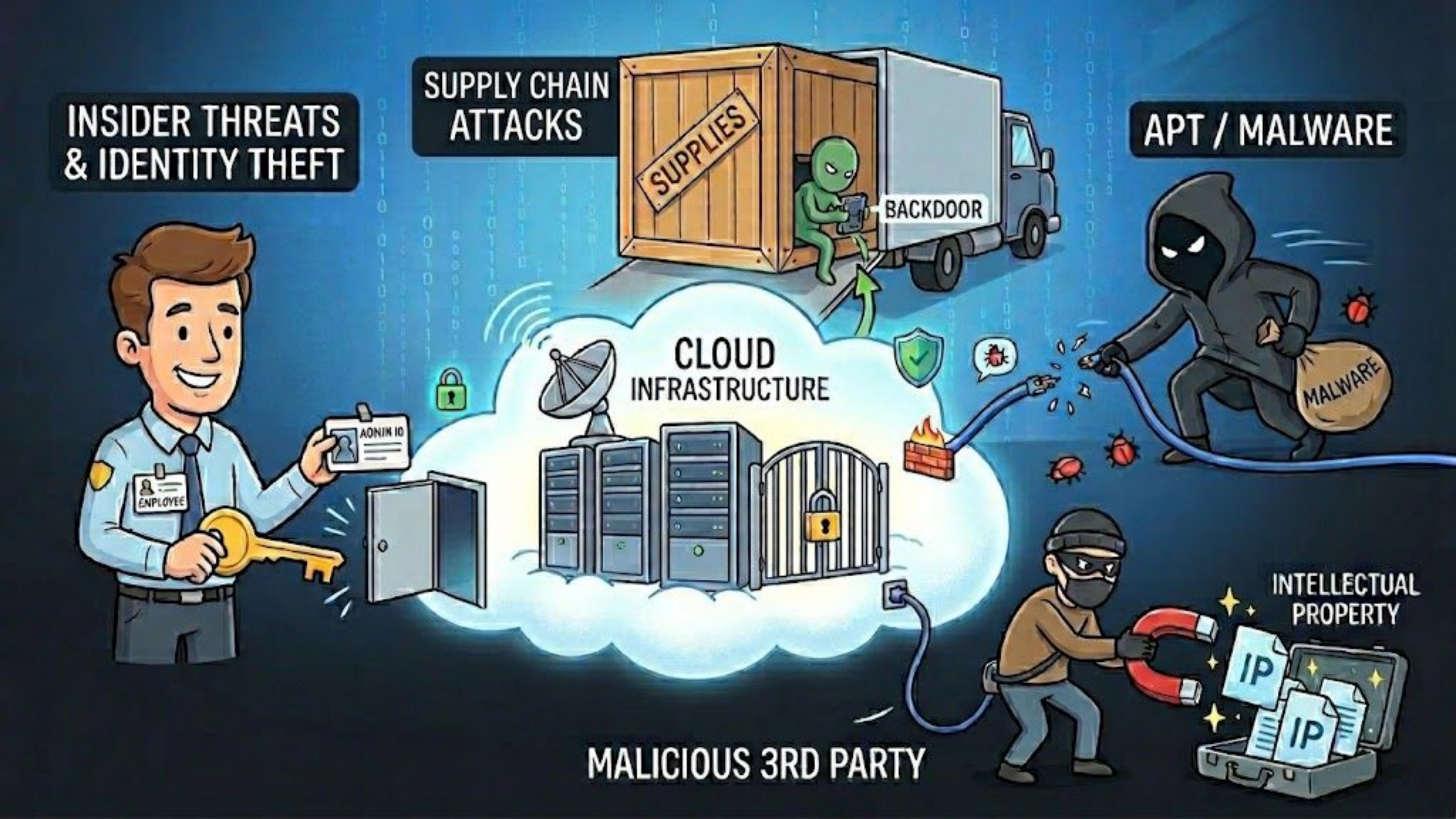
SNPeek

Side-Channel Analysis of Privacy Applications on Confidential VMs

Daniel Moghimi
Senior Research Scientist

danielmm@google.com

Joint work with **Ruiyi Zhang**[1], **Albert Cheu**[2], **Daniel Moghimi**[2], **Phillipp Schoppmann** [2], **Michael Schwarz**[1], **and Octavian Suciu**[2]
1. CISPA Helmholtz Center for Information Security
2. Google

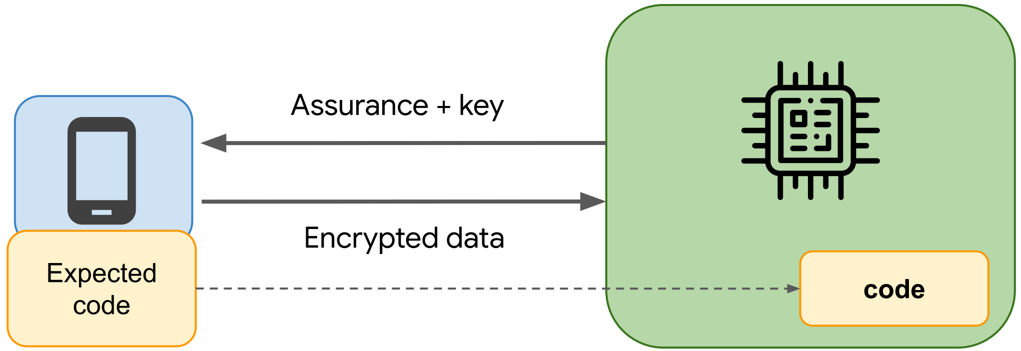

INSIDER THREATS & IDENTITY THEFT
SUPPLY CHAIN ATTACKS
SUPPLIES
BACKDOOR
APT / MALWARE
MALWARE
CLOUD INFRASTRUCTURE
ADMIN ID
EMPLOYEE
INTELLECTUAL PROPERTY
IP
IP
MALICIOUS 3RD PARTY

Confidential Compute

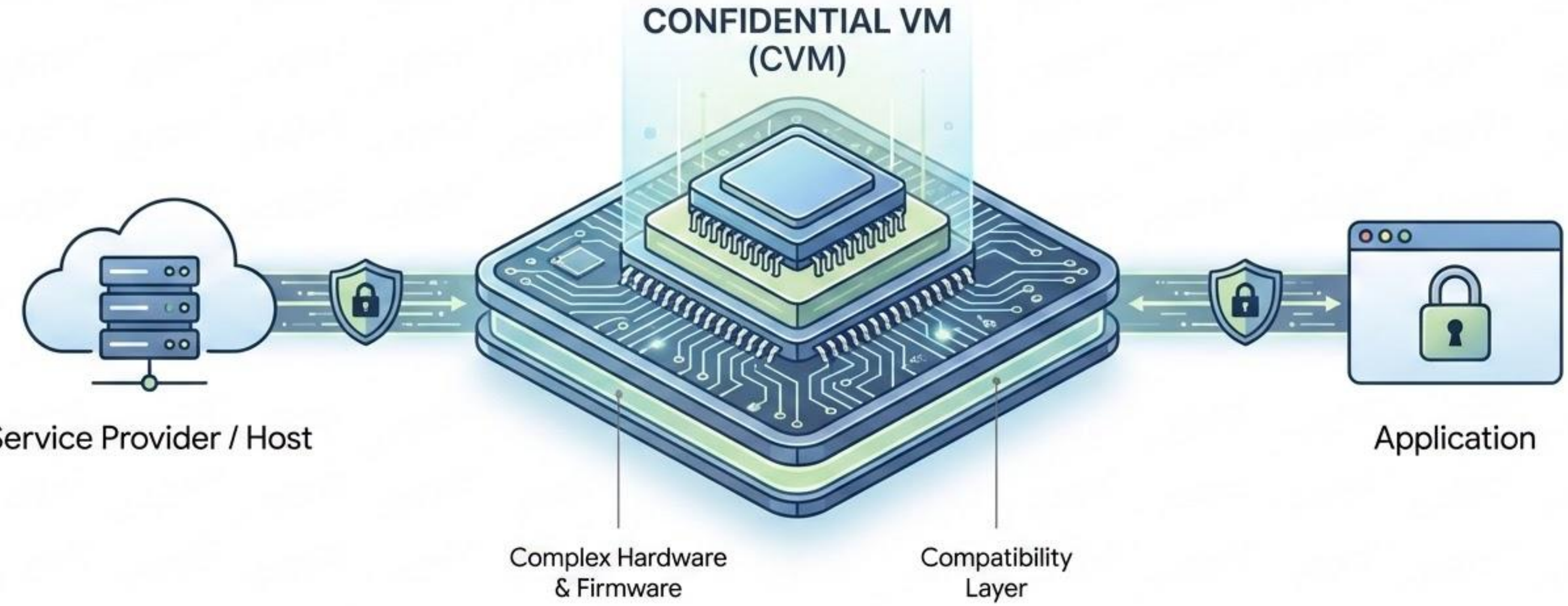
Background

Insufficient Idealized Privacy Holy Grail

A hardware-based inescrutable black box that computes a public function, and whose behaviour is externally verifiable.

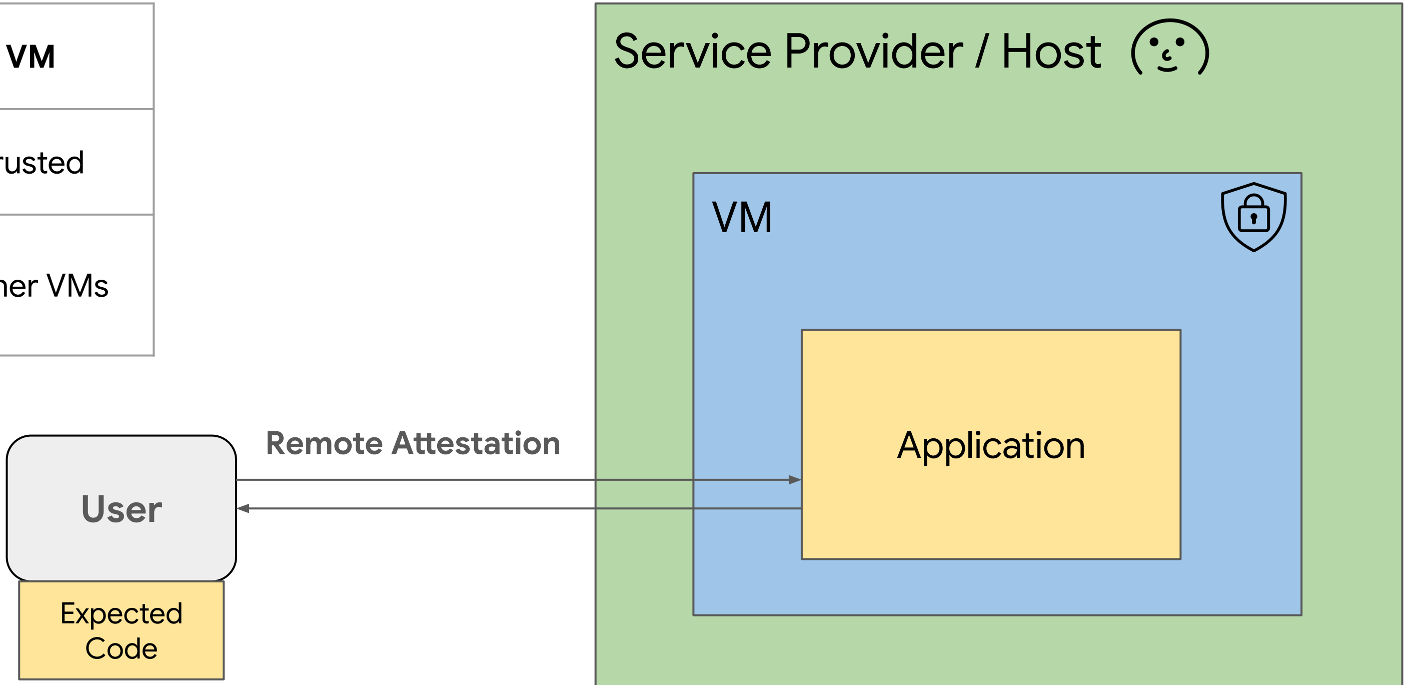


Confidential VMs use complex hardware/firmware to offer a compatibility-preserving middle ground.



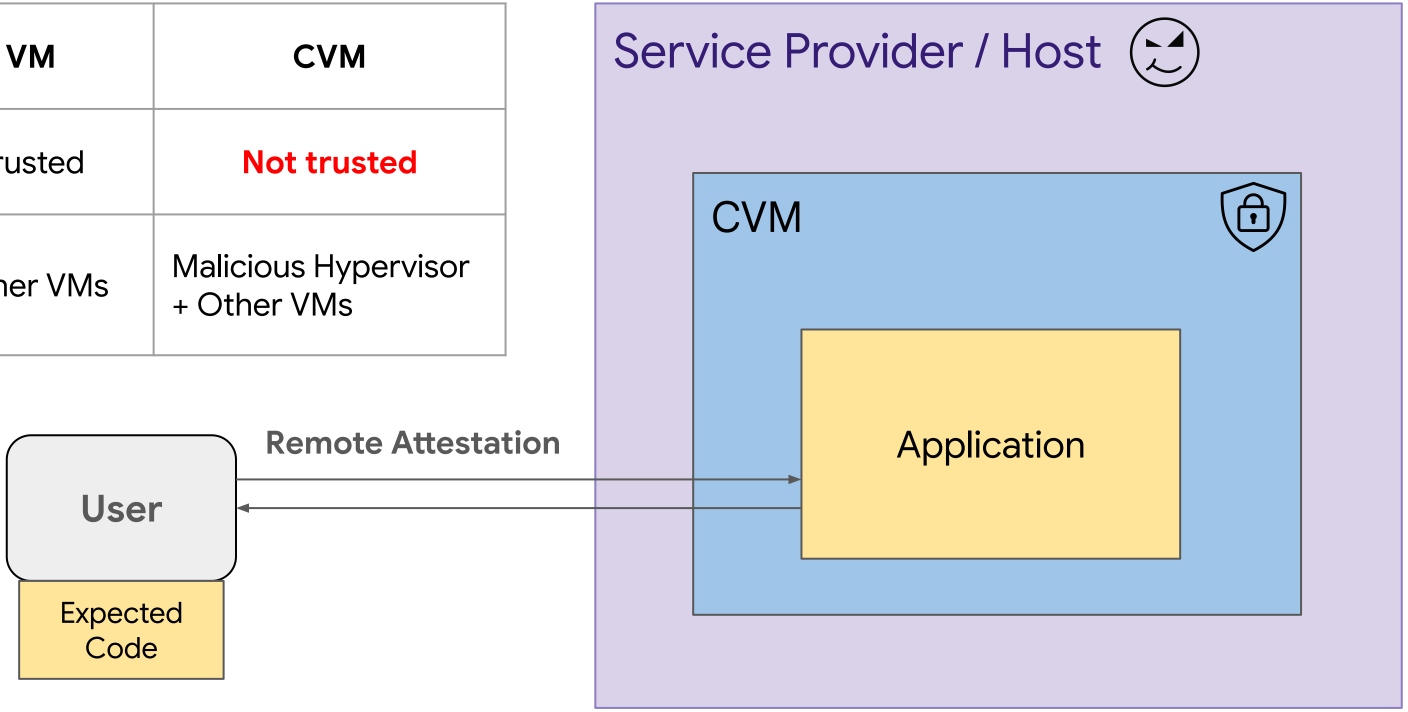
VM vs. CVM (Confidential VM)

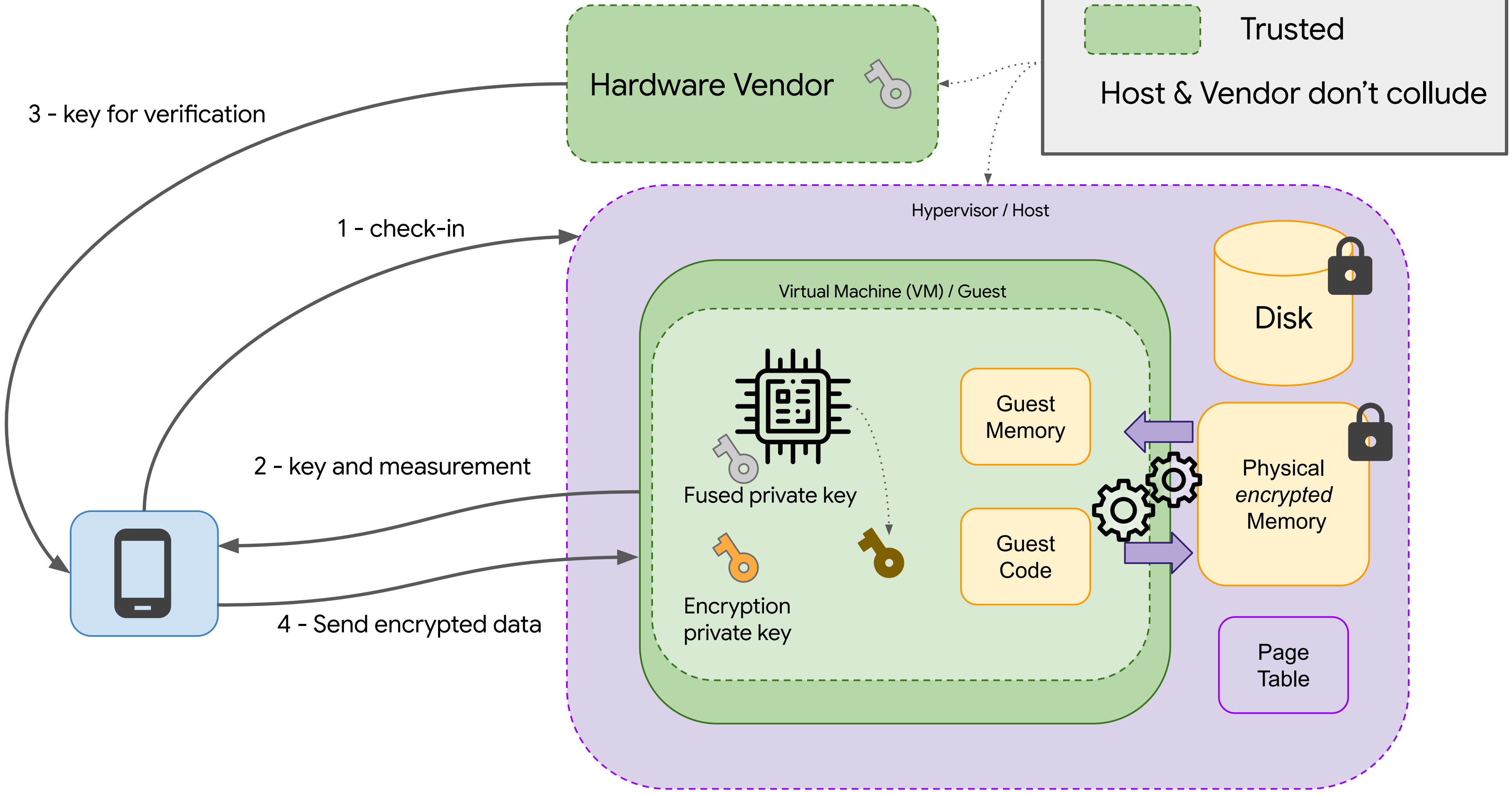
	VM
Hypervisor	Trusted
Potential Attackers	Other VMs



VM vs. CVM

	VM	CVM
Hypervisor	Trusted	**Not trusted**
Potential Attackers	Other VMs	Malicious Hypervisor + Other VMs




Trusted
Host & Vendor don't collude
Hardware Vendor
3 - key for verification
1 - check-in
Hypervisor / Host
Virtual Machine (VM) / Guest
Disk
Guest Memory
Guest Code
Physical *encrypted* Memory
Page Table
2 - key and measurement
Fused private key
Encryption private key
4 - Send encrypted data

CVM Side Channels

Who is responsible for side channels?

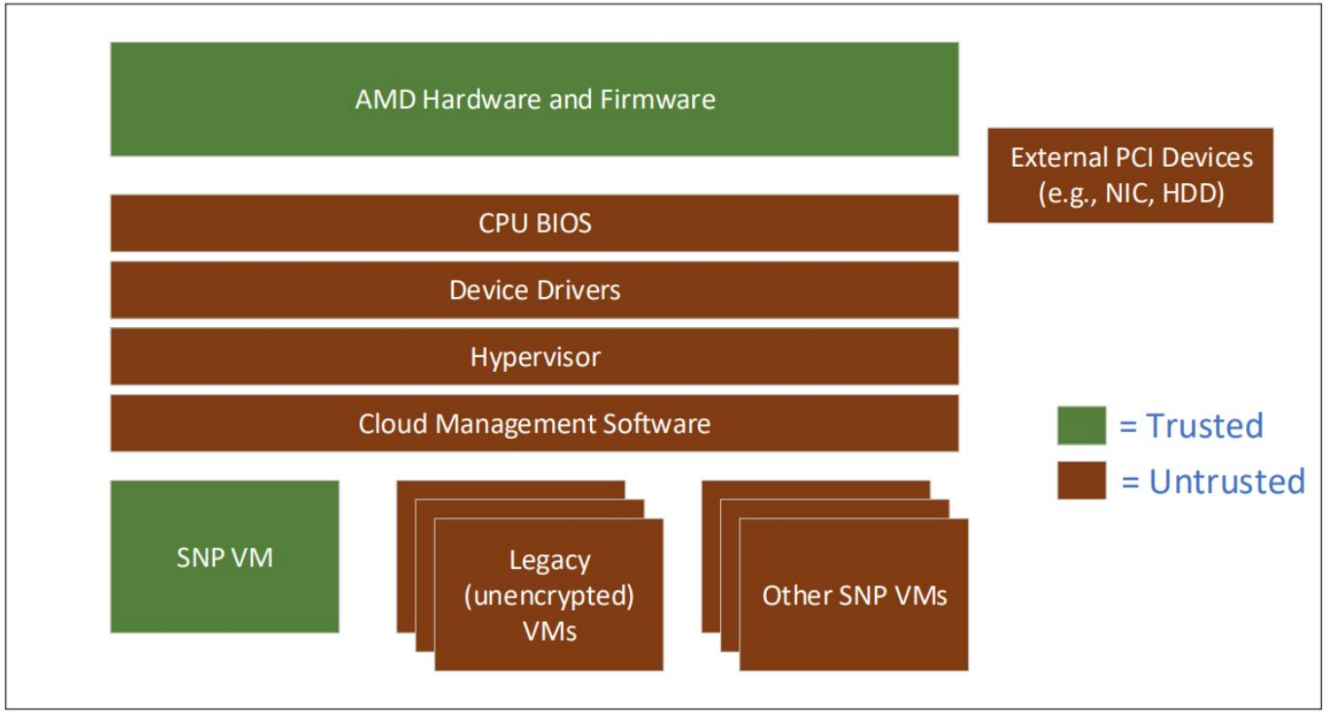
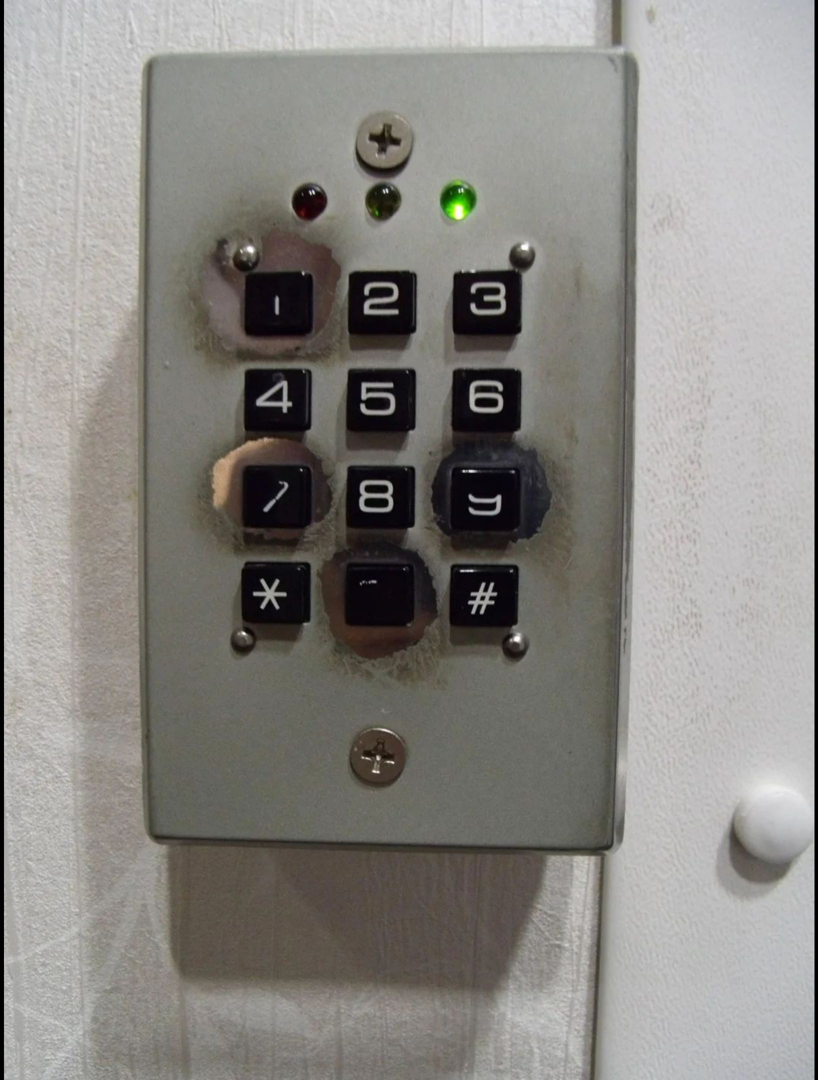
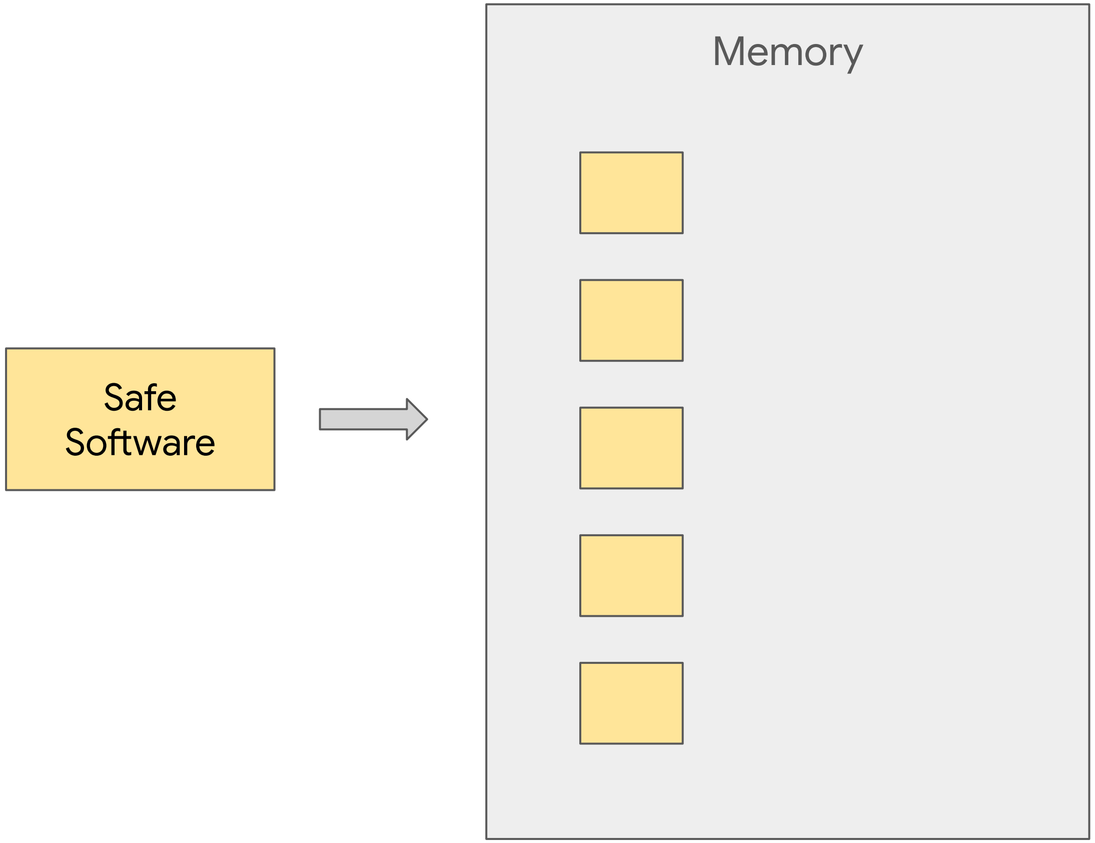
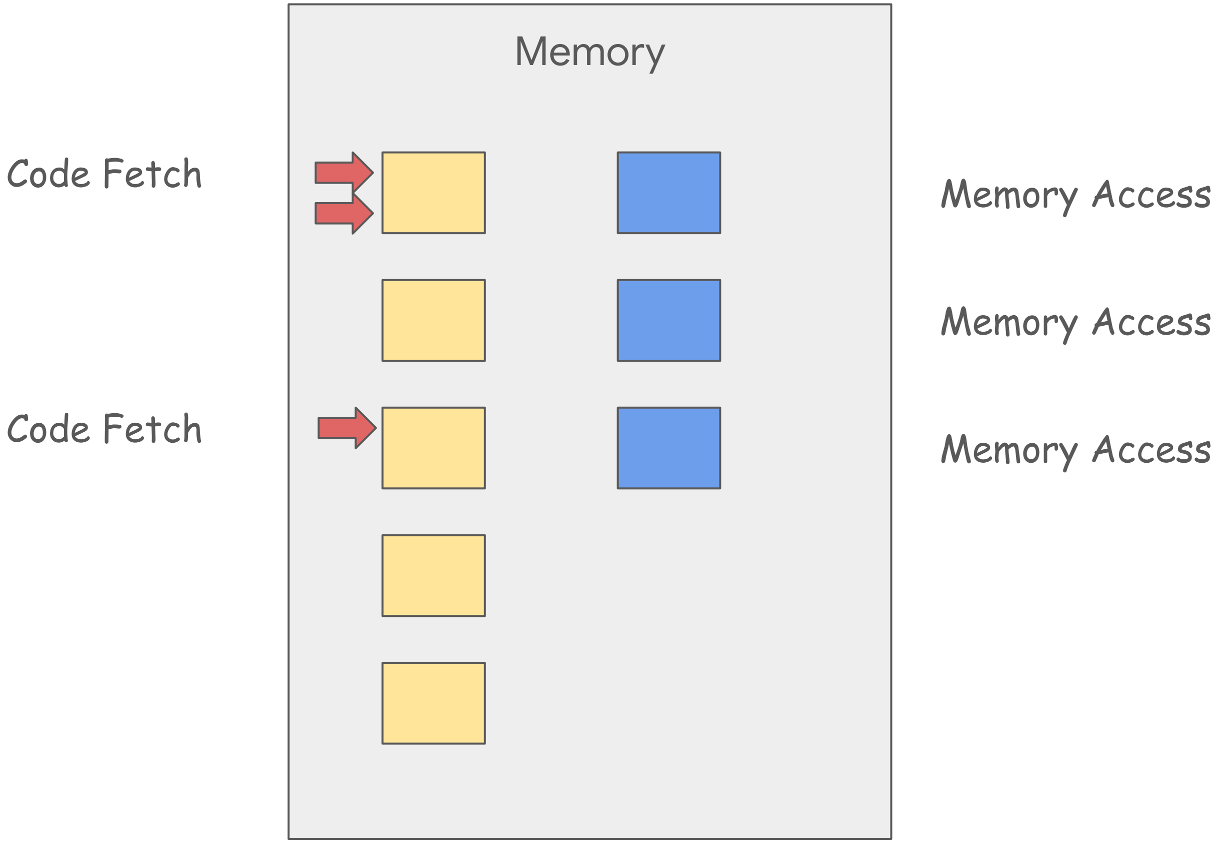


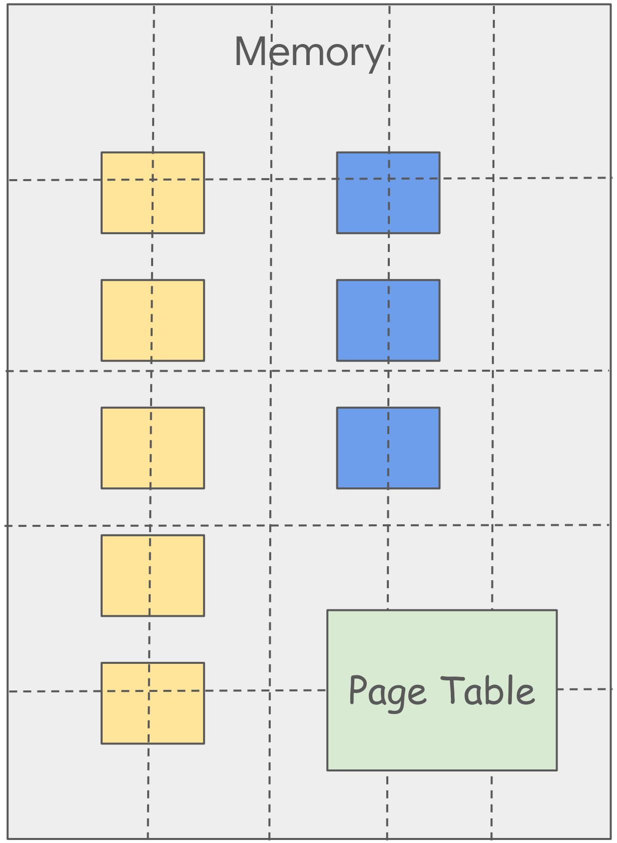
Figure 3: SEV-SNP threat model

(Diagram from AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More)


1
2
3
4
5
6
7
8
9
*
#


Safe Software
Memory


Memory
Code Fetch
Code Fetch
Memory Access
Memory Access
Memory Access

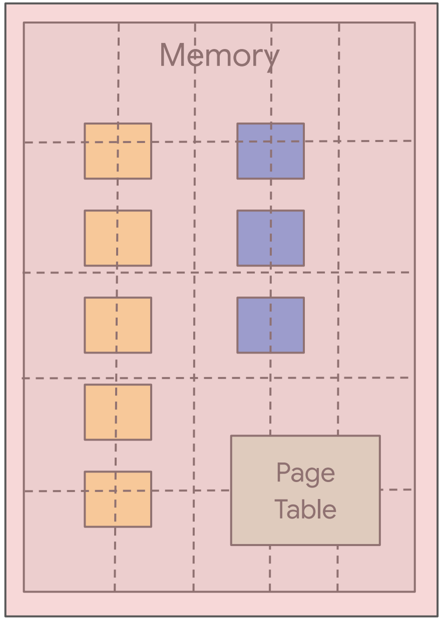


The memory is divided into chunks called pages, typically 4KB each

The guest OS has a page table to manage memory efficiently

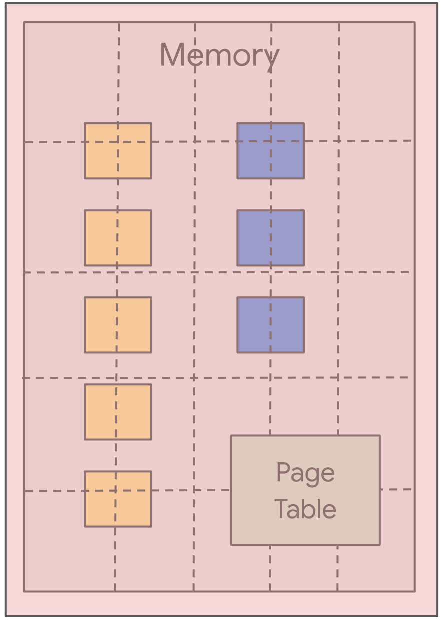
Yes,

All guest code and data within TEE are encrypted



Yes,

All guest code and data within TEE are encrypted



But...

1. The malicious hypervisor controls a nested page table
2. Cache side-channel attacks are not mitigated
3. The malicious hypervisor can still read the ciphertext of guest memory (AMD SEV-SNP)
4. Performance counter leakage (AMD SEV-SNP, will be mitigated on Zen 5)

But...

1. The malicious hypervisor controls a nested page table
2. Cache side-channel attacks are not mitigated
3. The malicious hypervisor can still read the ciphertext of guest memory (AMD SEV-SNP)
4. Leakage from performance counters (AMD SEV-SNP)

1. 4KB granularity - The attacker ~knows which code and memory pages are being executed / accessed / written
2. 64B granularity - The attacker ~knows which 64B memory blocks were accessed after previous code execution
3. 16B granularity - The attacker ~knows which 16B memory blocks changed after previous code execution
4. The number of assembly instruction / branches / taken branches... from the previously executed page

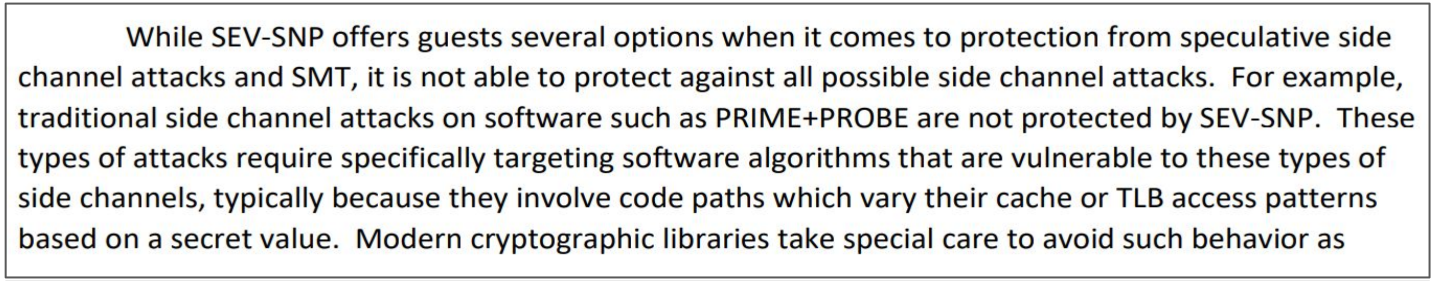
HW/FW mitigation

	Page Fault	Single-Step	Cache Attacks	Ciphertext	PMCs
AMD SEV-SNP	●	●	●	●	●
AMD SEV-SNP (Zen 5)	●	●	●	◑	○
Intel TDX	●	◑	●	○	○

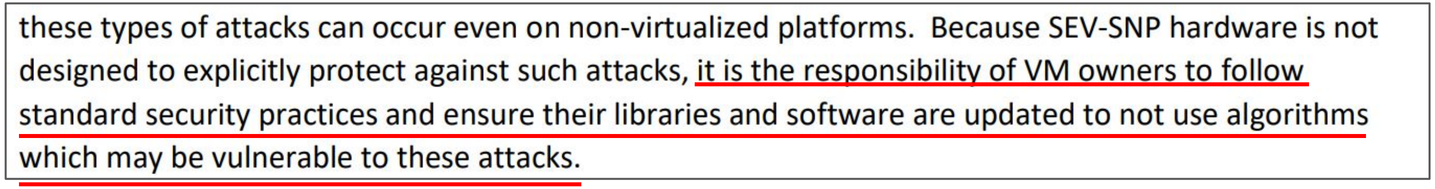
● : Vulnerable　　○ : Mitigated　　◑ : Compromised

AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More (white paper)

Side-channels are not in scope:



While SEV-SNP offers guests several options when it comes to protection from speculative side channel attacks and SMT, it is not able to protect against all possible side channel attacks. For example, traditional side channel attacks on software such as PRIME+PROBE are not protected by SEV-SNP. These types of attacks require specifically targeting software algorithms that are vulnerable to these types of side channels, typically because they involve code paths which vary their cache or TLB access patterns based on a secret value. Modern cryptographic libraries take special care to avoid such behavior as



these types of attacks can occur even on non-virtualized platforms. Because SEV-SNP hardware is not designed to explicitly protect against such attacks, it is the responsibility of VM owners to follow standard security practices and ensure their libraries and software are updated to not use algorithms which may be vulnerable to these attacks.

But extending protections is an option for future versions...

SNPeek

A Framework for Side-Channel Analysis of Privacy Apps

SNPeek

The SNPeek Framework: A dynamic analysis that (efficiently) extracts traces from AMD SEV-SNP host.

- Available here: https://security-and-privacy-group-research.googlesource.com/sevsca (kernel patch)
- Gathers page, cache, and block level info from arbitrary workloads

Formal definitions:

- Provided a setting and attacker definitions, i.e. pairwise distinguishability (MIA) & fingerprinting (reconstruction).

Analyses: Analyzed both toy examples and real (side-channel unaware) production code.

- Signal's ORAM implementation, TEE-based PIR, Stable DP histograms.

Mitigations: Proposed mitigation strategies inspired by DP.

Attack

(Differentially) Private Heavy Hitters

- ❏ Trusted Brella - Geo
  - *Membership Inference Attacks*
  - *Reconstruction Attacks*

UDF on private data

- ❏ Privacy Sandbox
  - *Covert channel with JS UDF*

Private Information Retrieval

- ❏ Oak - Geo
  - ……

Framework

- ❏ Side-channel Attacks
- ❏ Design
- ❏ Performance

Mitigation

- ❏ Coding practices
- ❏ Application specific ORAM
- ❏ HW/FW requests

Use Cases

Privacy workloads that are not so private

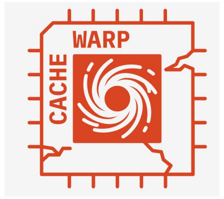
Confidential VMs: Applications (beyond cryptography)

Security Goal: Withstand a corrupted host

- Federated learning
- **Histogram computation**
- LLM Inference
- Joint data analyses

What can go wrong? (excluding physical attacks)

1. CVM guarantees are broken:
    a. Confidentiality
    b. Attestation



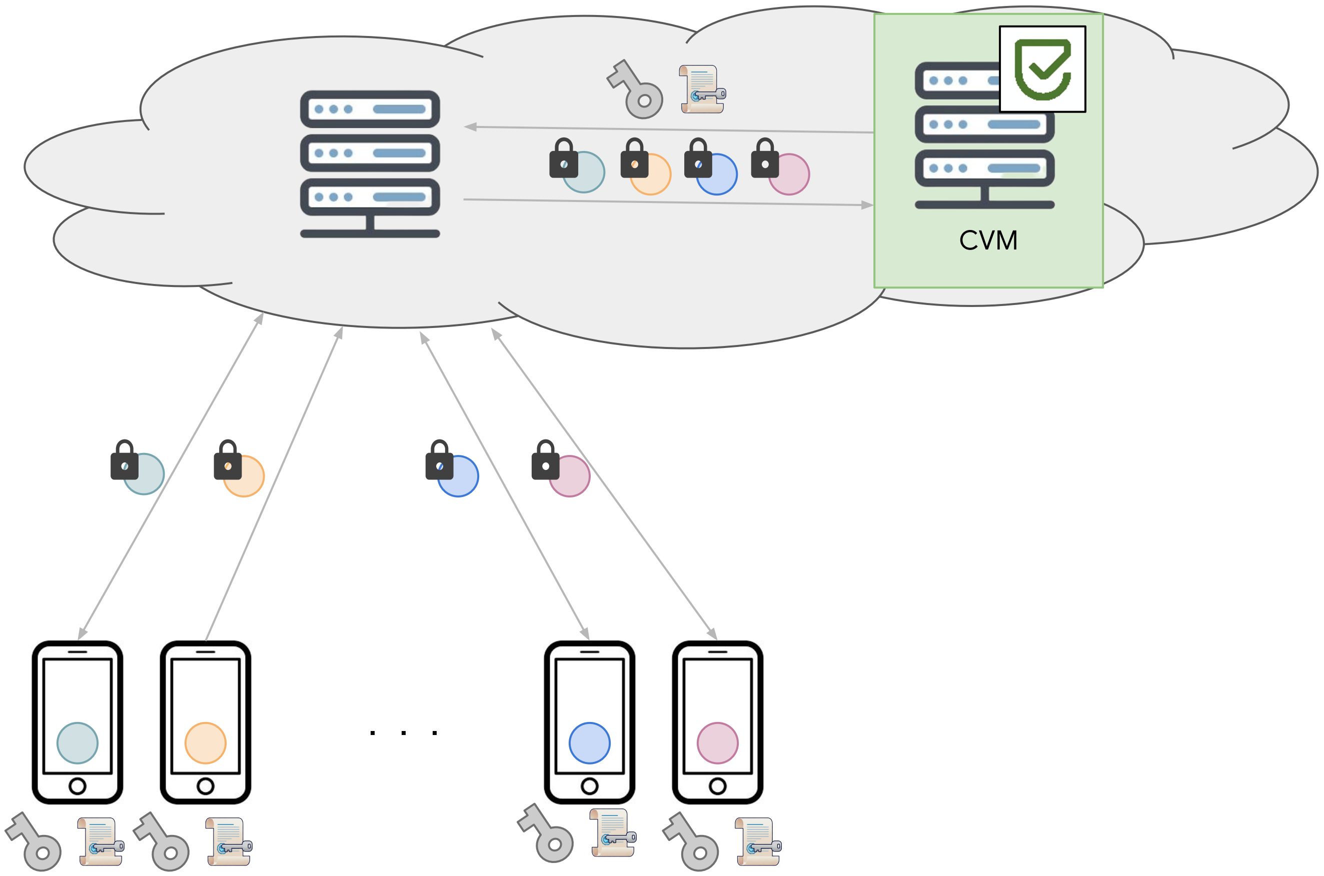
2. The application leaks, e.g. is not DP

3. The application leaks via (allowed) side channels

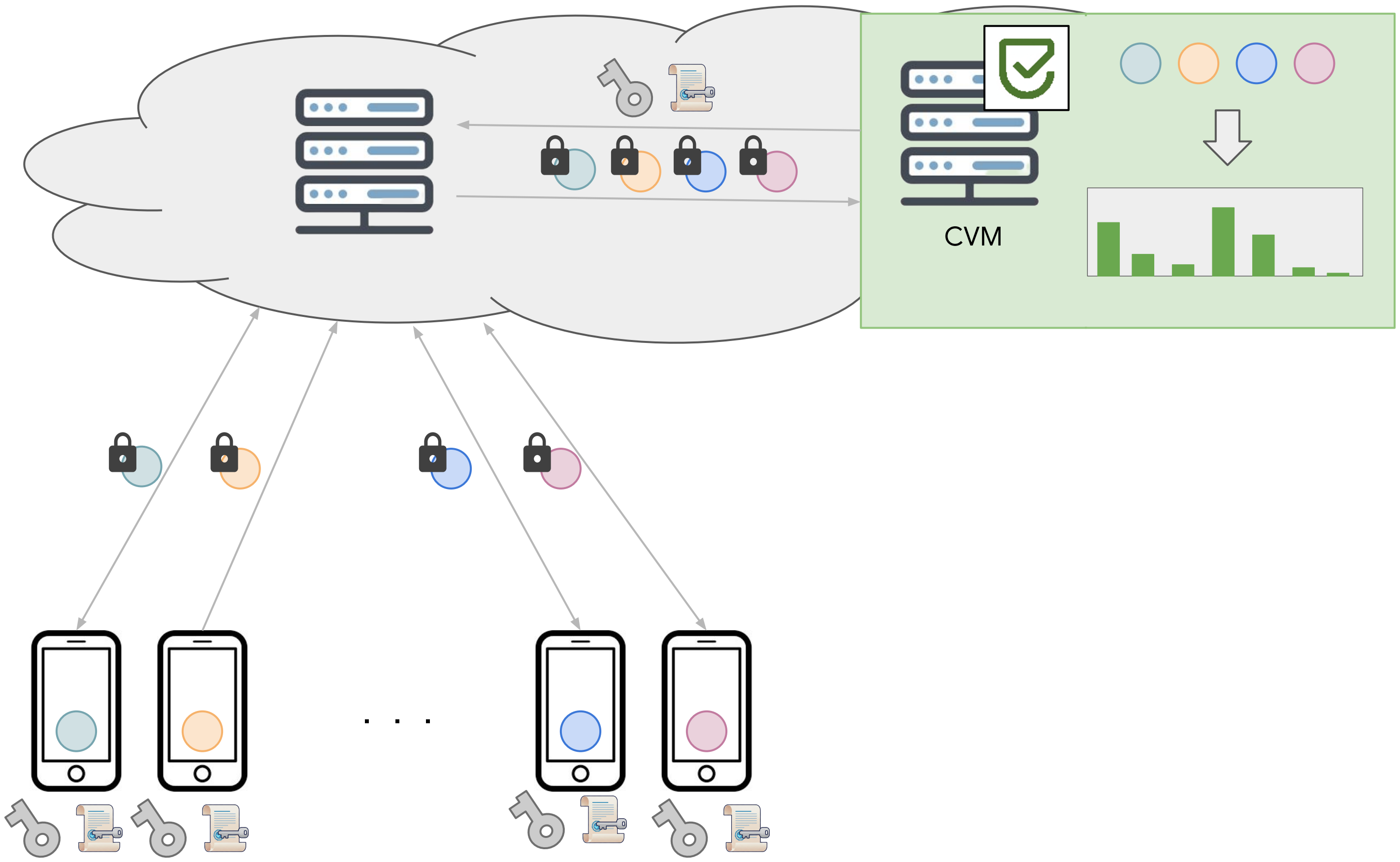
Application guarantees

CVM Threat model

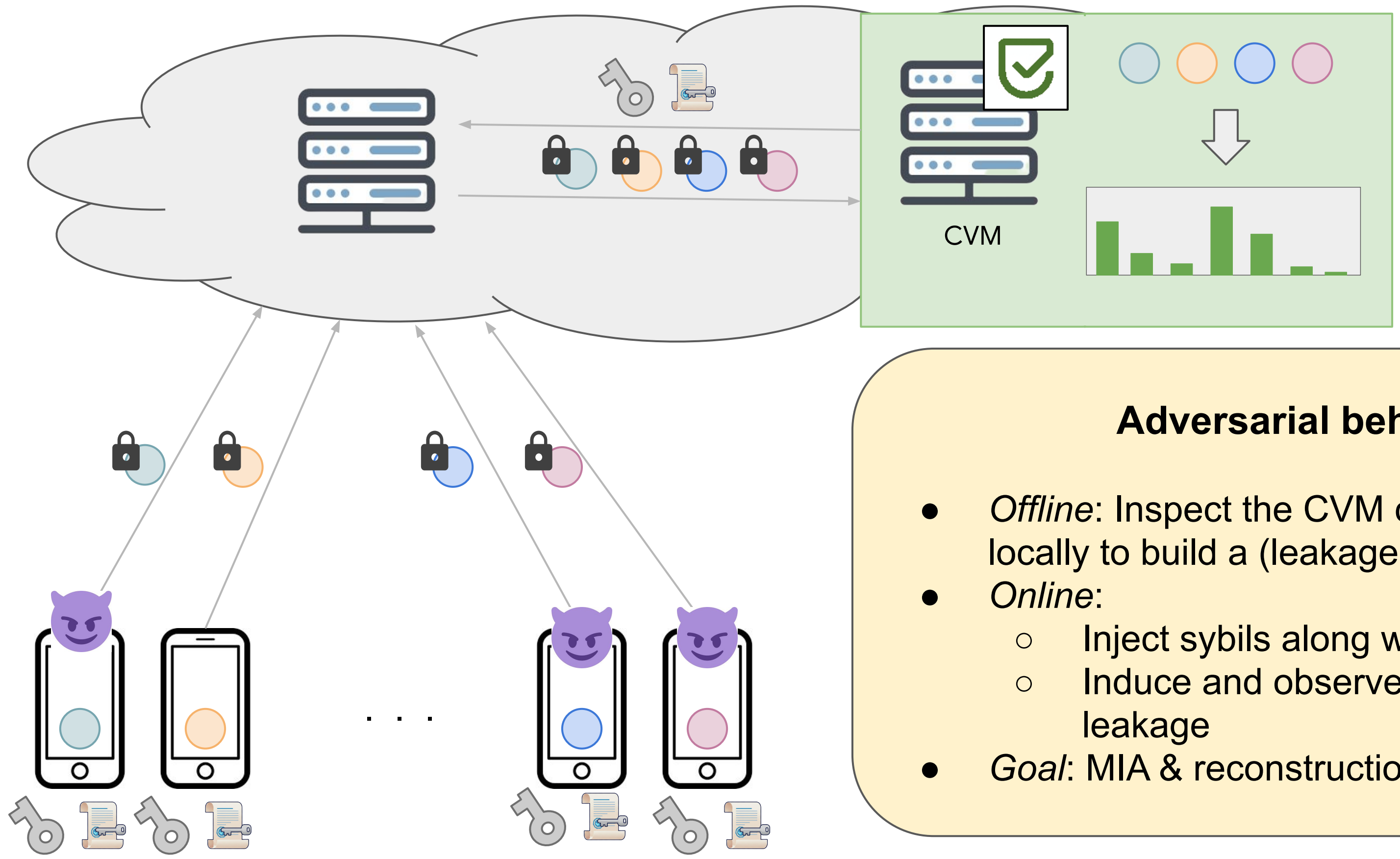
CVM-based Histograms



CVM-based Histograms



CVM-based Histograms



Adversarial behaviour

- *Offline*: Inspect the CVM code and run in locally to build a (leakage) model
- *Online*:
  - Inject sybils along with the victim's data
  - Induce and observe side channel leakage
- *Goal*: MIA & reconstruction attacks

Read input:

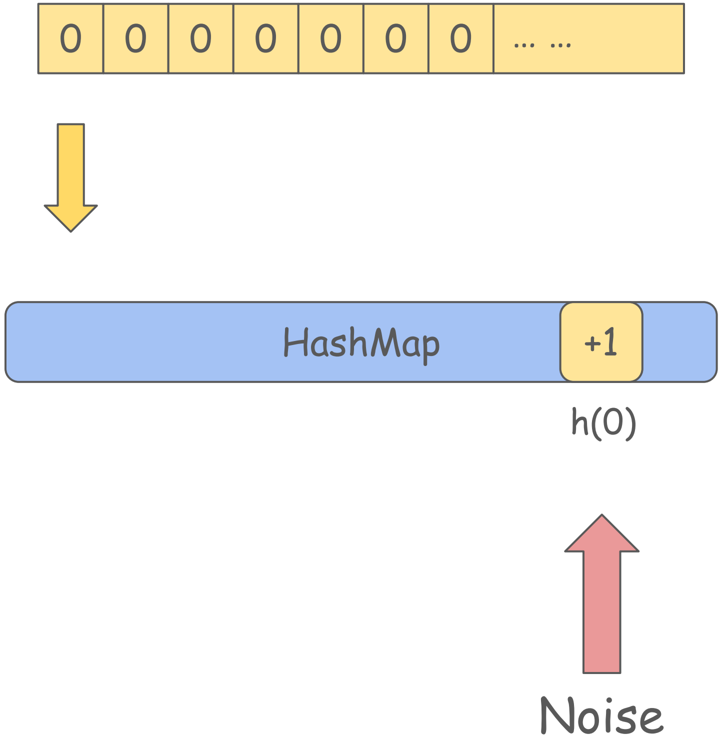
- 0, 0, 0 … 0, 0, 0   (num: 100)
- 0, 0, 0 … 0, 0, 1   (num: 100)

Accumulate:

```
std::unordered_map<int, int> hist;
while(*f >> x){
      hist[x]++;
}
```

Report:

```
std::vector<std::pair<int, int>> hist_out;
for (auto [k, v]: hist) {
      auto sample = sample_centered_laplace();
      if (v + sample >= kThreshold) {
            hist_out.push_back(std::make_pair(k, v));
      }
}
```



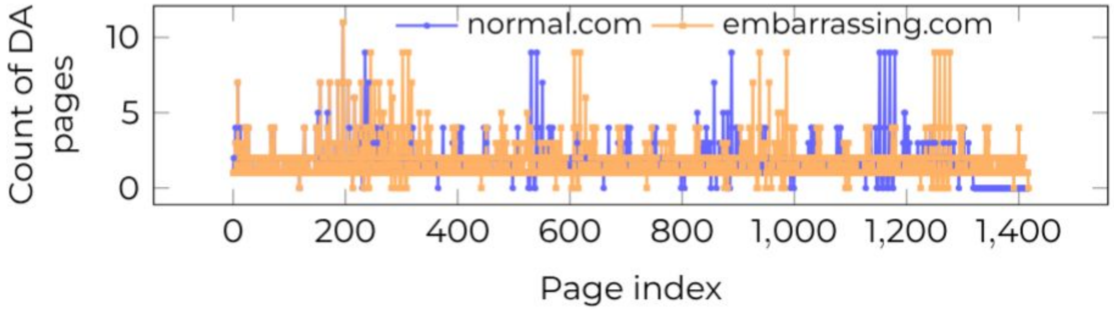
Toy example

Feed the algorithm n-1 copies of “normal.com” followed by target client data

So input is one of the following:

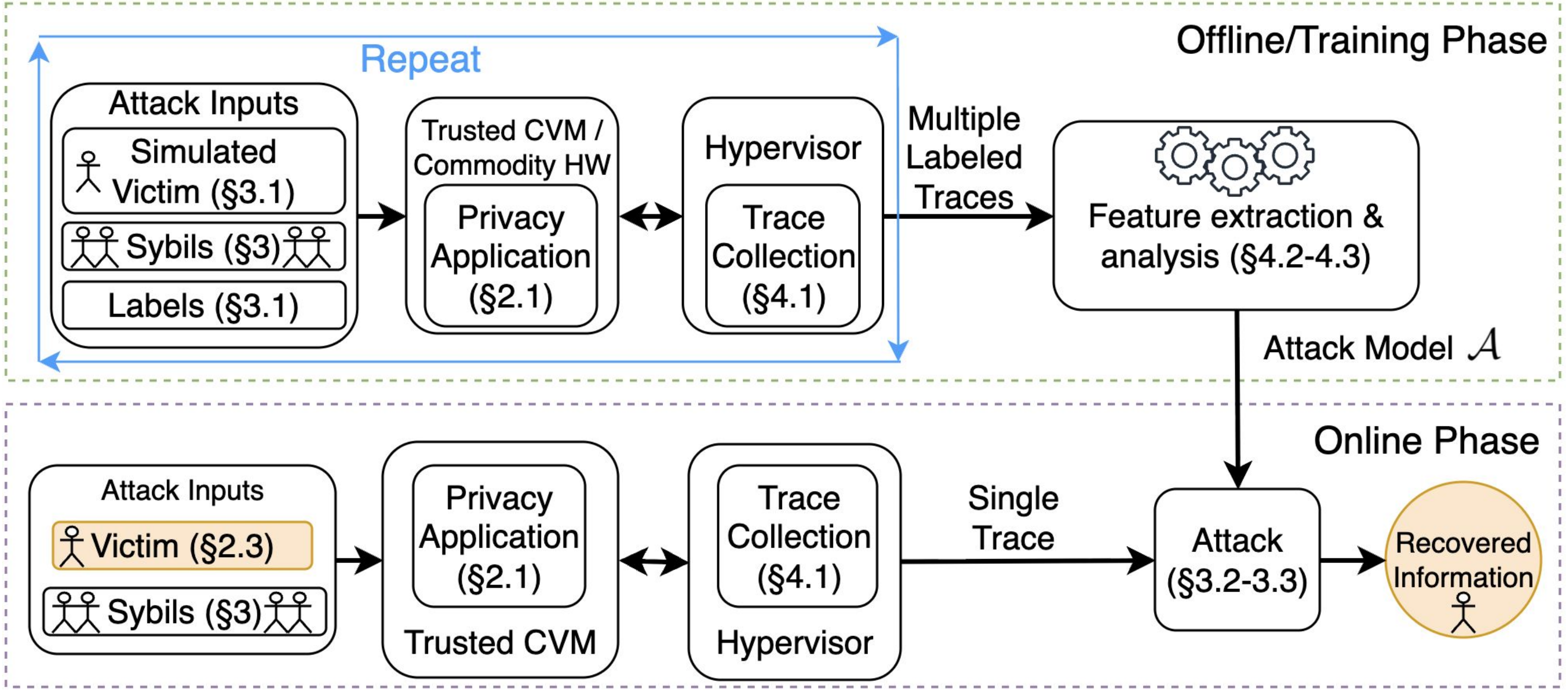
1. normal.com, normal.com, …, normal.com, embarrassing.com
2. normal.com, normal.com, …, normal.com, normal.com

SNPeek reveals that memory access behavior is divergent:



SNPeek Attack Framework

github.com/google-parfait/cvm-side-channel-analysis



Conclusion

Path Forward

Conclusion

- **Automated Side-Channel Analysis:** Essential for strengthening privacy in Confidential VMs, as software-only mitigations are insufficient against evolving threats.
- **Continuous Evaluation:** Developers must proactively analyze execution traces and threat models to ensure application-level protections remain effective.
- **SNPeek Framework:** Provides a comprehensive solution for quantifying leakage and validating the success of various mitigation strategies.
- **Future Defense-in-Depth:** Promising long-term security requires further research into architectural-level isolation, Sybil attack prevention, and oblivious data structures like ORAM.

Thanks

Daniel Moghimi

Senior Research Scientist @ Google | Security and Privacy Research Leader

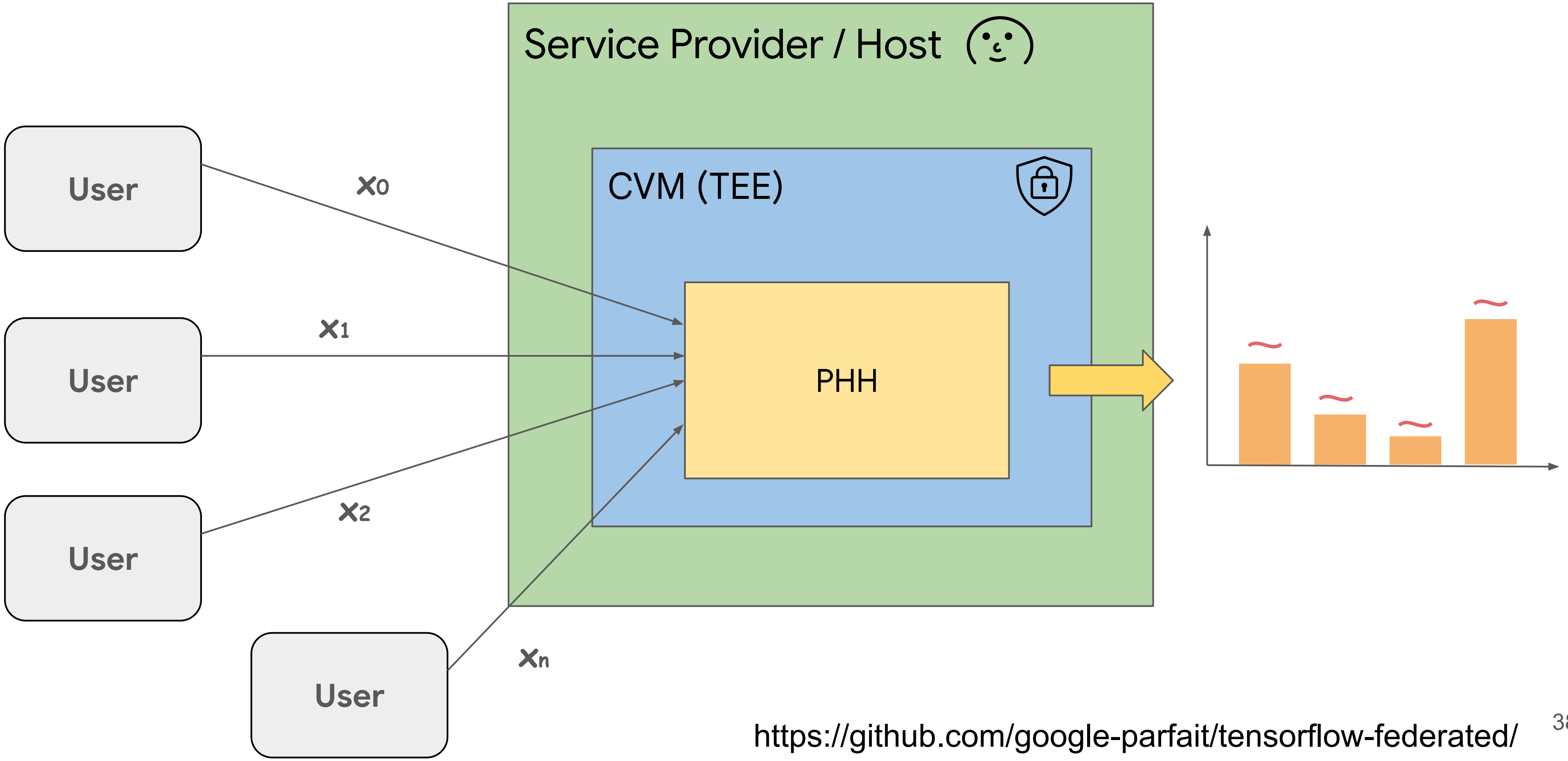
Back UP Slides

Framework - Performance

Given a 3.70 GHz CPU (3,700,000,000 CPU cycles per second),

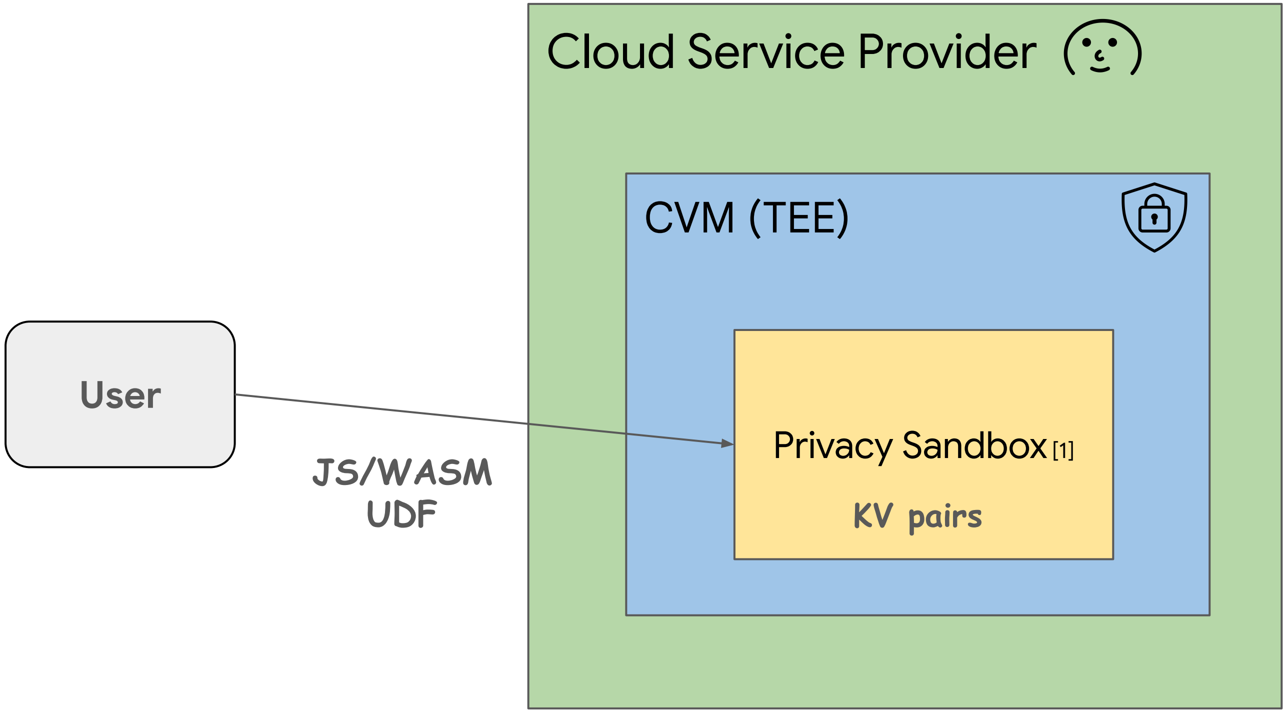
Leakage	The time spent on the context switch	# of entries per second
Page Fault	~16,000 cycles	~231,000
PMCs	~17,000 cycles	~217,000
Ciphertext	~23,000 cycles	~161,000
Cache Line	~200,000 cycles	~18,500

Note: We use AMD EPYC 9124 CPU - 16 Cores, 3.00/3.70GHz



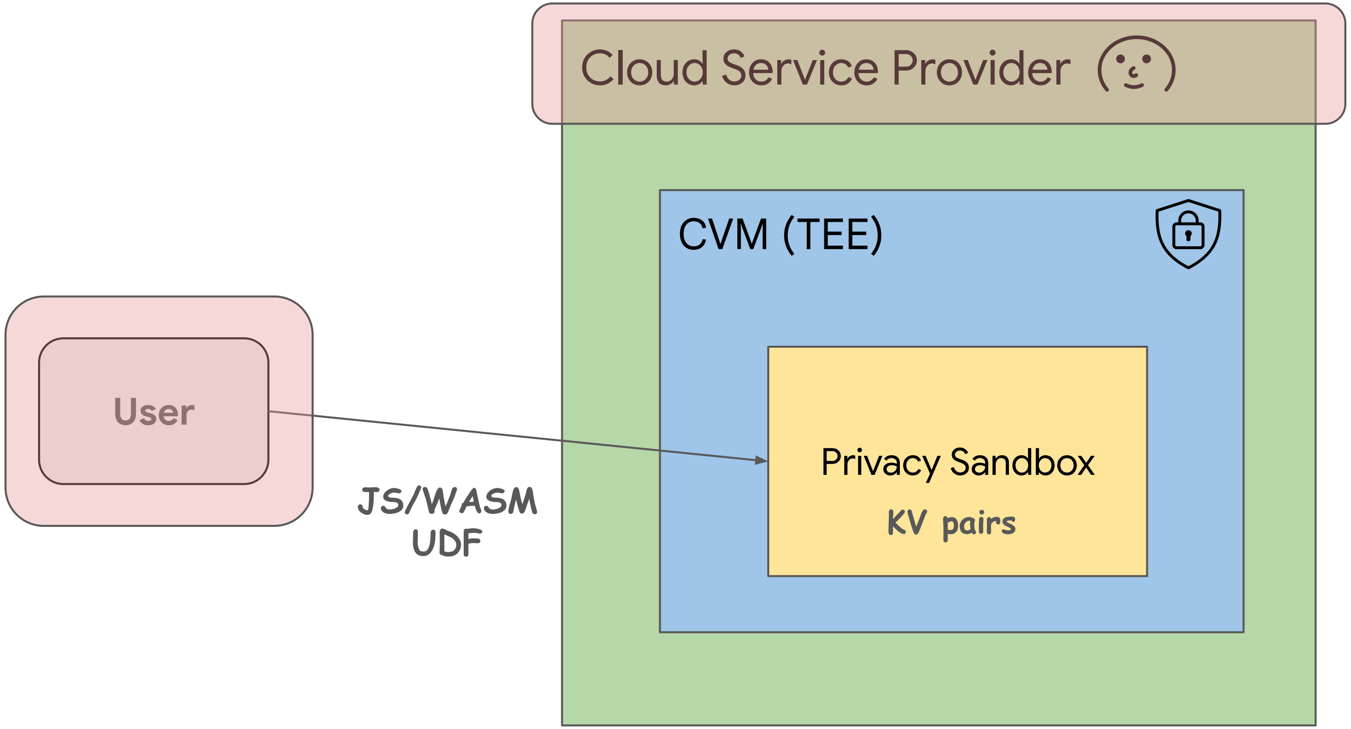
https://github.com/google-parfait/tensorflow-federated/

Covert Channel via JS UDF



[1] https://github.com/privacysandbox/protected-auction-key-value-service

Covert Channel



A Simplest Method to Encode Secret via JS UDF

var secret = [83, 69, 67, 82, 69, 84]

- ASCII of a string “SECRET”

83	69	67	82	69	84

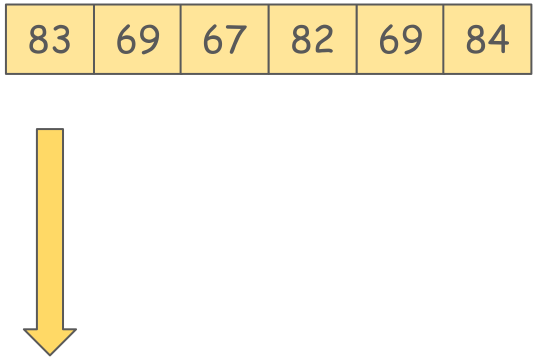
Encode:

```
for (let i = 0; i < secret.length;
i++)

        byte = secret[i]

        for (let c = 0; c < byte; c++) {
                ; // do nothing
        }
}
```

- Repeat “;” 83 times
    - Every inner loop triggers 5 CF page faults (even it does nothing in JS)
    - A different pattern when the “**i**” increments

var secret = [83, 69, 67, 82, 69, 84]

- ASCII of "SECRET"

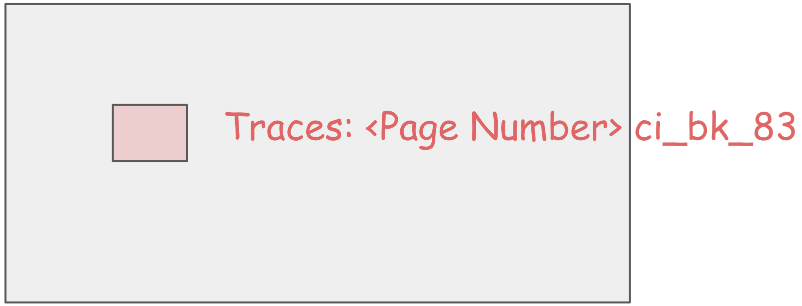


Encode:

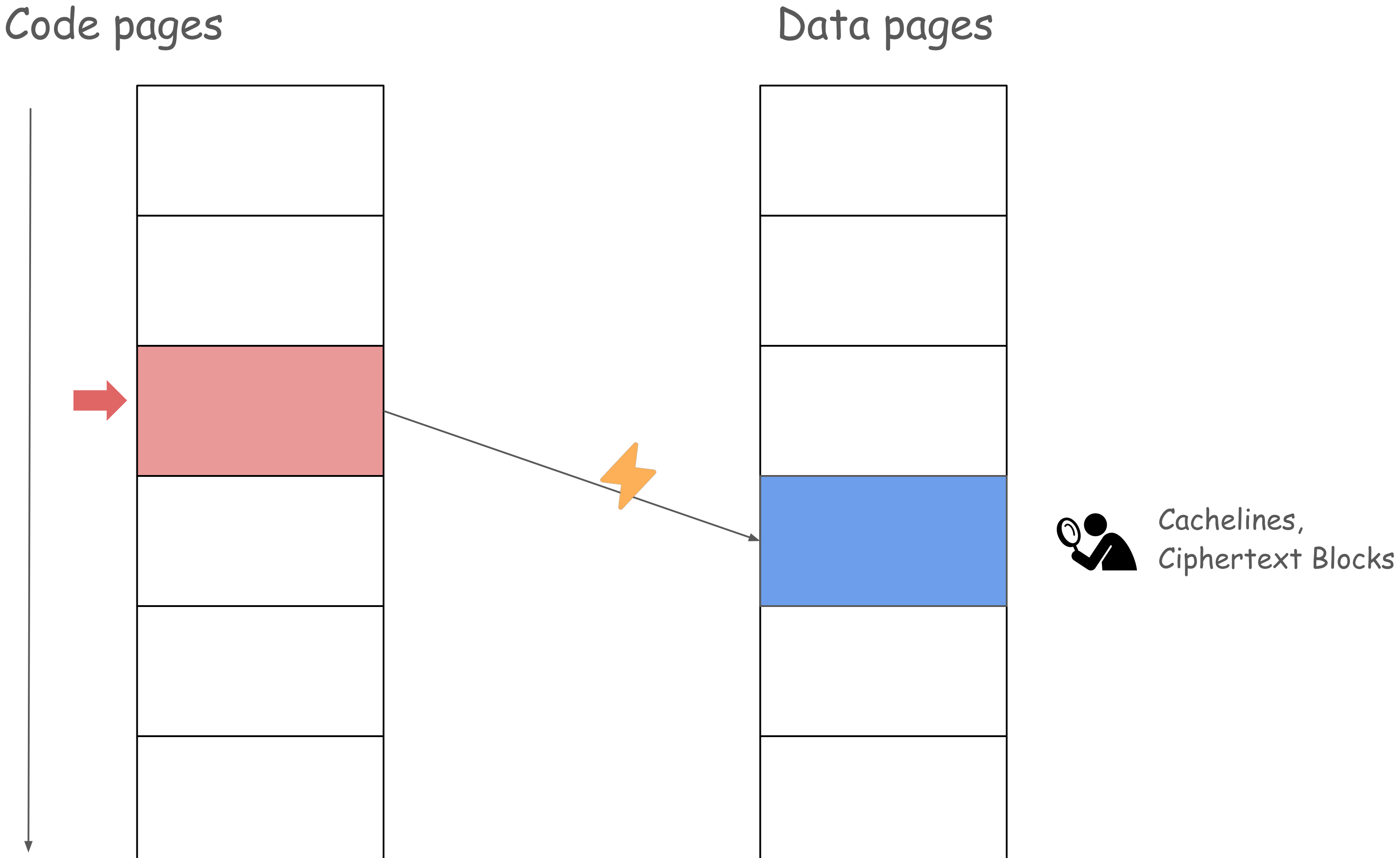
```
const test = new ArrayBuffer(4096);
const encode_buffer = new Uint8Array(test);
```

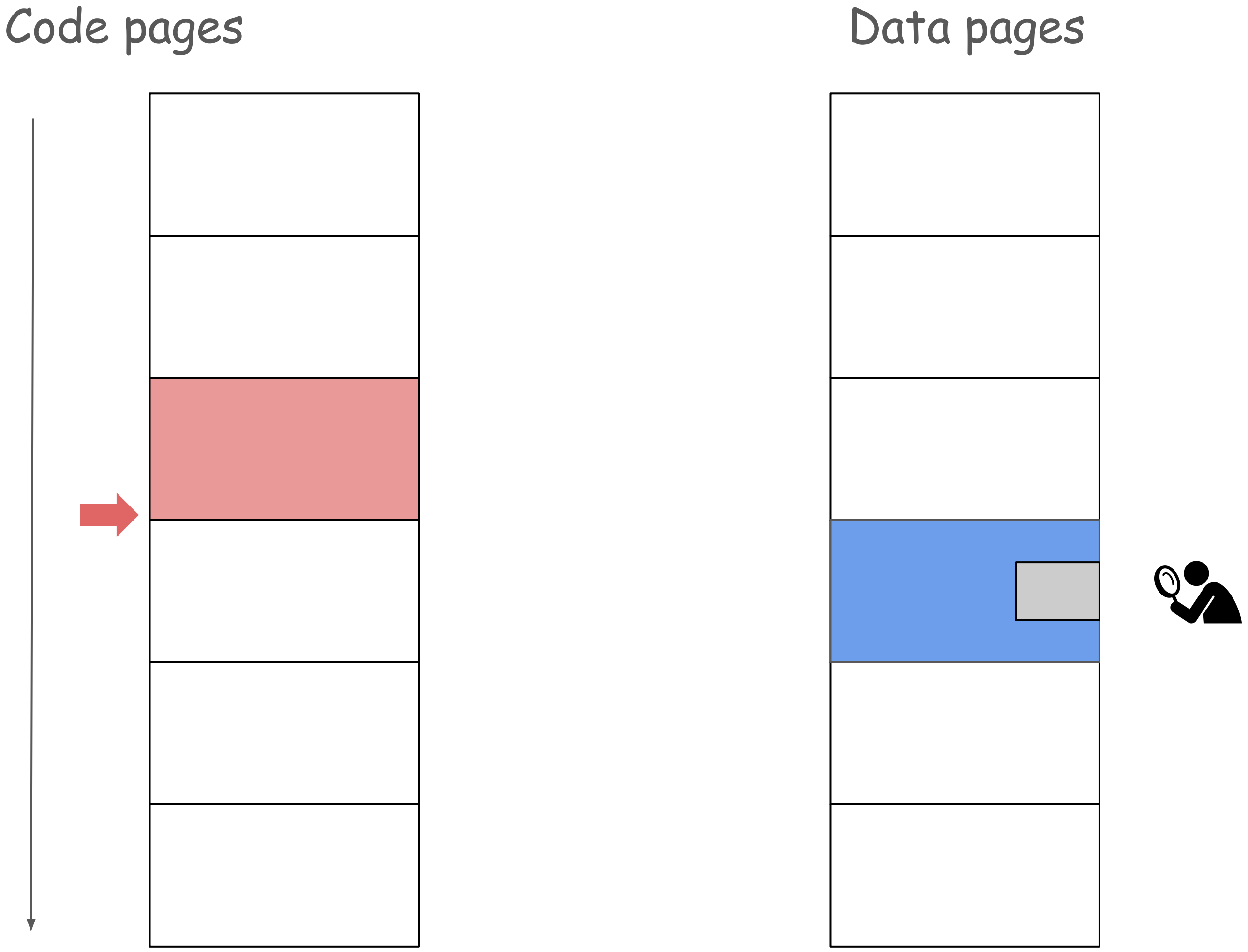
- Write to the 83rd ciphertext block within a page

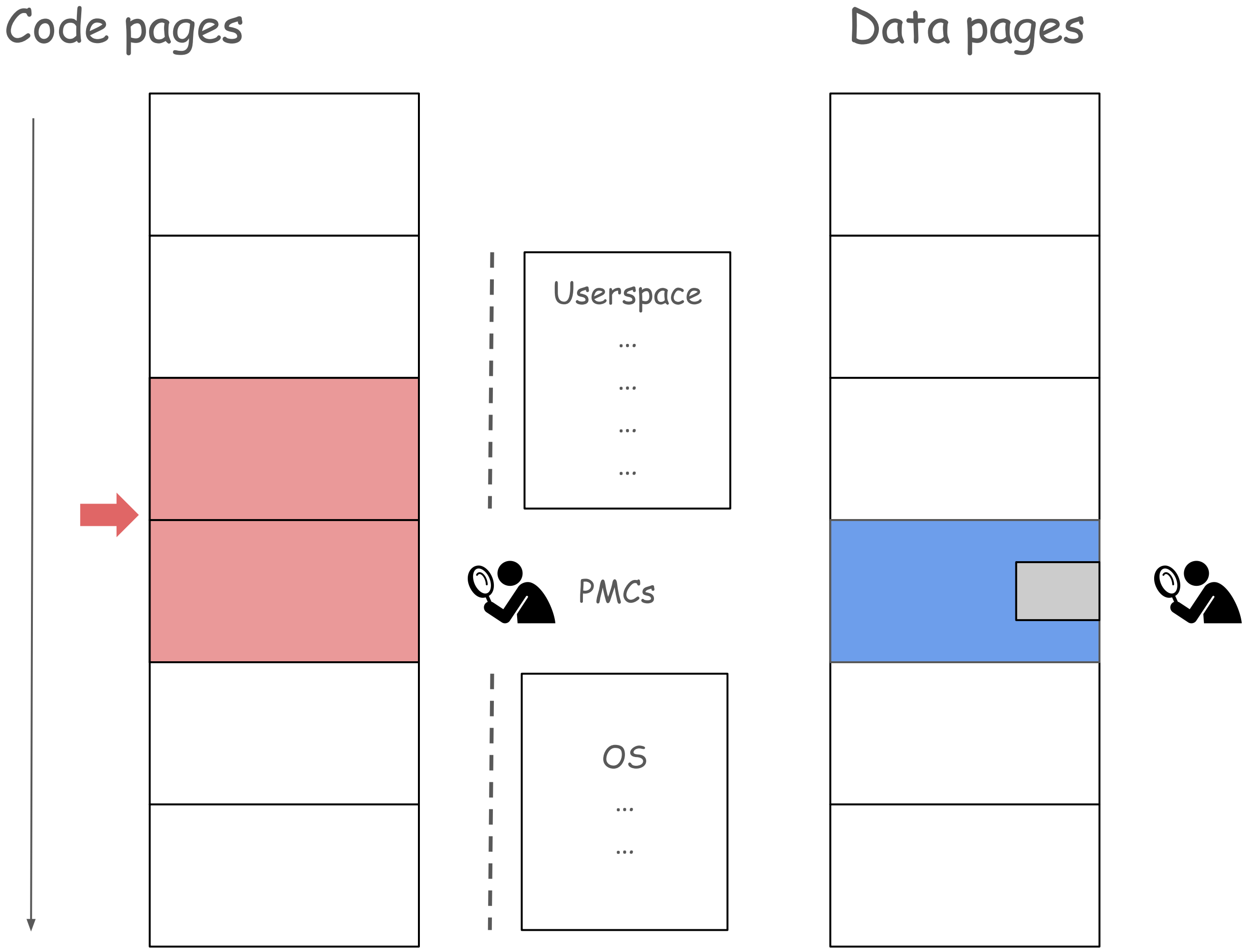
A 4kB page consists of 256 16B-ciphertext blocks



How does the framework work?


Code pages
Data pages
Cachelines,
Ciphertext Blocks


Code pages
Data pages


Code pages
Data pages
Userspace
...
...
...
...
PMCs
OS
...
...

Framework - Temporal resolution

Trigger Point: From guest VM's execution to the malicious hypervisor

- ❑ Page Fault Handler - by modifying PTEs in the host page table
- ❑ Timer Interrupt Handler - by setting an APIC timer to generate interrupts
  - Everytime the VM executes one single instruction, the APIC timer expires -> VM receives the interrupt -> Back to the hypervisor
  - Intel TDX mitigates it by checking the execution window of the VM
  - Slow, must be assisted with binary analysis

Framework - Reduce runtime noise

- ❏ Binary analysis
- ❏ Denoise environment (Offline other cores, Fix CPU frequency...)
- ❏ Heuristic strategies (The range of gPA, PMCs for guest OS)
- ❏ I/O / network patterns (Can be observed by Qemu)
- ❏ Special features (JIT -> W+X pages)

Mitigations

Coding Practice … is hard

1. Constant-time implementation
   - Put every “important code” within one cache line (64B)
   - Remove secret-dependant branches

2. Ciphertext side channel
   - Constant-time is not enough

- White Paper | Technical Guidance for Mitigating Effects of Ciphertext Visibility Under AMD SEV

Secure TSC

> Secure TSC allows guests to securely use RDTSC/RDTSCP instructions as the parameters being used cannot be changed by hypervisor once the guest is launched. More details in the AMD64 APM Vol 2, Section "Secure TSC".

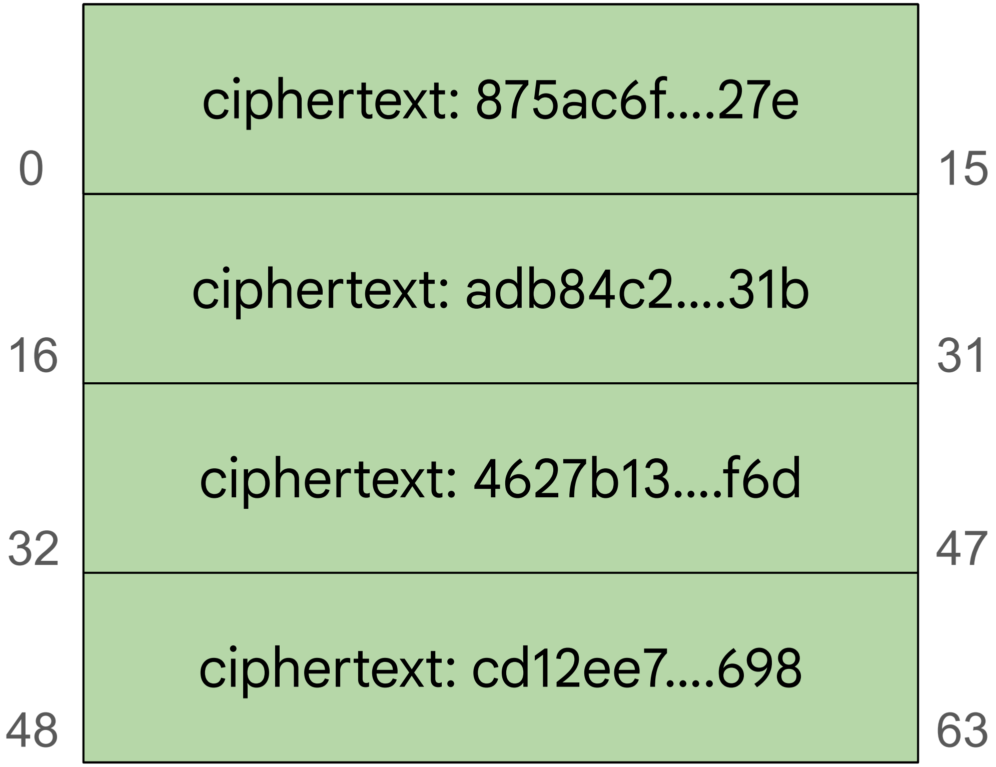
- **The victim knows the application's runtime**
- The threshold is hard to get though.
    - the attacker can make the attack faster by combining with many strategies

https://lwn.net/Articles/979296/

Full ORAM vs. DP-ORAM

- Several of our applications of interest, e.g. PHH, offer a per-user guarantee
  - We can aspire to offer make leakage DP, as opposed to completely remove it
- Example:
  - Full ORAM makes any two sequences of memory accesses indistinguishable
  - DP-ORAM makes any two neighboring of memory accesses indistinguishable
  - While not in general, this relaxation can be useful to mitigate PHH

Ciphertext Side-channel


ciphertext: 875ac6f....27e
ciphertext: adb84c2....31b
ciphertext: 4627b13....f6d
ciphertext: cd12ee7....698
0
15
16
31
32
47
48
63

"As AES uses a 16B (byte) block size for encryption, the XOR-Encrypt-XOR operation ensures that identical plaintext will encrypt to different ciphertext at different 16B locations in memory.

However, the same plaintext at the same location in memory will always encrypt to the same ciphertext value."

White Paper | Technical Guidance for Mitigating Effects of Ciphertext Visibility Under AMD SEV