TreVisor

OS-Independent Software-Based Full Disk Encryption Secure against Main Memory Attacks

Tilo Müller, Benjamin Taubmann, and Felix C. Freiling

Department of Computer Science
Friedrich-Alexander University of Erlangen-Nuremberg

Abstract. Software-based disk encryption techniques store necessary keys in main memory and are therefore vulnerable to DMA and cold boot attacks which can acquire keys from RAM. Recent research results have shown operating system dependent ways to overcome these attacks. For example, the TRESOR project patches Linux to store AES keys solely on the microprocessor. We present TreVisor, the first software-based and OS-independent solution for full disk encryption that is resistant to main memory attacks. It builds upon BitVisor, a thin virtual machine monitor which implements various security features. Roughly speaking, TreVisor adds the encryption facilities of TRESOR to BitVisor, i. e., we move TRESOR one layer below the operating system into the hypervisor such that secure disk encryption runs transparently for the guest OS. We have tested its compatibility with both Linux and Windows and show positive security and performance results.

1 Introduction

Why Disk Encryption Matters. Disk encryption is an increasingly popular method to protect sensitive data against physical loss and theft of nomadic computer systems. According to a Ponemon survey from 2010 [24], the majority of U. S. enterprises has an overall strategy for data protection from which full disk encryption (FDE) is the fastest growing favorite (59 %). Also the U. S. government recommends agencies to encrypt all data on mobile devices to compensate the lack of physical security outside their agency location [15].

However, widespread FDE solutions under Windows, such as BitLocker [17] and TrueCrypt [30], do *not* protect data effectively in all scenarios where an adversary has physical access to the computer.

Attacks against Disk Encryption. Since public weaknesses in common cryptographic primitives like AES are unknown, practical attacks against FDE often target the key management. Weak passphrases give rise to efficiently guessable keys, a fact that has been exploited by law enforcement authorities for many years. If strong passphrases are used, however, retrieving the cryptographic key can be circumvented by accessing the encrypted data through system subversion. Trojans and system level rootkits are usually sufficient to circumvent any kind of FDE and hard to prevent in general.

F. Bao, P. Samarati, and J. Zhou (Eds.): ACNS 2012, LNCS 7341, pp. 66–83, 2012.

But even if the user is not tricked into installing malicious software, physical access alone can be sufficient for system subversion. The master boot record (MBR) of an encrypted hard disk must necessarily be left unencrypted in software-based solutions for bootstrapping purposes. As a consequence, software keyloggers can always be placed in the master boot record of the disk. Such attacks have been called *evil maid attacks* [14] and typically require access to the target machine twice, once before, and once after the victim has entered the password.

However, evil maid attacks are based on system infiltration and are therefore not always applicable for lawful actions. Instead, physical memory access can be used to break FDE in many cases since widespread FDE products, including BitLocker and TrueCrypt, hold the encryption key in main memory. These attacks require access to the machine while it is running or suspended to RAM, a common scenario in the lawful seizure of evidence.

Historically, the first successful attacks on main memory used direct memory access (DMA). DMA allows devices to bypass the operating system (OS) and to access physical memory directly for performance reasons. Attacks exploiting this feature first came up in 2005 when an Apple Macintosh's system space was compromised by plugging in a malicious iPod via Firewire [3]. Later, similar attacks were used to unlock Windows Vista [21] and Windows 7 [4] even though BitLocker was active. Other interfaces, including PC Card [8,11], PCI Express [5], and Thunderbolt [25], are believed to have the same security issues as Firewire.

Another way to access main memory is to exploit the *remanence effect* of DRAM, i. e., the fact that RAM contents fade away gradually over time. Due to the remanence effect, encryption keys can be restored, e. g., after rebooting the system with a malicious USB flash drive. This type of attack became known as *cold boot* [10] in 2008. Cold boot attacks are generic and pose a threat to all current software-based FDE technologies, including Microsoft's BitLocker, Apple Macintosh's FileVault and Linux' dm-crypt.

Threat Model		Windows	Linux		OS-Independent	
Memory Attack	System Status	BitLocker/ TrueCrypt	dm-crypt/ TrueCrypt	TRESOR/ LoopAmnesia	BitVisor	TreVisor
cold boot	off or S4					
	run/locked	X	X		X	
	S3	X	X		X	
DMA	off or S4					
	run/locked	X	X	X		
	S3	X	X			

Fig. 1. Categories of our threat model. S3 and S4 stands for "suspended to RAM" and "disk", respectively. Vulnerable scenarios are marked with X.

Threat Model. Since it is impossible to protect FDE against system subversion by malware, we delegate this threat to the malware detection community and concentrate on memory attacks that rely on physical access. Such attacks can be classified according to the state of the system when an adversary gained access: (1) *switched off* or *suspended to disk*, (2) *running* but locked, and (3) *suspended to RAM*.

The resultant threat categories as well as countermeasures and vulnerable scenarios are depicted in Fig. 1. If a laptop was lost or got stolen, it was hopefully switched off or suspended to disk. In this category, all common FDE solutions protect the data on disk successfully. If the system was running (but the screen was locked[1]), widespread FDE systems for Windows and Linux are vulnerable to both cold boot and DMA attacks. The same holds for the third category: most machines that are suspended to RAM by the time an adversary gains access are vulnerable today.

Countermeasures. The current technological responses to cold boot attacks attempt to keep the key inside the CPU rather than in RAM: AESSE [18] stores the key inside SSE registers, TRESOR [19] inside debug registers, LoopAmnesia [23] inside MSRs, and FrozenCache [16] inside CPU caches. Besides AES keys, asymmetric keys are considered in the literature [22]. All of those systems treat registers and caches inside the CPU as more secure against attacks than RAM. As long as nobody finds a practical way to read out CPU contents, e. g., by injecting malicious code onto the bus, those systems are in fact more secure than conventional disk encryption systems. Unfortunately, these countermeasures require deep changes within the operating system. This is why all mentioned projects [18,19,23,16] are written for Linux and are not applicable to Windows.

The idea of keeping keys outside of RAM protects against DMA attacks on suspended machines, too, because in TRESOR and LoopAmnesia the key is irretrievably lost during suspend-to-RAM when the CPU is switched off. However, these solutions do not protect against DMA attacks on running machines, because DMA attacks generally allow to compromise the system space and consequently to read out key storage registers by executing code with ring 0 privileges. Currently, Intel's VT-d [1] technology can fully protect against such DMA attacks. Intel VT-d comprises an I/O Memory Mapping Unit (IOMMU) that enables address remapping for DMA data transfers. Just like traditional MMUs that translate virtual to physical addresses, the IOMMU translates device-visible addresses into physical ones. Hence, certain memory locations, e. g., the system space, can effectively be protected. However, Intel VT-d was introduced as virtualization technology and Windows 7 does not use it to protect against DMA attacks.

On the contrary, there exist hypervisor-based systems that do. BitVisor [28], for example, is a thin virtual machine monitor (VMM) which implements various security functionalities for a single guest, meaning that it exploits virtualization technology to enhance security and not to run multiple systems in parallel.

[1] If the screen was *not* locked but a privileged user was logged in, an adversary can access data trivially.

Among others, BitVisor supports full disk encryption and activates the IOMMU to enforce DMA security. But it does *not* protect against cold boot attacks.

Overall, we are not aware of any solution in the literature that (1) is operating system independent, and (2) secure against all threats listed in Fig. 1. Of course, another valid countermeasure is to revert to purely hardware-based FDE systems such as Intel's SSD 320 Series [13]. Since the key is held within the hard disk and cannot be read out, such approaches withstand attacks on main memory. But software-based solutions enjoy many advantages like reduced costs, vendor independence, configurability, and versatility; this is why we focus on software-based solutions in the remainder of this paper.

Contributions. In this paper, we present TreVisor. TreVisor is a software-based solution to FDE that is designed to be

- practically secure against the threats from Fig. 1 (i. e., cold boot and DMA),
- cryptographically secure by supporting the ciphers AES-128, -192, and -256,
- transparent to the operating system by using virtualization technology, and
- fast, in particular through Intel's new AES instruction set [9].

To this end, TreVisor unites two working projects (TRESOR [19] and BitVisor [28]) resulting in a free and robust FDE system that can be used by both Windows and Linux users. To the best of our knowledge, no other system exists that meets all these requirements.

We believe that integrating TRESOR's encryption routine into a hypervisor is a promising way for future FDE solutions, because using virtualization technology for disk encryption has several advantages:

- Hypervisors are isolated from the OS; even root users and local privilege escalations [2] cannot harm the encryption process or retrieve the key.
- All OSs are equally supported; it is possible to access the same encrypted partitions simultaneously from Windows and Linux.
- Hypervisors are small; the risk to have serious programming errors shrinks with the size of code.
- Building up a *full* disk encryption system is easy; only the hypervisor must be present unencrypted.
- DMA threats can centrally be counteracted by VT-d/IOMMU settings.

Altogether, TreVisor brings many of the advantages known from hardware-based FDE, such like resistance against memory attacks and transparency, into a software-based solution. As we show in this paper, TreVisor can achieve all these goals with a performance penalty of about one-third compared to unencrypted disks, which we find acceptable.

However, TreVisor requires both Intel's new AES instruction set (AES-NI) and support for VT-x/VT-d, which is currently available only with Intel Core i5 and Core i7 processors.

Outline. The remainder of our paper is structured as follows: first we give background information about TRESOR and BitVisor in Sect. 2. Then we introduce the implementation of TreVisor from a technical point of view in Sect. 3, followed by an evaluation regarding compatibility (Sect. 4), performance (Sect. 5), and, most notably, security (Sect. 6). Finally, we conclude in Sect. 7.

2 Background

We now briefly give the necessary background on TRESOR (Sect. 2.1) and BitVisor (Sect. 2.2). Readers familiar with these projects can safely skip this section.

2.1 TRESOR

TRESOR [19] runs encryption securely outside RAM; it is a Linux kernel patch for the x64 architecture designed to run AES resistant against cold boot attacks. TRESOR avoids RAM usage completely and runs the key management as well as the AES algorithm entirely on the microprocessor. To that end, some registers of the x64 architecture must permanently be used as cryptographic key storage and are not applicable for their intended purpose. The four breakpoint registers `dr0` to `dr3` have been chosen for that because they are (1) only accessible with ring 0 privileges, (2) large enough to store an AES-256 key, and (3) seldom used by end-users. Indeed, hardware breakpoints cannot be set by userland debuggers like GDB anymore. But given the fact that only developers and reverse engineers need to set breakpoints (and software breakpoints can still be set), the absence of these registers on end-user systems is acceptable as compared with the gain in security.

Other registers, like the SSE and general purpose registers (GPR), are used temporarily inside atomic sections, too. Before leaving the atomic section, these registers are reset to null and hence, they are safe to be swapped out to RAM during context switching. Only debug registers are not safe to be swapped out by context switching and cannot be used by other threads.

To overcome future cryptanalysis based on memory residues, TRESOR follows a strict security policy: no intermediate state of AES is ever going to RAM, meaning that nothing but the output block is written back to RAM after the input block has been read. Thereto, TRESOR's atomic sections encompass encryption (resp. decryption) of entire AES blocks.

AES uses a key schedule with 10-14 round keys based on the secret key. This key schedule is computed once and then stored inside RAM by all conventional AES implementations for performance reasons. But in TRESOR, the debug registers are fully occupied with the AES key itself and round keys cannot be stored permanently. Therefore, TRESOR uses a so called *on-the-fly key schedule*, meaning that each round key is re-computed inside the atomic section of each block. This contains a potential performance drawback; we get back on this in Sect. 5.

Despite this drawback, or because of it, TRESOR accelerates its AES computations by Intel's new AES instruction set (AES-NI) that implements AES

efficiently in hardware. It is currently available to Intel Core i7 processors, most Core i5 and will be available to many upcoming x86 CPU, including AMD processors.

To sum up, TRESOR is a cold boot resistant implementation of AES, primarily designed for hard disk encryption. It is currently restricted to CPUs of the Core-i series, and so is TreVisor.

2.2 BitVisor

Traditional VMMs like Xen and VMware require numerous components to provide virtual hardware devices that can be shared among parallel guests. To the contrary, BitVisor [28] is a thin hypervisor architecture based on Intel VT-x (and AMD-V) which is designed to enforce I/O device security of single VMs. It is OS-independent, meaning that the VM can run unmodified versions of Windows, Linux, or any other x86 operating system.

BitVisor minimizes the overhead introduced by virtualization, leading to a so called *parapass-through* architecture: hardware is directly passed through to the guest except for a few administrated devices. Thereby, the need for most device drivers is eliminated inside the VMM and the guest OS handles devices directly. The exceptional case are devices which must explicitly be administrated in order to enforce security functionalities. In our disk encryption scenario, these are primarily hard disks. Thereto, BitVisor comes with its own set of *parapass-through drivers* for (S)ATA disks; these drivers know the specification of the target device and can handle intercepted I/O accesses correctly without fully virtualizing them. Extracted data, such as the sectors of a hard disk, can be manipulated by the VMM, e. g., for encryption and decryption.

In order to prevent attacks from malicious I/O devices against the memory region of the hypervisor, external DMA accesses are restricted by the IOMMU. From the operating system's point of view, I/O devices still have access to the system space. But since the disk encryption routine runs with ring -1 privileges, BitVisor's IOMMU settings guarantee resistance against read- and writeable DMA attacks.

Using the IOMMU to restrict direct memory access, no parapass-through drivers are required. For example, DMA attacks based on Firewire are successfully defeated while parapass-through drivers for Firewire can be excluded.

To sum up, BitVisor is a secure lightweight hypervisor which, by running only a single VM, eliminates the need for most components that are required to share system resources among VMs.

3 Design and Implementation

The use cases of TreVisor are mobile single-user systems, i. e., laptops, because these get frequently lost and stolen at public places like airports while running or being suspended, and left unattended in hotel rooms. In other words, TreVisor is designed to protect systems that are especially exposed to attacks based on

physical access. Additionally, TreVisor should be available for Windows users, or, even better, be operating system independent.

The requirement to be OS-independent quickly brings hypervisor-based solutions into mind. A first idea is to move a solution like TRESOR [19] or LoopAmnesia [23] into a Linux-based hypervisor like Xen or VMware and to run Windows on top of that. Such a design would have had the advantage that TRESOR, which is a Linux kernel patch, could have been applied to the setup directly. However, we generally consider such a design as bad because running a second, full operating system introduces significant overhead. Instead, TreVisor is implemented as a thin hypervisor for single guests, meaning that no resource must be virtualized or shared among VMs, drastically facilitating the VMM implementation. Only hard disk accesses are intercepted in order to encrypt and decrypt them securely with TRESOR. Everything else, like the keyboard, mouse, printer, video, and sound card, is passed through to the guest without intervention.

In a nutshell, TreVisor is implemented as a patch for BitVisor and introduces the OS-independent parts of TRESOR to it. That is, to enforce security measures, TreVisor exploits hardware capabilities of modern Intel CPUs that are otherwise hardly used by end-users. Usually, end-user systems do neither utilize the debugging registers nor do they make use of the VT-d/IOMMU technology. TreVisor activates these, otherwise mostly idle, components to protect against cold boot and DMA attacks.

To sum up, TreVisor aims to be a disk encryption solution that prevents information leakages through main memory attacks while being transparent to the OS and to the end-user. Thereto it combines two technologies: BitVisor taken by itself is not resistant against cold boot attacks, and TRESOR does neither support Windows nor does it defeat DMA attacks on running machines.

In the following sections we describe technical challenges we faced when integrating TRESOR's encryption routines into the BitVisor code.

Key Storage Registers. TRESOR employs the debug registers `dr0` to `dr3` as cryptographic key storage, because those are (1) only accessible from ring 0, (2) large enough to store AES-256 keys, and (3) seldom used by end-users. While point two and three remain valid inside hypervisors, point one does not because we need to protect the key against ring 0 (the guest's kernel space) rather than ring 3 (the userland). In other words, in TreVisor we need a set of registers that is only accessible with ring -1 privileges.

As such a set of registers does not exist, we have to stay with the debug registers. Luckily, virtualization allows us to define sensitive events that lead the processor to switch context from the guest into the hypervisor, e. g., on executing privileged instructions or on accessing certain registers. Using VM execution control [12], we can effectively hide the debug registers from ring 0, i. e., from the guest OS, as follows:

- `cpuid` is an instruction that provides information about present CPU features. We hook into `cpuid` and forge its result by negating the DE (debugging extension) feature.

- `cr4` is an x86 control register that contains a flag to enable/disable debugging. We intercept `cr4` write accesses and deny all attempts to enable debugging.
- "MOV-DR exiting" is a virtualization control which causes the VM to exit on every `mov` instruction to or from debug registers. We enable this feature to ultimately prevent the guest from reading or overwriting the key.

All these measures cause unconditional VM exits and bring the hypervisor into action. The first measure is required to inform the OS about missing debugging capabilities. Since we generally disallow debugging in the second measure, an OS might react unexpectedly (e. g., crash) if it assumes debugging is present. The third measure is required to deny any read and write access to debug registers. Of course, a well programmed OS would never try to access a debug register if the CPUID negates its presence, but we want to be on the safe side for two reasons:

First, the OS (or any device driver) might be erroneous and write to debug registers despite their alleged absence. This immediately leads to data corruption on the encrypted hard disk because the key gets falsified. Second, an adversary who compromises the guest's system space could easily retrieve the secret key.

Summarizing, we used virtualization techniques to effectively change the privilege level of the debug registers from 0 to -1.

Key Management. As pointed out in the last section, we can get exclusive control over debug registers inside the hypervisor. We now take this exclusive access for granted and examine how to securely read the key from a user prompt into those registers. Due to numerous encryption features, BitVisor already comes with some kind of key management. Unfortunately, this key management is inadequate for our purpose as it stores passwords and keys in RAM (where they are vulnerable to cold boot attacks). Thus, we have to replace BitVisor's key management with a more secure one. To this end, we display a TreVisor specific password prompt at boot time.

The password is transformed into a 256-bit key by multiple SHA-256 iterations and then copied into the debug registers. We make sure that all residues of both the password and the key are erased from RAM thoroughly. Therefore we do *not* send the password or key into untrusted libraries; instead we use a custom implementation of the SHA-256 algorithm.

To support multicore CPUs, we have to write the secret key into the debug registers of all CPUs because we cannot assume that the TreVisor code is always executed on CPU0. To the contrary, hypervisor code is generally executed on the CPU which led to the VM-exit. And since process migration is a costly task, we distribute the secret key among all cores.

Writing the secret key into different cores turned out to be more complicated than we expected. The problem actually is that BitVisor itself does not fully initialize the APs (application processors) but runs on the BSP (boot strap processor) until the OS boots. In other words, BitVisor leaves it to the guest OS to set up remaining cores. As a solution we hook into the first inter-processor interrupt (IPI) of each CPU; IPIs are used by the OS to signal events between

processors. TreVisor intercepts the start-up signals in order to initialize the debug registers with keybits before they may be used for encryption. When a guest OS does not activate all cores, the key will only be present in those that have been activated.

The mechanism we implemented requires that the keybits are temporarily copied from the BSP into RAM and then further into the APs. Unfortunately, it is impossible to copy data between processors directly. That is, our solution might not be absolutely secure against cold boot attacks for the time of initializing a new CPU. But all OSs that we are aware of initialize CPU cores during boot-up or never and thus, we consider the risk as negligible.

Disk Encryption. TreVisor supports disk encryption with AES-128, AES-192, and AES-256. During startup, we always copy 256 keybits into the debug registers; if AES-128 or AES-192 are used, superfluous bits are just ignored.

In order to encrypt the disk, we have to hook into the guest's HDD activities. Fortunately, BitVisor already provides most of the necessary functionality, including parapass-through drivers for ATA and USB disks. To put it simply, we only have to replace the standard AES routines in BitVisor by TRESOR. To this end, we add TRESOR to an internal crypto API and activate it within the configuration.

Comparable to context switches inside an OS, hypervisors must switch the context between the guest and itself. Thereto Intel VT-x supports virtual machine control structures (VMCS) to hold guest states. These structures encompass all necessary registers, e. g., the GPRs and potentially the SSE and debug registers, too. On multicore systems, each processor has its own guest context.

Since TRESOR uses GPR and SSE registers, we have to run disk encryption routines inside a critical section where interrupts and context switches are disabled. Only by running TRESOR atomically, we can guarantee that no intermediate state of AES or the key schedule is ever going to the VMCS structures in RAM. Before leaving the atomic section, we zero-fill the GPR and SSE registers so that they are safe to be swapped out.

Additionally we have to take care of the SSE task switch bit (TS) and make sure that we save/restore the guest's SSE state before/after encrypting a disk block. The SSE registers are quite large (4 kilobits in total) and since a hypervisor usually does not use them, they are not saved by default for performance reasons.

Suspend to RAM. Lastly, we spent considerable effort to support ACPI suspend modes in TreVisor. Since the CPU is switched off during suspend modes, we have to re-read the key upon wakeup. Normally, all CPU registers are backed up in RAM during suspend modes, but in TreVisor the debug registers are naturally prevented from being stored inside RAM.

The ACPI mode S3 (suspend-to-RAM) is basically supported by BitVisor since release 1.2 (October 2011). We consider S3 as an important feature for mobile users as it speeds up the boot process and reduces power consumption. It just enhances the usefulness of mobile environments. Above that, it is especially

important to the cold boot scenario since the key of conventional FDE systems is *not* lost during S3.

Therefore we want to support S3 in TreVisor, and "all" we have to do is to re-read the key upon wakeup. This may sound easy, but in practice, video cards continuously fail to get re-initialized after S3 and the screen stays blank (until we return to the OS which re-initializes the video card smoothly thanks to the parapass-through architecture). We ended up with a workaround where the user must re-enter the password "blind" and thus, we must consider S3 support in TreVisor still as work in progress.

4 Compatibility

We analyzed TreVisor regarding its compatibility with userland programs, operating systems, encrypted partitions and hardware components.

Userland Programs. Generally, userland programs are supported unless they use one of the hardware components which are occupied by TreVisor; these are VT-x/VT-d and the debugging extensions. Although the TRESOR encryption routines additionally use multimedia (SSE) and general purpose registers, other programs are fully supported, including office and internet applications, 3D games, simulations, and more.

Debuggers are *not* fully supported. In particular hardware breakpoints cannot be set as we have verified with GDB in Linux and OllyDBG in Windows. Although the running debugger does not crash, setting a hardware breakpoint has just no effect (since TreVisor effectively prevents overwriting the debug registers). However, setting software breakpoints works fine and software breakpoints are the default breakpoints in most debuggers.

Virtualization software like VirtualBox and VMware is not fully supported either. Again, these programs do not crash but they run less efficiently as they cannot make use of Intel's VT-x (it is already used by TreVisor). The problem could be solved in the future because nested VT-x is generally possible when supported by the lowest hypervisor.

Operating Systems. One of the most important advantages of TreVisor compared to TRESOR is its capability to run an unmodified version of basically every x86 OS, including Windows 7, Linux, BSD variants, and more. We have verified this for the most important OSs, in particular for the 32- and 64-bit variants of Windows and Linux.

During development, the Windows operating system and its device drivers caused somehow more trouble than Linux. For example, booting up the 64-bit variant of Windows 7 fails with a blue screen when the CPUID negates debugging extensions. The problem can magically be solved by not changing the return value of CPUID but still disabling the debugging extensions (which is in fact a wrong configuration).

Encrypted Partitions. TreVisor supports full AES, including the 128-, 192-, and 256-bit variant; other ciphers are not supported.

Indeed, TreVisor allows to encrypt several partitions, but all of those must be encrypted with the same secret key because no space is left to store additional keys in debug registers. A possible solution to this problem is to store a keyring in RAM that is encrypted with a *master key*. As a downside, this solution induces a significant performance drawback because the key has to be decrypted before scrambling a block. Hence, and since TreVisor is primarily designed for single-user systems, we decided against it.

Partitions that are encrypted with TreVisor can be used by both Windows and Linux. Using partitions that were encrypted with another disk encryption software is not possible (at least not without re-encryption).

Hardware Components. Since we utilize recent Intel technologies, we had to write TreVisor close to the hardware and consequently, there *are* incompatibilities with many existing systems. It is almost impossible to support older CPUs because hardware virtualization simplifies the implementation of hypervisors considerably.

From current Intel CPUs, only the Core i5 and i7 series are compatible with TreVisor as these are the only 64-bit CPUs that support VT-x/VT-d as well as the AES instructions. We have tested TreVisor successfully with both, the i5 and i7. Most notably, CPU frequency scaling features can, unfortunately, not be used from within the guest at the time of this writing.

As mentioned above, we are also aware of problems with some video cards and ACPI wakeup. After TreVisor resumes from suspend-to-RAM, users must re-enter the password "blind" because the password prompt regularly fails to get displayed.

Tu sum up, guaranteeing the support of a wide range of commercial hardware components, including CPUs, video cards, and ACPI, is a difficult task. In this sense, TreVisor must be seen as an academic prototype, not as a market-ready product.

5 Performance

We now present TreVisor disk encryption benchmarks. We have evaluated the performance of TreVisor on two different systems: Windows 7 (64-bit) and Ubuntu Linux (64-bit, kernel 3.0). On both systems our tests revealed practical benchmark data; the performance drawback compared to unencrypted disks, and compared to disks encrypted with AES-NI, is about one-third.

Table 1 illustrates the encryption and decryption speed of TreVisor in megabytes per seconds. We list the encryption speed of all three TRESOR variants (TRESOR-128, -192, and -256) and compared TRESOR-256 with reference values of plain disk access (No-VMM), BitVisor without encryption (No-Crypto), BitVisor with our own, memory-based AES-NI implementation (AES-NI/256), and BitVisor with its default, OpenSSL-based AES variant (StdAES/256).

Table 1. Basic throughput data (a) of TreVisor in MB/s. The penalty of the reference values (b) is in comparison to the throughput of TRESOR-256.

	TRESOR-128	TRESOR-192	TRESOR-256
Linux/write	59.3	54.4	56.0
Linux/read	38.8	36.3	32.5
Windows	43.4	39.4	35.3

(a)

	No-VMM		No-Crypto		AES-NI/256		StdAES/256	
Linux/write	60.7	7.74 %	59.9	6.51 %	59.2	5.41 %	57.6	2.78 %
Linux/read	63.7	48.98 %	63.2	48.58 %	62.9	48.33 %	37.8	14.02 %
Windows	54.1	34.75 %	53.9	34.51 %	53.4	33.90 %	41.4	14.73 %

(b)

Under Linux, we evaluated the encryption speed (write to an encrypted partition) as well as the decryption speed (read from an encrypted partition) with the `dd` utility. To minimize the effect of disk caching, we copied large files (10 gigabytes) that do not fit into RAM and averaged over several test runs. Under Windows we used the PC analysis software SiSoft Sandra 2011 to create combined disk benchmarks (read and write). It remains unclear how the values are determined by SiSoft Sandra in detail and thus, the inferior throughput compared to Linux does not necessarily mean that hard disks under Windows work slower.

We took the disk speed of BitVisor without encryption (No-Crypto) into account to investigate the performance drawback that arises from hooking into disk writes without modifying them. As shown by Table 1, this effect is minor. Additionally we added our own, memory-based implementation of AES to BitVisor (AES-NI/256), because the default implementation of BitVisor does not utilize the AES instruction set. To investigate the exact performance drawback caused by TRESOR's on-the-fly key schedule, we patched TRESOR's encryption routine to store all round keys persistently in RAM. We were surprised by the outstanding acceleration attributed to AES-NI; the difference between AES-NI and no encryption is nominal.

As shown in Table 1, the performance drawback of TRESOR-256 compared to standard AES-256 is only about 3 % for writing but about 14 % for reading. This effect stems from the on-the-fly key schedule: decryption round keys are derived from encryption round keys by an additional, costly operation (namely `aesimc`, *inverse mix columns*). As the key schedule must be recomputed for each input block, this extra operation becomes noticeable. In comparison with plain (not encrypted) disks the effect intensifies and the performance decreases to almost 50 % for decryption/reading. Writing, on the other hand, is not affected by the extra operation and the performance decrease compared to all reference values is below 10 %.

In summary, TreVisor can practically be employed without a noteable performance drawback for encryption, but with a performance drawback of up to 50 % for decryption. However, compared to the gain in security, we consider this performance loss as acceptable. Furthermore, we want to point to the combined read/write benchmarks under Windows, revealing a decrease of about one-third on average. Lastly, we want to note that the decrease relative to traditional, non AES-NI implementations is below 15 %; and those systems were practically deployed for many years until AES-NI CPUs came up recently.

6 Security

First we show that TreVisor is, above all, secure against cold boot attacks and DMA attacks. Furthermore it can be configured in a way that is mostly secure against evil maid attacks. Last we discuss timing attacks and attacks by privileged users.

Cold Boot Attacks. The major development object of TreVisor is to offer Windows users the opportunity to encrypt their hard disks resistant to cold boot attacks. All existent, software-based FDE systems for Windows store necessary keys in RAM. As shown by a recent study about the practicalness of cold boot attacks [6], this problem *must* be taken seriously.

To defeat cold boot attacks we consistently treat RAM as insecure and adopted TRESOR's techniques to hold keys in CPU registers. But the correctness of TRESOR does not necessarily imply the correctness of TreVisor. We have to show that no programming error slipped into our implementation that, for example, leads the debug registers to be written out into RAM during context switching.

To prove resistance against cold boot attacks we analyzed the RAM of a TreVisor system in order to show that (1) debug registers are never swapped out into VMCS structures, and (2) GPR and SSE registers are not swapped out inside critical sections. The most comfortable way to analyze the memory of a TreVisor system is to run TreVisor itself inside a VM and examine its memory from outside. Unfortunately, nested VT-x does not work in current virtualization software; either it is not supported at all, or, as in VMware (4.0.2), it is officially supported but still causes TreVisor to crash.

Hence, we had to examine memory via cold boot attacks. We booted a mini OS from USB as well as over network (PXE) and dumped everything that was left in memory. Additionally we used BitVisor's debug console to examine the entire main memory at runtime and patched TreVisor to go through the crucial VMCS structures. Last but not least, we implemented log messages on access to debug registers.

Instead of using `aeskeyfind` [10], which is based on the AES key schedule that is not stored in TreVisor at all, we had advantage over real attackers by searching for known keybits. Despite this advantage, we were not able to find any match of the key that exceeds three bytes, a finding which can be explained by chance alone. On the other hand, when we explicitly wrote keybits into RAM, we *were* able to recover them.

Summarizing, we were not able to recover keybits, the key schedule or any part of them despite strenuous efforts.

DMA Attacks. Read-only DMA attacks are prevented by TreVisor as explained in the previous section – since the TreVisor key never enters RAM, it cannot be retrieved by reading from RAM. However, writeable DMA attacks may still be harmful because they can compromise the system space and execute privileged code, e. g., code that displays the debug registers.

Writeable DMA attacks can only be circumvented by restricting the memory space that is available to DMA capable I/O ports like Firewire. Such restrictions can effectively be implemented via the IOMMU of Intel's VT-d technology.

Normally, hypervisors use the IOMMU to protect themselves from access of untrusted guest OSs. However, the same mechanism can be used to improve security against DMA attacks, and so does BitVisor. BitVisor verifies that addresses, which are specified by guest DMA descriptors, do not point into hypervisor regions.

It should be mentioned that BitVisor protects only its own memory regions; the system space of the guest is *not* protected and still vulnerable to DMA attacks. But since the disk encryption logic of TreVisor resides in the hypervisor, the key cannot be accessed from the guest.

Summarizing, BitVisor implements an effective approach to protect its memory space against malicious I/O devices and hence, TreVisor is secure against DMA attacks.

Evil Maid Attacks. As a matter of fact, software-based FDE systems do not enforce *full* disk encryption due to bootstrapping reasons. At least the MBR and a small decryption routine must be stored unencrypted in order to launch the remaining system. This weakness affects most existent FDE solutions, including TrueCrypt, and allows for so called evil maid attacks: unencrypted MBRs can be infiltrated with bootkits (which, for example, may have keylogging functionality).

To overcome such attacks, we successfully tested a configuration of TreVisor that enables true FDE, meaning that the master boot record of the disk must not be left unencrypted. For this reason we store both the bootloader and the hypervisor onto an external USB flash drive which is required to be plugged in during boot time. (Additionally, reconfiguring the BIOS should be protected by a password.)

Although this countermeasure thwarts the most evident MBR attacks, it is not perfect. First, the USB device must be handled like a physical key, meaning that loss of the device threatens the protection mechanism. Second, only the integrity of the bootloader can be verified, not the integrity of the BIOS (e. g., BIOS kits [26]). Therefore, we are working on the support of trusted platform modules in future versions of TreVisor as it would be both more convenient and more secure. BitLocker, for example, already supports such a TPM configuration.

Timing Attacks. We want to mention briefly that TRESOR is resistant to another kind of side channel attacks – timing and cache-based attacks [20].

However, this is not special because all disk encryption systems which build upon Intel's AES-NI, including BitLocker and TrueCrypt, *are* resistant to timing attacks. Intel itself states that, beyond improving performance, the AES instruction set provides security benefits by running in data-independent time [27].

Privileged User Attacks. In TreVisor, an adversary who could gain root or administrator privileges *cannot* compromise the key. Of course, such an adversary is mostly able to read out the disk, but the encryption key itself cannot be accessed. This again is an improvement compared to BitLocker and TrueCrypt where the key resides in system space – always available to attacks by the superuser. In TreVisor the key does not reside in system space but ring -1 privileges are required to access it. TreVisor does not grant access from the untrusted guest OS and hence, it is impossible for an adversary to read the key without breaking out of the VM.

Preventing such VM escapes was one of the design goals of BitVisor. The authors argue that BitVisor comprises only about 20 KLOC (kilo lines of code) which is quite small compared to other VMMs. Reduced code size effectively increases the reliability of a hypervisor as it reduces the risk of serious programming errors. For example, local privilege escalations for the Linux kernel appear regularly (e. g., CVE-2009-2692, CVE-2010-3081, and CVE 2012-0056 [2]). Thus, even when using TRESOR, which is a Linux kernel patch, the secret key cannot effectively be protected against local attacks because once a user compromised system space, the key can easily be retrieved. In contrast, a TreVisor user would additionally have to break out of the VM – which is at least a further obstacle.

7 Conclusions and Future Work

In this paper we described TreVisor, a disk encryption scheme that is primarily designed to resist main memory attacks. Our proposal goes along with a prototype implementation that runs reliably in practice but allows for many improvements in future work, too.

Conclusions. Software based disk encryption solutions like BitLocker and TrueCrypt are designed to preserve confidentiality and integrity in the case of physical loss. But in many practical cases they do *not* fulfill these requirements as it has been shown by several known attacks: cold boot, DMA, and evil maid attacks.

Hence, there is practical need for a disk encryption solution that finally integrates countermeasures to all these threats [7]. TreVisor, which claims to be such a solution, has several advantages as compared with conventional disk encryption software:

- As only a hypervisor must be present unencrypted, truly full disk encryption can easily be enforced. For example, the small hypervisor can be stored on external bootable devices, improving protection against evil maid attacks.
- As VT-x technology is utilized to run below the operating system, TreVisor encrypts transparently for the OS. That is, TreVisor permits, for example, Linux and Windows to access the same encrypted partition simultaneously.

– As it is based on TRESOR and VT-d/IOMMU technology, TreVisor secures against attacks on main memory, namely cold boot and DMA attacks.

We believe that exploiting otherwise hardly used components, i. e., the debug registers and VT-x/VT-d, is a reasonable way to deploy more secure disk encryption in mobile end-user systems.

To conclude, TreVisor substantially increases the security of disk encryption systems. It is the first system which is secure against known main memory attacks, in particular against cold boot and DMA attacks. Before TreVisor, countermeasures against these attacks were spread over small, academic projects and serious disk encryption systems still do not implement them.

Future Work. We believe that utilizing virtualization technology in order to enforce OS-independent disk encryption is the most promising software-alternative to compete with hardware-FDE, which becomes increasingly popular. At the time of this writing, however, TreVisor can only be treated as a prototype of what *could* be done in future. To be deployable on the mass market, many issues have to be solved, mainly regarding usability and compatibility:

– Easy installation, particularly for Windows users. Currently, TreVisor must be compiled under Linux and manually be started from within GRUB before booting the OS.
– User-friendly suspend-to-RAM support. At the moment, suspend-to-RAM must be considered as "experimental" since TreVisor fails to display a visual password prompt.
– CPU frequency scaling from within the guest. This is especially important to save battery life of notebooks.

Above that, we are continuously working on further security improvements, primarily on the integration of trusted platform modules into the boot process. Proving the integrity of boot components by means of the TPM would be a secure and convenient add-on to TreVisor in the future.

Acknowledgments. We would like to thank *Richard Mäckl*, *Johannes Stüttgen*, and *Stefan Vömel* as well as the anonymous reviewers for reading a prior version of this paper and giving us valuable suggestions for improving it.

Availability. TreVisor is free software which is published under the GPL v2 [29]. Its source code is publicly available at `www1.cs.fau.de/trevisor`

References

1. Abramson, D., Jackson, J., Muthrasanallur, S., Neiger, G., Regnier, G., Sankaran, R., Schoinas, I., Uhlig, R., Vembu, B., Wieger, J.: Intel Virtualization Technology for Directed I/O. Intel Technology Journal 10 (August 2006)
2. Aedla, J.: Linux Kernel CVE-2012-0056 Local Privilege Escalation Vulnerability (January 2012); Common Vulnerabilities and Exposures, `http://www.securityfocus.com/bid/51625/`

3. Becher, M., Dornseif, M., Klein, C.N.: FireWire - All Your Memory Are Belong To Us. In: Proceedings of the Annual CanSecWest Applied Security Conference, Vancouver, British Columbia, Canada. Laboratory for Dependable Distributed Systems, RWTH Aachen University (2005)
4. Böck, B.: Firewire-based Physical Security Attacks on Windows 7, EFS and BitLocker. Secure Business Austria Research Lab (August 2009)
5. Carrier, B.D., Spafford, E.H.: Getting Physical with the Digital Investigation Process. IJDE 2(2) (2003)
6. Carbone, Bean, Salois: An in-depth analysis of the cold boot attack. Technical report, DRDC Valcartier, Defence Research and Development, Canada, Technical Memorandum (January 2011)
7. Cardwell, M.: Protecting a Laptop from Simple and Sophisticated Attacks (August 2011), https://grepular.com/Protecting_a_Laptop_from_Simple_and_Sophisticated_Attacks
8. Devine, C., Vissian, G.: Compromission physique par le bus PCI. In: Proceedings of SSTIC 2009. Thales Security Systems (June 2009)
9. Gueron, S.: Intel's New AES Instructions for Enhanced Performance and Security. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 51–66. Springer, Heidelberg (2009)
10. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest We Remember: Cold Boot Attacks on Encryptions Keys. In: Proceedings of the 17th USENIX Security Symposium, San Jose, CA, pp. 45–60. Princeton University, USENIX Association (2008)
11. Hulton, D.: Cardbus Bus-Mastering: 0wning the Laptop. In: Proceedings of ShmooCon 2006, Washington DC, USA (January 2006)
12. Intel Corporation. Intel 64 and IA-32 Architectures Developer's Manual, Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C edition (December 2011)
13. Intel Corporation. Solid-State Drive 320 Series (2011), http://www.intel.com/content/www/us/en/solid-state-drives/solid-state-drives-320-series.html
14. Rutkowska, J.: Evil Maid goes after TrueCrypt. The Invisible Things Lab (October 2009), http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html
15. Johnson, C.: Protection of Sensitive Agency Information. U.S. Executive Office of the President, Washington, D.C. 20503 (June 2006)
16. Pabel, J.: Frozen Cache (January 2009), http://frozenchache.blogspot.com/
17. Microsoft Corporation. Windows BitLocker Drive Encryption: Technical Overview. Microsoft (July 2009)
18. Müller, T., Dewald, A., Freiling, F.: AESSE: A Cold-Boot Resistant Implementation of AES. In: Proceedings of the Third European Workshop on System Security (EUROSEC), Paris, France, pp. 42–47. RWTH Aachen / Mannheim University, ACM (April 2010)
19. Müller, T., Freiling, F., Dewald, A.: TRESOR Runs Encryption Securely Outside RAM. In: 20th USENIX Security Symposium, San Francisco, California. University of Erlangen-Nuremberg, USENIX Association (August 2011)
20. Osvik, D.A., Shamir, A., Tromer, E.: Cache Attacks and Countermeasures: The Case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)

21. Panholzer, P.: Physical Security Attacks on Windows Vista. Technical report. SEC Consult Vulnerability Lab, Vienna (May 2008)
22. Parker, T.P., Xu, S.: A Method for Safekeeping Cryptographic Keys from Memory Disclosure Attacks. In: Chen, L., Yung, M. (eds.) INTRUST 2009. LNCS, vol. 6163, pp. 39–59. Springer, Heidelberg (2010)
23. Simmons, P.: Security Through Amnesia: A Software-Based Solution to the Cold Boot Attack on Disk Encryption. CoRR, abs/1104.4843. University of Illinois at Urbana-Champaign (2011)
24. Ponemon, L.: 2010 Annual Study: U.S. Enterprise Encryption Trends. Ponemon Institute, Symantec (2010)
25. Graham, R.D.: Thunderbolt: Introducing a new way to hack Macs. Errata Security, `http://erratasec.blogspot.com/2011/02/thunderbolt-introducing-new-way-to-hack.html` (February 2011)
26. Sacco, A.L., Ortega, A.A.: Persistent BIOS Infection: The early bird catches the worm. In: Proceedings of the Annual CanSecWest Applied Security Conference, Vancouver, British Columbia, Canada. Core Security Technologies (2009)
27. Gueron, S.: Intel Advanced Encryption Standard (AES) Instruction Set White Paper. Intel Corporation, Rev. 3.0 edn. Intel Mobility Group, Israel Development Center (January 2010)
28. Shinagawa, T., Eiraku, H., Omote, K., Hasegawa, S., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y., Kato, K.: In: International Conference on Virtual Execution Environments, Washington, DC, USA. University of Tsukuba (March 2009)
29. Richard Stallman and Jerry Cohen. GNU General Public License Version 2. Free Software Foundation (June 1991)
30. TrueCrypt Foundation. TrueCrypt: Free Open-Source Disk Encryption Software for Windows, Mac OS and Linux (2010), `http://www.truecrypt.org/`