid schemes [23].HTM usually updates data quickly in CPU-bound caches or store buffers before the transaction commits, and discards the up to data statistics on aborts. LogTM updates memory at once and saves unmodified values in a according to the per-thread log; on aborts, the state is restored by using through the logs.

9. CONCLUSION

We present Mimosa, an implementation of the RSA cryptosystem with notably improved forward security ensures on the private keys. With the assist of HTM, Mimosa ensures that best of Mimosa itself is able to access plaintext private keys in a private-key computation. Any unauthorized get access to might routinely cause a transaction abort, which immediately right away clears all sensitive data and terminates the cryptographic computations. This software memory disclosure attacks that exploit vulnerabilities to stealthily read sensitive data from memory with out breaking the integrity of executable binaries. Meanwhile, the whole protected computing environment is constrained in CPU caches, so Mimosa is proof against immune to cold-boot attacks on RAM chips.

[1] REFERENCES

[2] O. Aciicmez, W. Schindler, and C. Koc, “Cache based remote timing attack on the AES,” in Proc. 7th Cryptographers’ Track RSA Conf. Topics Cryptology, 2007, pp. 271–286.

[3] AMD, “Advanced synchronization facility, proposed architectural specification (revision 2.1),”2009.http://developer.amd.com/wordpress/media/2013/09/432ASF_Spec_2.1.pdfus/articles/intel-advanced-encryption-standard-aes-instructions.

[4] C. Arnaud and P.-A. Fouque, “Timing attack against protected RSA-CRT implementation used in PolarSSL,” in Proc. Cryptographers Track RSA Conf., 2013, pp. 18–33.

[5] J. Bauer, M. Gruhn, and [4] . Bauer, M. Gruhn, and F. Freiling, “Lest we forget: Cold-boot attacks on scrambled DDR3 memory,” Digital Investigation, vol. 16, pp. S65–S74, 2016.

[6] A. Baumann, M. Peinado, and G. Hunt, “Shielding applications from an untrusted cloud with Haven,” in Proc. USENIX Symp. Operating Syst. Des. Implementation2014.

[7] M. Becher, M. Dornseif, and C. Klein, “Firewire: All your memory are belong to us,” in Proc. CanSecWest Conf., 2005. http://www.cansecwest.com/core05/2005-firewire-canecwest.pdf.

[8] D. Bernstein, “Cache-timing attacks on AES,” 2004. https://cr. yp.to/antiforgery/cachetiming-200414.pdf.

[9] A. Birgisson and U. Erlingsson, “An implementation and semantics for transactional memory introspection in Haskell,” in Proc. ACM SIGPLAN 4th Workshop Program. Languages Anal. Security, 2009, pp. 87–99.

[10] A. Birgisson, M. Dhawan, U. Erlingsson, V. Ganapathy, and L. Iftode, “Enforcing authorization policies using transactional memory introspection,” in Proc. 15th ACM Conf. Comput. Commun. Security, 2008, pp. 223–234.

[11] E.-O. Blass and W. Robertson, “TRESOR-HUNT: Attacking CPUbound encryption,” in Proc. 28th Annu. Comput. Security Appl. Conf., 2012, pp. 71–78.

[12] M. Herlihy and J. Moss, “Transactional memory: Architectural support for lock-free data structures,” in Proc. 20th Annu. Int. Symp. Comput. Archit., 1993, pp. 289–300.

[13] Intel,, “Chapter 8: Intel transactional memory synchronization extensions,” in Intel Architecture Instruction Set Extensions Programming Reference, 2012. https://software.intel.com/sites/default/ files/m/9/2/3/41604.

[14] Intel, “Intel advanced encryption standard (AES) new instructions set,” 2012.

[15] C. Koc, “High-speed RSA implementation,” RSA Laboratories, Bedford, MA , USA, 1994, TR201.

[16] P. Simmons, “Security through Amnesia: A software-based solution to the cold boot attack on disk encryption,” in Proc. 27th Annu. Comput. Security Appl. Conf., 2011, pp. 73–82.

[17] f.Zyulkyarov, V. Gajinov, et al., “Atomic Quake: Using transactional memory in an interactive multiplayer game server,” in Proc. 14th ACM SIGPLAN Symp. Principles Practice Parallel Program., 2009, pp. 25–34.

[18] Y. Zhao, J. Lin, W. Pan, C. Xue, F. Zheng, and Z. Ma, “RegRSA: Using registers as buffers to resist memory disclosure attacks,” in Proc. IFIP Int. Conf. ICT Syst. Security Privacy Protection, 2016, pp. 293–307.

[19] R. Yoo, C. Hughes, K. Lai, and R. Rajwar, “Performance evaluation of Intel transactional synchronization extensions for highperformance computing,” in Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal., 2013, Art. no. 19.

[20] T. Karnagel, R. Dementiev, et al., “Improving in-memory database index performance with Intel transactional synchronization extensions,” in Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit., 2014, pp. 476–487.

[21] F. Brasser, U. Muller, A. Dmitrienko, K. Kostiainen, S. Capkun, € and A.-R. Sadeghi, “Software grand exposure: SGX cache attacks are practical,” in Proc. 11th USENIX Conf. Offensive Technol., 2017, pp. 11–11.

[22] J.-W. Chung, M. Dalton, H. Kannan, and C. Kozyrakis,

“Threadsafe dynamic binary translation using transactional memory,” in Proc. IEEE 14th Int. Symp. High Perform. Comput. Archit., 2008, pp. 279–289.

[23] C. Cowan, S. Beattie, J. Johansen, and P. Wagle, “PointGuardTM: Protecting pointers from buffer overflow vulnerabilities,” in Proc. 12th Conf. USENIX Security, Symp., 2003, pp. 7–7.

[24] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum, “Hybrid transactional memory,” in Proc.12th Int. Conf.Architectural Support Program. Languages Operating Syst., 2006, pp.336-346.

[25] D. Dice, Y. Lev, M. Moir, D. Nussbaum, and M. Olszewski, “Early experience with a commercial hardware transactional memory implementation,” in Proc. 14th Int. Conf. Architectural Support Program. Languages Operating Syst., 2009.

[26] B. Garmany and T. Muller, “PRIME: Private RSA infrastructure € for memory-less encryption,” in Proc. 29th Annu. Comput. Security Appl. Conf., 2013.