2014 IEEE Fourth International Conference on Big Data and Cloud Computing

Fine Grain Cross-VM Attacks on Xen and VMware

Gorka Irazoqui*, Mehmet Sinan Inci*, Thomas Eisenbarth*, Berk Sunar*
*Worcester Polytechnic Institute
{girazoki,msinci,teisenbarth,sunar}@wpi.edu

Abstract—**This work exposes vulnerabilities in virtualized cloud servers by mounting Cross-VM cache attacks in Xen and VMware VMs. We show for the first time that AES implementations in a number popular cryptographic libraries including `OpenSSL`, `PolarSSL` and `Libgcrypt` have non-constant execution times and are vulnerable to Bernstein's correlation attack when run in Xen and VMware (bare metal version) VMs. We show that the vulnerability persists even if the VMs are running on different cores in the same machine. Experiments on Amazon EC2 and Google Compute Engine highlight the practical implications of the found vulnerability. The results of this study show that there remains a security risk to AES implementations of popular libraries and data encrypted under AES on popular cloud services.**

Keywords: Cryptographic libraries, Cross-VM attacks, cache timing attacks, virtualization, AES key recovery attack.

I. Motivation

In as little as a decade cloud computing and cloud enabled applications have become mainstream. IT services of many businesses and even popular personal computing applications such as Dropbox and Netflix run on the infrastructure provided by Cloud Service Providers (CSPs) such as Amazon and Google. The biggest concern preventing companies and individuals from further adopting the cloud is data security and personal privacy. Systems running on the cloud use libraries designed for single-user server world, where adversaries can only interact over well-defined network interfaces. In contrast, in the cloud, malicious code may be running on the same machine as the crypto stack, separated only by a virtual machine manager (VMM). Indeed, the main security principle in the design and implementation of VMMs has been that of process isolation achieved through *sandboxing*. However, sandboxing has traditionally been defined in the software space, ignoring leakages of information through subtle side-channels shared by the processes running on the same physical hardware. In non-virtualized environments, a number of effective side-channel attacks were proposed that succeeded in extracting sensitive data by targeting the software layer only. For instance, the early proposal by Bernstein [1] (and later in [2], [3]) targets the time variation in accessing the cache to extract AES encryption keys while the proposal by Acıiçmez targets the key dependent time variations in the hardware branch prediction unit [4].

Given the level of access to the server and control of process execution required to realize a successful physical attack, these attacks have been largely dismissed in the cloud setting until fairly recently.

As early as in 2003, D. Brumley and Boneh [5] demonstrated timing side-channel vulnerabilities in `OpenSSL 0.9.7` [6]. The authors ran several experiments with processes running on the same processor and managed to recover RSA private keys from an `OpenSSL`-based web server using packet timing information. As a consequence of this attack, RSA implementations in `OpenSSL` use blinding to counter timing attacks [7]. Nevertheless, as recently as in 2011, B.B. Brumley and Tuveri [8] demonstrated that the ladder computation operation in the ECDSA computation [9] is vulnerable to a timing attack. The authors were able to steal the private key used for server authentication in a TLS handshake implemented using `OpenSSL 0.9.8o`.

Later, in 2011, Suzaki et al. [10], [11] exploited an OS optimization, namely KSM, to recover user data and subsequently identify a user from a co-located guest, hosted in a Linux Kernel-based Virtual Machine (KVM).

In 2009, Ristenpart et al. [12] demonstrated that it is possible to solve the logistics problems and extract sensitive data across VMs. Specifically, using the Amazon EC2 service as a case study, Ristenpart et al. demonstrated that it is possible to identify where a particular target VM is likely to reside, and then instantiate new VMs until one becomes co-resident with the target VM. Even further, the authors show how cache contention between co-located Xen VMs may be exploited to deduce keystrokes with high success probability. By solving the co-location problem, this initial result fueled further research in Cross-VM side-channel attacks.

Shortly later, Zhang et al. [13] presented an access-driven side-channel attack implemented across Xen VMs that manages to extract fine grain information from a co-located victim VM. More specifically, the authors manage to recover an ElGamal decryption key from a victim VM using a cache timing attack. To cope with noise and thereby reduce the errors, the authors utilize a hidden Markov model. The significance of this work is that for the first time the authors were able to extract *fine grain* information across VMs, in contrast to the earlier work of Ristenpart et al. [12] who managed to extract keystroke patterns. Yarom et al. [14] describe a trace-driven flush and reload attack and note that it could be applied in virtualized environments. In [15] Weiß et al. for the first time present a cache timing attack on AES in a L4Re VM running on an ARM Cortex-A8 processor inside a `Fiasco.OC` microkernel. The attack is realized using Bernstein's correlation attack and targets several popular AES implementations including the one in `OpenSSL`. The significance of this work is that it showed that it is possible to extract even finer grain information (AES vs. ElGamal keys in [13]) inside a VM.

In this work, we push the envelope even further by utilizing Bernstein's correlation attack (and its last round variation) to recover AES keys from popular cryptographic libraries in Xen, VMware and KVM virtualized environments. Our work shows that it is possible to recover fine grain information, i.e. AES keys, in commonly used cryptographic software packages, running inside and across Xen, VMware and KVM, even if VMs are located on *different cores* on the same machine. We repeat the experiments, by running the attack natively as well as Cross-VM—establishing that for most cases the attack success degrades only mildly when the target and victim processes are taken from native to Cross-VM execution. We further validate that the attacks are possible in commercial clouds running on Xen and KVM hypervisors, namely Amazon's EC2 and Google's GCE.

II. Background

Our goal is to set up a testbed that allows us to evaluate whether fine-grain Cross-VM attacks can be applied in a real world scenario.

As previously mentioned, the first fine-grained side-channel attack on a server system was Bernstein's attack on AES [1]. The attack exploits data-dependent timing behavior that is introduced by the underlying cache architecture of the host system. Bernstein's attack has often been disregarded as it is considered impractical when performed over a network (e.g. in [16]). However, due to the quickly increasing popularity of the cloud services, timing noise stemming from the network connection is no longer an obstacle. In public clouds, determined adversaries are able to place malicious hosts on the same machine as a target host [12]. The malicious hosts can then monitor and manipulate shared resources, including caches and other hardware resources in order to extract critical information from the target. In the following we review the cache architecture as well as attacks that exploit this behavior. We will focus on Bernstein's attack, as it gives us a simple tool to study and compare the viability of fine-grained Cross-VM attacks in a *realistic* attack scenario.

A. Cache Side Channel Attacks

a) Cache Architecture: The cache architecture consists of a hierarchy of memory components located between CPU cores and RAM with the purpose of reducing the average access time to the main memory. When the CPU needs to fetch data from memory, it queries the cache memory first to check if the data is in the cache. If it is, then it can be accessed with much smaller delay and in this case we say that a cache *hit* has occurred. On the other hand, if the data is not present in the cache, it needs to be fetched from a higher-level cache or maybe even from the main memory which results in greater delays. This is referred to as a cache *miss*. When a cache miss occurs, the CPU retrieves the data from the memory and stores it into the cache. This action is motivated by the *temporal locality* principle: recently accessed data is likely to be accessed again soon.

However, the CPU also takes advantage of *spatial locality*: when a data is accessed, the values stored in nearby locations to the accessed data are likely to be accessed again. Therefore, when a cache miss occurs, an entire block of data, including data in nearby locations, is loaded into cache. Therefore, the cache is organized into fixed size *cache lines*, e.g of l bytes each. A cache line represents the partitions of the data that can be retrieved or written at a time when accessing the cache.

When an entry of a table stored in memory is accessed for the first time, the memory line containing the retrieved data is loaded into the cache. If the algorithm accesses data from the same memory line again, the access time is faster since the data is located in the cache, not in the memory, i.e. a cache hit occurs. Therefore the *encryption time* depends directly on the accessed table positions, which in turn depend on the secret internal state. This fact can be exploited to gain information of the used secret key. In case there are no empty (invalid) cache lines available, one of the data bearing lines needs to be reallocated to the incoming line. Therefore, the cache lines that are not accessed recently are *evicted* from the cache.

There are three main families of cache-based side channel attacks: *time driven*, *trace driven*, and *access driven* cache attacks. The difference between these attack types is the capabilities of the attacker. Time driven attacks are the least restrictive ones where the only assumption is that the attacker can observe the aggregated timing profile of a full execution of the target cipher. Trace driven attacks are assumed to be able to monitor the cache profile when the targeted program is running. Finally, access driven attacks assume only to know which sets of the cache have been accessed during the execution of a program.

b) Cache Leakage for AES: The run-time of fast software implementations of ciphers like the AES often heavily depends on the speed at which table look ups are performed. A popular implementation style for the AES is the T-table implementation of AES [17]. It combines the `SubBytes`, `ShiftRows` and `MixColumns` operations into one single table look up per state byte along with xor operations. Compared to standard S-boxes, T-table based implementations use more memory, but the encryption time is significantly faster, especially on 32-bit CPUs. Because of this almost all current software implementations of AES for high performance CPUs are T-table implementations. Please note that the index of the loaded table entry is determined by a byte of the cipher state. Hence, information on which table values have been loaded into cache can reveal information about the secret state of AES. Such information can be retrieved by monitoring the cache directly, as done in *trace-driven* cache attacks. Similar information can also be learned by observing the timing behavior of an AES execution. Above we ascertained that a cache miss takes more time to access the data than a cache hit. Hence, the look up of values from a table will take variable time, depending on whether the accessed data has been loaded to a cache before or not.

c) Related Attacks on AES: Kocher showed in his 1996 study that data-dependent timing behavior of cryptographic implementations can be exploited to recover secret keys [18]. It took almost ten years for the cryptographic community to become aware of the fact that the microarchitecture of modern CPUs can introduce exploitable timing behavior into seemingly constant run-time implementations such as AES. Bernstein showed in [1] that timing-dependencies introduced by the data cache allow key recovery for the popular T-table implementations of AES [17]. Subsequent papers explored reasons [16] and improved attack techniques, such as collision attacks [2], [19] and attacks targeting the last round of AES [20]. In [21] new cache-specific attack styles, such as access-driven cache attacks are explored. Mowery et al. analyze the effects of new microarchitectural features commonly present in modern CPUs on Bernstein's attack [22]. Examples of access-driven cache attacks on AES are the ones proposed in [14], [3]. This attack flushes the desired data from the cache prior to the encryption process execution. Later on it reloads the previously flushed memory. If the data has been accessed by the encryption process, it takes less time to access it (since the encryption process fetched it to the cache).

Osvik et al. describe a trace driven *prime+probe* attack [21]. The prime and probe technique fills the cache before the encryption. After the encryption is finished, the attack re-accesses its data to discover which sets have been accessed by the target process. Moreover, the same paper describes a time driven attack called *evict+time*. The approach is simple: Certain cache lines are evicted between two identical encryption processes. If the correct cache lines were evicted, the second encryption takes more time than the first one.

In [23] several cache attacks such as Prime+Probe and Evict+Time are implemented, targeting the OpenSSL AES implementation in a non-virtualized environment. Even though this work discusses possible side-channel weaknesses of the sandboxing and the virtualization systems, it lacks an actual implementation of an attack on a virtualized system.

In a real virtualized scenario, we are only aware of the research in [15], where the authors attack AES on ARM processors. To the best of our knowledge, this work is the first to analyze the side-channel cache attack vulnerability of current AES in virtualized settings.

B. Bernstein's Attack

Bernstein's attack [1] is a four step side-channel cache attack on AES. Bernstein originally proposed his attack in a *non-virtualized server-client* setting, however, we adapt it here to a virtualized setting to serve the purposes of this paper. The attack mainly consists of four stages:

- **Profiling Stage** In the *profiling stage*, the attacker creates a duplicate *profile server* either on the attacker's own VM or on a third VM on the same physical machine. The *profile server* encrypts plaintexts using a *known key* and sends back the timing information of the encryption to the client. The received timing information is put into a timing table T of size 16×256 that holds the timing information for each byte of the key for all possible key values. Note that the table is initialized with all zero values. New values are added and then averaged for each entry. For the profiling stage, t is calculated for each i, j such that $0 \leq i < l$ and $0 \leq j < 16$, where l is the number of plaintexts sent to the server. This way, the following equation is calculated continuously until the desired sample number l is reached.

$$T[j][p_j, i] := T[j][p_j, i] + \text{time}(\mathsf{AES}(p_i, k)) \qquad (1)$$

- **Attack Stage** In the *attack stage* the attacker sends known plaintexts to the target server just like in the profiling stage. However, this time the *secret key* k is unknown to the attacker. During this stage, the attacker profiles the timings for known plaintexts with an unknown key and records the timing information for each plaintext.
- **Correlation Stage** In the *correlation stage*, timing profiles from the first two stages are correlated and a new table is created. This new table holds a correlation coefficient for every possibility of each key byte. Then the elements of this table are sorted starting from the possible key with the highest correlation coefficient to the one with the lowest correlation coefficient. Using a threshold, key candidates with a low posterior likelihood are eliminated and the remaining key candidates are displayed.
- **Brute Force Stage** In the *final stage*, all possible key combinations are tried using the key candidates created at the end of the third stage. Since the possible key space has been significantly reduced by the correlation stage, this stage takes much less time than a brute-force AES key search. The remaining key recovery step is not actually performed in our experiments described in Section V, since we are only interested in the amount of information learned from the side channel.

Note that the described attack is a pure timing attack, i.e. the only information needed by the attacker is the the sent plaintexts as well as the needed encryption time. The attacker does not require access to the victim's cache during the attack phase (and no access to the victim at all during the other stages). However, the timing variations that are exploited by the attack are caused by the cache architecture, making this attack a cache timing attack. These timings depend on deterministic system-based evictions. When an attacker requests an encryption for all possible plaintext bytes with a known key (profiling stage), she profiles the performance of the cache for a certain entry in the corresponding table look up. Then she profiles the cache performance for an unknown entry in the corresponding table look up, with a known plaintext. Since the entries for the table look ups in the first round are just an XOR operation between key and plaintext, she can compute the correlation as a function of the key byte and thereby determine the correct key.

The cache attack scenario does not significantly change when moved to a cloud environment. Users set up virtual machines with different OSs and the VMM handles their accesses to the hardware. In this specific scenario we show a two core physical machine with three cache levels, where each core has private L1 and L2 caches, but the L3 cache is shared.

III. AES IN COMMON CRYPTO LIBRARIES

We believe that cache based side-channel leakage is a big concern for cryptographic libraries. In this section we analyze the software implementations of AES for three widely used crypto libraries, namely `OpenSSL` [6], `PolarSSL` [24], and `Libgcrypt` [25]. All studied AES implementations use the T-tables optimization and are implemented in C. However, these libraries differ in their implementation of the last round. Also, some of the AES implementations already contain methods to reduce or nullify cache-based side channel leakage.

Due to the lack of the MixColumns operation, the last round has to be handled differently to the preceding rounds. We have encountered three different implementation styles: The classical approach is to have a special 256×32-bit table T_4; alternatives are a 256×8-bit S-box or a reuse of one of the T-tables, followed by masking and shift operations. While using T_4 is very efficient, it can introduce a particularly strong leakage since it is only being used in the last round. The S-box usage for the last round to the contrary, is more secure, since each memory line holds four times more data than the look up table. Less leakage is introduced when reusing T-tables, since they are used in all the rounds of the encryption.

From this point on, we will mention first and last round attack referring to Bernstein's attack applied to first and last round. Berstein's last round attack was introduced in [26], and requires known ciphertexts instead of known plaintexts.

`OpenSSL 0.9.7a` This version of `OpenSSL` implements the T-table implementation of AES, using T_4 for the last round. This library has been a perfect target for researchers since it is highly vulnerable against cache attacks. Although outdated, we decided to include it in our study as a reference for an inappropriately protected implementation. Some examples of researchers attacking this library are [1], [2], [26]. The advantage that this version of the library gives to attackers is that the last round is a good target, since T_4 is being used. Therefore profiling the last round with Bernstein's attack should lead to measurements with less noise and hence better results.

```
Rounds 1-9: T0,T1,T2,T3.
Last Round:   T4;
```

`OpenSSL 1.0.1f` This recent[1] version of the `OpenSSL` library features two different implementations of AES. One of them tries to prevent cache-based side channel leakage in the first and last rounds. The method is simple: a small s-box is used in the outermost rounds and prefetching is performed before the round. This means that the table is loaded to the cache before the mentioned rounds, resulting in constant-time accesses for all possible look ups.

```
Prefetch(T4).
First Round: T4.
Rounds 2-9: T0,T1,T2,T3.
Prefetch(T4).
Last Round: T4.
```

[1]The more recent `OpenSSL 1.0.1g` and `OpenSSL 1.0.2` have fixes for the Heartbleed vulnerability, but did not update the AES implementations.

The other AES core file is similar to the one of `OpenSSL 0.9.7a`. However, it differs only in the last round. Instead of using a dedicated T-table for the last round, the tables T_0 to T_3 are reused. Hence, the timing behavior for the last round is at least hindered. This implementation is the default choice if `OpenSSL` is compiled with the `no-asm` and `no-hw` flags. We also verified that this implementation is the one installed by default in Ubuntu id no AES hardware support is available by reverse engineering the automatically installed dynamic `OpenSSL` library in Ubuntu. The former leakage-shielded implementation one is still experimental.

`Rounds 1-10:` T_0, T_1, T_2, T_3.

`PolarSSL 1.3.3` This library uses an implementation based on the four T-tables for rounds 1-9 and a single S-box for the last round. Unlike T-tables, that hold 4 byte values, S-boxes hold 1 byte values and the entire S-box occupies very few cache lines. Since Bernstein's attack is profiling cache lines, `PolarSSL`'s implementation makes the library a suitable target for the first round attack, but not for the last round attack for practical purposes.

`Rounds 1-9:` T_0, T_1, T_2, T_3.
`Last Round: S-box.`

`Libgcrypt 1.6.0` The implementation of AES in `Libgcrypt` also uses the T-table approach for the first nine rounds. The last round uses only T_1. This is a similar scenario as with latest version `OpenSSL`, where a last round attack is not likely to recover any key bytes, since many of T_1 values are already in the cache. However the first round is still vulnerable to Bernstein's correlation attack.

`Rounds 1-9:` T_0, T_1, T_2, T_3.
`Last Round:` T_1.

Note that the latest versions of all discussed cryptographic libraries support AES-NI instructions. AES-NI is an instruction set extension for Intel CPUs that accelerate the execution of AES. According to Intel [27], the performance is from 3 to 10 times faster when compared to state-of-the-art software implementations. AES-NI performs a full round of AES on a 128-bit register. Hence, data-dependent table look ups are no longer necessary, consequently preventing all known microarchitectural side channel attacks, including cache attacks. Most of the virtual machine monitors allow a guest OS to use the AES-NI instructions to perform an AES encryption. However, there are still numerous scenarios where AES-NI is not available, accessible or the implementation might not be aware of its presence.

IV. Experiment Setup

Fine-grain cache-based side channel attacks work best if the target and the attacker are on the same physical machine, making a cloud environment a promising and realistic scenario for the attack. Two well-known and widely used cloud service providers are the Amazon EC2 and the Google CE services. These CSPs use the XEN and KVM hypervisors respectively to provide virtualization. A popular alternative VMM is VMware, which is frequently found in companies to consolidate a large number of servers into a few machines to reduce IT costs. First set of our experiments are performed on local installations of Xen and VMware virtualization environments, more precisely Xen 4.1 and VMware ESXi 5.5 build number 162338. Second set of our experiments are performed on Amazon EC2 and Google CE.

We assume the attacker to have a copy of the target server's implementation so that the profiling stage can be performed on the attacker's set of VMs. This assumption is practical, since there is typically a small set of popular implementations of certain services. In our experiments we used Ubuntu Linux, which comes with `OpenSSL` and `Libgcrypt` preinstalled as dynamic libraries. `PolarSSL` has to be installed manually, but it is also automatically installed as a dynamic library.

Attacker (the client in Bernstein's attack) and the profile server are under the adversary's full control. The target server is running in another virtual machine with an unknown secret key. We assume that the attacker has already solved the co-location problem using the methods listed at [12]. Next, the adversary performs the profiling stage. Since both the target and profiling servers are in the same physical machine, they profile the same cache architecture. During the subsequent attack stage, the attacker queries the target server and records plaintexts (or ciphertexts for the last round attacks) together with the measured timings of the AES encryptions. As the final step, the attacker correlates both results and recovers the corresponding key bytes. Timings were obtained using the Intel x86 RDTSC instruction on the target server, as done in [1], [22], [15].

Note that even though we are using a Linux based operating system and the Linux kernel utilizes ASLR by default, it also automatically aligns dynamic libraries to page boundaries. Since Bernstein's attack only depends on the page offset of the look up table positions, the tables are aligned from one execution to the other. In other words, the attack works in spite of the enabled ASLR.

For the commercial cloud experiments, we use Amazon EC2 compute optimized c3.large instances with 2 vCPUs featuring Intel Xeon E5-2680 v2 (Ivy Bridge) processor as well as Google Compute Engine n1-standard-1 instances with 1 vCPU featuring 2.6 GHz Intel Xeon processor. We launched 8 instances of aforementioned types in each environment. The reason to launch 8 instances each instead of 2 is to make sure that we get consistent results and statistical accuracy. Also, instead of using dedicated servers which goes against the idea of cloud, we used shared resource instances that host several VMs/customers on the same physical machine. This ensures realistic background noise and co-resident VM load for the experiments.

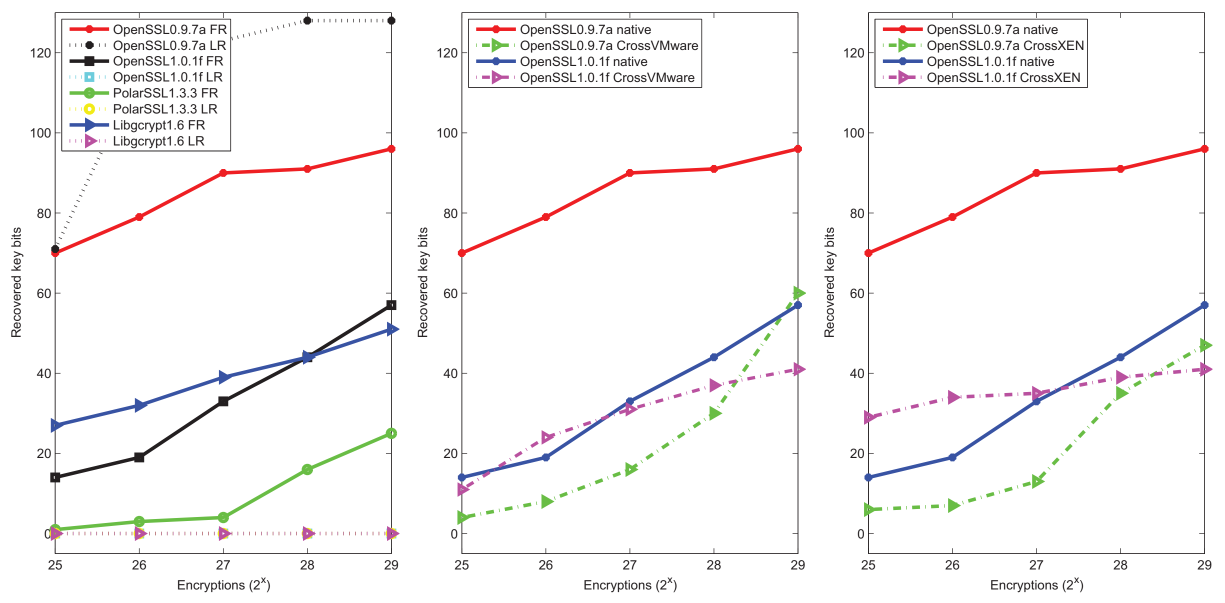
V. Experiment Results

This section provides the details of our measurement setup and the results obtained for various crypto libraries under a number of attack scenarios. Our main concern is the impact of Bernstein's attack when it is translated to a Cross-VM attack scenario.

A. Results for Native Execution

We first analyzed whether the latest versions of the targeted libraries are still vulnerable against Bernstein's attack. We attack each of the analyzed libraries with both Bernstein's first round and last round attack. In Figure 1(a) we summarize the results of our first set of experiments. The plot shows the number of recovered key bits as a function of the number of plaintexts sent to the server. The measurements have been performed on two workstations: a dual core Intel i5 3320 2.6 GHz running Ubuntu v12.04 and an eight core Intel i7 4702MQ 3.2 GHz running Ubuntu v12.04.

Observe that for `OpenSSL 0.9.7` major parts of the key can be recovered with the first round attack and the entire key with the last round attack (with sufficient plaintexts). For `OpenSSL 1.0.1f` we see a degradation in the first round attack, and a complete failure in the key recovery for the last round attack. This fact is due to the design of AES with respect to look up tables, as explained in Section III.



(a) (b) (c)

Figure 1. Comparison of Bernstein's attack: for different libraries on native machine (a); in native and cross VM scenarios for VMware (b) and for XEN (c).

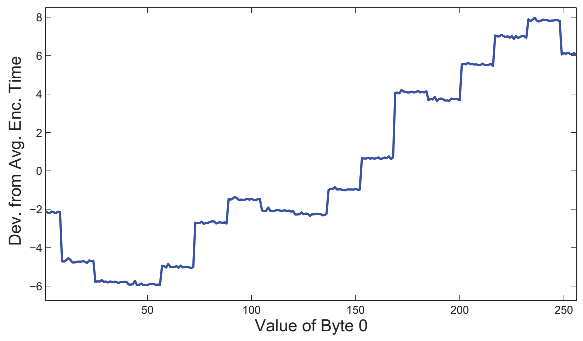


Figure 2. Execution time deviation of different key values of Byte 0.

We observe a similar behavior for PolarSSL and Libgcrypt. For both libraries the first round attack works well, i.e. it recovers 25% to 50% of the key. However, the last round attack is completely mitigated. We further notice that between the latest versions of the three libraries PolarSSL is less vulnerable than OpenSSL and Libgcrypt.

B. Cross-VM Attack Results

The next set of experiments are performed in a Cross-VM environment with the setup described in Section IV. Measurements were collected on two workstations, one of them running XEN 4.1 VMM and the other running a VMware ESXi 5.5 VMM. Both machines feature a dual core Intel i5 650 3.20GHz CPUs. All the VMs created for the profiling server, target server and attacker have the same settings and use the same OS, Ubuntu 12.04.

The results are presented in Figure 1(b) and Figure 1(c). The first graph shows a comparison between the attacks running natively and the attacks running in the Cross-VM setting on VMware. Figure 1(c) shows the same attacks for XEN. Clearly there is a degradation in the attack performance when the attack is moved from native execution to a virtual machine. This is due to the increased noise of the virtual machine environment. When compared to the case where we recovered almost the whole key in native machine for OpenSSL0.9.7a, in the case of XEN we could only recover up to 41 bits. For OpenSSL1.0.1f fewer than 40 bits of the key were recovered.

We observe a similar situation in the VMware experiments. In the Cross-VM scenario we were still able to recover around 60 bits for OpenSSL 0.9.7a. In the case of OpenSSL 1.0.1f we observe again a degradation, from around 60 bits recovered during native execution down to 41 bits recovered in Cross-VM scenario. When comparing the behavior of XEN and VMware for the Cross-VM attacks, we see similar results. For OpenSSL 1.0.1f between 30 and 40 bits were recovered in both XEN and VMware once the number of encryptions collected was sufficiently high.

C. Commercial Cloud Results

Next we evaluate the attacks efficiency on commercial cloud systems. For this, both client and server are executed as VMs in Amazon Elastic Compute Cloud (EC2) and Google Compute Engine (GCE). The results in Figure 3(c) show that—even with the real-world background noise and co-located VM CPU load—the attack partially recovers secret AES keys. In Amazon EC2 and GCE environments respectively, we were able to recover 36 and 20 bits of an 128 bit AES. As expected, the number of recovered bits decreases compared to the lab experiment setups. We expect this to be due to the co-resident VM and VMM load as well as modified versions of Xen and KVM that Amazon and Google uses respectively as the VMM, as we observed a highly noisy environment and drifting in encryption time.

The execution time of AES is affected significantly by the physical CPU load. This load is created by both the numerous co-resident VMs and the Hypervisor. Since there is no pattern to the co-resident VM noise, it is not possible to predict this noise and eliminate its effects before it causes spikes in encryption times. To improve the

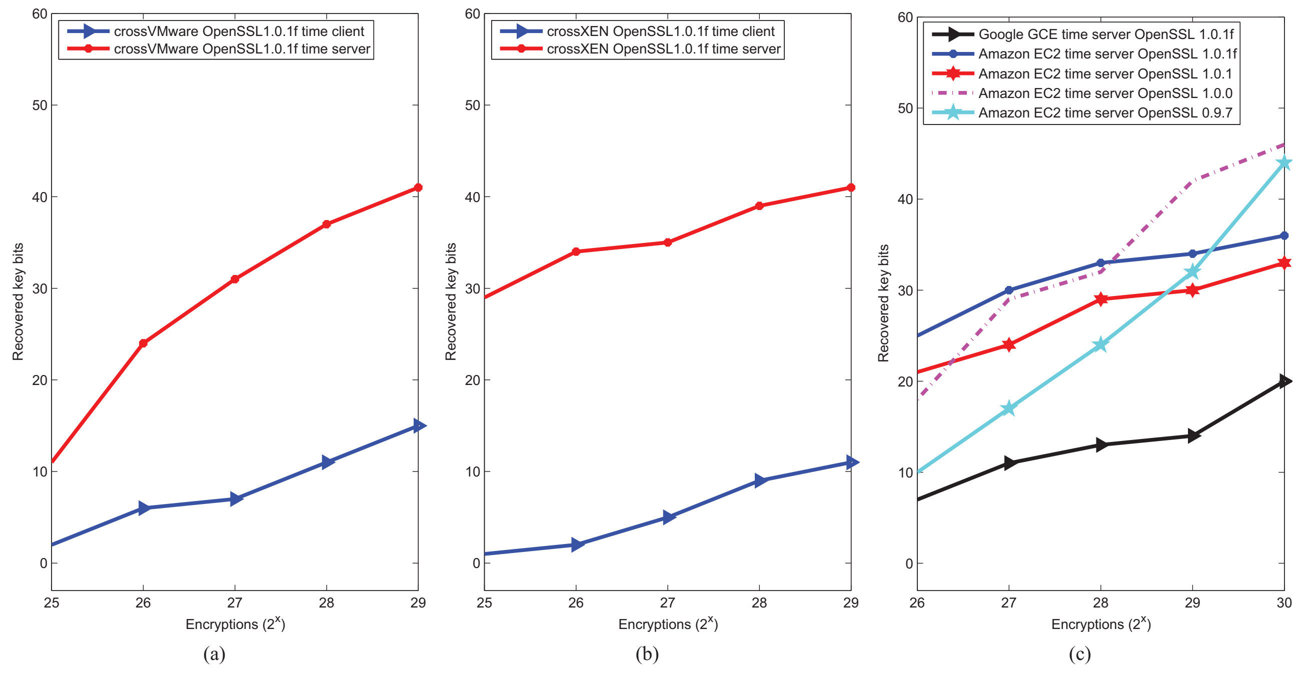


Figure 3. Comparison of Bernstein's attack when timing measurements are performed by the *client* in Cross-VM scenario in VMware (a) and in XEN (b). Figure (c) shows commercial cloud results with timing measurements performed by the *server*.

success rate, a dynamic outlier threshold that doesn't throw away useful information must be used. To do so, we first ran the attack for a short time to get the average encryption time which depends on that specific physical machine's work load. After getting the average encryption time for that specific VM instance, we added the offset necessary to account for the encryption time of the fastest executed plaintext and the slowest one. Using this method, we were able to utilize a useful and dynamic outlier threshold that cut off outliers without throwing away useful information.

A unique observation for the GCE instances are very similar encryption times for 64 consecutive plaintexts instead of 16 that we observed for both our lab setup using Intel Xeon E2670 and the the Amazon EC2 experiments. This results in a weaker side-channel leakage. In fact, only two bits per key byte can be reliably recovered in such a case, degrading the attack performance significantly. This degradation in leakage is unexpected, since the used GCE n1-standard instance are also specified to feature Intel Xeon CPUs, which we expect to have cache lines of the same size and hence hold the same number of T-table entries as seen in Figure 2.

Our experiment results show that the use of non-constant time implementations of cryptographic implementations such as AES poses a security risk in commercial cloud environments. The necessary precautions should be taken accordingly.

D. Moving Timing to the Client Side

Previous works mounting Bernstein's cache timing attacks on AES (e.g. the VM attack in [15]) perform the timing measurements on the server side, as we did in the previous measurements. While this approach is good for proving and quantifying an existing vulnerability, one might wonder how much the attack will degrade if performed over the virtual network between two co-located VMs. Thus, the next set of experiments were performed in a scenario were the *client* measures the execution time with an important additional source of noise: the network. We mount Bernstein's attack in Cross-VM for both XEN 4.1 and VMware ESXi 5.5 VMMs for `OpenSSL 1.0.1f` using client side timing measurements.

The results for VMware are presented in Figure 3(a). Not surprisingly, the performance degrades significantly due to the new source of noise coming from the network. While being able to recover 41 bits with server side timing measurements, we recover only 15 bits with the same number of encryptions when performing timing measurements on the client side. Results for XEN are presented in Figure 3(b) and are in the same ballpark, with 11 key bits recovered for 2^{29} observations. Note that this verifies that the timing leakage is still exploitable. Taking more sample measurements for the study will further improve the attack and increase the number of recovered key bits.

E. Experiment Outcomes

The conclusions we have reached after analyzing the experiment results are as follows:

- **AES execution path is clearly data-dependent**: As clearly seen in Figure 2, the execution time of the AES encryption is data dependent, revealing information about the secret key. This is due to the fact that different cache lines are accessed with varying time delays during the AES execution, depending on the key.
- **First round vulnerable**: The first round of the three libraries analyzed is still vulnerable to Bernstein's attack in particular and cache attacks in general.
- **Last round attack mitigated**: We did not obtain any key byte from the last round attacks in latest versions of the libraries.
- **Noise in VM scenario**: We observed that there is a performance drop when moving from native execution to the Cross-VM setting. We believe that this is due to the additional layers

between the guest OS and the hardware in a cloud environment, such as the VMM.

- **Network adds noise**: While our experiments show a significant degradation on the performance of the attack when the time measurements were done by the attacker, the results also clearly indicate that the leakage remains exploitable in the presence of network noise.
- **Bit recovery**: We recovered a minimum of 30 bits for the Cross-VM scenarios and 20 bits in cloud environment. Note that this analysis does not consider the posterior likelihood of surviving key candidates, which is far from uniform and could be used to speed up exhaustive key search.

VI. Preventing the Attack

It is possible to prevent the data leakage and the resulting Cross-VM cache attacks on cloud environments with simple precautions:

Cache Prefetching Prefetching the tables into the cache prior to the first and last rounds of AES evaluation can mitigate cache-based attacks. Prefetching should ensure constant access times during the first and the last rounds, since all T table entries are preloaded. Since it is significantly more difficult to implement a second round attack, this is a viable option. `OpenSSL` developers are aware of the cache attack vulnerabilities and provide assembly codes for various CPU platforms, many of which have prefetching enabled. As for `Libgcrypt` and `PolarSSL`, a new prefetching scheme for AES should be implemented to provide protection from AES cache attacks.

AES-NI Setting up the virtual machine on a computer with AES-NI support and using AES-NI instructions instead of the software implementation completely prevents cache-based attacks on AES. The AES-NI instructions do no longer require data-dependent accesses to cache, giving constant time execution. Moreover, AES-NI is nearly 10 times faster than the software implementation [27]. Fortunately, AES hardware support is in the process of becoming standard for cloud applications.[2]

VII. Conclusion

We showed that popular cryptographic libraries, namely `OpenSSL`, `Libgcrypt` and `PolarSSL` have little to no protection against a cache based timing side-channel attack such as Bernstein's attack if the Intel AES-NI instructions are not used. We showed that all of the libraries are still susceptible to first round cache attacks, while last round attacks have been mitigated in the current versions. When the attack is moved to a virtualized setting, i.e. co-located guest OS under Xen, KVM or VMware ESXI Hypervisors, the attack still succeeds. While Cross-VM attacks are noisier, the degradation is mild, and it is still possible to recover fine-grain information, i.e. an AES encryption key, from a co-located guest OS, even on commercial cloud servers, as the results for Amazon EC2 and Google CE show.

References

[1] D. J. Bernstein, "Cache-timing attacks on AES," 2004, uRL: `http://cr.yp.to/papers.html#cachetiming`.

[2] J. Bonneau and I. Mironov, "Cache-Collision Timing Attacks against AES," in *Cryptographic Hardware and Embedded Systems—CHES 2006*, ser. Springer LNCS, vol. 4249. Springer, 2006, pp. 201–215.

[3] D. Gullasch, E. Bangerter, and S. Krenn, "Cache Games – Bringing Access-Based Cache Attacks on AES to Practice," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 490–505. [Online]. Available: http://dx.doi.org/10.1109/SP.2011.22

[2]Amazon just retired the last machines on EC2 without AES-NI in June 2014.

[4] O. Aciiçmez, "Yet Another MicroArchitectural Attack:: Exploiting I-Cache," in *Proceedings of the 2007 ACM Workshop on Computer Security Architecture*, ser. CSAW '07. New York, NY, USA: ACM, 2007, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/1314466.1314469

[5] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[6] The OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS," April 2003, www.openssl.org.

[7] ——, "OpenSSL Security Advisory: Timing-based attacks on RSA keys," March 2003, https://www.openssl.org/news/secadv_20030317.txt.

[8] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *Computer Security–ESORICS 2011*. Springer, 2011, pp. 355–371.

[9] I. F. Blake, G. Seroussi, and N. Smart, *Elliptic curves in cryptography*. Cambridge university press, 1999, vol. 265.

[10] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Memory deduplication as a threat to the guest OS," in *Proceedings of the Fourth European Workshop on System Security*. ACM, 2011, p. 1.

[11] ——, "Software side channel attack on memory deduplication," *SOSP POSTER*, 2011.

[12] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653687

[13] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 305–316. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382230

[14] Y. Yarom and K. E. Falkner, "Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack." *IACR Cryptology ePrint Archive*, vol. 2013, p. 448, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/iacr/iacr2013.html#YaromF13

[15] M. Weiss, B. Heinz, and F. Stumpf, "A Cache Timing Attack on AES in Virtualization Environments," in *14th International Conference on Financial Cryptography and Data Security (Financial Crypto 2012)*, ser. Lecture Notes in Computer Science. Springer, 2012.

[16] M. Neve and J.-P. Seifert, "Advances on Access-Driven Cache Attacks on AES," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, E. Biham and A. Youssef, Eds. Springer Berlin Heidelberg, 2007, vol. 4356, pp. 147–162. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74462-7_11

[17] J. Daemen and V. Rijmen, *The Design of Rijndael*. Springer-Verlag, 2002.

[18] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology — CRYPTO '96*, ser. Lecture Notes in Computer Science, N. I. Koblitz, Ed., vol. 1109. Springer Verlag, 1996, pp. 104–113.

[19] O. Aciiçmez, W. Schindler, and Çetin K. Koç, "Cache Based Remote Timing Attack on the AES," in *Topics in Cryptology CT-RSA 2007, The Cryptographers Track at the RSA Conference 2007*. Springer-Verlag, 2007, pp. 271–286.

[20] J. Bonneau, "Robust Final-Round Cache-Trace Attacks against AES."

[21] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 1–20. [Online]. Available: http://dx.doi.org/10.1007/11605805_1

[22] K. Mowery, S. Keelveedhi, and H. Shacham, "Are AES x86 Cache Timing Attacks Still Feasible?" in *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '12. New York, NY, USA: ACM, 2012, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2381913.2381917

[23] E. Tromer, D. Osvik, and A. Shamir, "Efficient Cache Attacks on AES, and Countermeasures," *Journal of Cryptology*, vol. 23, no. 1, pp. 37–71, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00145-009-9049-y

[24] PolarSSL, "PolarSSL: Straightforward,secure communication," www.polarssl.org.

[25] Libgcrypt, "The libgcrypt reference manual," http://www.gnupg.org/documentation/manuals/gcrypt/.

[26] A. C. Atıcı, C. Yılmaz, and E. Savaş, "An approach for isolating the sources of information leakage exploited in cache-based side-channel

attacks," in *Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on*, June 2013, pp. 74–83.

[27] L. Xu, "Securing the Enterprise with Intel AES-NI," White Paper, September 2010, http://www.intel.com/content/www/us/en/enterprise-security/enterprise-security-aes-ni-white-paper.html.