# Hardware Security: Intel SGX and VC3

Scribe: Andrew Low

September 2018

## 1 Introduction

The rise of cloud computing has driven demand for a way to securely execute programs with confidential data on untrusted, remote machines. Under traditional systems, a user would send programs and confidential data to remote machines for execution. However, this requires the user to trust multiple components of the remote machine, including the operating system and memory storage. Any attacks that compromise the remote machine would also compromise the user's security. The user's program and data would thus be vulnerable to a wide variety of attacks, including OS rootkits, privilege escalation attacks, and compromised hardware devices. Several approaches such as fully homomorphic encryption (FHE) and multi-party computation enable trusted program execution on untrusted platforms and achieve the desired security properties, but are often too slow in practice to scale for many applications.

Hardware enclaves are hardware-backed container abstractions that enable trusted execution on untrusted platforms. Hardware enclaves can guarantee secure, verifiable program execution over confidential data with little performance overhead compared to native computation.[2] By using secure enclaves, the user only has to trust their code, the processor, and Intel's Attestation Service, which greatly reduces the size of the trusted computing base (TCB). Intel SGX is one enclave technology that enables users to instantiate secure enclaves using specialized Intel processors.

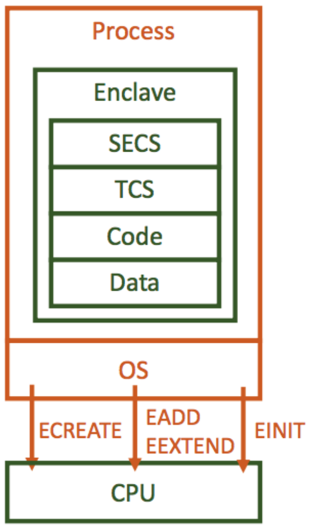


Figure 1: Enclave Design based on [3]

## 2 What is a Hardware Enclave?

Fundamentally, a hardware enclave is a protected region of an application's address space that has its own stack, heap, and code regions [Figure 1]. The processor enforces the confidentiality and integrity of the enclave by preventing anything but the enclave process from accessing virtual addresses within the enclave. This affords protection from not only other processes, but also the operating system itself. Hardware enclaves provide the following properties to guarantee confidentiality and integrity:

1. Secure boot
   - The system hardware verifies (measures) the program before it is run to ensure code and data integrity. The first instruction is also static to protect against timing/selective execution attacks
2. On-chip program isolation
   - The CPU's registers and memory cannot be read by any external parties while the enclave program is running on the CPU chip

3. Cryptographically protected external memory
    - Any enclave program data that is stored outside the enclave must be encrypted with integrity checks.
4. Execution integrity
    - External parties cannot change the control flow or execution of enclave programs
5. Attestation and secret sealing
    - A program must be able to prove that it is running on a genuine enclave that has been set up correctly.

## 3 Threat Model

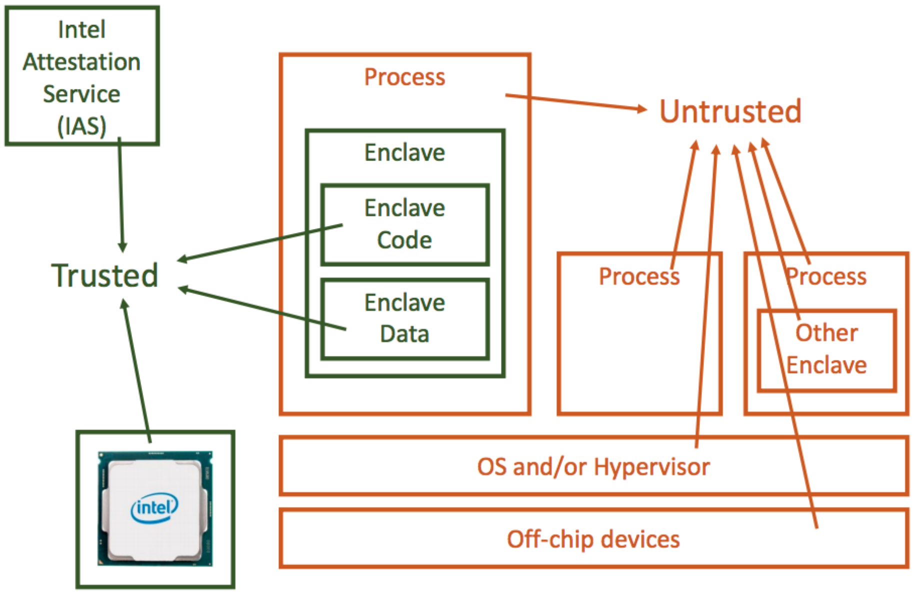


Figure 2: Threat Model based on [3]

Hardware enclaves operate under a threat model in which the attacker can compromise almost the entire software stack, including the operating system, hypervisor, and other processes running on the same machine. The trusted computing base consists of the CPU, the code and data running in the enclave, and the Intel Attestation Service (IAS) [Figure 3].

## 4 Creating and Using an Enclave

Enclave creation in Intel SGX is a multi-step process, involving initialization of the enclave control structure, allocating protected memory for the enclave, loading and measuring content into the protected memory, and establishing the enclave identity. The relevant instructions needed for enclave creation are described in Table 1.

1. Initialization of enclave control structure
    - This initial step is triggered by the ECREATE instruction and creates the SGX Enclave Control Structure (SECS), which stores enclave metadata. Each enclave has its own SECS.

2. Allocate protected memory for the enclave
   - In this step, we allocate pages from the EPC to the enclave with the EADD instruction, and measure the loaded content with EEXTEND. EEXTEND measures 256 bytes and updates the enclave's measurement (MRENCLAVE), a cryptographic tool used to ensure enclave integrity.
3. Establishing Enclave Identity
   - Next, we formally initialize the enclave, which locks the enclave address range and allows the processor to enter the enclave. This is triggered via the EINIT instruction.
4. Entering the Enclave
   - At this point we have a fully functional enclave containing our program and data, We can enter the enclave to start execution with the EENTER instruction. Note that EENTER always enters at a static enclave address, to prevent attackers from interfering with the control flow. A separate instruction, ERESUME, allows the enclave to continue execution from the point at which it was last interrupted.
5. Exiting the Enclave
   - We can exit the enclave using the EEXIT instruction to exit to any accessible address. If the exit is unplanned, the processor uses the AEX instruction instead. In both cases, the processor flushes all enclave information from the registers to preserve enclave confidentiality.

Table 1: Enclave Instructions, based on Tables 3-1, 3-2 in [1]

| **Instruction** | **Description** |
|---|---|
| ECREATE | Declare base and range, start build |
| EADD | Add 4k page |
| EEXTEND | Measure 256 bytes |
| EINIT | Declare enclave built |
| EENTER | Enter enclave |
| EEXIT | Leave enclave |
| AEX | Asynchronous enclave exit |

### 4.1 Execution Integrity

Enclave execution can be interrupted arbitrarily by the Operating System. Interrupts and exceptions trigger an asynchronous exit (AEX) operation, which causes the hardware to save all secrets and state inside the enclave, erase secrets from the processor's registers, and return to the external program flow.[1] This ensures that no enclave secrets are leaked during interrupts or exceptions.

## 5 Enclave Memory Isolation

Intel SGX provides several mechanisms to isolate and protect sensitive data from attackers. There are three primary places where sensitive data can live: within the trusted processor, in memory (DRAM), and on other storage devices outside of memory such as the disk. Within the processor, which encompasses the registers and various caches, the data is unencrypted. In memory (DRAM), the sensitive data is encrypted and MAC-ed by the Memory Encryption Engine (MEE). On other storage devices outside of memory, sensitive data is also encrypted and MAC-ed, but is managed by the processor itself.

Table 2: Enclave Data Structures and Components, based on Table 3-6 in [1]

| **Structure** | **Description** |
|---|---|
| Enclave Page Cache (EPC) | Contains protected code and data in 4K pages |
| Enclave Page Cache Map (EPCM) | Contains meta-data of enclave page |
| SGX Enclave Control Store (SECS) | Meta data for each enclave |
| Thread Control Structure (TCS) | Meta data for each thread |
| VA Page | Version Array of evicted pages |

## 5.1 Protecting EPC Memory

All enclaves allocate their protected memory from a shared protected memory region called the Enclave Page Cache (EPC). The EPC is divided into 4Kb pages, which are allocated among the running enclaves. The EPC's page mappings are stored in the EPC Map (EPCM). Enclaves are isolated from the operating system as well as each other. Any attempt to read or write EPC memory by non-enclave threads returns an abort page value.

To protect information stored in the EPC, Intel SGX utilizes a Memory Encryption Engine (MEE), which sits between the CPU and the Memory Controller. When an EPC cache line is evicted from the cache to memory, the MEE encrypts and generates a MAC of the cache line and stores the encryption key and MAC. When the cache line is retrieved during a cache miss, the MEE decrypts the cache line and verifies its integrity using the stored MAC before transferring the cache line back to the CPU. The MEE stores the encryption keys and MACs in a hash tree data structure for efficient access.

## 5.2 Protecting Evicted Enclave Pages

Another issue arises when the EPC pages are swapped out of the EPC to main memory by the OS, where they are no longer protected by the MEE. During paging operations, data protection is managed by the processor directly with the special EWB and ELDU instructions.

The EWB instruction is used to evict an EPC page to a non-EPC page. When executing an EWB instruction, the processor securely encrypts and MACs the page before flushing it to the main memory buffer. Replay attacks are thwarted by assigning a unique version value to each page. The version numbers are stored in a special VA page in the EPC.

The ELDU and ELDB instructions are used to load a non-EPC page back into the EPC memory. When executing an ELDU or ELDB instruction, the processor essentially performs the reverse of the EWB instruction. The processor verifies the page's integrity with the MAC, decrypts the page, and updates the EPCM with the new mapping. Replay attacks are avoided by removing the VA page entry corresponding to the page.

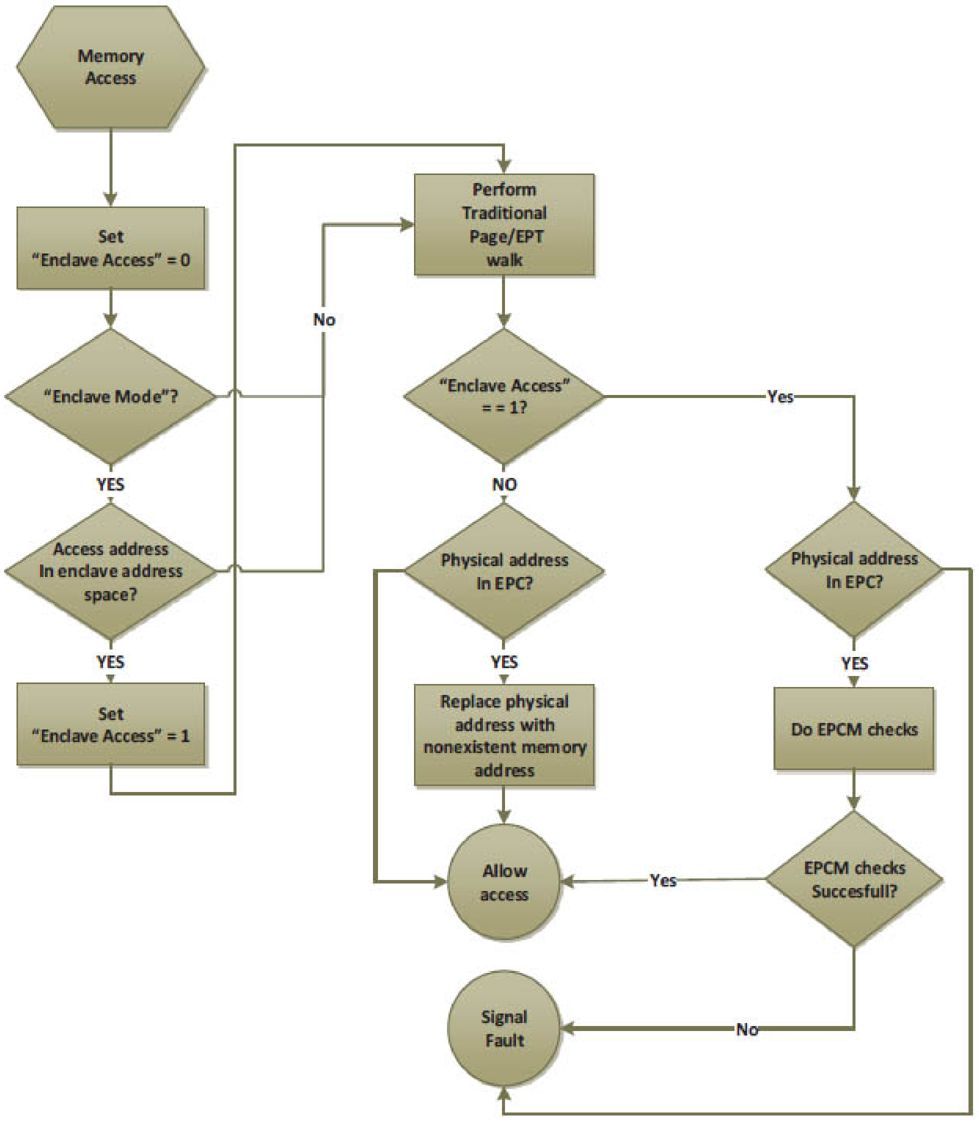


Figure 3: Memory Access Decision Graph. Source:[3]

As described in Section 5.2 above, EPC pages can only be accessed by enclave threads from the owning enclave. If a non-enclave thread or a thread from a different enclave attempts to access the page, the controller replaces the physical address with a nonexistent memory address to avoid memory leakage. If the requesting thread has the necessary permissions but attempts to access an invalid non-EPC address, a signal fault is triggered.

## 6 Attestation

Attestation is a mechanism to prove to a third party that a given program is running on a genuine enclave. This enables users to trust that remote machines are executing user programs correctly. Each enclave uses the processor's intrinsic root secret (EPID) to generate a set of unique keys:

Table 3: Enclave Keys, based on [3]

| **Key** | **Purpose** |
|---|---|
| Report key | Intra-platform (local) attestation |
| Attestation key | Inter-platform (remote) attestation |
| Seal key | Sealing enclave secrets |
| And more... | ... |

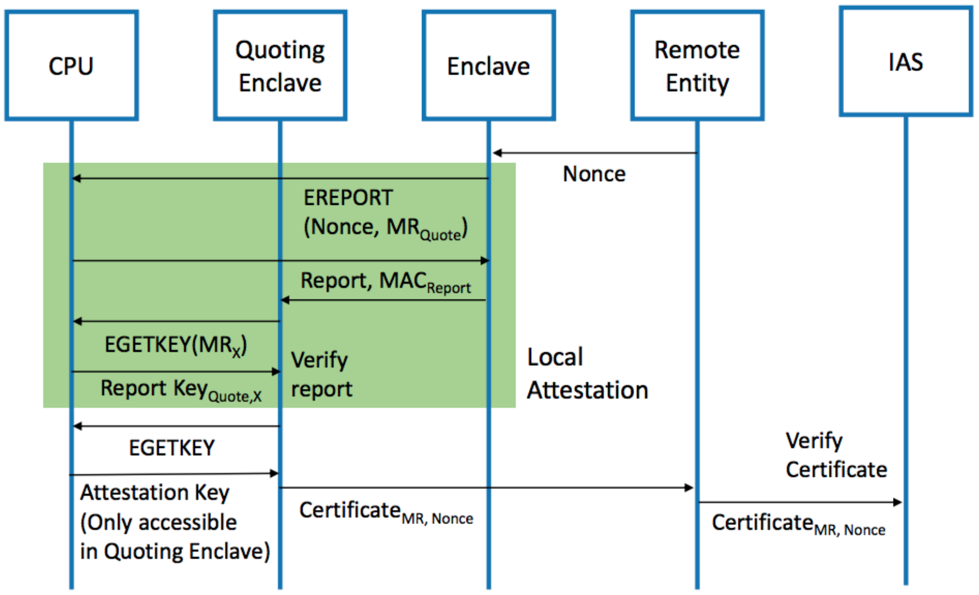


Figure 4: Remote Attestation [3]

Only the report key and attestation key are used during attestation. The basic attestation process works as follows:

1. A remote user sends a nonce to the enclave $E$. Note that the user does not trust the enclave yet.

2. The enclave $E$ must now prove to a special Intel-provided quoting enclave $Q$ that it is a real enclave. This process is also called local attestation.

   (a) Enclave $E$ asks the processor generate a report to certify $E$'s authenticity as a legitimate enclave.

   (b) Enclave $E$ then sends the report to the quoting enclave $Q$, which verifies the report.

3. At this point, the quoting enclave $Q$ now trusts enclave $E$. The quoting enclave $Q$ now retrieves the attestation key from the CPU and uses it to generate a certificate proving the legitimacy of enclave $E$. The certificate is sent to the remote user.

4. The remote user can verify the attestation by checking certificate using the Intel Attestation Service (IAS).

## 7 Applications

There are many potential applications for hardware enclaves, including Digital Rights Management (DRM), computation outsourcing, distributed systems and blockchains, and protection for antivirus programs or JIT compilers. [3] However, the security properties provided by hardware enclaves can also be abused. For instance, attackers can run programs within hardware enclaves to conceal their attacks.

## 8 VC3

VC3 [4] is a system that leverages Intel SGX and other protocols to enable users to securely run distributed, verifiable, MapReduce computations on distributed systems. MapReduce [5] is a widely-used programming model for processing and generating large datasets on computing clusters. The MapReduce framework consists of two functions: *Map*, which shards incoming key-value data across machines in the cluster for processing, and *Reduce*, which aggregates the outputs of the cluster machines.

Traditional MapReduce implementations do not protect the confidentiality of input data from cluster. VC3 addresses this issue by providing two security guarantees:

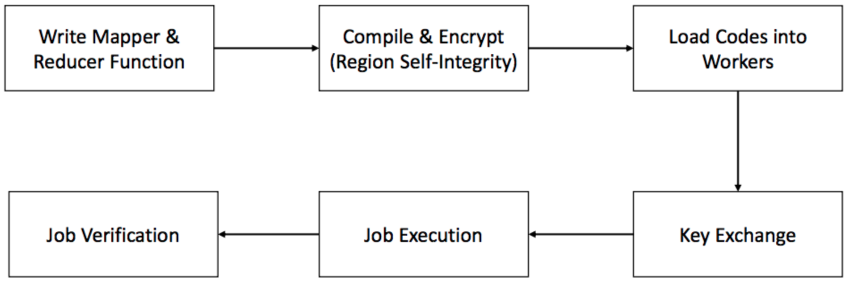
1. Confidentiality and integrity for code and data. [4]
2. Verifiability of execution of the code over data. [4]

### 8.1 Adversarial Model

VC3 operates under a threat model in which the adversary can control the entire software stack in a cloud provider's infrastructure [4]. The adversary can replay packets, block traffic, and access other jobs running on the same machine as the VC3 MapReduce job. However, the attacker cannot compromise SGX processors

### 8.2 Protocol Overview

To run a VC3 job, users craft standard Map and Reduce functions, which are then encrypted, linked to a standard VC3 public code snippet, and sent to the cloud. As show in Figure 5, once in the cloud, the public code snippet runs an key-exchange protocol with the user to authenticate the cloud enclave. The key exchange protocol is also used to generate several keys for protecting the input, output, and intermediate values generated by the MapReduce computation. With the keys, the cloud enclaves are able to decrypt the MapReduce code and data and execute the job securely. Each cloud node involved in the computation also generates verifiable summaries of the executed job and verifies the summaries of other nodes.


Write Mapper & Reducer Function
Compile & Encrypt (Region Self-Integrity)
Load Codes into Workers
Key Exchange
Job Execution
Job Verification
Figure 5: VC3 Workflow Design

VC3 relies on SGX to maintain confidentiality, integrity, and verifiability on individual machines [4]. However, guaranteeing integrity over the entire distributed computation is a more complex challenge. VC3 introduces a global job execution protocol in which individual computing nodes generate verifiable summaries of the work they have performed. Each node aggregates the summaries received from peers, and the protocol repeats this aggregation on several levels to generate a final summary that the user can check to verify the correctness of the entire computation[Figure 6]. VC3 runs on unmodified Hadoop. To minimize the size of the TCB, VC3 excludes Hadoop, the operating system, and the hypervisor from the TCB.

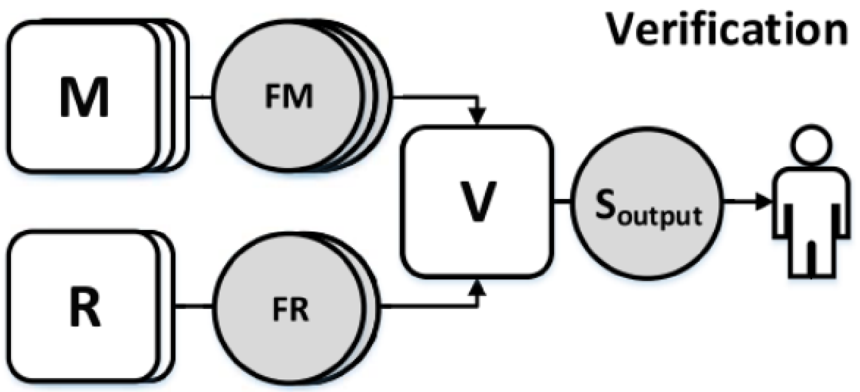


Figure 6: Conceptual VC3 Design

In Figure 6, the M's represent mapper nodes and R's represent reducers. Each produces a job verification summary FM and FR, respectively. These summaries are then used to verify the final ouput of the MapReduce computation.

## 8.3 Performance

In practice, VC3 performs similarly compared to native Hadoop, and is able to provide several security guarantees while only introducing marginal performance overhead of 4 to 8 percent. VC3 is more efficient traditional MPC computation protocols because VC3 is optimized for the MapReduce protocol specifically.

## References

[1] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos Rozas, Hisham Shafi, Vedvyas Shanbhogue and Uday Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. https://inst.eecs.berkeley.edu/~cs261/fa18/readings/sgx.pdf

[2] Keystone, an Open Source Secure Hardware Enclave. https://keystone-enclave.org/.

[3] Hardware Enclaves and Intel SGX. https://inst.eecs.berkeley.edu/~cs261/fa18/slides/Hardware_Enclaves.pdf.

[4] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, Mark Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/vc3-oakland2015.pdf

[5] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters https://ai.google/research/pubs/pub62