Cache Side Channel Attacks on Intel SGX

Princeton University Technical Report
CE-L2017-001

January 2017

Zecheng He Ruby B. Lee
{zechengh, rblee}@princeton.edu

Department of Electrical Engineering
Princeton University

This work was supported in part by a research gift from Intel Labs

1. Introduction

Cache side channel attacks [1][2][3][4][5] have become serious threats against computer security. Crypto keys can be leaked through these side channels. These attacks have been well studied and many defense mechanisms have been proposed. Based on the attack targets, these attacks can be classified as hits/misses based attacks. Based on the attacker's measurement of his own memory access time or the victim's total execution time, attacks can be classified as access-based attacks or timing-based attacks, respectively.

Intel Software Guard Extensions (SGX) [6][7][8][9] and SGX-2 [10] are designed for protecting a user's data in the memory. The threat model claims that the OS and hypervisor are both untrusted. The only hardware trust base is the CPU chip. Data in the memory are encrypted. They are only decrypted when transmitted to the processor. However, this threat model does not consider cache side channel attacks. It is possible that SGX still leaks information through cache side channel attacks. Moreover, because transmitting data to/from memory requires encryption and decryption, the difference of access time between memory hits and misses becomes larger under the protection of SGX. This feature may cause the leakage of information through cache side channel attacks to be even faster than without the protection of SGX.

In this report, we show one representative class of cache side channel attacks, the evict-and-time attacks, still works under the protection of SGX. Also we find a novel phenomena which can be used as a new side-channel attack targeted at the implementation of SGX.

This report is organized in five sections. In Section 2, we introduce our experimental settings and configurations for evict-and-time attacks. In general, we have done the following evict-and-time attack experiments:

1. AES encryption on an SGX machine with SGX enclave, all-zero keys
2. AES encryption on an SGX machine with SGX enclave, non-zero keys
3. AES encryption on an SGX machine but without using enclave, all-zero keys
4. AES encryption on an SGX machine with SGX enclave and no evicting of any of the cache sets, all zero keys
5. AES encryption on an SGX machine using enclave, swap table location of encryption Table 1 and Table 2, all zero keys
6. AES encryption on a non-SGX machine, all zero keys
7. AES encryption on a non-SGX machine, with memory fences and all zero keys
8. AES encryption on a SGX machine with enclave, with memory fences and all zero keys

We also show two interesting observations in this section:

- **Observation 1:** A very high point which could make evict-and-time attack easier.
- **Observation 2:** The mixture of unusual high and low points, instead of the expected all high points.

In Section 3, we provide more experimental results for Observation 1. We prove that the crypto keys can leak through Observation 1, resulting in a more accurate attack when SGX enclaves are used. We show more experiments comparing an SGX machine and a non-SGX machine and give our explanation for Observation 2 in Section 4. We discuss the feasibility of integrating new cache architectures into SGX to avoid cache side-channel attacks in Section 5. We conclude our work and propose some future work directions in Section 6.

2. Configuration of Experiments and Result Figures

We have implemented an evict-and-time attack on an SGX machine with enclaves. The general workflow of the attacker is shown below:

For i < Total number of tables for AES encryption
For j < Total number of plaintext (random bytes) blocks
Evict a whole cache set of Table i
Start time T1
Pick plaintext block j (16 bytes) for encryption
End time T2
Store execution time (T2-T1) according to plaintext bytes
End for
Store encryption time minus the average
End for

In our experiments, AES uses five lookup tables for encryption (four for regular rounds and one for the last round). The total number of plaintext blocks is at most 2^{28} in our experiments. The L1 cache on the SGX machine is 32KB 4-way set-associative. The cache line size is 64 bytes. The SGX machine uses Intel Core i7-6700 processor and Ubuntu 14.04 operating system.

The result figures in the rest of this report follow the rules below:

- Each experiment has 4 figures, according to which Table is evicted. The first figure shows results when Table 0 is evicted, the second figure shows results when Table 1 is evicted and so forth.

- There are 16 subfigures in each figure, arranged as a 4X4 array. We index the subfigures from left to right, and from top to bottom, i.e. subfigure 0,1,2,3 in the first row, subfigure 4,5,6,7 in the second row and so forth. In order to clearly show our results, the scale of the y-axis may be different. For example, the column 3 in Figure 1.1-1.4 is an order of magnitude larger than column 1,2 and 4 in the same figures.

- The x-axis of subfigure i shows the value of plaintext byte i, from 0 to 255. The y-axis of subfigure i shows the average encryption time (number of cycles) when the value of plaintext byte i is 0-255. For example, “x=150 and y=100 in subfigure 3” means the average encryption time when “byte 3 of the plaintext equals to 150” is 100 cycles. The cycle number in each subfigure has subtracted the average of that subfigure.

As mentioned in Section 1, we list the figure number of each our experiments here.

1. **AES encryption on an SGX machine using an enclave, all-zero keys (Section 2, Figure 1.1-1.4)**

2. **AES encryption on an SGX machine using an enclave, non-zero keys (Section 3, Figure 2.1-2.4)**

3. **AES encryption on an SGX machine but without using enclave, all-zero keys (Section 3, Figure 3.1-3.4)**

4. **AES encryption on an SGX machine using enclave and no eviction on any of the cache sets, all zero keys (Section 3, Figure 4.1-4.4)**

5. **AES encryption on an SGX machine using enclave, swap table location of encryption Table 1 and Table 2, all zero keys (Section 3, Figure 5.1-5.4)**

6. **AES encryption on a non-SGX machine, all zero keys (Section 4, Figure 6.1-6.4)**

7. **AES encryption on a non-SGX machine, with memory fences and all zero keys (Section 4, Figure 7.1-7.4)**

8. **AES encryption on a SGX machine using an enclave, with memory fences and all zero keys (Section 4, Figure 8.1-8.4)**

We first implement an evict-and-time attack against AES encryption with enclave and all zero keys. The results are shown in Figure 1.1-1.4.

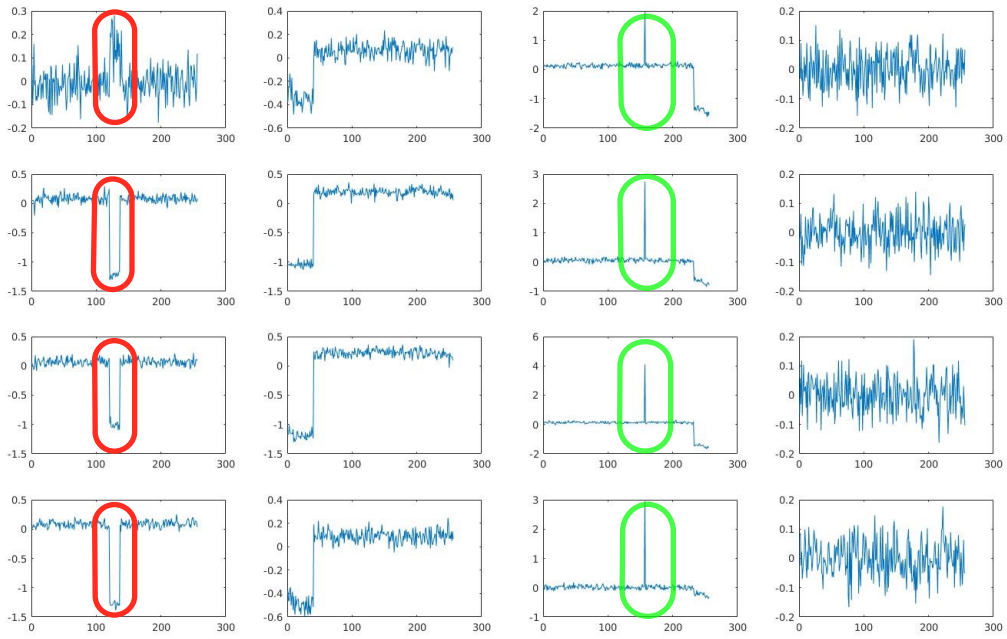

Figure 1.1

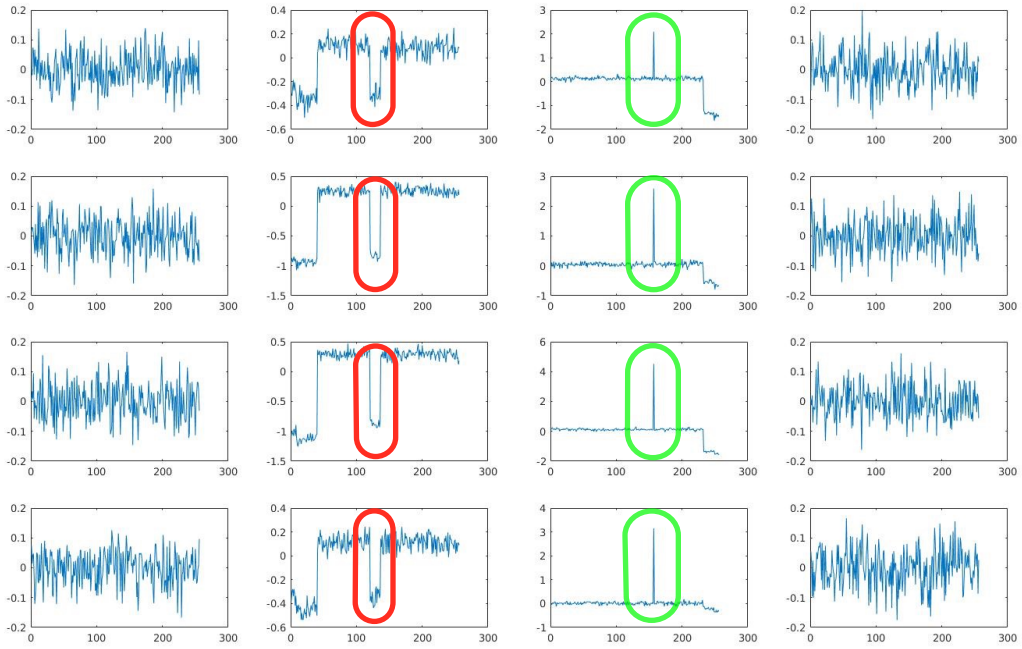

Figure 1.2

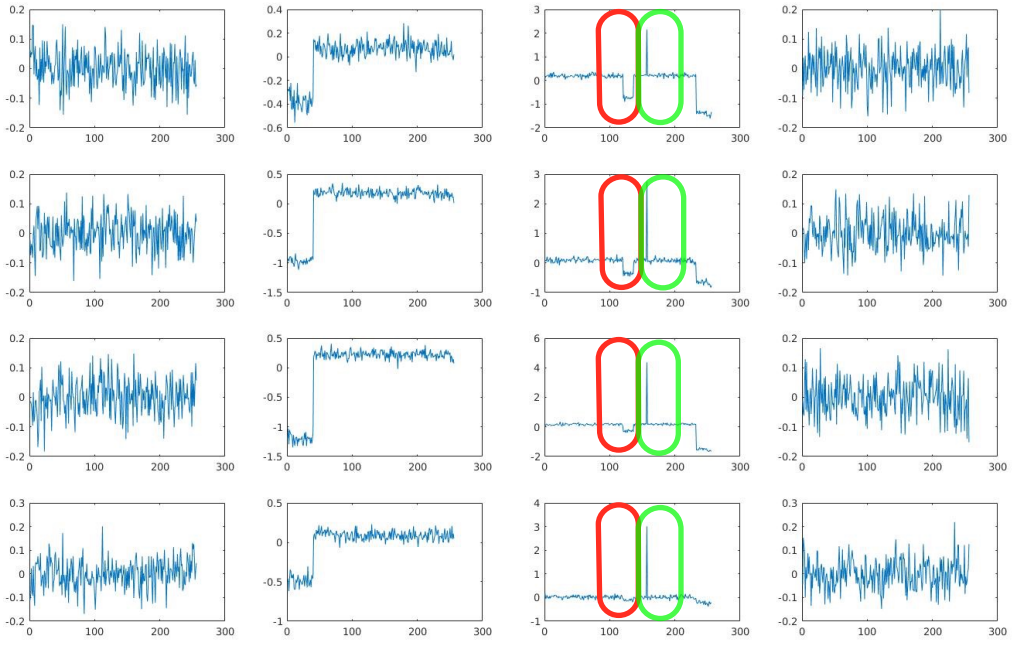

Figure 1.3

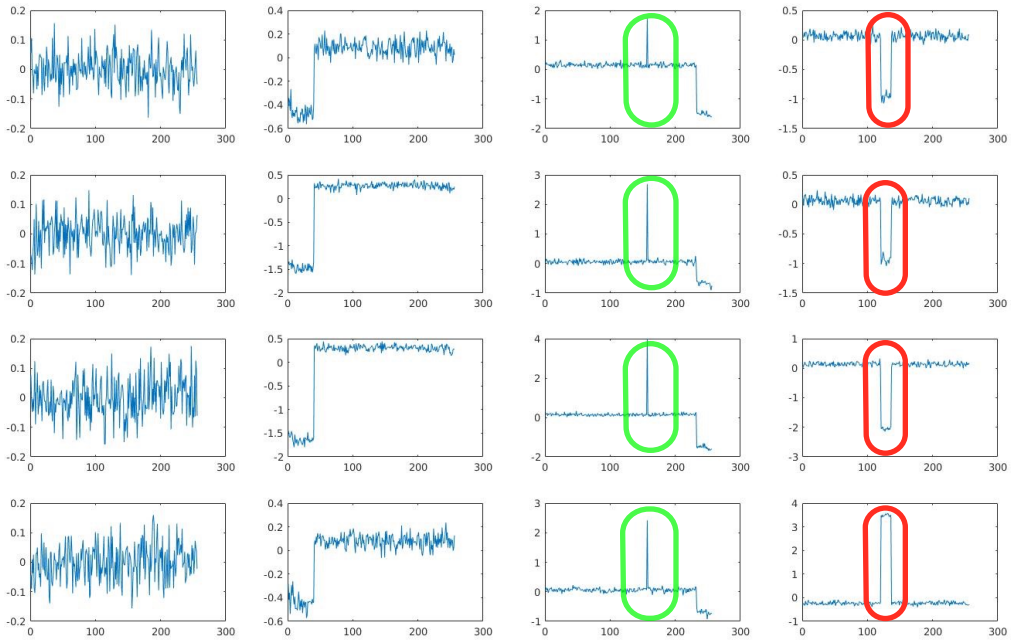

Figure 1.4

Figure 1.1-1.4 show the traditional evict-and-time attack results on an SGX machine using an SGX enclave. We set all keys to zeros. This attack succeeds because we can see clear high and low points (marked red) at x=120~135 in column 1-4 subfigures of figure 1.1-1.4, respectively. 4 out of 8 bits of each byte of AES keys can be obtained through these high and low points.

Moreover, we have two interesting observations:

Observation 1: There are very high points (marked green) in column 3 in all subfigures at x=156. We will show in the next two sections that these high points leak more crypto key information and are easier to be exploited by the attackers. Our experiments reveal that only 2^{23} samples are needed to observe these very high points.

Observation 2: The high and low points (marked red) are supposed to be all high points, according to the mechanism of the evict-and-time attack. However, we find some of these points are high but some are low.

Naturally, these observations lead to two open questions:

Question 1: Why do the very high points (marked green) occur?
Question 2: Why are the points which are supposed to be all high (marked red) actually a mixture of high and low points?

We try to give more clues and explanations on these two open questions in the following two sections.

3. A Better evict-and-time attack on SGX enclaves (from Observation 1)

In section 2, we have observed some high points in our experiments. In this section, we show these high points can actually be misused to get crypto keys, even easier than traditional evict-and-time attacks. We also give more results to help trace back the cause of Observation 1.

We first implement the same evict-and-time attack on the same SGX machine with enclave but with non-zero keys. We show that the positions of the very high points change according to the key bytes values. The results are shown in Figure 2.1-2.4. In column 3, we see a single high point in each of the subfigures (marked green). These high points enable us to easily recover all 8 bits of a non-zero key byte, by simply XOR the x-axis values of the high points in all-zero key experiments and non-zero key experiments.

For example, in Figure 1.1 (all zero key), the x-axis values corresponding to the high points are {156,156,156,156}. In Figure 2.1 (non-zero key), the x-axis values corresponding to the high points in the 3rd column are {140,188,172,220}. Four key bytes of the non-zero key can be obtained by {156,156,156,156}⊕{140,188,172,220}={0x10, 0x20, 0x30, 0x40}. These are the key bytes 2, 6, 10 and 14 used in our experiment.

The same is also true for Figure 2.2-2.4 when XORed with Figure 1.2-1.4, respectively. All eight bits (not just 4 bits) can be obtained for 4 key bytes of the non-zero key.

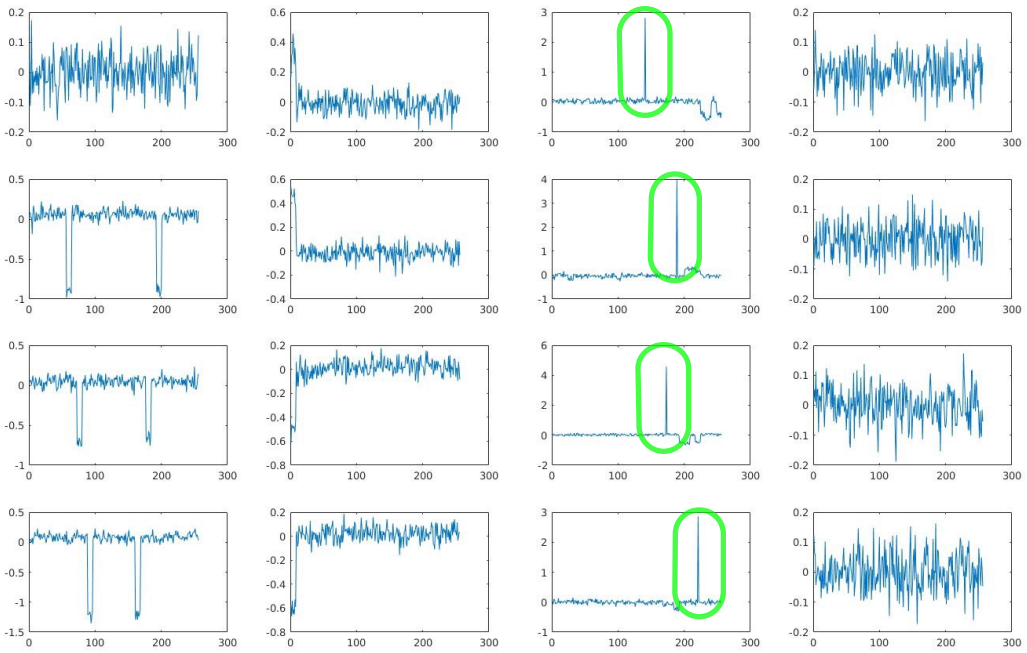

Figure 2.1

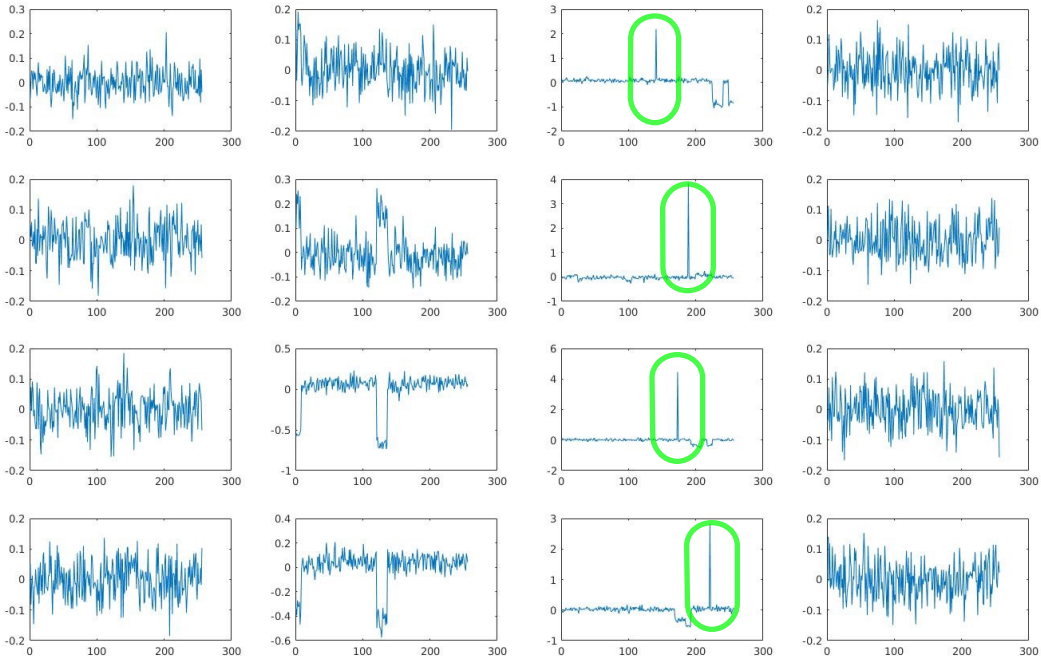

Figure 2.2

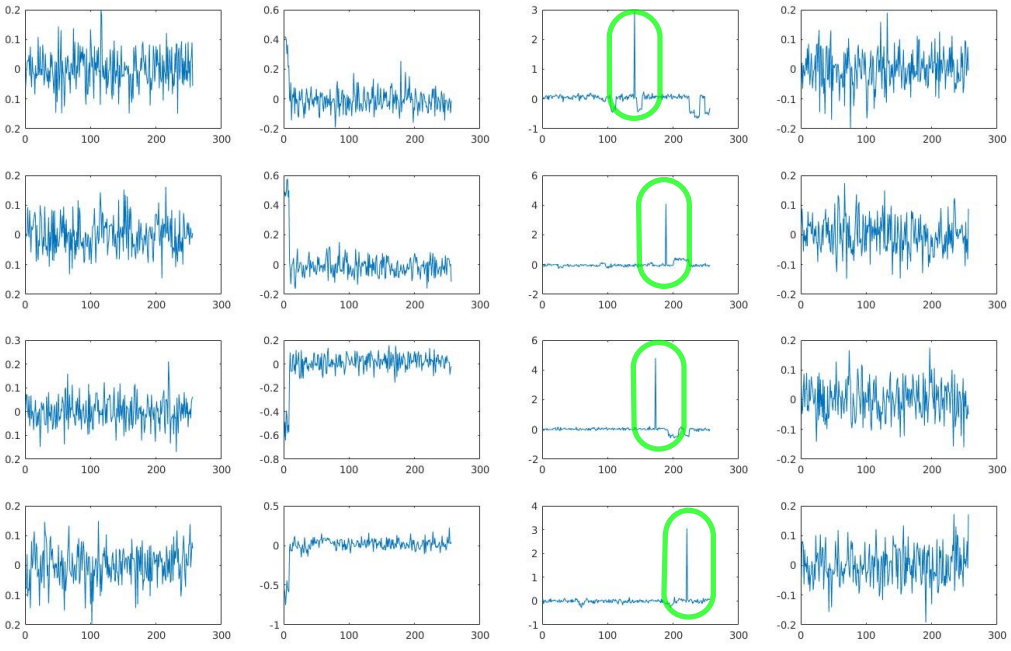

Figure 2.3

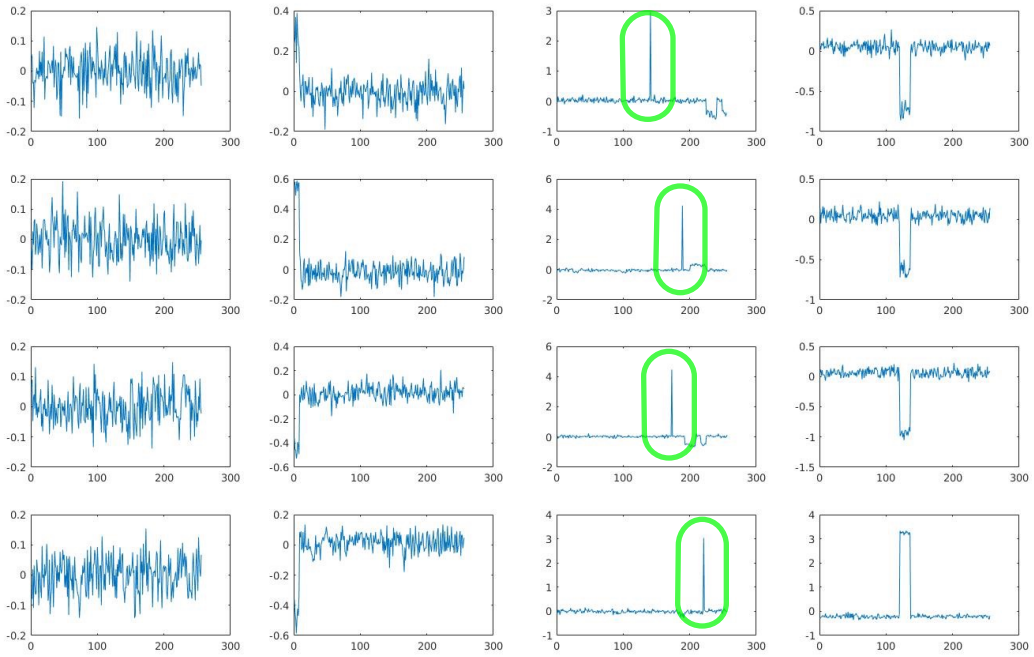

Figure 2.4

Our second experiment in this section is an evict-and-time attack on the same machine but not using an enclave. The goal of this experiment is to find whether the high points are caused by the particular SGX machine implementation or by the SGX enclave architecture and implementation. The results are shown in Figure 3.1-3.4.

The high points are still there (marked red) but we find two significant differences.

1. There are four to five (continuous) high points rather than only one in column 3 of each figure. The x-axis values of these high points are around 250.

2. Moreover, we find that the locations of the high points never move if we use non-zero keys. This means although the high points are still there, they do not leak information about the crypto keys.

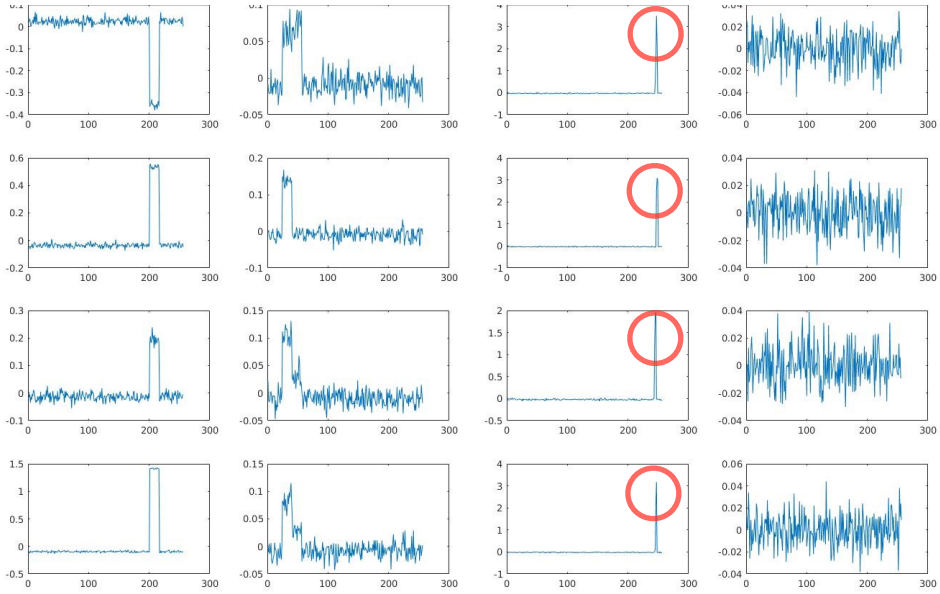

Figure 3.1

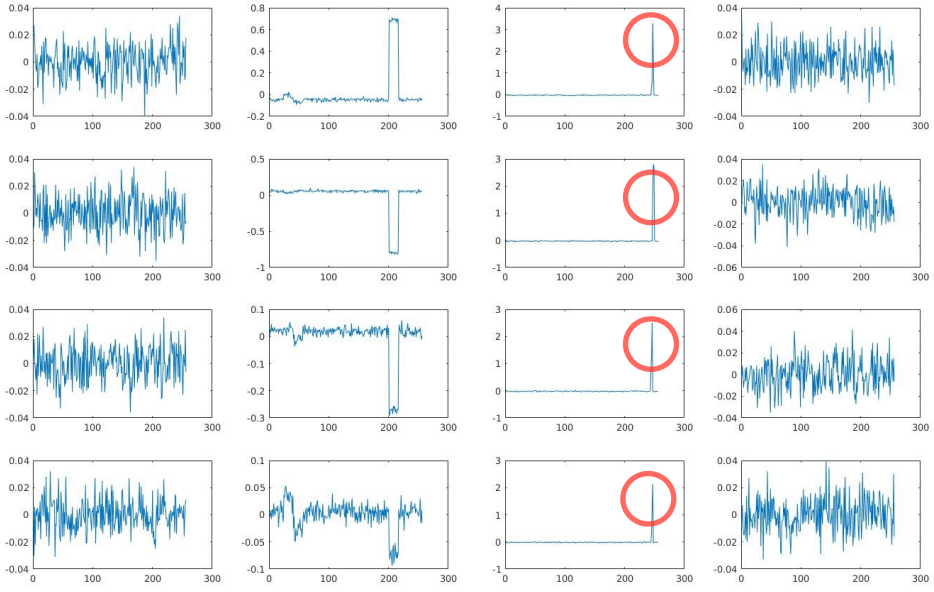

Figure 3.2

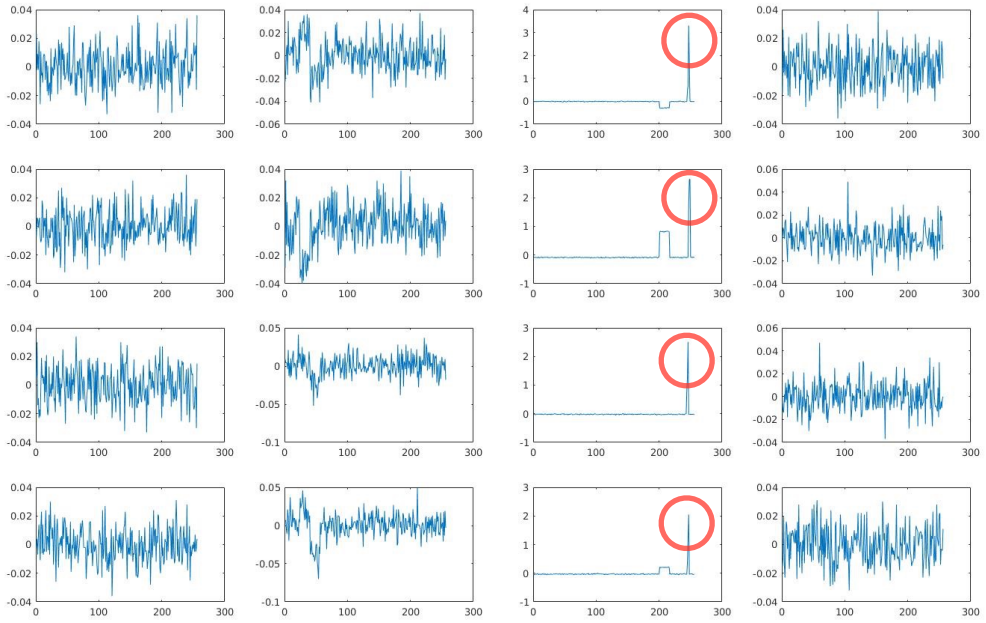

Figure 3.3

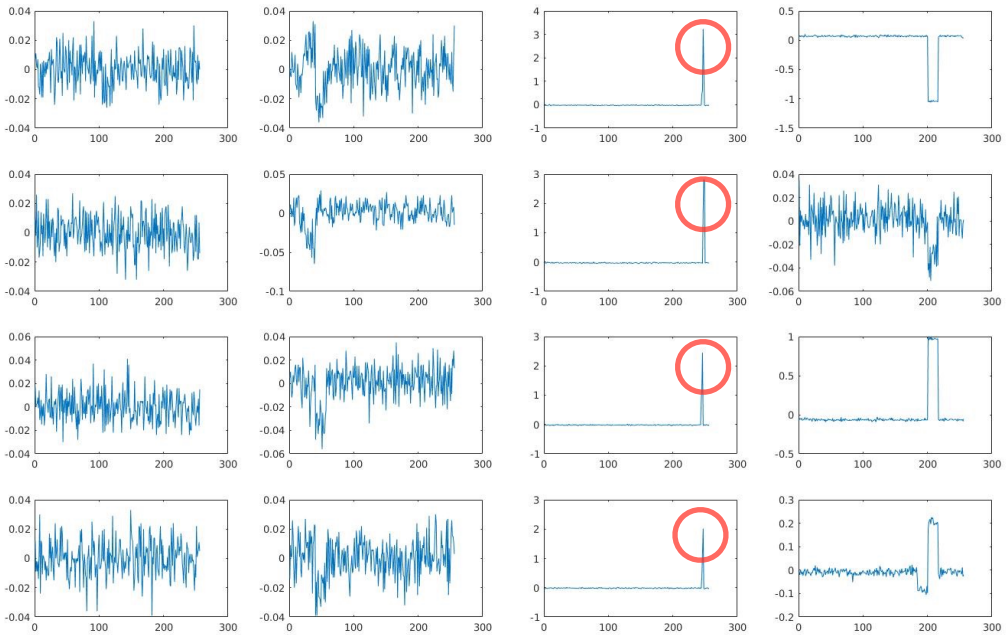

Figure 3.4

From the previous experiments, we speculate that the occurrence of these high points may be because of the accesses of some specific memory addresses in the implementation of SGX enclave. We use the following experiment to support our hypothesis. In this experiment, the attacker does not evict any cache set but only runs encryptions of random plaintext blocks, in an SGX machine using an enclave. The results of this experiment are shown in Figure 4.1-4.4.

In Figure 4.1-4.4, we can still clearly see the high points in the subfigures. Because we are using all zero keys, the x-axis of the high points are 156 as in Figure 1.1-1.4. These results support our hypothesis that some specific memory addresses are always accessed in the implementation of an SGX enclave, which behaves very similar to “eviction of a whole cache set”.

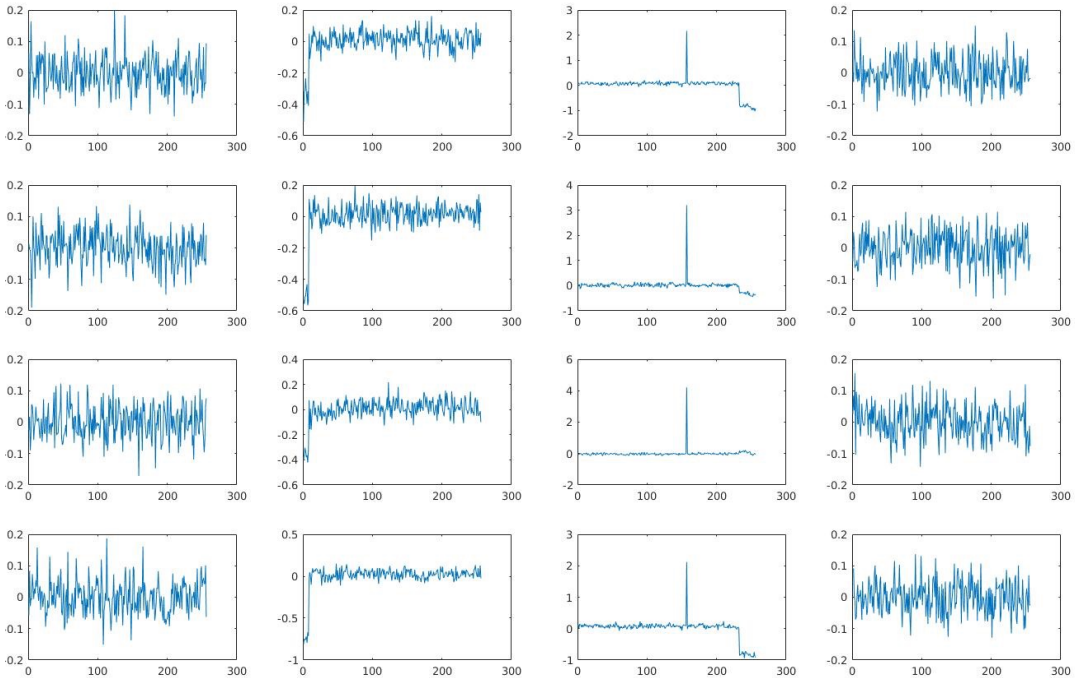

Figure 4.1

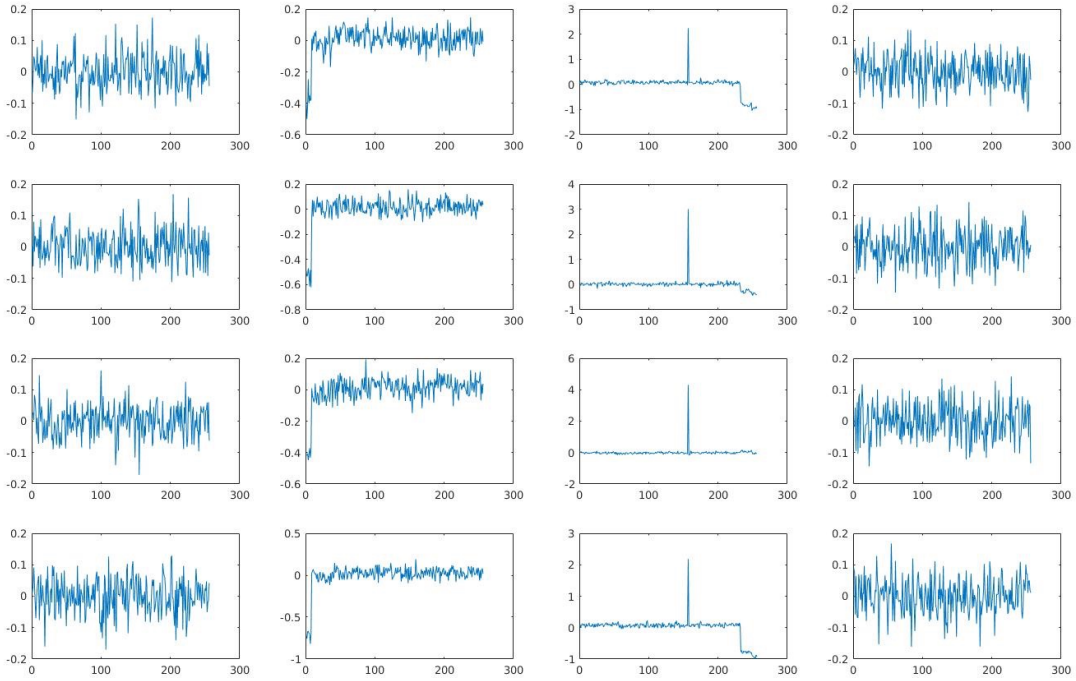

Figure 4.2

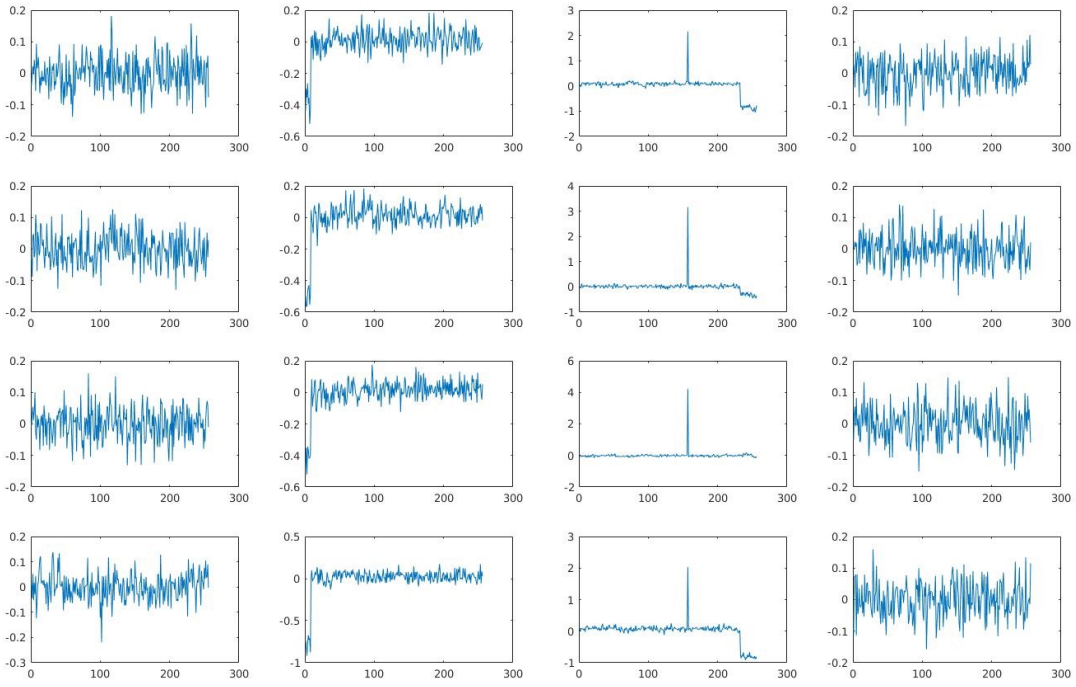

Figure 4.3

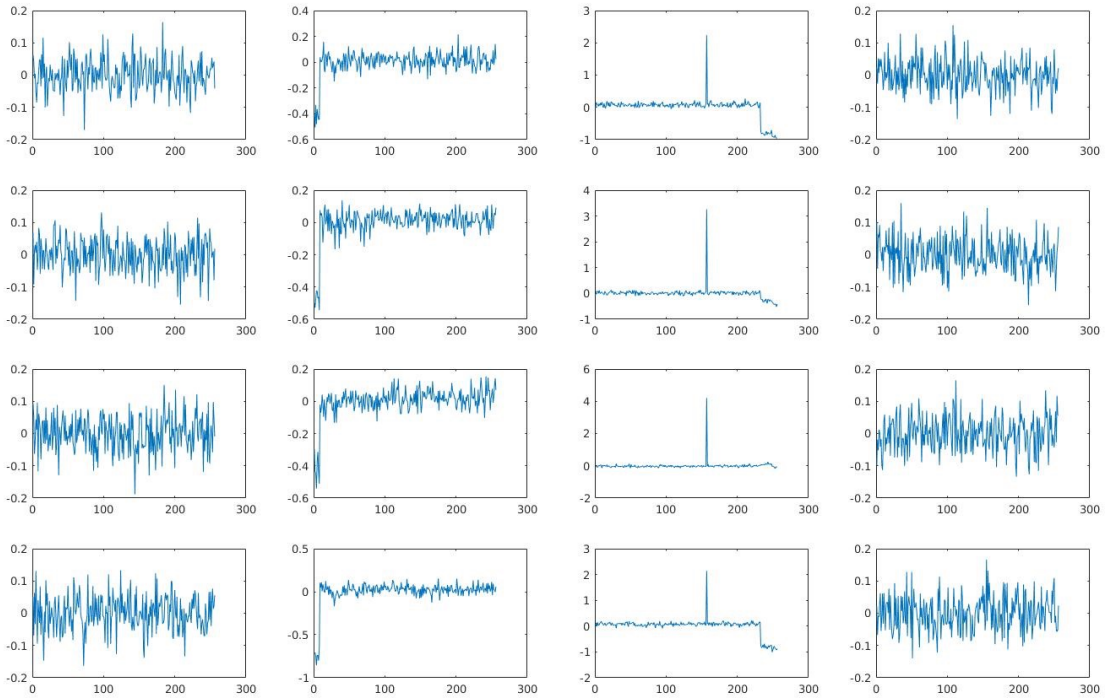

Figure 4.4

In the previous experiments, we find that no matter which encryption table entries we evict, the high points always occur in the third column. This feature means the high points only leak bytes 2, 6, 10, 14 of the crypto key. In order to see whether it is possible to leak other key bytes, we do the following experiment that swaps the location of Encryption Table 1 and Encryption Table 2. The results are shown in Figures 5.1-5.4.

In Figures 5.1-5.4, we find that the high points change to the second column of the subfigures. By the results, the attacker is able to obtain full bits of crypto key bytes 1, 5, 9, 13. We believe if we swap the locations of Table 0 and Table 2, we can obtain key bytes 0, 4, 8, 12, and if we swap the locations of Table 3 and Table 2, we can obtain key bytes 3, 7, 11, 15 respectively.

Therefore, the attacker can do the following to obtain all 16 bytes of the key:

1. If he has the ability to determine AES table locations, he can simply swap table locations.

2. If not, he can restart the AES service and let the OS (in the SGX threat model, the OS can be compromised) map the tables to different locations.

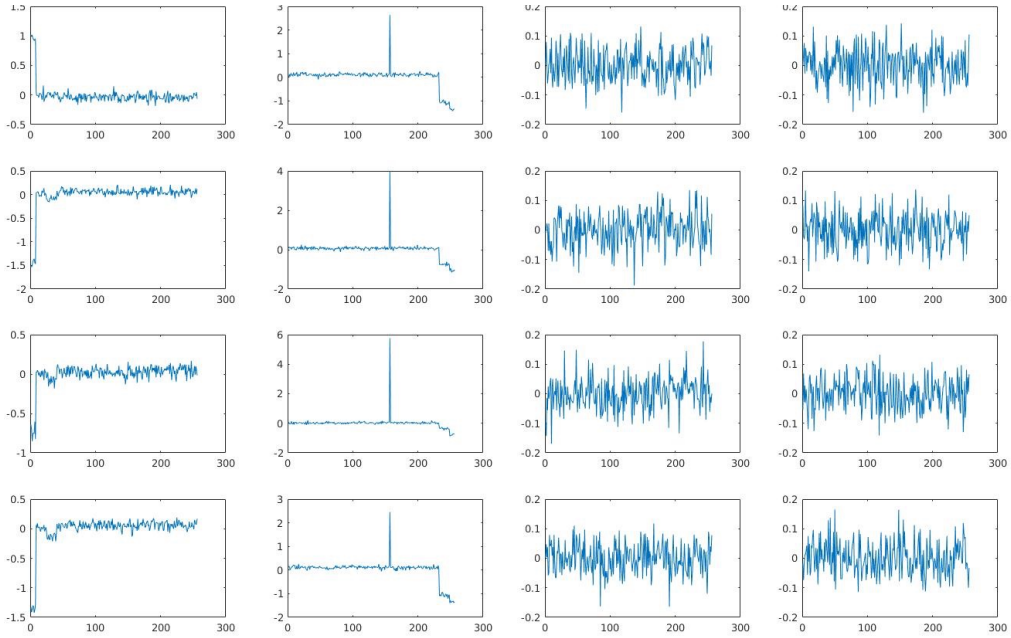

Figure 5.1

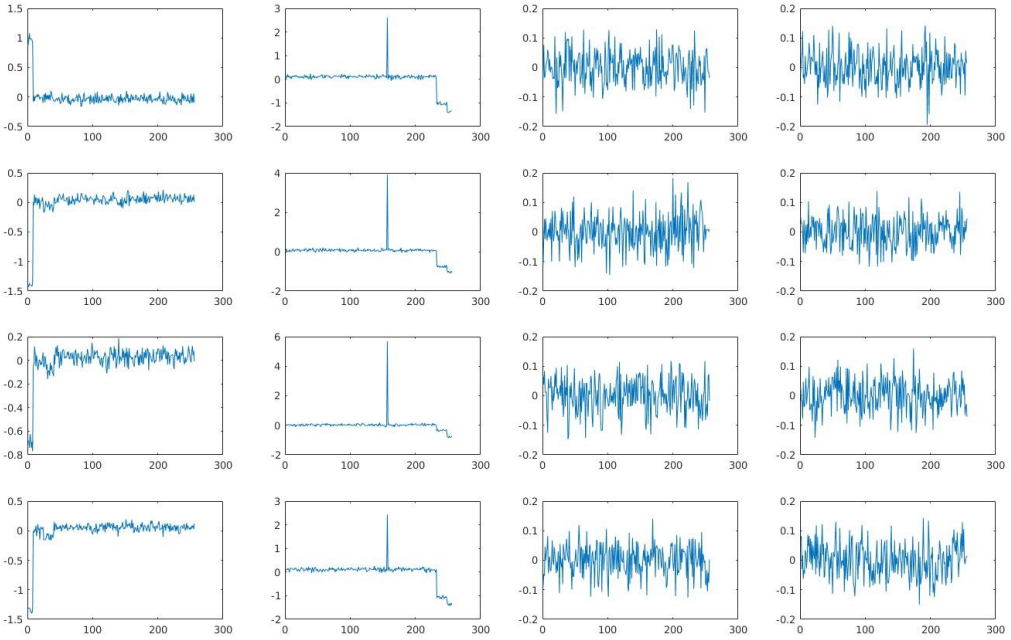

Figure 5.2

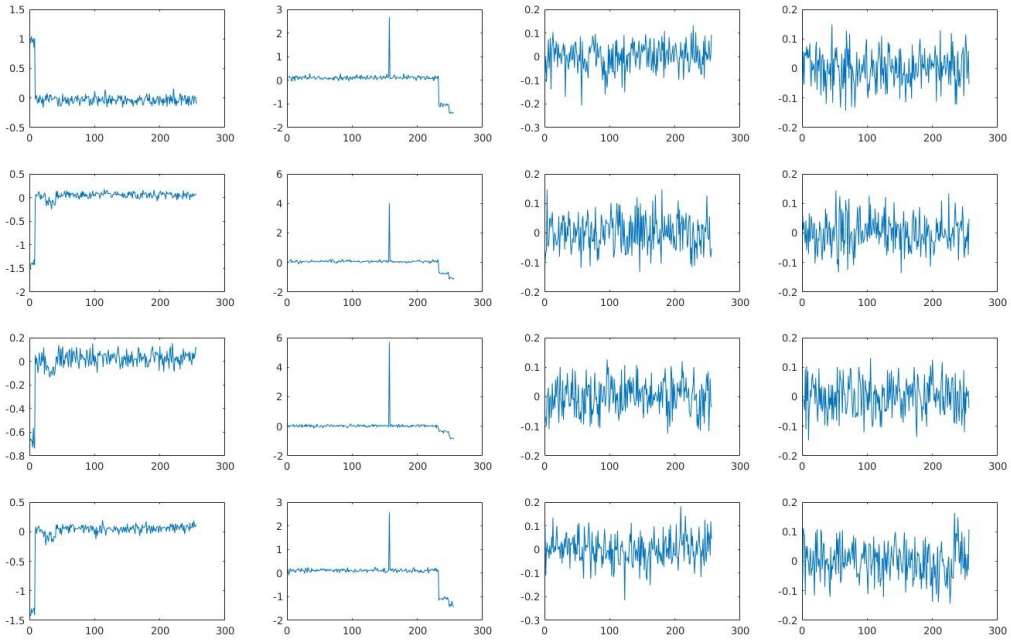

Figure 5.3

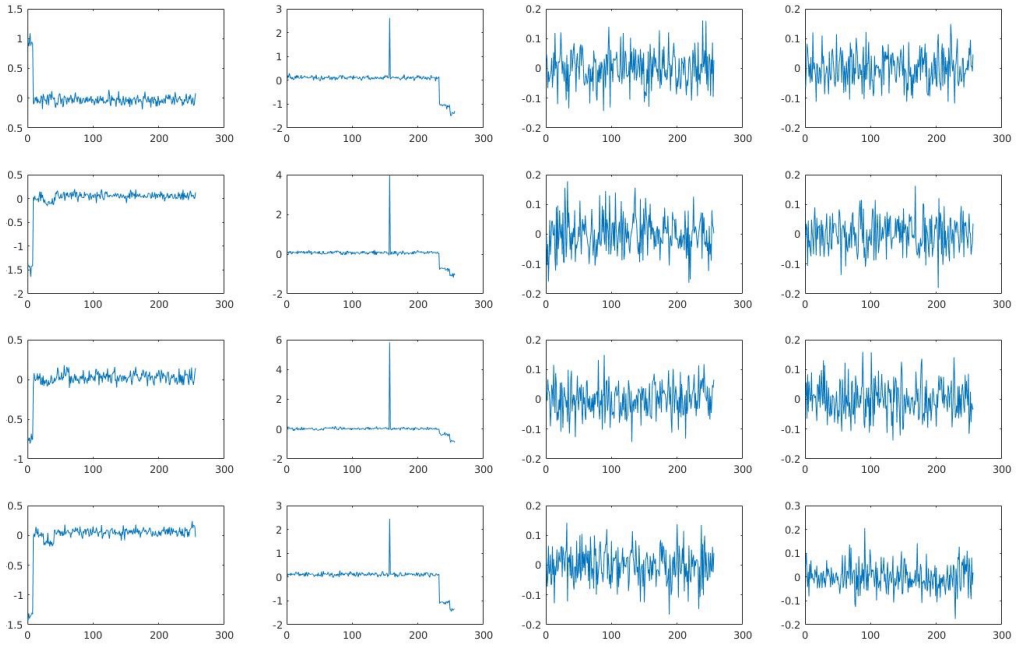

Figure 5.4

4. More experiments on Question 2

In this section, we discuss the second question proposed in Section 2: why are the points which are supposed to be high actually a mixture of high and low points?

Intuitively, the low points in the subfigures reveal that some memory accesses are faster than the others. It is very similar to cache collision attacks, where some table accesses benefit from the earlier memory accesses. For the same reason, we speculate that these low points are memory accesses which benefited from the previous ones. However, we have not proved that in this report. For the rest of this section, we try to prove a looser conclusion that the low points are related to the order of the memory accesses. An out-of-order implementation may optimize memory accesses by re-ordering the order of these memory accesses.

We first implement the evict-and-time attack on a non-SGX machine. The results of this experiment are shown in Figure 6.1-6.4. We find that in each figure two columns have high/low points (marked red in Figure 6.1 and similar in Figure 6.2, 6.3 and 6.4). That is because the L1 cache of the non-SGX machine is 8KB and 4-way set-associative with line size of 64 bytes. Total encryption table size (4KB) is larger than cache way size (2KB). Hence, the eviction of a specific cache set affects two tables. Table arrangements in L1 cache of an SGX machine (32 Kbytes) and a non-SGX machine (8 Kbytes) are shown in the figure below.

Because the unusual points are still a mixture of high and low points, we can conclude that Question 2 is not because of the SGX machine nor enclave implementation.

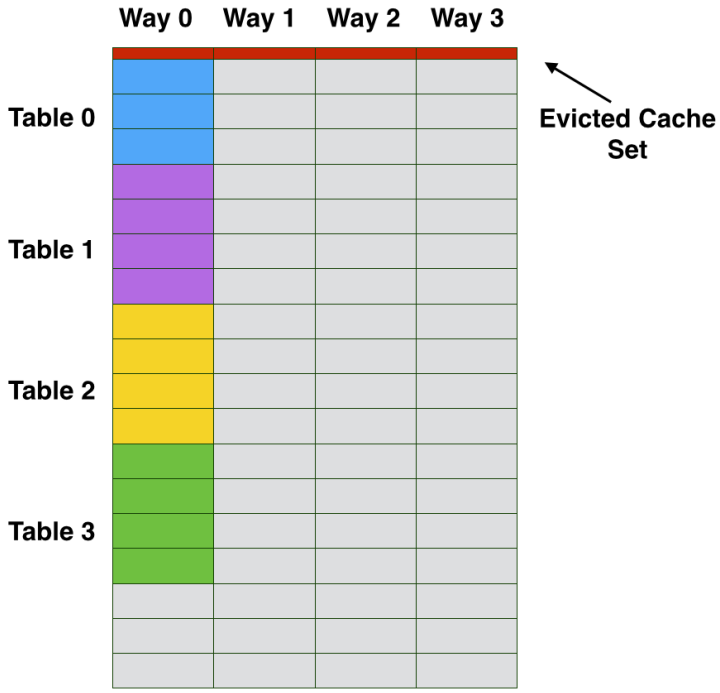

Table arrangment in L1 cache on SGX machine

Table arrangment in L1 cache on non-SGX machine

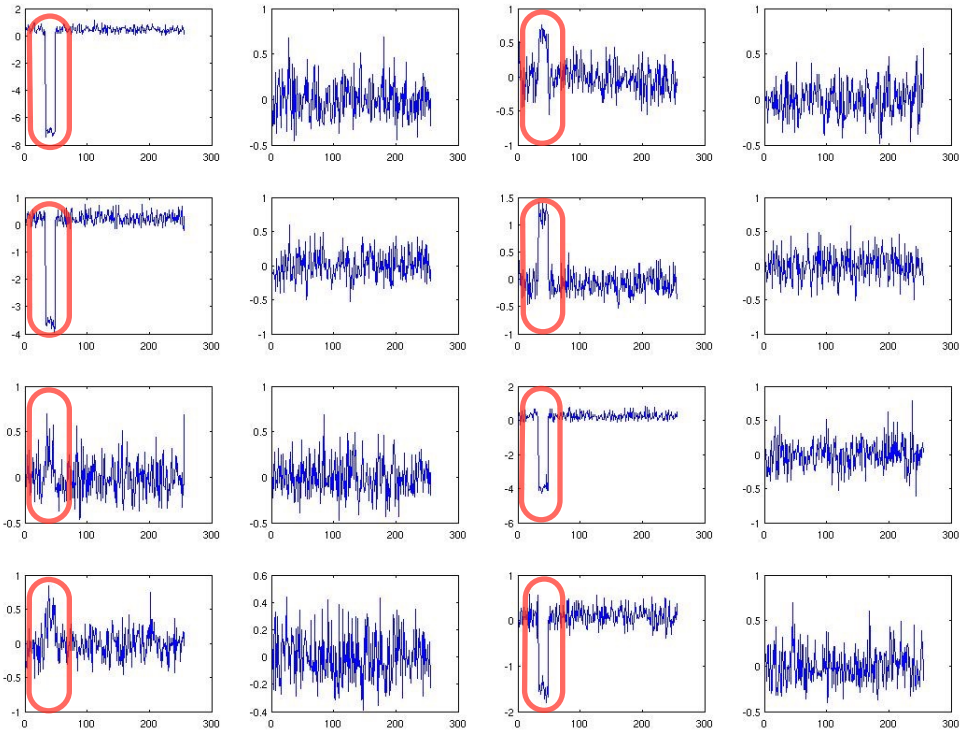

Figure 6.1

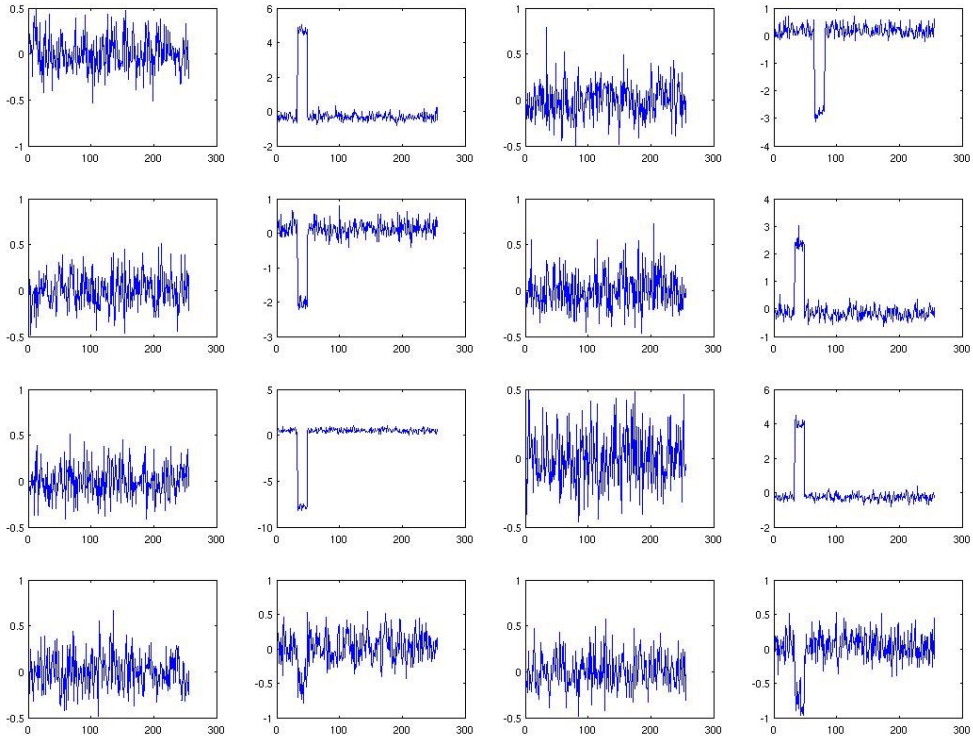

Figure 6.2

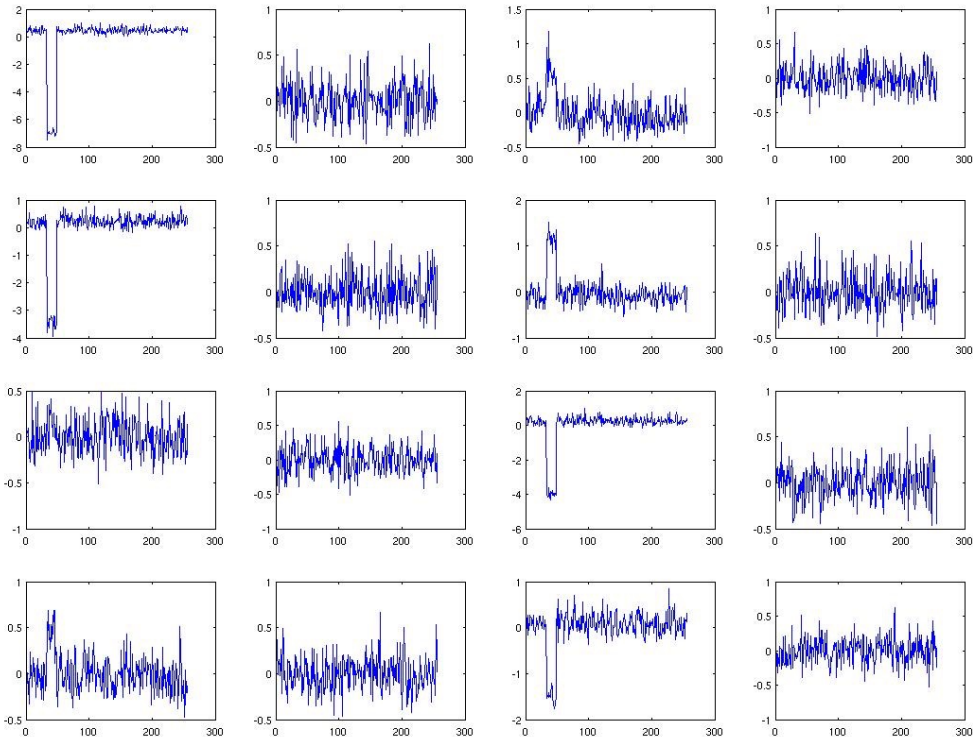

Figure 6.3

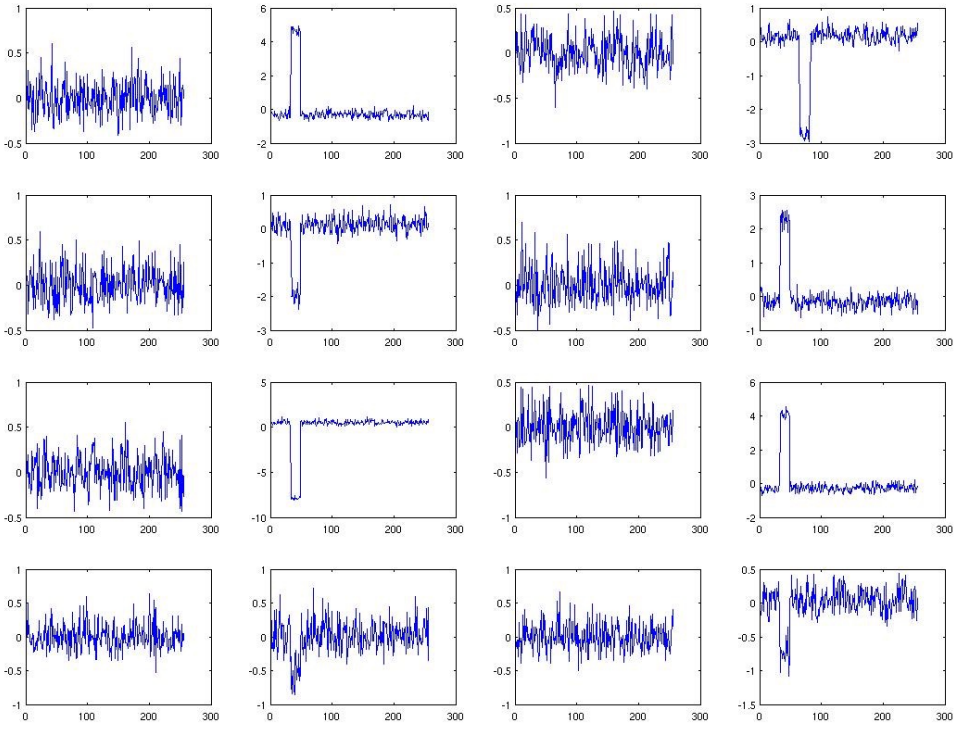

Figure 6.4

As mentioned earlier in this section, we speculate that the cause of the unusual high and low points is the order of the memory accesses. In the following experiment, we add memory fences between table accesses during encryption to prevent the out-of-order execution of memory operations. Specifically, we add memory fences in the following way to separate memory reads of the same table as shown below:

For i = 1 to 9 (rounds)
Memory Fence
Read an entry in Table 0
Read an entry in Table 1
Read an entry in Table 2
Read an entry in Table 3
XOR operation and other
Memory Fence
Read an entry in Table 0
Read an entry in Table 1
Read an entry in Table 2
Read an entry in Table 3
XOR operation and other
Memory Fence
Read an entry in Table 0
Read an entry in Table 1
Read an entry in Table 2
Read an entry in Table 3
XOR operation and other
Memory Fence
Read an entry in Table 0
Read an entry in Table 1
Read an entry in Table 2
Read an entry in Table 3
XOR operation and other

The results of this experiment are shown in Figure 7.1-7.4. All unusual points become high points. This supports our hypothesis that the occurrence of unusual high and low points is because of the order of memory accesses on a non-SGX machine.

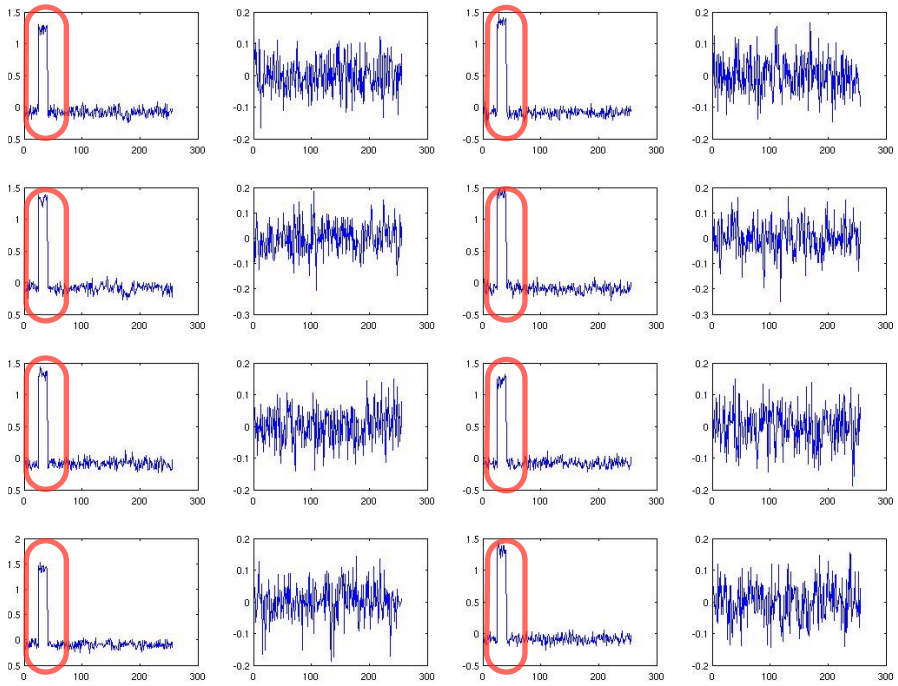

Figure 7.1

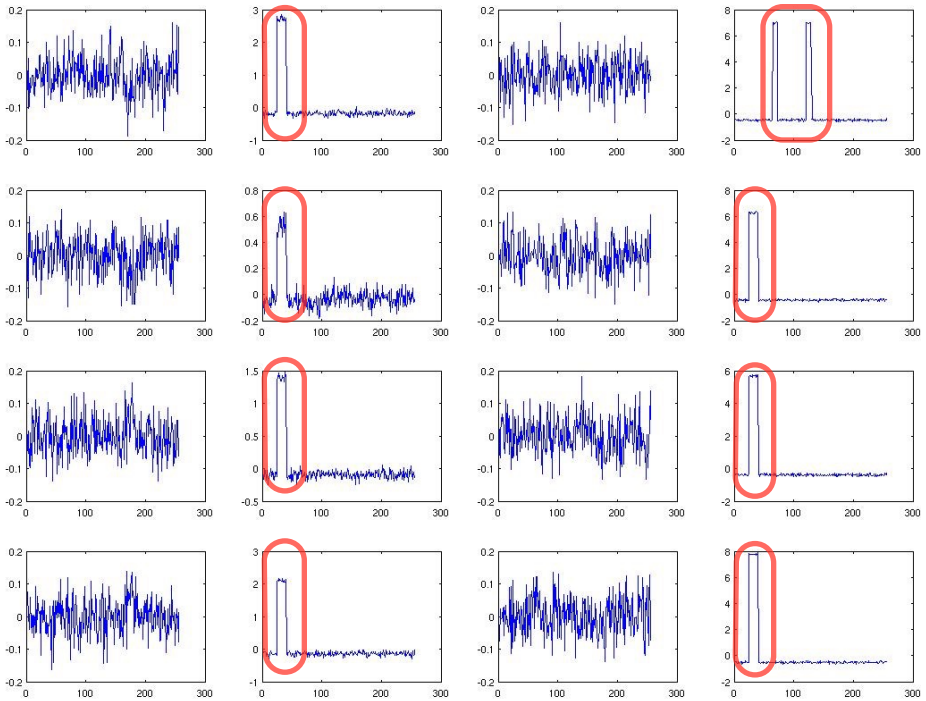

Figure 7.2

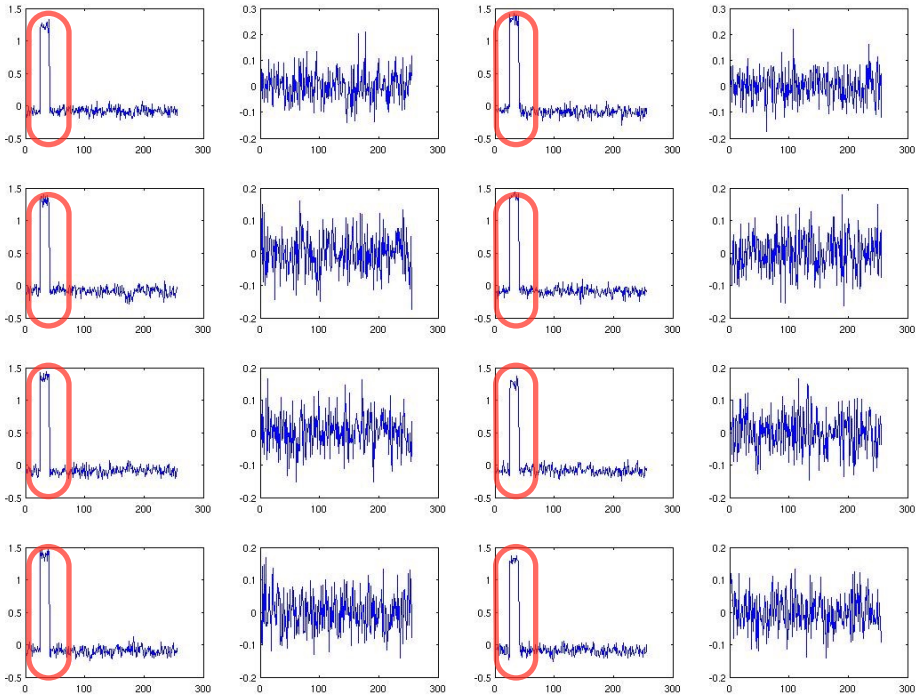

Figure 7.3

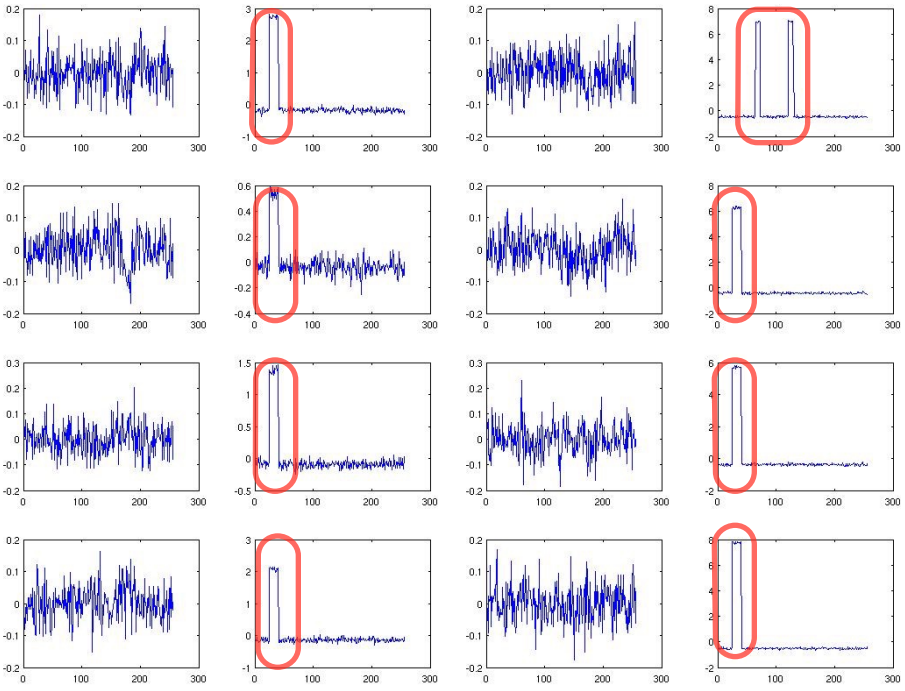

Figure 7.4

Figures 8-1, 8-2, 8-3, 8-4 show results of the evict-and-time attack on an SGX machine using enclave and with memory fences. We find that all unusual points are high points now (marked red), which supports our assumption that the high and low points are caused by the ordering of memory accesses in the SGX machine.

However, we find two new phenomena which we cannot explain now when memory fences are used in an SGX enclave:

1. Only one very high point occurs in subfigure column 3 row 3, at the same position x=156 as in Section 2 (marked green). The y-axis is scaled up to 15. Compared with this, Figures 2.1, 2.2, 2.3 and 2.4 have very high points in all 4 subfigures in column 3.

2. The high points (marked red in Figure 8.1 column 1, Figure 8.2 column 2, Figure 8.3 column 3 and Figure 8.4 column 4) disappear in Figure 8.3 subfigure column 3 row 3 (marked with dotted red line).

3. Some other strange high points (marked yellow). However, we do not find that the position of these high points are related to the key bytes.

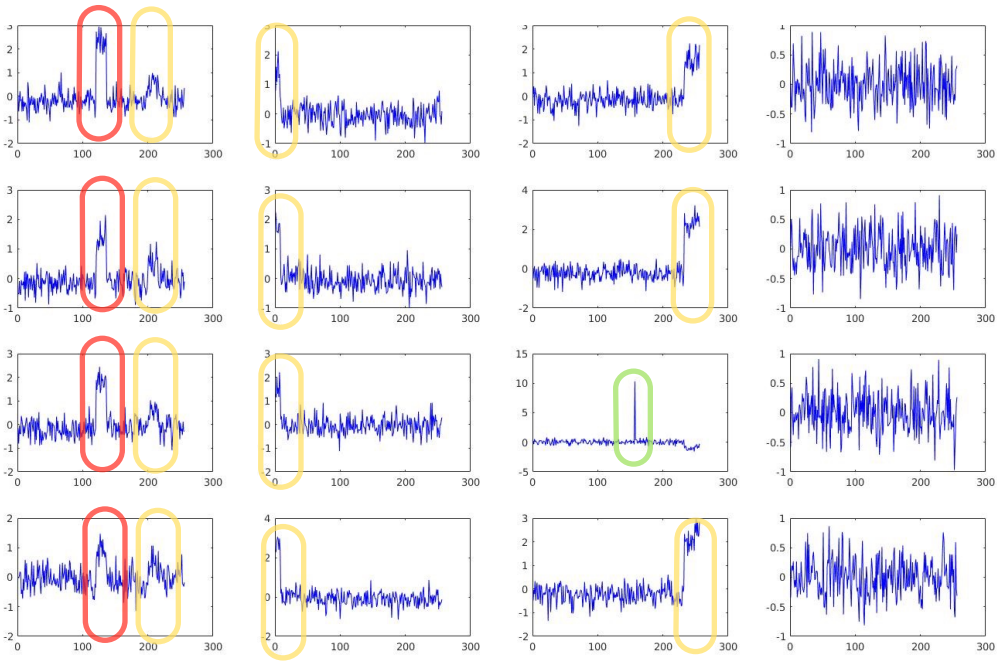

Figure 8.1

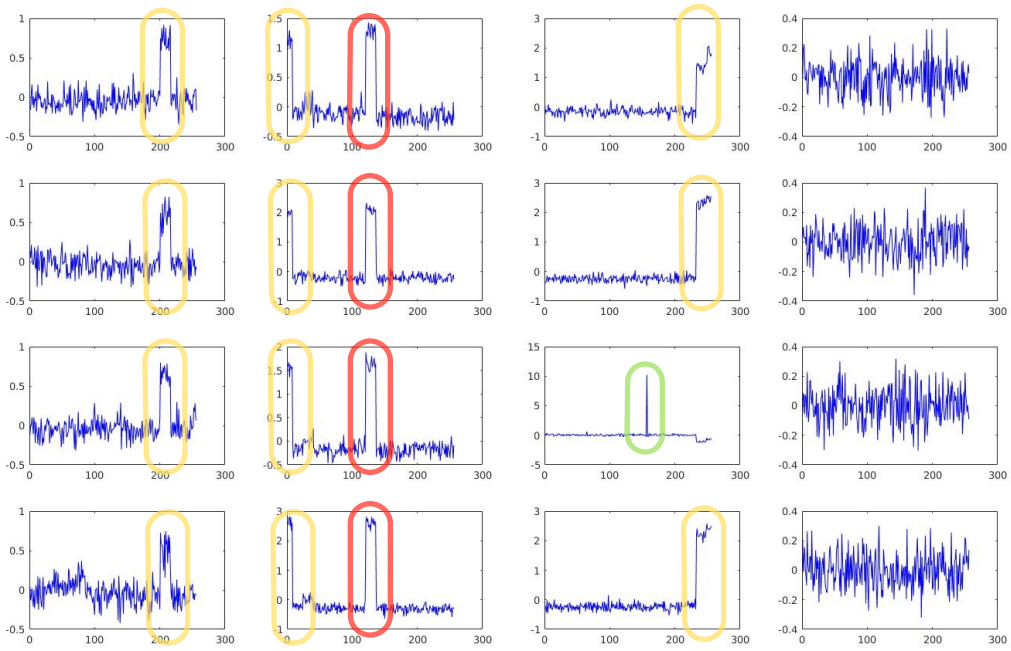

Figure 8.2

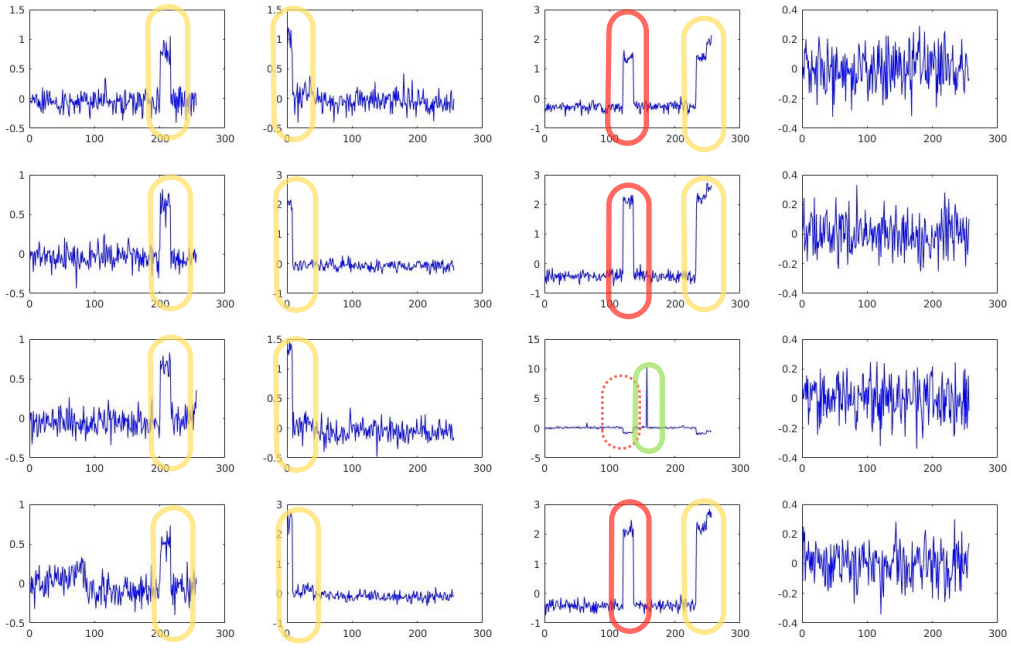

Figure 8.3

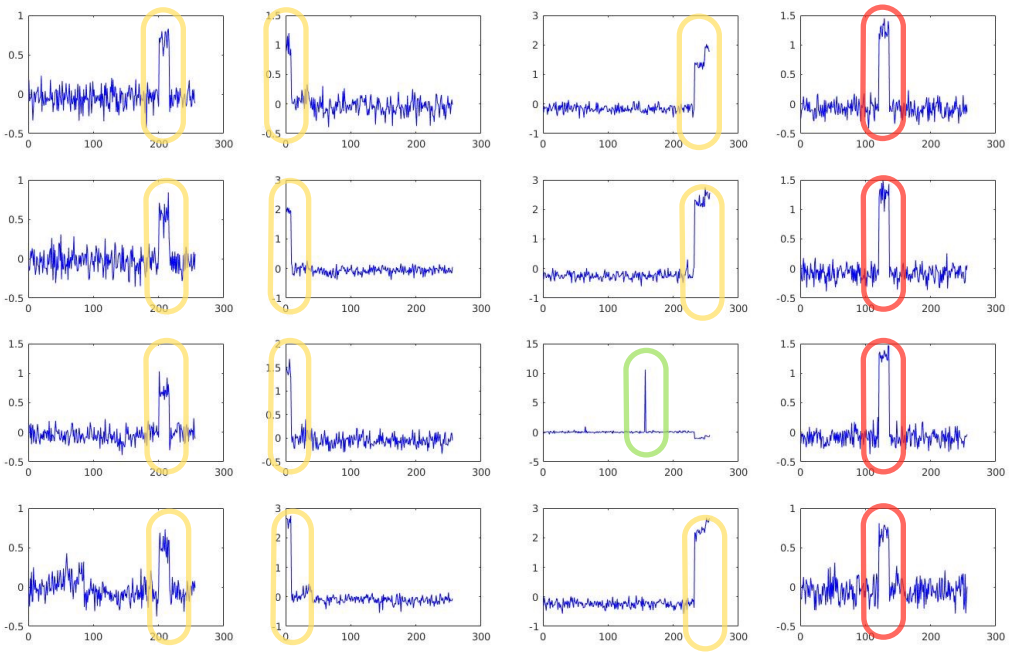

Figure 8.4

5. Integrate A Secure Cache in SGX ?

As shown in Section 2, Section 3 and Section 4, evict-and-time attacks still work, and are even easier for the attacker on SGX. In this section, we talk about possible solutions of mitigating cache side channel attacks in SGX.

Secure cache architectures which provide protection against cache side-channel attacks have been proposed [11][12][13]. In particular, Newcache [11][12] is an architecture that is able to defeat several cache attacks, including evict-and-time and prime-and-probe attacks, without degrading performance.

We have two ways to verify whether Newache is really able to defeat these attacks in SGX. First and the best way is integrating Newcache into SGX simulators. However, currently we do not have access to the SGX simulator and we would be very glad if we can have a chance to collaborate with Intel and simulate Newcache on SGX.

Second, we can try to model the Newcache and SGX with the side channel attacks. We are currently investigating "probabilistic information flow graph (PIFG)" model. This consists of nodes and edges (labelled e_i), with functions on the edges. Probabilities of states and edge transitions can also be given. The attack is successful if there exists a path from the victim's memory accesses to the attacker's observation.

6. Conclusions and future work

In this report, we show the evict-and-time attack still works in SGX. We also find two interesting observations:

- **Observation 1:** A very high point which could make evict-and-time attack easier.
- **Observation 2:** The mixture of unusual high and low points, instead of the expected all high points.

We do some comparison experiments and find that the cause of Observation 1 may be particular memory accesses in the implementation of an SGX enclave. We also show that Observation 2 is related to the order of memory accesses. We also briefly talk about modeling the cache attacks.

We would like to work in the following directions:

1. Figure out with Intel collaborators the reason why there is a clear very high point in evict-and-time attacks in SGX (Observation 1), and how to prevent this new cache side-channel attack on an SGX enclave.

2. Implement the prime-and-probe attacks on SGX.

3. How can we mitigate cache side channel attacks in SGX? Can we integrate Newcache and see if it can help to defeat cache side channel attacks when SGX enclaves are used?

4. (Future) Investigate modeling of caches and side channel attacks in Probabilistic Information Flow Graph (PIFG). We also would like to include SGX if possible.

5. (Future) What are other solutions to prevent nullifying the confidentiality provided by SGX enclaves due to cache side-channel attacks on SGX?

References

[1] C. Percival, “Cache missing for fun and profit,” in Proc. of BSDCan, 2005.

[2] Bernstein, Daniel J. "Cache-timing attacks on AES." (2005).

[3] Bonneau, Joseph, and Ilya Mironov. "Cache-collision timing attacks against AES." International Workshop on Cryptographic Hardware and Embedded Systems. Springer Berlin Heidelberg, 2006.

[4] Yarom, Yuval, and Katrina Falkner. "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack." USENIX Security. Vol. 2014. 2014.

[5] Liu, Fangfei, et al. "Last-level cache side-channel attacks are practical." Security and Privacy (SP), 2015 IEEE Symposium on. IEEE, 2015.

[6] Hoekstra, Matthew, et al. "Using innovative instructions to create trustworthy software solutions." HASP@ ISCA. 2013.

[7] McKeen, Frank, et al. "Innovative instructions and software model for isolated execution." HASP@ ISCA. 2013.

[8] Anati, Ittai, et al. "Innovative technology for CPU based attestation and sealing." Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy. Vol. 13. 2013.

[9] Intel Software Guard Extensions (SGX) Programming Reference, https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[10] McKeen, Frank, et al. "Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave." Proceedings of the Hardware and Architectural Support for Security and Privacy 2016. ACM, 2016.

[11] Wang, Zhenghong, and Ruby B. Lee. "New cache designs for thwarting software cache-based side channel attacks." ACM SIGARCH Computer Architecture News. Vol. 35. No. 2. ACM, 2007.

[12] Liu, Fangfei, et al. "Newcache: Secure Cache Architecture Thwarting Cache Side-Channel Attacks." IEEE Micro 36.5 (2016): 8-16.

[13] Liu, Fangfei, and Ruby B. Lee. "Random fill cache architecture." Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on. IEEE, 2014.