Performance of Large Scale Data-Oriented Operations under the TEE Constraints

Robin Carpentier, Nicolas Anciaux, Iulian Sandu Popa, Guillaume Scerri

Performance of Large Scale Data-Oriented Operations under the TEE Constraints

Robin Carpentier[1, 2], Nicolas Anciaux[1, 2], Iulian Sandu Popa[1, 2], Guillaume Scerri[1, 2]

[1] INRIA Saclay-IDF
Palaiseau, France
<Fname.Lname>@inria.fr

[2] DAVID Laboratory
Univ. of Versailles, France
<Fname.Lname>@uvsq.fr

1. INTRODUCTION

Trusted execution environments (TEE), like Intel SGX [3] or ARM TrustZone [1], are now embedded in end-users' personal computers and in cloud servers. TEEs offer an interesting set of security primitives, ranging from *isolation* from the host OS, *confidentiality* and *integrity* of the executed code, to *attestation* capabilities, all of this being combined with the efficiency of modern CPUs.

With such promises, TEEs obviously raise new hopes towards highly secure and efficient data management, as attested by the several recent contributions in the data management community to leverage these security properties in database processing.

EnclaveDB [8] and HardIDX [4] are examples of database works which leverage the SGX security properties to secure database operations. EnclaveDB considers a transactional database running inside an SGX enclave. EnclaveDB investigates the case of using Intel SGX to host an existing database engine (typically, the Hekaton engine) and thus convert it into a secure DBMS. HardIDX identifies the critical security operation subset of a search algorithm to be implemented within an SGX enclave to reach the same security level as the best searchable encryption schemes, but with much better performance. Such proposals mainly consider SGX as a secure black box, and identify the minimal codebase of existing database solutions to be included in SGX to improve the security level for the entire database system.

In this context, we follow a rather complementary approach consisting in opening the black box and studying the impact of Intel SGX specificities on the design of the underlying database structures and algorithms. The main limitations include the cryptographic overhead of accessing persistent data outside the TEE enclave [2], the limited RAM amount of each TEE enclave [3], the cost of external function calls [10] and memory access overheads, which can slow the computing performance by orders of magnitude compared to a regular environment, and have to be taken into account.

More precisely, our ongoing work focuses on (i) identifying the specific constraints of trusted execution environments (in particular Intel SGX) and understand how side channel attacks (timing, memory access patterns) affect classical data processing; and (ii) proposing appropriate design rules for data structures, algorithms and countermeasures, for efficient, scalable and secure personal data management using secure enclaves.

The rest of the paper is organized as follows: Section 2 introduces the main properties of Intel SGX, Section 3 gives a preliminary overview of the potential impact of these properties on database processing and the consequences on the underlying data structure and algorithms, and Section 4 sketches the outline of our ongoing work.

2. INTEL SGX PROPERTIES

Intel SGX [3] offers a mechanism for isolating processes from the rest of the system in *enclaves*. The security goals of SGX are twofold. First, under the assumption that the CPU is not compromised, programs running in enclaves cannot be observed nor affected by the rest of the system (including other programs in enclaves, and the untrusted OS). Second, enclaves should be able to prove their identity and the integrity of their code to third parties through a process called attestation. Our main concern in this work is the impact of SGX security on elementary data processing operations. However, the memory protections of enclaves have strong impact on memory management, as does the interaction of enclaves with the rest of the system. We give here an overview of the impact SGX has on code execution.

Note that the memory is protected from the rest of the system (including other enclaves) through cryptographic means, i.e., the memory belonging to an SGX enclave needs to be encrypted/decrypted before being usable. This process has a relatively low impact on program execution with respect to the more classical context. Indeed, the encryption is performed online by a dedicated hardware module and does not incur a big overhead.

A more important restriction is that enclave memory is set once and for all at enclave initialization. Additionally, enclave memory is restricted to 90MB currently. Therefore, if an enclave ever needs more memory than its initially allocated memory, it needs to store its additional memory outside of its dedicated part of the RAM. This can be done through encrypting a memory block using a key known only by the enclave and passing the page back to the OS encrypted. However, this process is much costlier than a typical RAM access or RAM copying. To some extent it is similar to running out of memory and paging memory to disk in more classical setups. In order to obtain efficient algorithms, this should be avoided or, at least, carefully controlled.

Finally, communication between an enclave and the rest of the system needs to be considered quite carefully. First, context switching between enclave mode and non-enclave mode in the CPU incurs additional cost compared to a normal context switch as the CPU has to perform a significant amount of bookkeeping during this context switch. Second, passing data between an enclave and the rest of the system requires such a context switch, and additionally the encryption/decryption of said data for the enclave, which are both non negligible costs.

3. IMPACT ON DATABASE RELATED PROCESSING

At first, we focus on benchmarking elementary database operations. While other works [8] require loading the entire data system within the RAM allocated to an enclave, we consider this approach as unrealistic in the general case since this requires allocating up to hundreds of GB of RAM to an enclave. Since the RAM is statically allocated to an enclave, such solutions are too costly in practice.

For instance, it would not be acceptable (neither in the cloud nor home machine contexts) to permanently allocate large quantities of RAM (which cannot be easily reclaimed) to an enclave belonging to a specific user. Indeed, in the cloud context statically allocating amounts of RAM to mostly offline users is unrealistic as it would require much more RAM than effectively needed to run the system in an unsecure context.

For a home computer this assumption is even more unrealistic as user cannot effectively block large amounts of memory for a personal data management system. An alternative would be to page the enclave memory to external memory managed by the OS. However, as mentioned above, this would break optimizations made to data processing algorithm due to the high cost of paging. Therefore, this calls for data processing techniques where private data mostly resides encrypted outside the enclave and is consumed by the enclave on request within the limit of the enclave RAM.

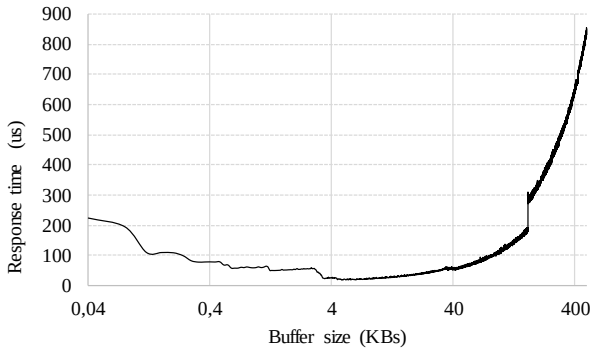

Figure 1. Binary search in a sorted list.

Hence, our first performance evaluation consists in measuring the data transfer cost in between the enclave RAM to the untrusted system RAM and conversely. We measured both the cost of importing/exporting large chunks of data at once, and the cost of importing/exporting large amounts of small pieces of data. The additional cost compared to classical RAM access comes from two main factors: the cost of context switching to the untrusted mode in order to request data, and the cost of encrypting the data for the enclave RAM. We observed that importing large chunks of memory at once is much faster than importing a large number of small pieces of memory summing up to the same size. Also, as soon as the size of the data imported is larger than 90MB (i.e., the maximum RAM enclave size in our test system), the overhead due to paging (managed automatically by the Linux SGX driver) is clearly visible.

But an important question is in which way the above noted constraints can impact a data-intensive processing. Let us consider the basic example of item search in a large sorted list in RAM. In the classical context, a binary search on the list offers the best performance. In the SGX context, each access to a list item from the enclave requires a context switch to load the item from the unprotected RAM, which is costly. Alternatively, the enclave could load multiple list items at each access (e.g., every 10^{th} element in the list), which would proportionally reduce the number of context switches but increase the data access cost. Figure 1 shows the average time search in a sorted list with increasing access buffer size. The left-most point corresponds to a buffer size of one item (i.e., a binary search). We can see that the optimum buffer size is around 4KB, which leads to search times one order of magnitude smaller than the binary search. On the other hand, increasing further the buffer size leads to increasing the search cost, especially if the buffer size exceeds the enclave RAM size (see right-most part of the graph).

4. FUTURE WORKS

The results presented here cover the processing efficiency within SGX enclaves, and derive design principles for SGX data processing. We aim at formalizing these insights and extend our results to a more extensive set of data processing algorithms. The next essential step is considering the security issues related to side channel attacks both on SGX and on the access patterns on external data.

First, vulnerabilities and side channel attacks exist on SGX. The most significant one being the recent Foreshadow [9] attack. Mostly, these attacks rely on out of order execution in a way somewhat similar to Spectre [5] and Meltdown [6]. The mitigation techniques for these attacks typically rely on forcing a L1 cache flush on each enclave enter/exit. An important study would be to examine how these cache flushes influence performance. This study will make our analysis somewhat robust to future versions of SGX which will include these mitigations. Additionally, one should consider timing attacks on the algorithms used for data management, as timing of enclave computations can be easily observed from the untrusted OS. In particular, we plan to study the dependency between sensitive data and computation time in classical data-oriented algorithms.

Second, an important leakage source resides in the interactions between the untrusted OS and the data processing enclave. For example, in our search example the OS gets the position of the searched element in the list from the position of the queried elements. Mitigations for this kind of leakage is an important research direction, as well as the impact of such mitigations on performance and algorithm designs.

5. REFERENCES

[1] Alves, Tiago, and Don Felton. TrustZone: Integrated Hardware and Software Security-Enabling Trusted Computing in Embedded Systems. 2004.

[2] Brenner, Stefan, et al. SecureKeeper: Confidential ZooKeeper using Intel SGX. Middleware, 2016.

[3] Costan, Victor and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016.086 (2016): 1-118.

[4] Fuhry, Benny, et al. HardIDX: Practical and secure index with SGX. *IFIP Annual Conference on Data and Applications Security and Privacy*, 2017.

[5] Kocher, Paul, et al. Spectre attacks: Exploiting speculative execution. *S&P*, 2019.

[6] Lipp, Moritz, et al. Meltdown: Reading Kernel Memory from User Space. *USENIX Security Symposium*, 2018.

[7] McKeen, Frank, et al. Innovative instructions and software model for isolated execution. *HASP@ ISCA 10*, 2013.

[8] Priebe, Christian, Kapil Vaswani, and Manuel Costa. EnclaveDB: A Secure Database using SGX. EnclaveDB: A Secure Database using SGX. *IEEE*, 2018.

[9] Van Bulck, Jo, et al. FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. *USENIX Security Symposium*, 2018.

[10] Zhao, ChongChong, et al. On the Performance of Intel SGX. *IEEE Web Inf. Systems and Applications Conf.*, 2016.