CS 350S: Privacy-Preserving Systems

Secure Hardware

Outline

1. Background on secure hardware
2. VC3 paper
3. Logistics
4. Student presentation on controlled-channel attacks

Secure hardware

- High-level goal: provide hardware-level protections against an attacker that has access to the machine
- Properties hardware might try to provide against such an attacker:
  - Confidentiality: Attacker cannot learn device secrets
  - Integrity: Attacker cannot tamper with device state
  - Attestation: Client can confirm that it is interacting with “real” (i.e., manufacturer-certified) secure hardware
- Different types: HSMs, TPMs, enclaves, confidential VMs

Hardware security modules (HSMs)

- HSMs manage cryptographic keys
- Typically limited API for key generation, signing, encryption, and attestation
- Cryptographic key should never leave the HSM

https://www.ssl2buy.com/wiki/what-is-hardware-security-module

Hardware security modules (HSMs)

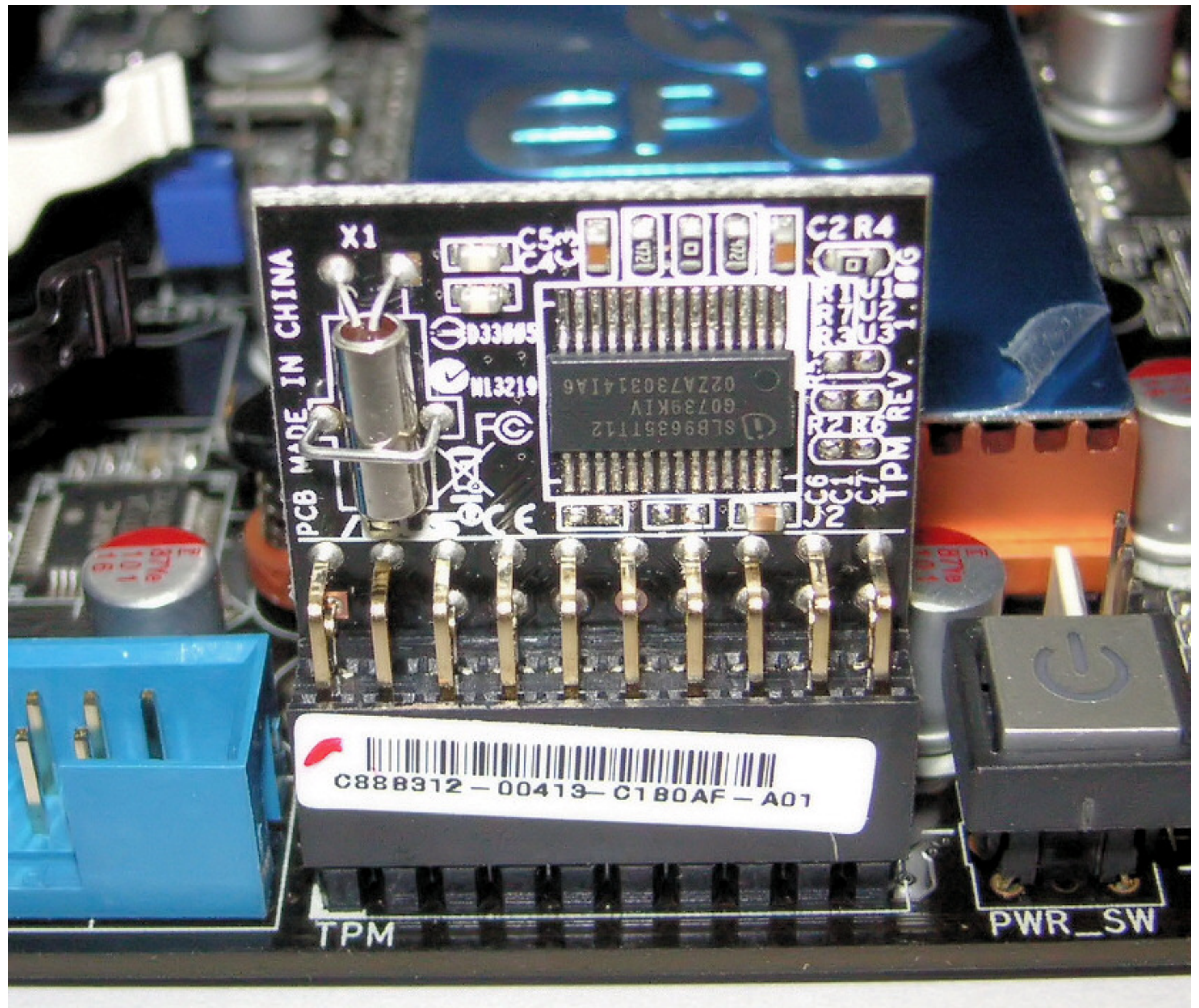
- Limited API limits the attack surface — does not run arbitrary external code
- “Hardened” against physical attacks: difficult (but not impossible) to extract secrets from HSMs given physical access
  - FIPS 140-3 certification levels can require hardware to resist physical attacks (e.g., temperature, voltage, fault-injection)
  - When hardware detects an attack, it can erase its secret state

HSM use cases?

- Storing certificate authority keys
- Signing software binaries
- Encrypted backups
- Financial transactions

Trusted Platform Modules (TPMs)

- Whereas HSM is a standalone device, TPM is a secure cryptoprocessor that provides cryptographic support to rest of the system
- Like HSMs: manages cryptographic keys via a limited API and resists physical tampering
- TPM manages secure boot process: ensures that correct code starts when a machine boots up
- TPM can also remotely attest to the code that is running on the machine (i.e., attesting to platform integrity), manage full disk encryption, and more

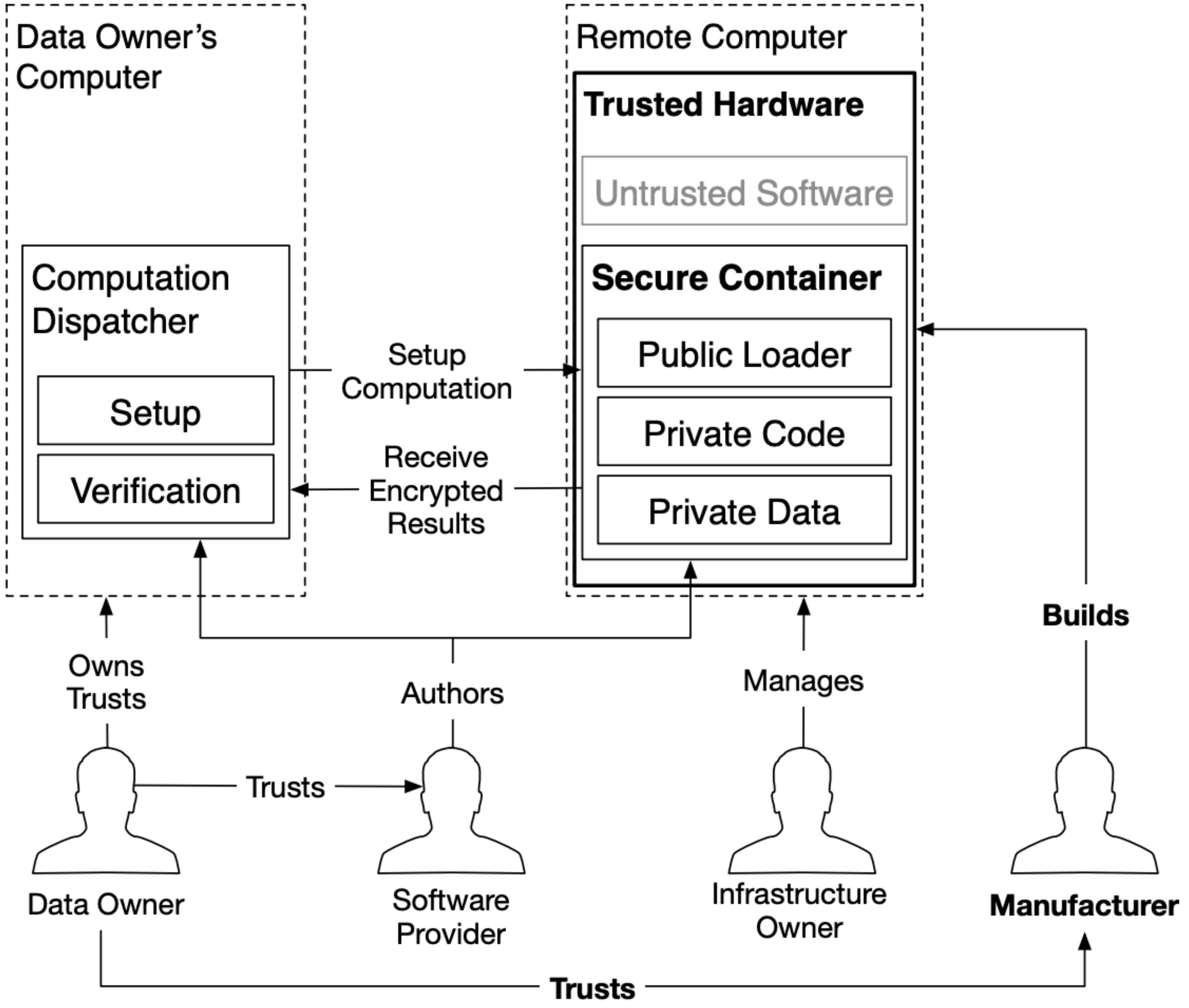


https://commons.wikimedia.org/wiki/File:TPM_Asus.jpg

Secure enclaves

- Execute arbitrary code in an environment “protected” from malicious OS
- Idea: no process on the same machine can view or modify the application running inside the enclave, *regardless of privilege level*
- EX: Intel SGX, ARM TrustZone, …
- Secure Enclave built into iPhones

Intel SGX workflow



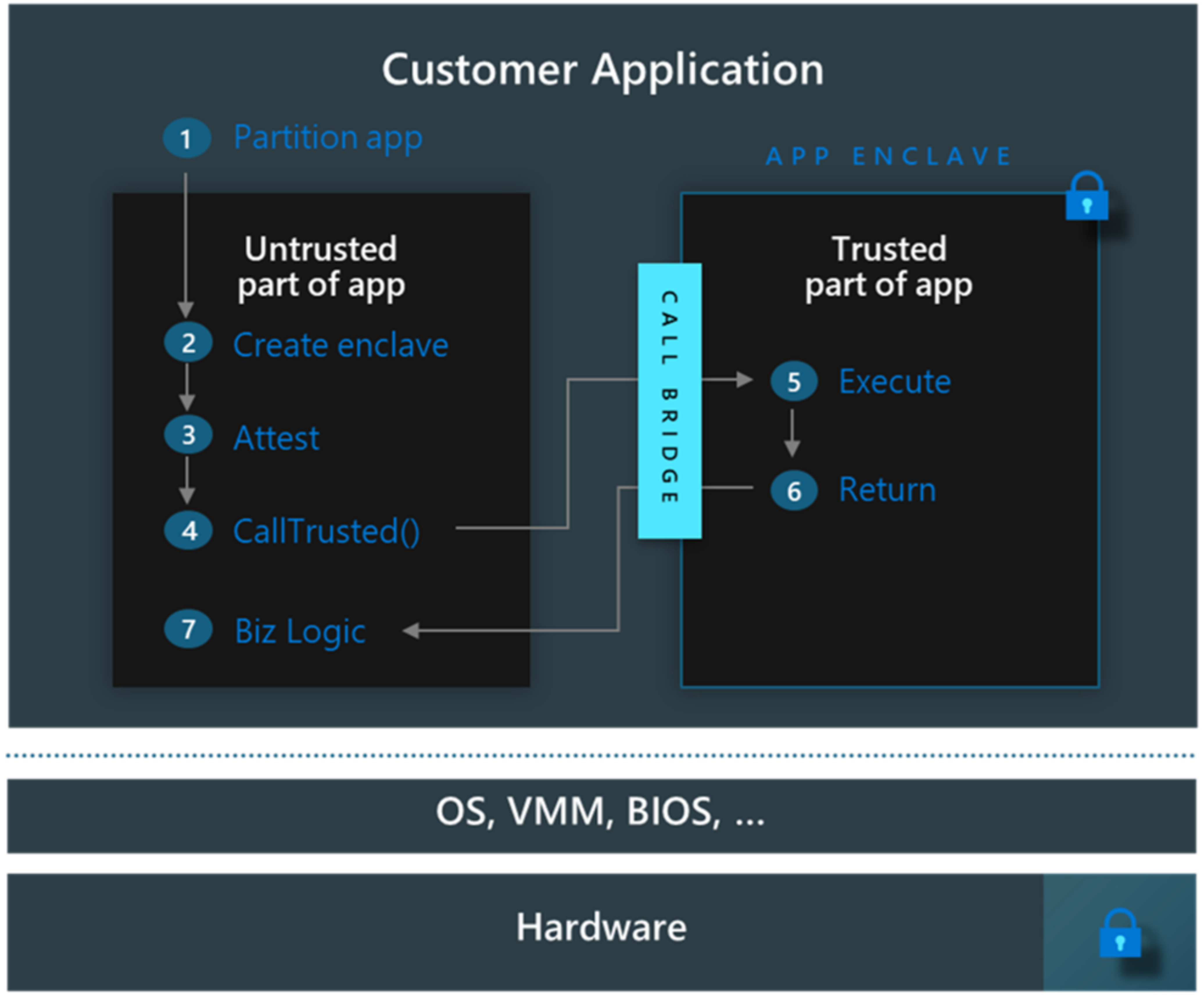
Victor Costan and Srini Devadas. *Intel SGX Explained.*

Intel SGX security properties

- High-level security properties against an attacker with root privileges:
  - **Confidentiality:** Attacker should not be able to view enclave memory
  - **Integrity:** Attacker should not be able to modify enclave memory
  - **Attestation:** An enclave will only attest to the code that is running
    - An attacker not running on an enclave cannot generate a valid enclave attestation
    - An attacker running inside the enclave cannot convince the enclave to attest to code that is different than the code running

Writing an application for Intel SGX

- Partition the application into two pieces
  - Component that does not need SGX protections and runs outside enclave (does not handle sensitive data or make security-critical decisions)
  - Component that needs SGX protections and runs inside enclave (handles sensitive data or makes security-critical decisions)
- Code inside enclave is part of trusted computing base (TCB)



https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-computing-deployment-models

Secure enclave use cases?

- Secure map-reduce (VC3)
- Medical databases (attacker cannot learn medical records)
- Secure training (attacker cannot learn training data)
- Secure inference (attacker cannot learn user input)
- …

Attacks on Intel SGX

- Remote attacker cannot learn data contents, but can view memory access patterns
  - Student presentation on controlled-channel attacks
- Given physical access to an SGX enclave, attacker can break SGX guarantees
  - Original goal of SGX was to protect against the hardware owner
  - In reality, need some confidence in physical security to have confidence in SGX guarantees

FORESHADOW

CacheOut
Leaking Data on Intel CPUs via Cache Evictions

SGAxe
How SGX Fails in Practice

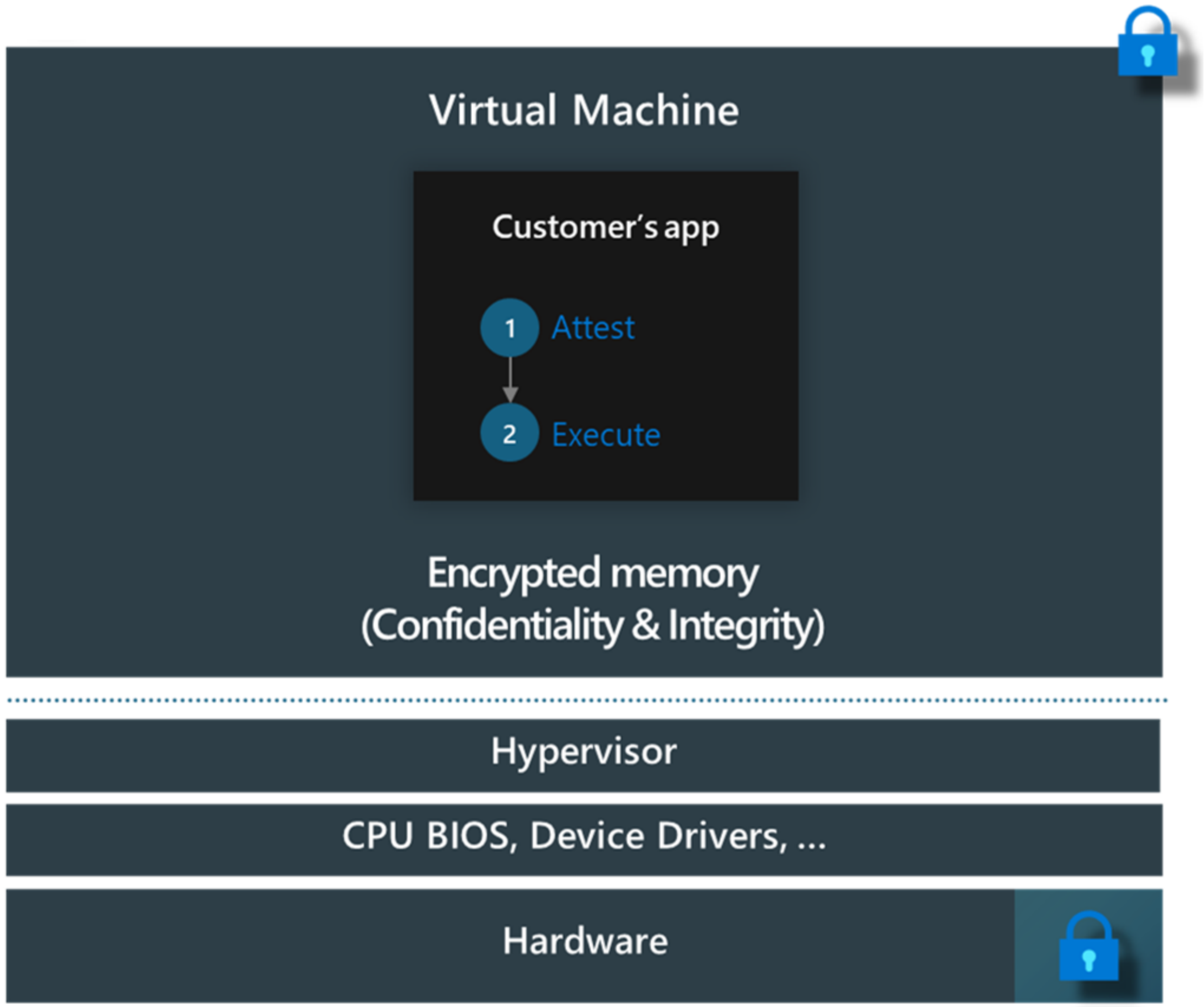
ZombieLoad Attack
Return of the Leaking Dead

PLUNDER VOLT

. . .

Confidential VMs

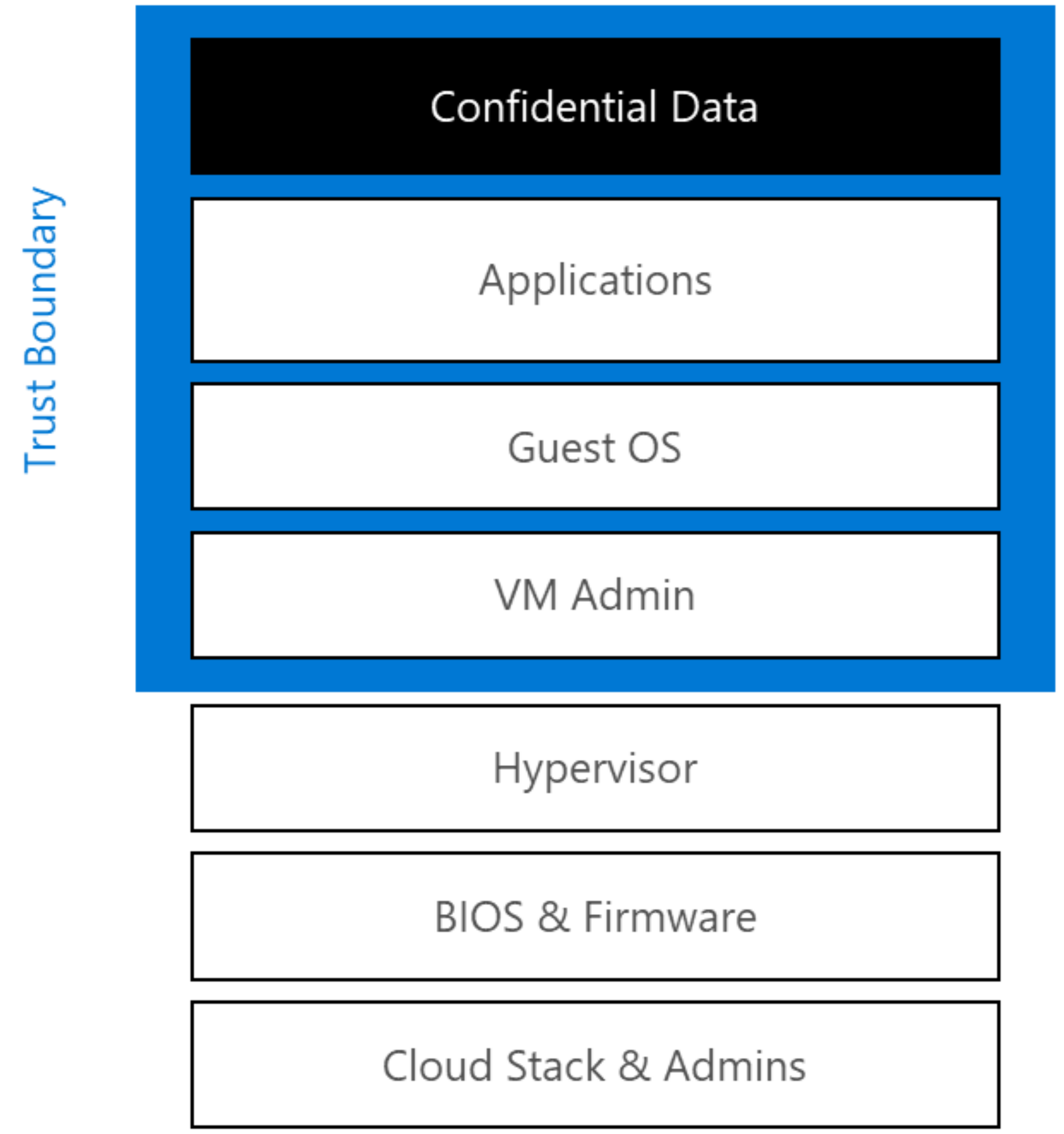
- Provides protections for a virtual machine against an attacker that controls the hypervisor
- Advantage: virtual machines are a simple abstraction to deploy
- EX: Intel TDX, AMD-SEV SNP



https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-computing-deployment-models

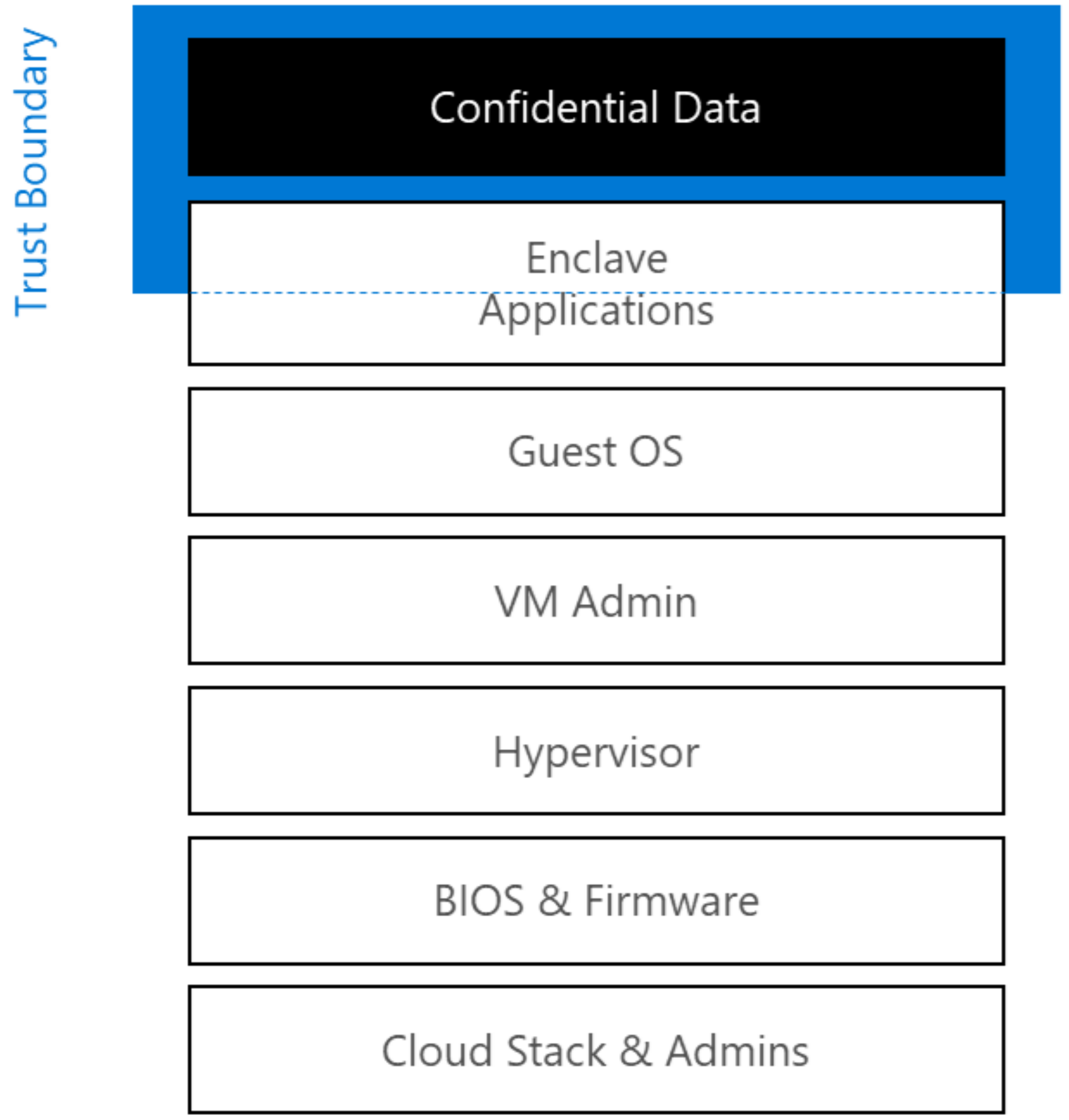
Enclaves vs confidential VMs

VM Isolation



Most straightforward path for legacy apps

App Isolation



Smallest trust boundary

https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-computing-deployment-models

Outline

1. Background on secure hardware
2. **VC3 paper**
3. Logistics
4. Student presentation on controlled-channel attacks

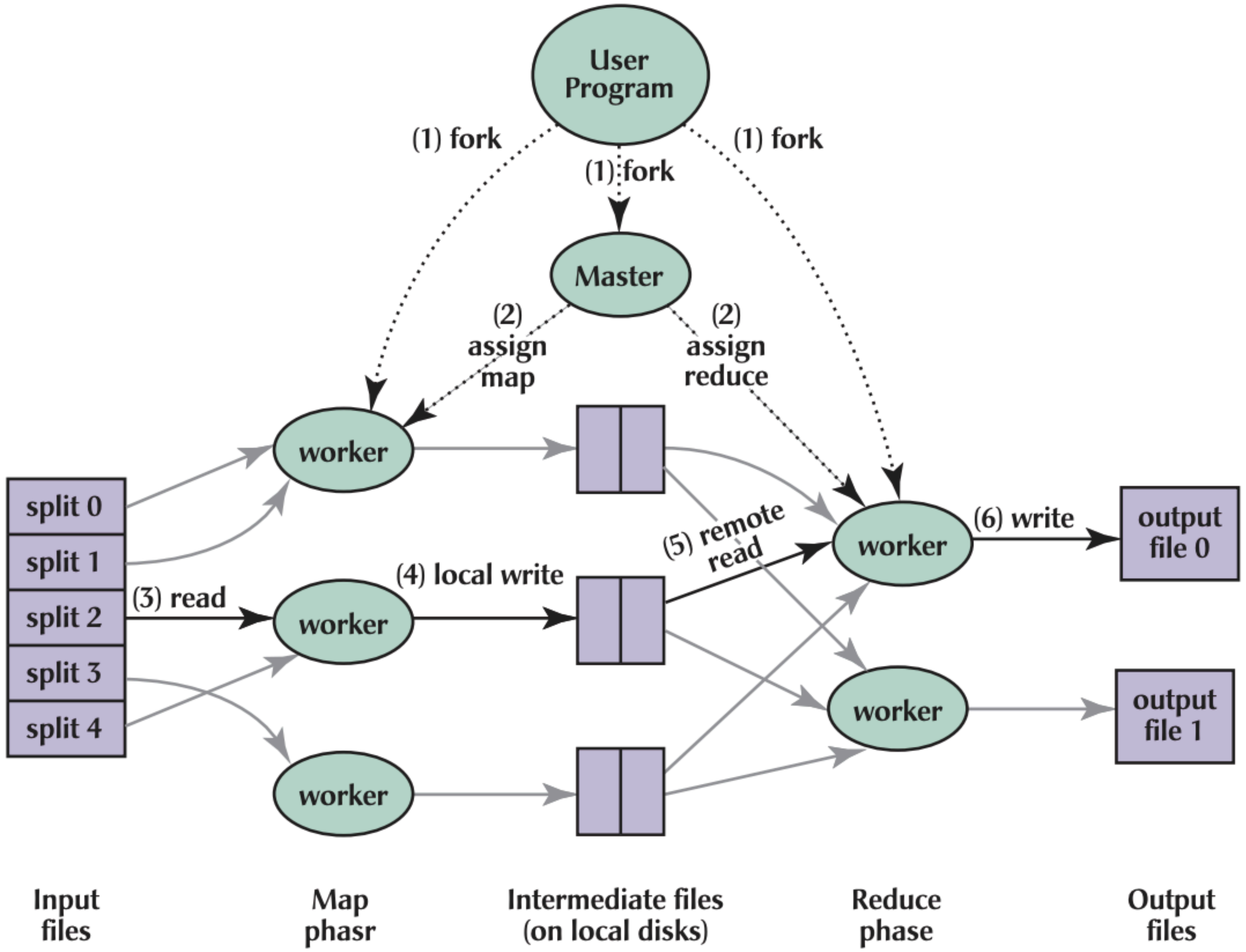
Motivation: MapReduce for sensitive computations

- Users need to run large, distributed computations over sensitive data
- A powerful framework for expressing distributed computations: MapReduce
- At a high level, users should be able to outsource their computation to a cloud provider and have the following guarantees:
    - An attacker controlling the cloud provider cannot view or modify the data
    - An attacker controlling the cloud provider cannot tamper with the computation
- Tool: Intel SGX

Background: MapReduce

Jeffrey Dean and Sanjay Ghemawat

- Framework for expressing distributed computations
- Takes input key-value pairs and outputs key-value pairs
- Map function: takes input key-value pairs, and outputs intermediate key-value pairs
- Reduce function: takes as input some intermediate keys and all values for those keys, and outputs a value

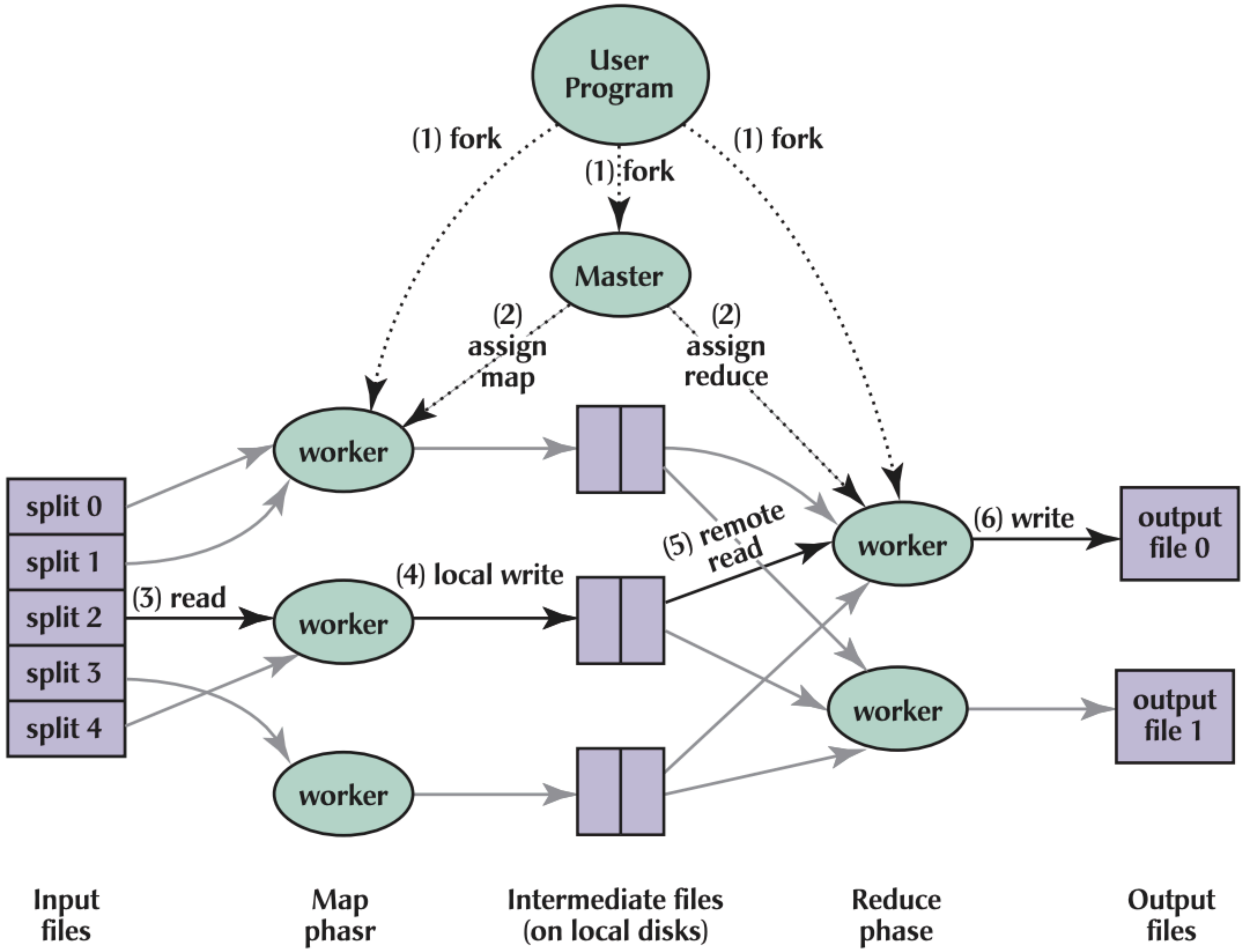


Background: MapReduce

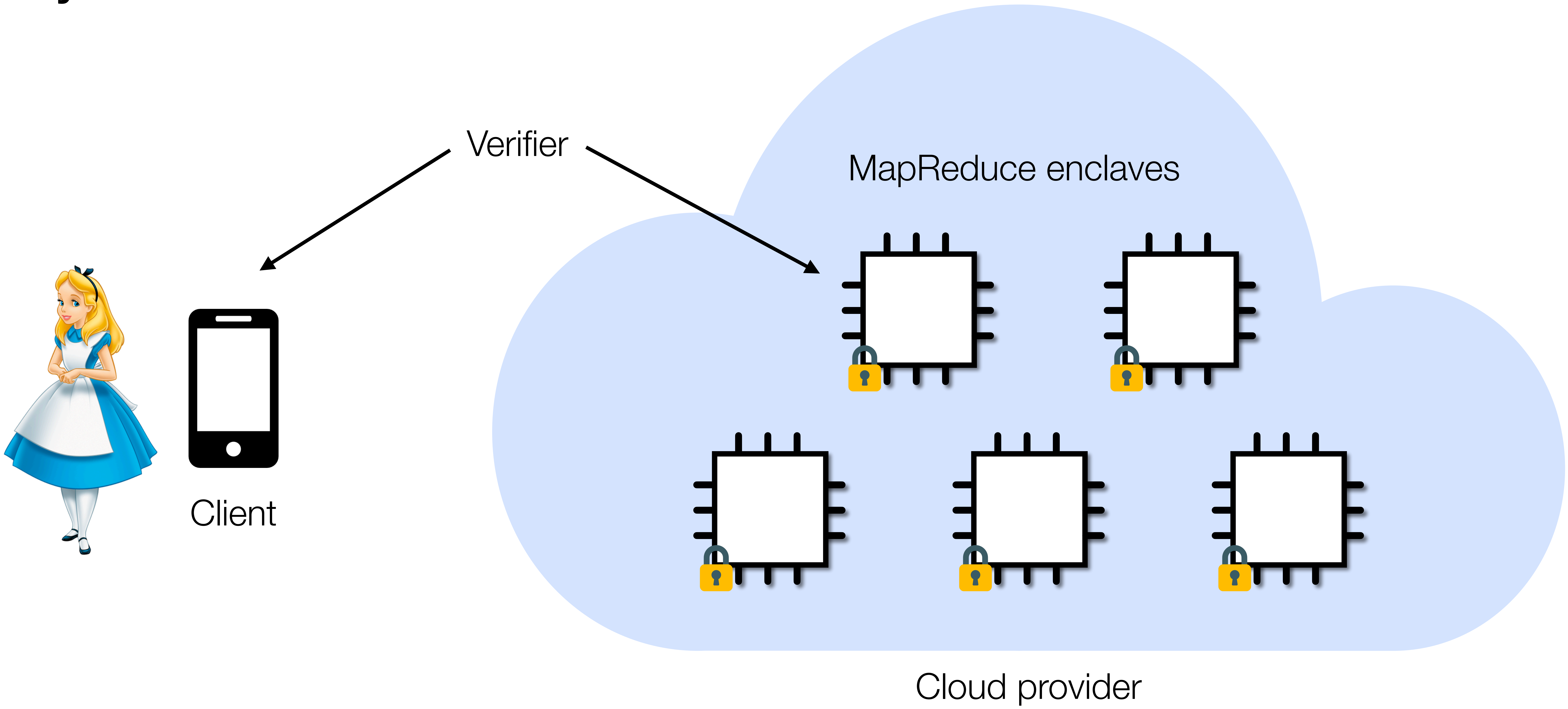
Jeffrey Dean and Sanjay Ghemawat

```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
      EmitIntermediate(w, "1");

reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
```



System entities



Attacker model

- Protect against attacker that can:
    - Control the software stack for the cloud provider’s infrastructure
    - View and tamper with network traffic
- Does not protect against an attacker that:
    - Has physical access to SGX enclaves
    - Runs a network traffic-analysis, side-channel, or fault-injection attack
    - Launches a denial-of-service attack

Security and privacy properties

An attacker that controls the software stack for the cloud infrastructure and the network learns at most:

- Encrypted sizes for the code, input splits, intermediate key-value pairs, and output key-value pairs
- Key-repetition patterns in intermediate key-value pairs (see how intermediate key-value pairs are mapped to reducers)

Design challenges

- Encrypting and authenticating messages between nodes is not enough
- Attacker that controls cloud can still drop or duplicate data — can affect computation outputs
- Want to preserve integrity without affecting ability to load balance and schedule computation

Protocol steps

- Step 0: Deployment
- Step 1: Setup
- Step 2: Mapping
- Step 3: Reducing
- Step 4: Verification

Step 0: Deployment

- Client runs attestation procedure with enclave
- If client is convinced that the enclave is legitimate and deployed by a cloud provider, the client sends decryption keys
  - Decryption keys used to access code and later execution data

Step 1: Setup

- Client encrypts and uploads input splits
- Client generates the job specification
  - Which input splits are provided as input
  - Number of reducers
- Client securely sends the job specification to the verifier

Step 2: Mapping

- MapReduce distributes input splits to mappers for processing
- Mappers output intermediate key-value pairs — need to send all pairs with the same intermediate key to the same reducer (for correctness)
- For each intermediate key-value pair (k_i, v_i) where there are R reducers, mapper reveals (simplified):

  $\mathsf{PRF}_{\mathsf{kprf}}(k_i) \mod R, \mathsf{Enc}_{\mathsf{kinter}}(k_i \,||\, v_i)$

  where kprf is a PRF key and kinter is an encryption key from step 0
- At the end, mapper sends
  - Each reducer an encryption of number of intermediate key-value pairs to expect
  - The verifier an encryption of the input key set it processed

Step 3: Reducing

- Reducer checks that it received the expected number of intermediate key-value pairs from each mapper
- Reducers process encrypted intermediate key-value pairs to produce encrypted output key-value pairs
- Each reducer sends the verifier an encryption of:
  - Set of IDs of output key-value pairs
  - Mappers where received an encryption of number of intermediate key-value pairs

Step 4: Verification

- Verifier collects verification messages from mappers and reducers
- Verifier checks that
  - Each mapper and reducer sent a verification message
  - Each input has been processed exactly once by the mappers
  - Each reducer received all the relevant intermediate key-value pairs from each mapper
- If all the checks pass, then verifier accepts that union of output IDs from reducers are the encrypted job output

Security and privacy properties

An attacker that controls the cloud software stack and network learns:

- Encrypted sizes for the code, input splits, intermediate key-value pairs, and output key-value pairs
- Key-repetition patterns in intermediate key-value pairs (see which intermediate keys map to which reducers)

What can this information reveal?

How could we mitigate this leakage?

Limitation of VC3 attacker model

- Attacker model does not account for fact that memory and network access patterns are straightforward for enclave attacker to view
  (student presentation)

- Based on how mapper access patterns, an attacker can trace inputs through the computation, leaking information about input data

- Property we want: how the application accesses memory reveals no information about sensitive data (obliviousness)

Opaque: oblivious distributed analytics

Wenting Zheng, Ankur Dave, Jethro Beekman, Raluca Ada Popa, Joseph Gonzalez, and Ion Stoica

- Distributed analytics with Intel SGX while hiding access patterns
- Oblivious building block: oblivious sorting
- Use this to construct more general oblivious operators
    - Filtering, aggregating, sort-merge join
- Show how to go beyond access pattern leakage to also hide the output size of an operator (e.g., how many items match a filter)

Outline

1. Background on secure hardware
2. VC3 paper
3. **Logistics**
4. Student presentation on controlled-channel attacks

Final project proposals

- Due next Thursday 10/16
- For project proposal, need:
  - Group members
  - Problem that you’re trying to solve
  - Privacy goals of system
- Not sure what to work on? Come to office hours!

Security properties for reading questions

"System X provides privacy." 🧐

- Privacy against who?
- What information is actually protected?

"System X hides queries from an attacker who compromises the server." 😃

- Explains what information is protected from an attacker with specific capabilities

Outline

1. Background on secure hardware
2. VC3 paper
3. Logistics
4. **Student presentation on controlled-channel attacks**

References

Costan, V., & Devadas, S. (2016). Intel SGX explained. *Cryptology ePrint Archive*.

Cheng, Pau-Chen, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. "Intel tdx demystified: A top-down approach." *ACM Computing Surveys* 56, no. 9 (2024): 1-33.

Zheng, Wenting, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. "Opaque: An oblivious and encrypted distributed analytics platform." In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 283-298. 2017.

https://sgx101.gitbook.io/sgx101/sgx-bootstrap/overview

https://www.blackhat.com/docs/us-16/materials/us-16-Aumasson-SGX-Secure-Enclaves-In-Practice-Security-And-Crypto-Review.pdf

https://www.wolfssl.com/difference-hsm-tpm-secure-enclave-secure-element-hardware-root-trust/

https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-computing-deployment-models

https://www.thesslstore.com/blog/what-is-tpm-security-trusted-platform-modules-explained/