

Fine grain Cross-VM Attacks on Xen and VMware are possible!

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

Abstract

This work exposes further vulnerabilities in virtualized cloud servers by mounting Cross-VM cache attacks in Xen and VMware VMs targeting AES running in the victim VM. Even though there exists a rich literature on cache attacks on AES, so far only a single work, demonstrating a working attack on an ARM platform running a L4Re virtualization layer has been published. Here we show that AES in a number popular cryptographic libraries including OpenSSL, PolarSSL and Libgcrypt are vulnerable to Bernstein’s correlation attack when run in Xen and VMware (bare metal version) VMs, the most popular VMs used by cloud service providers (CSP) such as Amazon and Rackspace. We also show that the vulnerability persists even if the VMs are placed on different cores in the same machine. The results of this study shows that there is a great security risk to AES and (data encrypted under AES) on popular cloud services.

1 Motivation

In as little as a decade cloud computing and cloud enabled applications have become mainstream running IT services of many businesses and have even personal computing through popular applications such as cloud based storage, e.g. Dropbox, and media sharing and delivery systems, e.g. Netflix. 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 a 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 [5] (and later in [8, 12]) targets the time variation in accessing the cache to extract AES encryption keys while the proposal by Acıçmez targets the key dependent time variations in the hardware branch prediction unit [1].

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 [10] demonstrated timing side-channel vulnerabilities in OpenSSL 0.9.7 [22], in the context of sandboxed execution in VMMs. 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. Nevertheless, as recently as in 2011, B.B. Brumley and Turever [9] demonstrated that the ladder computation operation in the ECDSA computation [6] 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. [20, 21] exploited an OS-optimization, namely KSM, to recover user data and subsequently identify a user from a co-hosted guest OS hosted in a Linux Kernel-based Virtual Machine (KVM).

In 2012, Ristenpart et al. [19] demonstrated that it is possible to solve the logistics problems and extract sensitive data across VMs. Specifically, using the Amazon

*This material is based upon work supported by the National Science Foundation under Grant No. #1318919.

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. [27] presented an access-driven side-channel attack implemented across Xen VMs that manages to extract fine-grain information from a 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. [19] who managed to extract keystroke patterns. Yarom et al. [26] describe a trace-driven flush and reload attack and note that it could be applied in virtualized environments. In [24] 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 [27]) 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 and VMWare 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 and VMware, even if the VMs are located on *different cores* on the same machine. We repeat the experiments, by running the attack natively, inside the same VM and Cross-VM—establishing that for most cases the attack success degrades only mildly when the target and victim processes are taken from native to virtualized to Cross-VM execution.

2 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 [5]. It exploits data-dependent timing-behavior that is introduced by the un-

derlying cache architecture of the host system. Bernstein’s attack has often been disregarded as impractical when performed over a network (e.g. in [16]). However, due to the quickly increasing popularity of 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 [19]. 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 will review the cache architecture as well as attacks that exploit its 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.

2.1 Cache Side Channel Attacks

Cache Architecture The cache architecture consists of a hierarchy of memory components located between CPU cores and RAM. Its purpose is to reduce 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* 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. We refer to this 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 is going to take advantage of *spatial locality* as well: when a data is accessed, the values stored in close locations to the accessed data are likely to be accessed again. Therefore, when a cache miss occurs it makes sense to load not only the missed data, but an entire block which includes data in nearby locations also. 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 lower since the data is located in the cache, not in the memory, i.e. a cache hit occurs. Therefore the encryption *time* will depend 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, cache lines not recently accessed are *evicted* from the cache.

The position that a certain memory line occupies in the cache depends on the cache structure. There are three kinds of cache structures: fully-associative, direct mapping and set-associative. The *fully-associative* structure states that a memory line can occupy any of the cache lines of the cache. The *direct mapping* policy states that each memory line has a fixed position in the cache. Finally, the *set-associative* policy divides the cache into s sets, each containing k lines. With this structure a memory line has to be loaded in one specific set, but it can occupy any of the k lines in that set.

There are three main families of cache-based side channel attacks: *time driven*, *trace driven*, and *access driven* cache attacks. The difference between them are the capabilities of the attacker. Time driven attacks are the least restrictive ones. The only assumption is that the attacker can observe the aggregated timing profile of a full execution of a target cipher. Trace driven attacks are assumed to be able to catch 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 a execution of a program.

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 [11]. 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 it takes more time to access the data than a cache hit. Hence, the look up of values from a table will take different time, depending on whether the accessed data has been loaded to a cache before or not.

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 [13]. It took almost ten years for the cryptographic community to become aware of the fact that that the microarchitecture of modern CPUs can introduce exploitable timing behavior into seemingly constant runtime implementations such as AES. Bernstein showed in [5] that timing-dependencies introduced by the data cache allow key recovery for the popular T-table implementations of AES [11]. Subsequent papers explored reasons [16], improved attack techniques such as collision attacks [8, 2] and attacks targeting last round [7]. In [17] new cache-specific attack styles, such as access-driven cache attacks are explored. Mowery et al. analyze in [15] the effect of new microarchitectural features commonly present in modern CPUs on Bernstein’s attack. An example of access-driven cache attack in AES is the one proposed in [26, 12] where Gullasch et al flush the desired data from the cache prior to the encryption and then at some point of the encryption, they try to load the flushed data again. If the data has been accessed in the encryption it takes less time to access it (since it is in the cache).

Dag Osvik et al describe a trace driven attack called *prime+probe* [17]. The prime and probe technique fills the cache before the encryption and after it checks which sets have been accessed by reloading the data. Moreover in the same paper we find a description of a time driven attack called *evict+time*. The approach is simple: The cache is evicted before the encryption and then the time of the encryption is measured.

Only little work has been performed on the influence of virtualization on cache-based side-channel leakage. In a real virtualized scenario, we only know about Michael Weiss research in [24], where they tried to 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 implementations on modern x86-type CPUs in virtualized environment.

2.2 Bernstein’s attack

Bernstein’s attack [5] is a four step side-channel cache attack on the AES. Bernstein originally proposed his attack in the *non-virtualized server-client* setting, however, we adapt it here to the 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. After the server is run with

```

correlation results;

48 0 4e 4c 4f 4d 43 41 40 42 67 66 64 5e 5f 60 65 5d 5c 63 62 61 6a 69 6b 68 44 46 45 47 48 4a 49 4b 50 53 79 52 7a 7b 78 77 1f 74 51
76 1c 1e 75 1d
8 1 0a 0b 08 09 05 04 06 07
32 2 04 05 06 07 0a 0b 09 08 11 13 23 12 10 20 22 21 24 27 26 25 29 2a 2b 28 0f 0e 0c 0d 1d 1e 1f 1c
8 3 a5 a6 a7 a4 ab aa a9 a8
20 4 b1 b0 b3 b2 be 9a 98 bf 99 bc bd 9b 95 97 94 96 83 81 80 82
8 5 39 3a 3b 38 35 36 34 37
32 6 a7 a5 a6 a4 aa a8 a9 ab 8f 8e 8d 8c bc be bf bd 8b 89 88 8a 86 85 84 87 a3 a0 a2 a1 b0 b1 b2 b3
8 7 75 74 77 76 7a 78 79 7b
24 8 d4 d6 d5 d7 d9 da d8 db f3 f0 f2 f1 fe fc ff fd c1 c3 c2 c0 ce cc cf cd
8 9 2e 2c 2f 2d 22 21 20 23
24 10 7a 79 78 7b 74 76 77 75 52 53 50 51 57 56 54 55 62 61 60 63 5b 58 59 5a
8 11 ba b9 b7 b4 b6 bb b8 b5
24 12 13 12 11 10 1c 1e 1d 1f 37 36 35 34 3a 3b 38 39 07 06 04 05 0a 08 09 0b
8 13 de df dd dc d0 d3 d2 d1
24 14 9a 99 9b 98 95 97 96 94 bc be bf bd b4 b7 b6 b5 b9 ba bb b8 8e 8c 8f 8d
4 15 20 21 22 23

```

Figure 1: Output of Bernstein’s attack after the correlation phase.

a *known key*, the *profile server* encrypts these plaintexts and sends back the timing information of the encryption to the attacker. 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. So, for the learning 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.

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

- **Attack Stage** In the *attack stage* the attacker C sends known plaintexts to the target server S just like in the study stage but with a slight difference. Unlike the study stage, the *secret key* k in this stage is unknown to the attacker. So 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. Then using a threshold, low possibility key candidates are eliminated and the remaining key candidates are displayed. We can see an example of such output correlation in Figure 1.
- **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.

For different executions of AES, the attacker gets different timings. These timings depend on deterministic system-based evictions. When an attacker requests an encryption for all possible key bytes with a known key (profiling stage), it profiles the performance of the cache for a certain entry in the corresponding table look up. Then he profiles the cache performance for an unknown entry in the corresponding table look up, although he knows the plaintext. Since the entries for the table look ups in the first round are just an XOR operation between key and plaintext, he can compute the correlation and figure out what key has been used.

In a cloud environment the cache attack scenario does not change by much. In Figure 2 we see an example of a typical scheme in a cloud. Users set up virtual machines with different OS and the VMM handles their accesses to the hardware. In this specific scheme we show a four core physical with three cache levels, where each core has private L1 and L2 caches, but the third level cache is shared. Usually the last level cache is attacked since it is where the four cores share data, making it suitable for a cross-VM leakage [26].

2.3 Challenges & Opportunities in VM Memory Management Technologies

Here we briefly summarize key VM memory management technologies which were developed to improve memory utilization, and memory access performance in general, and to enhance security. We also comment on how they affect the performance of side-channel attacks in a virtualized server/cloud setting. As it turns out while some VM memory management technologies provide a

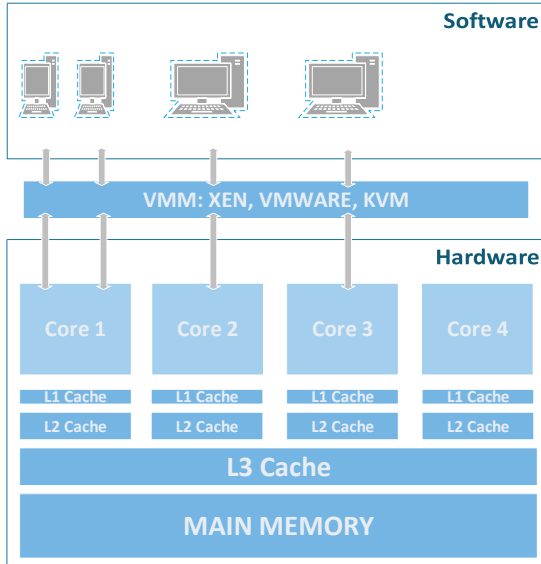


Figure 2: Hypervisor allocating different virtual machines into the same physical machine

challenge to mounting a cache timing attack, others actually do enable attacks.

Address Space Layout Randomization (ASLR): ASLR is a security technique implemented to avoid buffer overflow attacks. In a nutshell, the ASLR technique randomizes the memory positions of various memory sections of the program (heap, stack, static and dynamic libraries) each time the program is executed. The randomization prevents an adversary from exploiting a weak function that can be exploited in the memory. Most of the modern operating systems support ASLR and for the majority of them, e.g. most Linux distributions, Microsoft Windows, is enabled by default. However there are still some operating systems such as FreeBSD which do not support ASLR.

Second Level Address Translation (SLAT): SLAT schemes such as Intel’s Extended Page Tables (EPT) and AMD’s Nested Page Tables (NPT) as shown in Figure 3 are used to manage the virtualized memory directly from the processor. Using a larger Translation Lookaside Buffer (TLB) with additional logic circuitry inside the processor, these schemes provide faster virtual machine memory management by eliminating the intermediary step between the virtual memory address (VA) and the physical memory address (PA). Instead of managing the virtual machine memory through the host operating system by performing the VA to PA and then PA to Memory Address (MA) mapping at the software level, EPT and NPT technologies use a larger TLB that supports both native and virtual memory entries to be mapped to the

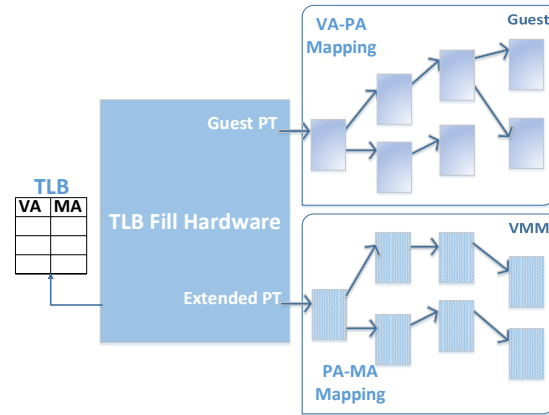


Figure 3: Nested/Extended Page Tables

memory addresses directly from the guest and the host operating systems. As shown in Figure 3 the TLB table has the option that indicates if the received data is from a virtual machine or the native machine. Also, if the data is generated by a virtual machine, then it is tagged with that specific VM’s Address Space Identifier (ASID). Using this tag, the TLB can keep track of entries from different virtual machines in the physical machine. This method provides a significant performance improvement in VM memory management but also introduces a security risk by giving direct memory access to the guest VMs.

Kernel SamePage Merging (KSM): KSM is the Linux memory deduplication scheme that was originally developed to allow the execution of multiple virtual machines on the same physical machine but later was adopted to the native machines as well and became a standard feature with the Linux kernel v2.6.32. What KSM does is simply to search the memory contents for duplicate entries and merge the duplicate pairs into single page entries. But instead of going over the whole memory continuously which would be CPU and memory consuming, KSM looks for duplicates using a more sophisticated scheme. When an application is called, its assigned a memory space. If the memory space is marked as mergeable then the KSM adds it to its search space and starts to look for duplicates using two red-black trees method. Using the knowledge that some of the pages are shared between processes and clients, one can steal sensitive information. This information can be part of a process running on the co-resident guest OS such as a visited website or even a possible secret key. Suzaki et al [20] have reported that by exploiting the KSM they were able to detect web browsers, visited websites and even a downloaded file used by the co-hosted guest OS in a Linux Kernel-based Virtual Machine (KVM) system. In another study, using a secret memory marker, Suzaki et al

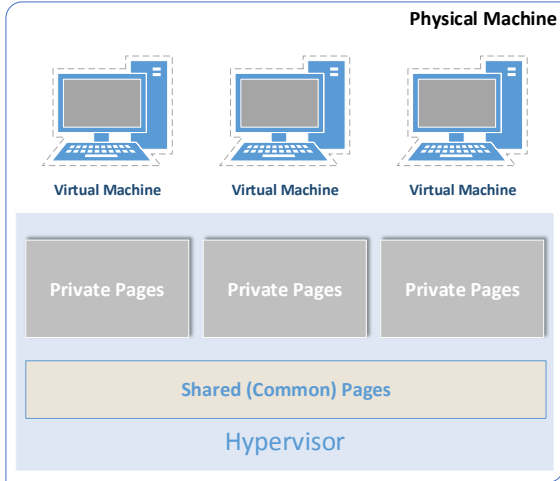


Figure 4: Memory Sharing Across VMs

[21] were able to identify a VM user on a processor in a multi-user cloud environment.

Transparent Page Sharing (TPS): Another important security risk for virtualization is TPS which is implemented in both the VMware and the XEN VMMs [23] [4]. A high level depiction of the TPS is shown in Figure 4. TPS continuously scans the memory for duplicate 4KB sized data regardless of when, where or by whom the data was created. Duplicate entries are found by first matching their hashes and then if there is a hash match, checking their actual data bit by bit. If a duplicate data in memory is found, the VMM automatically discards one of the duplicates and allows multiple guests to use the same data until the data is modified by one of the clients. With this sharing, TPS utilizes the system memory in a more efficient manner but at a cost of compromising memory space isolation. Similar to the custom SLAT case, one of the clients using the same physical machine can set traps and collect information on a neighboring VM through cache timing attacks. Even though the shared data cannot be modified or corrupted by an adversary, it can be used to reveal secrets about the target VM client. An adversary can fill the cache with the candidate data used by the target VM to find out which data is shared with therefore also used by the other VM client.

3 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[22],

PolarSSL [18], and Libgcrypt [14]. All studied AES implementations use the T-tables optimization. However, the implementations differ in the implementation of the last round. Also, some of the implementations do 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 is holding four times more data than the look up table. Less leakage is introduced when reusing T-tables, since they are used in the 10 rounds of the encryption.

From now on, we will mention first and last round attack referring to Bernstein’s attack applied to first and last round (described in [3]), respectively.

OpenSSL 0.9.7 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 [5, 8, 3]. 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. However, we show in the following section that results for the first round attack on this library are very good as well when comparing to other libraries.

Rounds 1–9: T_0, T_1, T_2, T_3 .
Last Round: T_4 ;

OpenSSL 1.0.1 The latest version of the OpenSSL library features two different implementations of AES. One of them tries to completely prevent cache-based side channel leakage in the first and last rounds. The method is simple: a simple and small s-box is used in the outermost rounds. 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(T_4).
First Round: T_4 .

Rounds 2-9: T_0, T_1, T_2, T_3 .
 Prefetch(T_4).
 Last Round: T_4 .

The other AES core file has more in common with the older version. However, it differs in the last round, since instead of using a single and different T-table, the tables T_0 to T_3 are reused. Hence, there is no special leakage for the last round. Instead, the additional round with the same tables adds more noise to measurements, so we expect the results to be worse than for the older version. This implies more possible candidates for the key, but no mitigation of cache attacks.

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

One might wonder which of the two implementations is used by default? According to OpenSSL source comments, the leakage-shielded implementation one is still experimental. Second, the makefile in the library compiles the vulnerable one by default. Finally, we reverse engineered the static library that is installed by default in Ubuntu. It is, in fact, the vulnerable one.

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 to be profiled. Since the 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.

Libcrypt 1.6 The implementation of AES in Libcrypt 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. A first round attack makes more sense for this implementation, but more noise in our measurements comparing to OpenSSL 0.9.7.

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 [25],

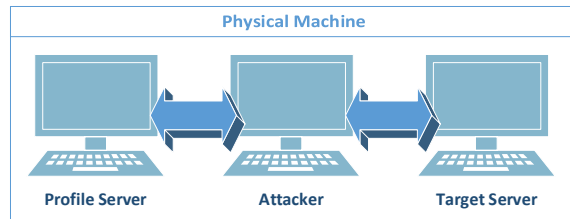


Figure 5: Our attack setup

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.

4 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 Rackspace services. Both of these CSPs use the XEN VMM to provide virtualization. Also companies are typically deploying VMware to consolidate a large number of servers into a few machines to reduce IT costs. Our study focuses on these two virtualization environments. We assume the attacker to have a copy of the target server’s implementation so that the profiling stage can be performed on in 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 Libcrypt preinstalled as dynamic libraries. PolarSSL has to be installed manually, but it is also automatically installed as a dynamic library. Figure 5 gives a visual description of our attack scenario. 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 [19]. 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 [5, 15, 24]. Although practical server implementations will not provide timestamps along with the response messages, the setup translates to a practical one for the following reasons: Unlike in classical over-the-network attacks, the inter-domain delay between two VMs in the same physical machine is insignificant and almost constant [24]. Since Bernstein’s attack algorithm uses correlation, the constant offset is irrelevant. The remaining jitter is expected to only introduce very little additional noise. However, using the RDTSC instruction minimizes the sources of noise in our setup and thus allows us to quantify the amount of leaked information faster, reducing the time spent for each experiment.

Note that even though we are using a Linux based operating system and the Linux kernel utilizes the 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.

5 Experiment Results

This section details on our measurement setup and the results obtained for different crypto libraries in different attack scenarios. Our main concern is the impact of Bernstein’s attack when it is translated to a cross-VM attack scenario.

5.1 Results for Native Execution

We first analyzed whether the latest versions of the studied 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. the results are given in the number of bits of the key recovered by the attack as a function of the number of plaintexts sent to the server. The measurements were done in two workstations: a four core I5 2.6GHz of speed under Ubuntu v13 and an eight core I7 3.2GHz of speed under Ubuntu v12.04.

In Figure 6 we can observe the resilience of each library against Bernstein’s attack. Note that for OpenSSL 0.9.7 major parts of the key were recovered with the first round attack and the entire key with last round attack (with sufficient plaintexts). For the latest version of

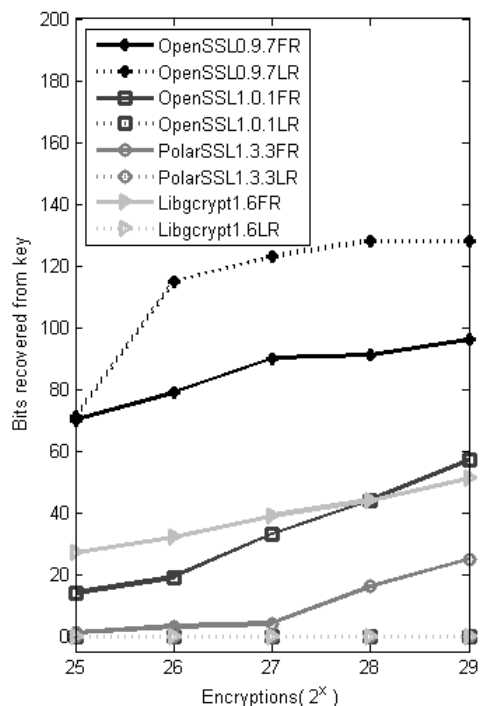


Figure 6: Comparison of Bernstein’s attack for different libraries on native machine.

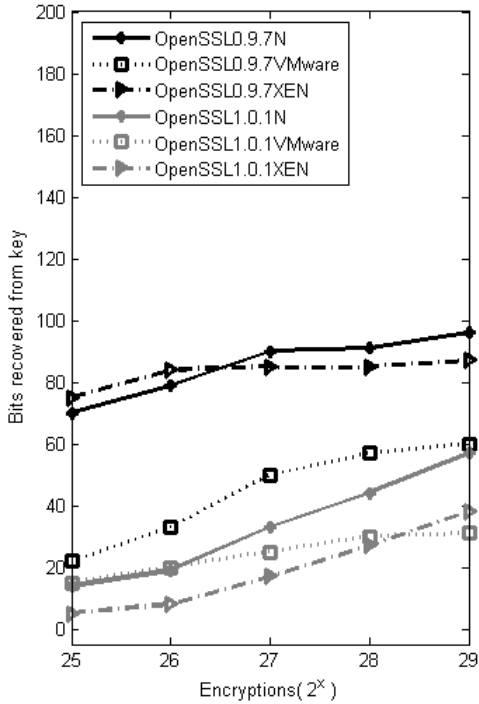


Figure 7: Comparison of Bernstein’s attack in native and single VM scenarios.

OpenSSL, 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 3.

We observe a similar behavior for PolarSSL and Libgcrypt. For both libraries the first round attack works well (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.

5.2 Results in Single VMs

Next step was to analyze the degradation of Bernstein’s attack when it is moved to a virtual machine. Measurements were performed on two workstations, one of them running XEN hypervisor, and the other one running a VMware hypervisor. Both machines feature two Intel Core i5 3.20GHz CPUs. The operating system for our attacks was Ubuntu 12.04. At this point we only performed profile and attack stages in the same virtual machine. The results are presented in Figure 7.

We can see that there is a degradation of the attack when moving it to a virtual machine. This is due to the

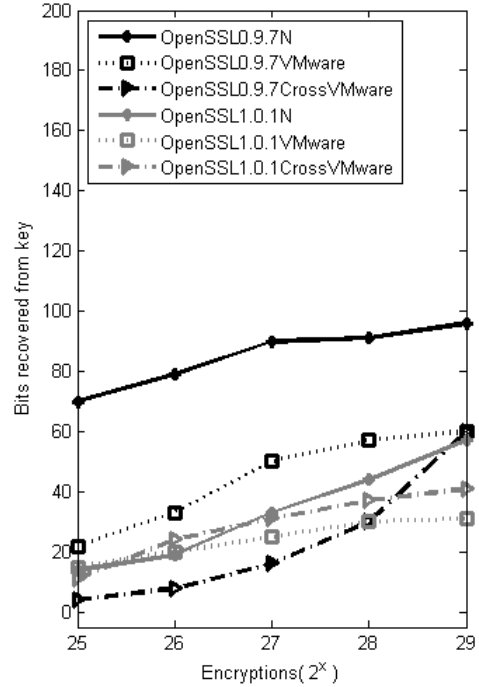


Figure 8: Comparison of Bernstein attack in native, single VM and cross VM scenarios for VMware

increased noise of the virtual machine environment. In contrast to the case where we recovered almost the whole key in native machine for OpenSSL 0.9.7, we could only recover up to 81 bits in the case of XEN and up to 61 bits in the case of VMware. We believe that the difference between both of the results is due to external noise. In fact, the single VM attack on VMware was the only in the study that was performed in a noisy environment with typical user traffic in the background.

Indeed when we analyze the case for OpenSSL 1.0.1 we see a similar behavior for both hypervisors. For this library we also decreased the amount of bits recovered from 58 to 38 in the case of XEN and to 31 in the case of VMware. Although we did not even recover half of the bits of the key, we believe that recovering 25% of the key is a reason to be concerned.

5.3 Cross-VM Attack Results

The last set of experiments is performed in a cross-VM environment with the setup that we described in Section 4. The setup of our server is identical to the one in the single VM scenario (XEN and VMWARE hypervisors), but with more VMs. All the VMs created for the profiling server, target server and attacker have the same settings and have the same OS, Ubuntu 12.04.

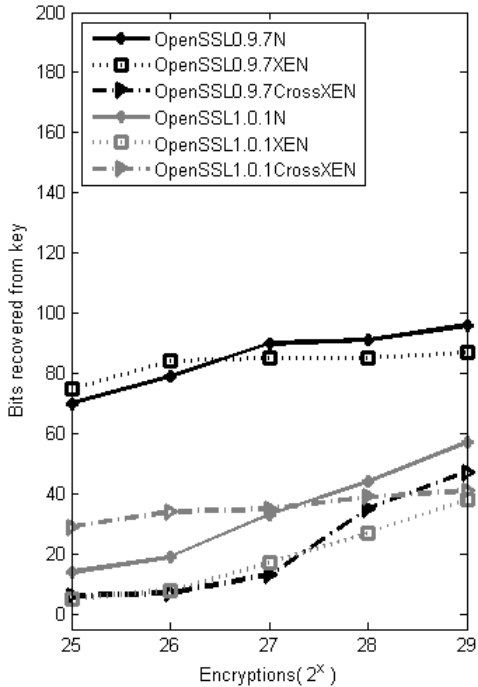


Figure 9: Comparison of Bernstein attack in native, single VM and cross VM scenarios for XEN

The results are presented in Figure 8 and Figure 9. The first graph shows a comparison between the attack running in native machine, single VM running in VMware and cross-VM running in VMware. The second one shows the same comparison for the attack in XEN.

In the first plot we can see that again we have more sources of noise in the virtual environment so the difference to the native machine attack is easily observed. However we see similar results in the case of VMware for single VM and cross-VM attack. This means that we are not losing any performance by moving the attack to a cross-VM scenario (indeed we recovered some additional bits for cross-VM), although we have an increased latency due to not communicating via local sockets.

In the second plot we notice that there is a huge drop from single VM to cross VM for OpenSSL 0.9.7. However, the results were better than expected when moving from native to virtual machine (where we only lost 10 bits). We believe there was very little noise when running OpenSSL 0.9.7 in single VM. Further work should be done here to investigate noise sources. In the cross-VM scenario we were still able to recover around 50 bits for OpenSSL 0.9.7. In the case of OpenSSL 1.0.1 it can be seen that for 2 million encryptions the results for sin-

gle VM and for cross-VM are similar. We conclude that the results depend more on different noise for different runs than on the migration from single to cross-VM.

When comparing the behavior of XEN and VMware for the cross-VM attacks, we see similar results. For the latest version of OpenSSL in both of them between 30 and 40 bits were recovered once the number of encryptions is high enough.

We want to highlight that apart from discarding some key byte candidates, the attack sorts the rest of them by their posterior likelihood. The attacker can take advantage of this additional knowledge by starting the exhaustive search of the remaining key space by checking the most likely values for the key bytes first. Figure 1 gives an example output obtained from Bernstein’s attack. The first column refers to the remaining key possibilities after the attack; the second column refers to key byte number; in the third column, key byte candidates are sorted according to their posterior likelihood. In our experiments we have observed that the correct key is either in the first or the first two quartiles with a very high likelihood. Table 1 shows how often the correct key byte value was within the first, first two, and first three quartiles, depending on the number of remaining candidates. The table shows that if the attack was only able to narrow the space down to 16 or more possibilities, the probability of finding the correct key in the 50% most likely values is of more than 95%. This means that even if a remaining key space seems to be too large to be searched in its entirety (this usually means remaining lists of 16 or more for each key byte, or a remaining key space of more than 2^{64} possibilities), an attacker might still be able to narrow down the search space to manageable sizes without significantly sacrificing the success probability of the attack.

5.4 Experiment Outcomes

The conclusions we made after analyzing the experiments are the following:

- **Last round mitigated:** We did not obtain any key byte from the last round attacks in latest versions of the libraries.
- **First round vulnerable:** The first round of the three libraries analyzed is still vulnerable to cache attacks.
- **Noise in VM scenario:** We observed that there is a drop when moving from native machine to virtual machine. We believe that it is due to more layers that are introduced (i.e. VMM) in a cloud environment between the guest OS and the hardware.

Table 1: Probability of the correct key byte having a high rank within the sorted remaining candidates.

Remaining key byte space	Correct key in 1st quartile	Correct key in 2nd quartile	Correct key in 3rd quartile
1-8	43%	65%	86%
8-16	36%	61%	75%
16-32	57%	95%	95%
32-64	66%	100%	100%
64-128	71%	96%	100%
128-256	65%	100%	100%

- **No degradation when moving to cross-VM:** We did not observe a degradation in the results due to moving from a single VM to a cross-VM scenario.
- **Noise dependent results:** We obtained cleaner results when we were not running other processes together with the attack (i.e web browsing).
- **Bit recovery:** We recovered a minimum of 30 bits for the cross-VM scenarios, without any optimization of the exhaustive remaining key search.

6 Countermeasures

It is possible to prevent the data leakage and the resulting cross-vm cache attacks on cloud environment with simple precautions. The countermeasures that we propose are easy solutions to avoid a problem that can be very painful when working with sensitive data.

- **AES-NI** Setting up the virtual machine on a computer with AES-NI support and use AES-NI instructions instead of the software implementation. Using AES-NI mitigates completely the cache side-channel attacks since it does not use the memory. Moreover is nearly 10 times faster than the software implementation [25]. Even though this might seem like an obvious countermeasure, a quick search shows that there are still some VMs offered by popular CSPs that do not have AES-NI hardware support.
- **Cache Prefetching** Prefetch the tables in to the cache prior to first and last round. When the data is prefetched there is no possibility of attacking either the first or last rounds. Although this leads to a less noisy environment for a second round attack, the fact that it is much more difficult to implement the second round attack still makes this a viable option. We already know that the OpenSSL developers are aware of the cache attack vulnerability and we encourage them to work with the experimental file that they already implemented and included in the OpenSSL 1.0.1' distribution. As for Libgcrypt

and PolarSSL a new prefetching scheme for AES should be implemented to provide protection from AES cache attack.

- **Using AES-256** Use 256-bit key AES encryption instead of 128-bit and 196-bit versions. This will lead in more rounds, 12 for the 192-bit and 14 for the 256-bit versions, and will introduce more noise to the side channel measurements. We believe that the study samples required to analyze a 256-bit AES key would deem the cache attack impractical and the amount of the key bytes recovered in such case would be drastically low.
- **Preventing Cartography** As noted earlier in [19] another countermeasure would be making the server cartography harder. If the attacker cannot locate the target, it is possible to prevent the attack before it even begins since the attacker has to be on the same physical machine to perform the attack. Frequent core-migration might be used as a primary mechanism to prevent cartography.
- **Avoiding Co-location** This countermeasure was cited in [19]. Although this option is not performance efficient and goes against the idea of cloud based systems, using separate physical machines for each customer/client would make the side channel attacks impossible. Even if an adversary discovers the target's IP address and connects to the target using this IP address, she still can not perform the cache attack since she won't have a study file (revise this) to correlate with the attack timings.

7 Conclusion

We showed that popular cryptographic libraries, namely OpenSSL, Libgcrypt and PolarSSL have little to no protection against a cache type 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 seem to have been mitigated in the current versions. When the

attack is moved to a virtualized setting, i.e. co-located guest OS in Xen and VMware, 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.

References

- [1] ACIİÇMEZ, O. Yet another microarchitectural attack:: Exploiting i-cache. In *Proceedings of the 2007 ACM Workshop on Computer Security Architecture* (New York, NY, USA, 2007), CSAW '07, ACM, pp. 11–18.
- [2] ACIİÇMEZ, O., SCHINDLER, W., 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* (2007), Springer-Verlag, pp. 271–286.
- [3] ATICI, A. C., YILMAZ, C., AND SAVAŞ, E. 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.
- [4] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review* 37, 5 (2003), 164–177.
- [5] BERNSTEIN, D. J. Cache-timing attacks on AES, 2004. URL: <http://cr.yp.to/papers.html#cachetiming>.
- [6] BLAKE, I. F., SEROUSSI, G., AND SMART, N. *Elliptic curves in cryptography*, vol. 265. Cambridge university press, 1999.
- [7] BONNEAU, J. Robust final-round cache-trace attacks against aes.
- [8] BONNEAU, J., AND MIRONOV, I. Cache-collision timing attacks against aes. In *Cryptographic Hardware and Embedded Systems—CHES 2006* (2006), vol. 4249 of *Springer LNCS*, Springer, pp. 201–215.
- [9] BRUMLEY, B. B., AND TUVERI, N. Remote timing attacks are still practical. In *Computer Security—ESORICS 2011*. Springer, 2011, pp. 355–371.
- [10] BRUMLEY, D., AND BONEH, D. Remote timing attacks are practical. *Computer Networks* 48, 5 (2005), 701–716.
- [11] DAEMEN, J., AND RIJMEN, V. *The Design of Rijndael*. Springer-Verlag, 2002.
- [12] GULLASCH, D., BANGERTER, E., AND KRENN, S. Cache games – bringing access-based cache attacks on aes to practice. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2011), SP '11, IEEE Computer Society, pp. 490–505.
- [13] KOCHER, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology — CRYPTO '96* (1996), N. I. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 104–113.
- [14] LIBGCRYPT. The libgrypt reference manual. <http://www.gnupg.org/documentation/manuals/gcrypt/>.
- [15] MOWERY, K., KEELVEEDHI, S., AND SHACHAM, H. Are aes x86 cache timing attacks still feasible? In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop* (New York, NY, USA, 2012), CCSW '12, ACM, pp. 19–24.
- [16] NEVE, M., AND SEIFERT, J.-P. Advances on access-driven cache attacks on aes. In *Selected Areas in Cryptography*, E. Biham and A. Youssef, Eds., vol. 4356 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pp. 147–162.
- [17] OSVIK, D. A., SHAMIR, A., AND TROMER, E. Cache attacks and countermeasures: The case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology* (Berlin, Heidelberg, 2006), CT-RSA'06, Springer-Verlag, pp. 1–20.
- [18] POLARSSL. PolarSSL: Straightforward,secure communication. www.polarssl.org.
- [19] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. 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* (New York, NY, USA, 2009), CCS '09, ACM, pp. 199–212.
- [20] SUZAKI, K., IJIMA, K., YAGI, T., AND ARTHO, C. Memory deduplication as a threat to the guest os. In *Proceedings of the Fourth European Workshop on System Security* (2011), ACM, p. 1.
- [21] SUZAKI, K., IJIMA, K., YAGI, T., AND ARTHO, C. Software side channel attack on memory deduplication. *SOSP POSTER* (2011).
- [22] THE OPENSLL PROJECT. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org, April 2003.
- [23] WALDSPURGER, C. A. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 181–194.
- [24] WEISS, M., HEINZ, B., AND STUMPF, F. A cache timing attack on aes in virtualization environments. In *14th International Conference on Financial Cryptography and Data Security (Financial Crypto 2012)* (2012), Lecture Notes in Computer Science, Springer.
- [25] XU, L. 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>.
- [26] YAROM, Y., AND FALKNER, K. E. Flush+reload: a high resolution, low noise, l3 cache side-channel attack. *IACR Cryptology ePrint Archive 2013* (2013), 448.
- [27] ZHANG, Y., JUELS, A., REITER, M. K., AND RISTENPART, T. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 305–316.