# Haven

Scribes: Nathan & Yi

November 5, 2015

**Abstract**

This week, we continue our theme of trusted hardware by discussing Haven, one of the first applications of Intel's SGX technology (and the best paper at OSDI 2014). Haven allows client applications to run using a cloud service's resources, while preventing the cloud provider from seeing, or tampering with, the data.

**Goal:** We want to outsource applications to the cloud without the cloud provider being able to see or change code or data — it just provides processor cycles, storage, and networking. The result of running the application should be the same as if the program ran on a local cluster.

**Threat model:**

- All cloud software is compromised (the entire stack, including the operating system).
- The adversary is assumed to be an active attacker.
- But the processor has not been tampered with.
- Also trusted: Intel (as the manufacturer) and its keys.

**Out of scope:** Denial of service is still possible.

**Shielded execution:** The paper introduces the concept of shielded execution. It provides the following properties:

1. *confidentiality:* the client's application is a black box to the server; the provider only sees inputs and outputs.[1]
2. *integrity:* results of the computation on the server are always the same, as if the application was running on a local cluster.

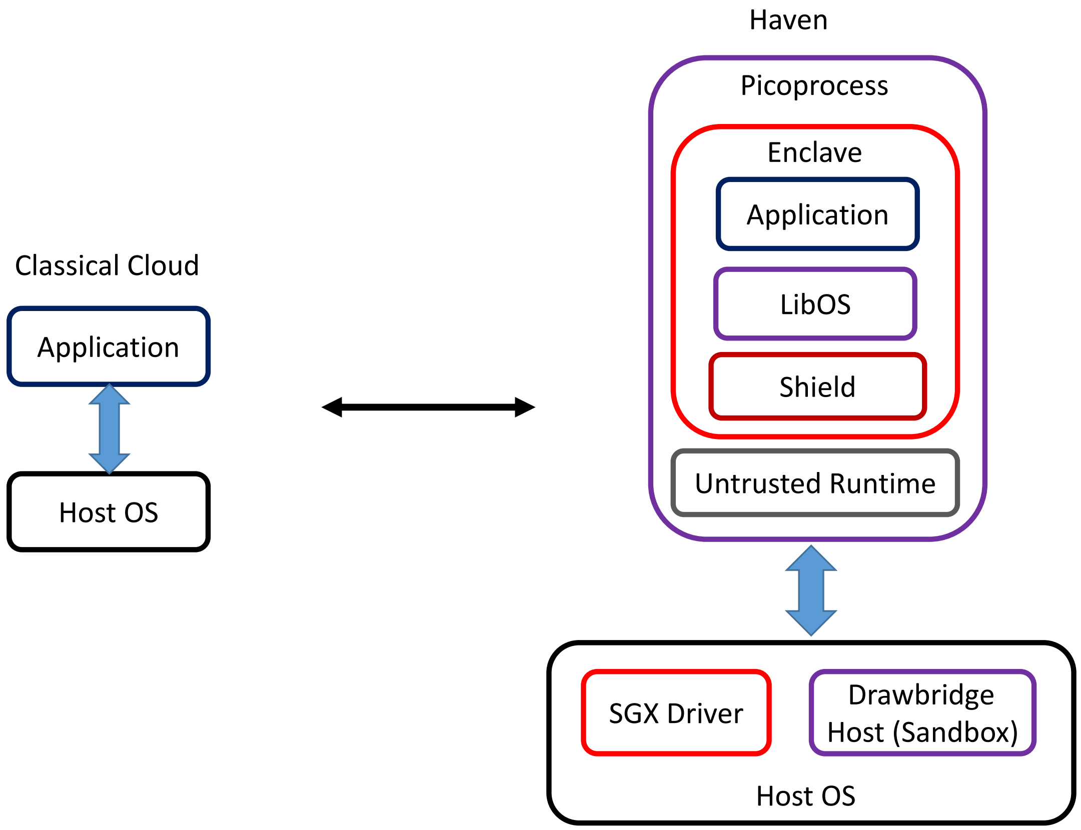
Shielded execution can be thought of as the inverse of sandboxing, which protects the system from an untrusted application.

[1]To avoid input/output network leakage to the cloud provider, you can use a proxy server and talk to it using SSL.

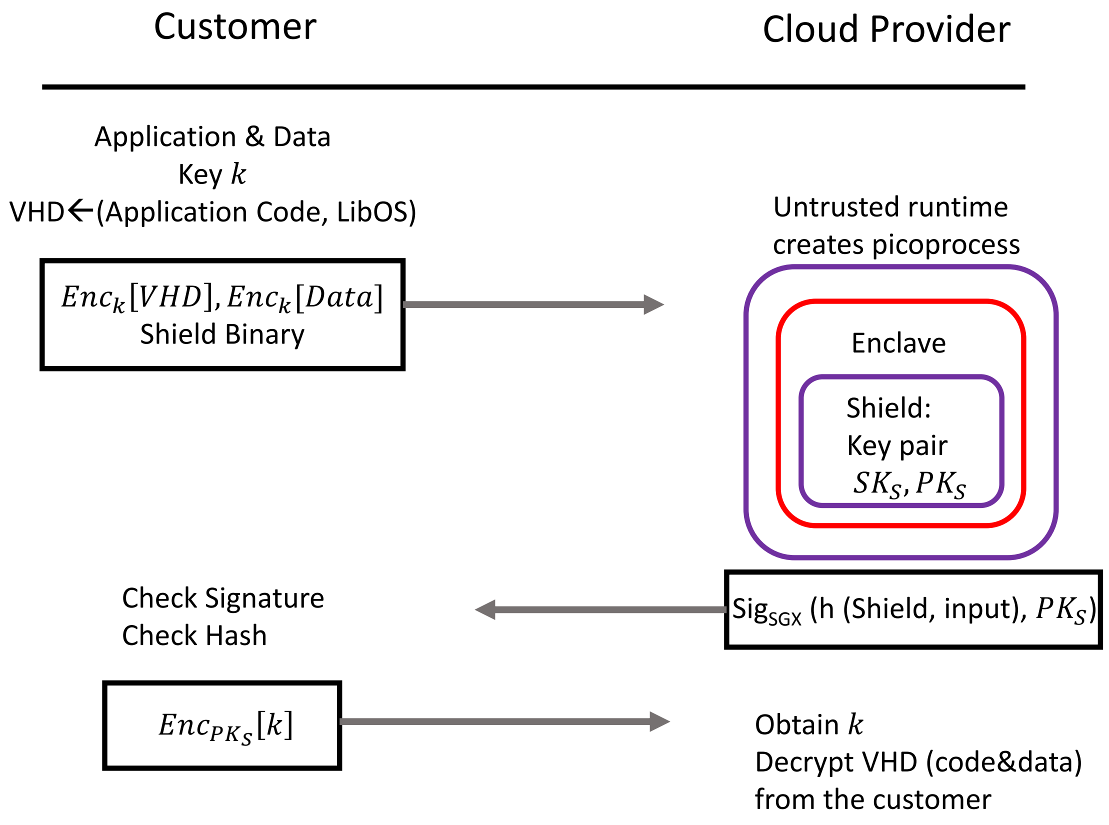
**Challenge:** Haven's specific challenge is to support unmodified binaries, to avoid the need to rewrite or recompile existing server applications.

**Strawman solution:** We could take an existing binary and run it, unmodified, inside SGX. The problem with this approach is that the application, as written, will make system calls and, when they return, assume that the results can be trusted.

**Key idea:** Haven's insight is that this problem can be solved by putting the syscall handler inside the secure enclave. In Haven, the handler is called LibOS; it will handle syscalls inside the enclave. To make sure it's trusted, the LibOS implementation is supplied by the customer.



**Attestation:** While code can run securely inside the SGX enclave, getting it there requires interaction with the untrusted operating system. To prevent it from tampering, or seeing the data, Haven securely loads the data with a process that depends on the SGX attestation feature.


Customer
Cloud Provider
Application & Data
Key $k$
VHD←(Application Code, LibOS)
$Enc_k[VHD], Enc_k[Data]$
Shield Binary
Untrusted runtime creates picoprocess
Enclave
Shield:
Key pair
$SK_S, PK_S$
Check Signature
Check Hash
Sig$_{SGX}$ (h (Shield, input), $PK_S$)
$Enc_{PK_S}[k]$
Obtain $k$
Decrypt VHD (code&data) from the customer

Customer

1. Starts out with the application code and the data they want to run it with
2. Generates key $k$
3. Uses $k$ to encrypt the code and data: $Enc_k$[VHD = {app code + LibOS}], $Enc_k$[input data]
4. Sends this to the cloud
5. Also sends shield binary (this doesn't have to be encrypted)

Cloud

1. Spawns a picoprocess to isolate this task
2. Creates an enclave, puts the shield inside
3. The shield generates a private & public keypair

4. Generates "measurement": a hash of the shield & its inputs, as well as the public key it generated.[2]
5. All of this is signed by an SGX key and sent back to the client

Customer

1. Checks the message signature: SGX's public key is certified by Intel
2. Checks hash
3. Uses shield's public key to encrypt $k$
4. Sends the encryption back to the server

Shield

1. Obtains $k$
2. Decrypts VHD code & data from customer
3. Begins execution!

## Presentations

### TrInc, the trusted incrementor[3]

In distributed systems, a big potential problem is equivocation: a malicious server can tell different clients different things. This problem is solvable; however, if you assume no machine can be trusted, the solution requires many extra machines. Fewer are required if you have trusted hardware. This paper seeks to find the minimum level of trust required. The solution is TrInc, a small trusted hardware module.

> "To gain the benefits of TrInc, a user must attach a trusted piece of hardware we call a trinket to his computer."

TrInc operates a secure counter that is guaranteed to only ever increment. It is used to calculate hashes, ensuring that it will only sign a single hash once. In doing so, TrInc prevents fork attacks.

### BitLocker, Windows disk encryption

The motivation is laptop loss, which happens at a rate of 1-2% each year. The traditional solution is to encrypt the computer's disk with a passphrase or token; however, users tend not to do this because of the extra hassle. BitLocker does this transparently by using a trusted platform module (TPM), a security chip on the motherboard. The chip is tamper resistant and provides crypto primitives & key management. Here's how it works:

[2] You can ask for specific values or metadata to be included in the attestation.
[3] NSDI 2009 best paper

"The TPM keeps several Platform Configuration Registers, or PCRs. At power-up the PCRs are set to zero. PCRs are only modified by the extend function which (effectively) sets a PCR to the hash of its old value and a supplied data string."

"During the boot process the PCRs are used to keep track of the code that runs. The key used to encrypt the disk is sealed against a particular set of PCR values. During a normal boot the PCRs reach the same values, and the key can be unsealed by the TPM. If an attacker boots into any other operating system, the machine will be fully functional but the PCR values will be different and the TPM will not unseal the key. Thus, other operating systems cannot read the data on the disk, or find out how to modify the disk to reset the Administrator password."