EPTI: Efficient Defense against *Meltdown* Attack for Unpatched VMs

Zhichao Hua, Dong Du, Yubin Xia, Haibo Chen, Binyu Zang

Institute of Parallel and Distributed Systems (IPADS)
Shanghai Jiao Tong University

Meltdown

Break user/kernel isolation

- Allow attacker to read arbitrary kernel data

Hardware bug in architecture

- Hard to be fixed by micro-code patch

Exist in almost all Intel CPUs produced in past 20 years

Meltdown

key = 0x01

Mapped with **kernel privilege** in page table

Kernel

User

Access **key**

Permission Error!

Meltdown

key = 0x01

Mapped with **kernel privilege** in page table

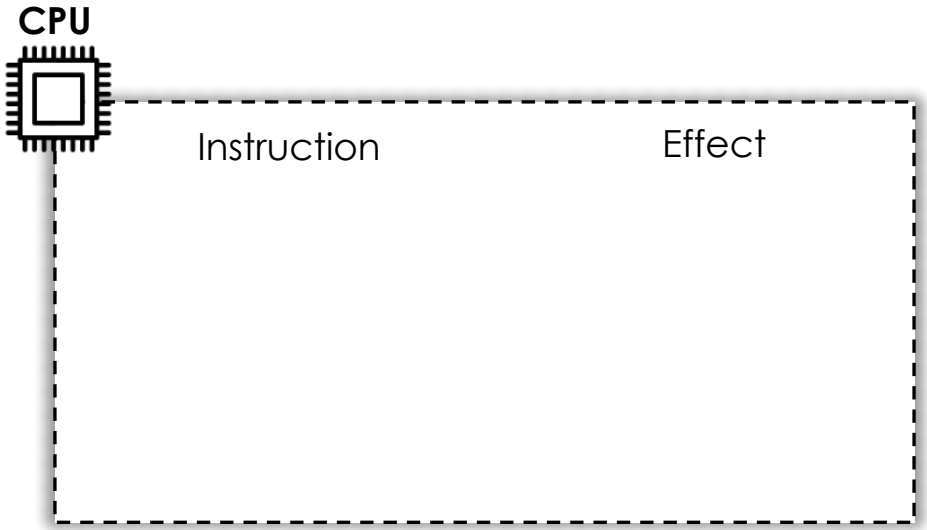
Kernel

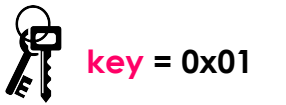
User

Meltdown

Can access **key**!

Meltdown



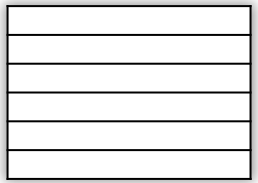


Kernel

User

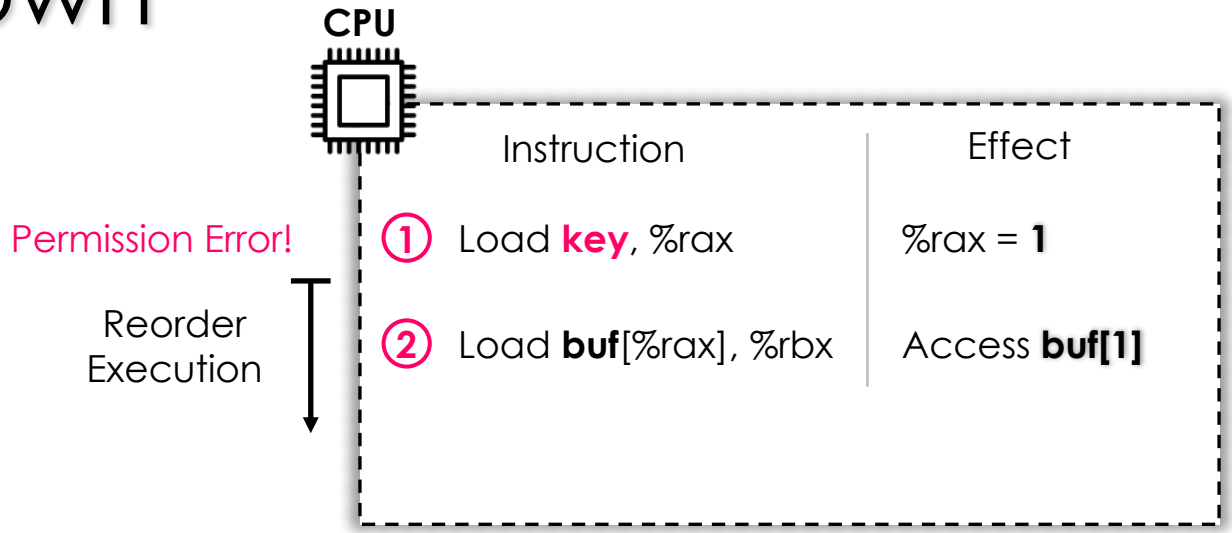
Meltdown

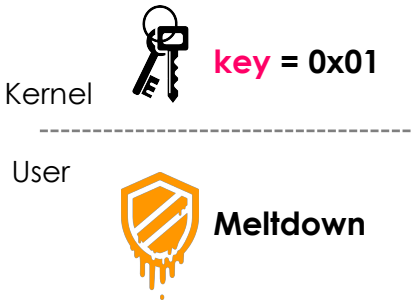
Cache

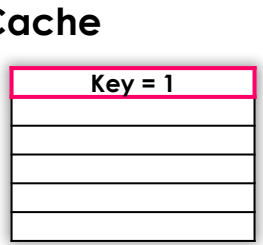


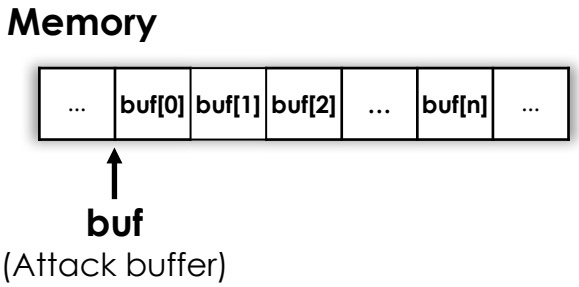
Memory

Meltdown

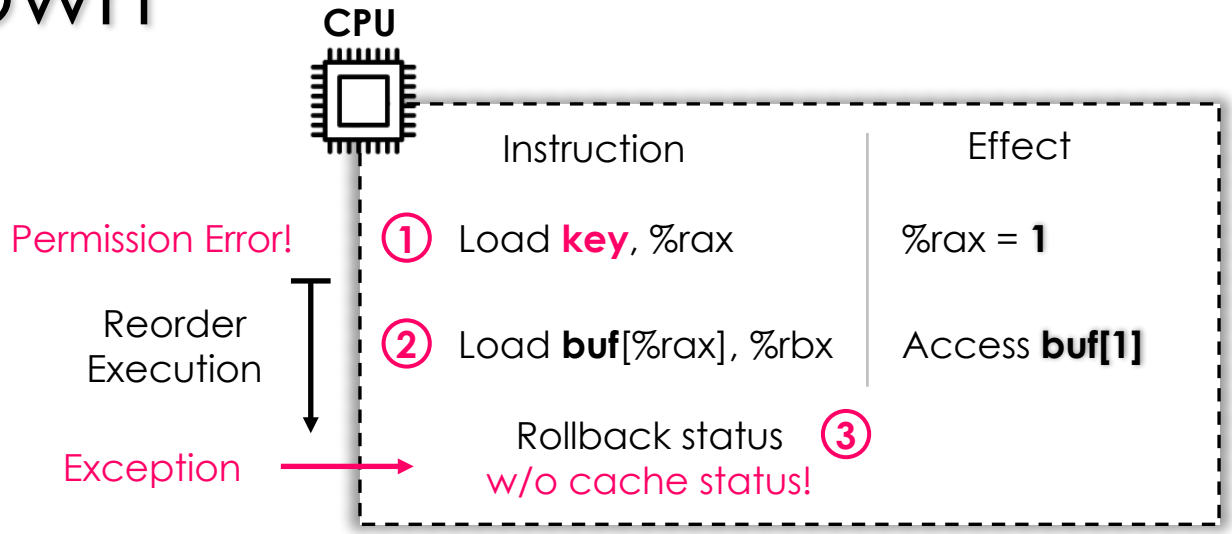

CPU
Instruction
Effect
Permission Error!
① Load key, %rax
%rax = 1
Reorder Execution
② Load buf[%rax], %rbx
Access buf[1]

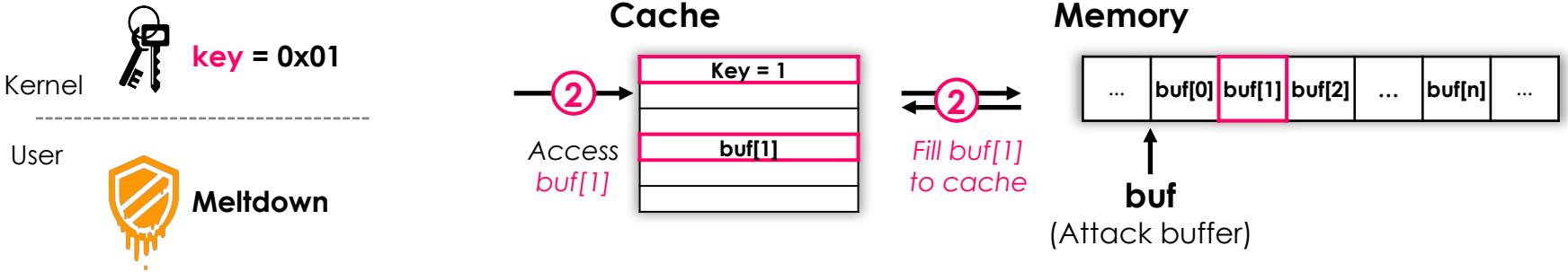

Kernel
key = 0x01
User
Meltdown


Cache
Key = 1

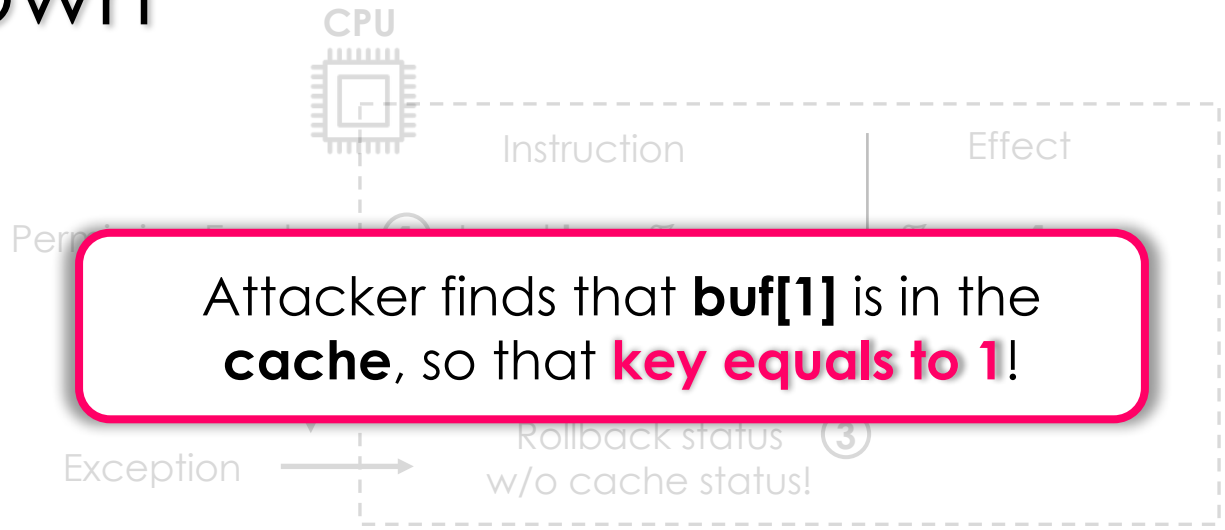

Memory
...
buf[0]
buf[1]
buf[2]
...
buf[n]
...
buf
(Attack buffer)

Meltdown

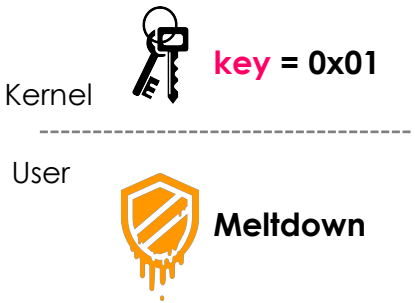

CPU
Instruction
Effect
Permission Error!
① Load key, %rax
%rax = 1
Reorder Execution
② Load buf[%rax], %rbx
Access buf[1]
Exception
Rollback status ③
w/o cache status!

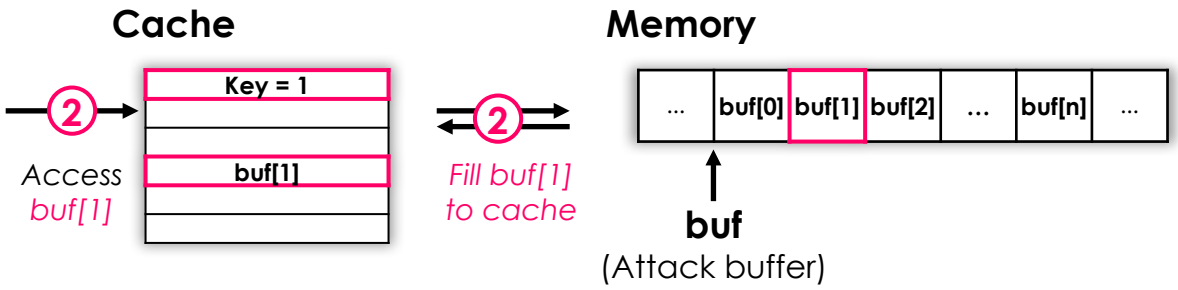

Kernel
key = 0x01
User
Meltdown
Cache
Key = 1
buf[1]
② Access buf[1]
② Fill buf[1] to cache
Memory
... buf[0] buf[1] buf[2] ... buf[n] ...
buf
(Attack buffer)

Meltdown



Attacker finds that **buf[1]** is in the **cache**, so that **key equals to 1**!





Existing Solution

KPTI (Kernel Page Table Isolation)

- Two page tables for user and kernel space
  - User page table only maps user space
  - Kernel page table maps both user and kernel space
- Switch the page table during user/kernel switching
  - Add latency to syscalls, signal handler,…

Not suitable for the cloud environment

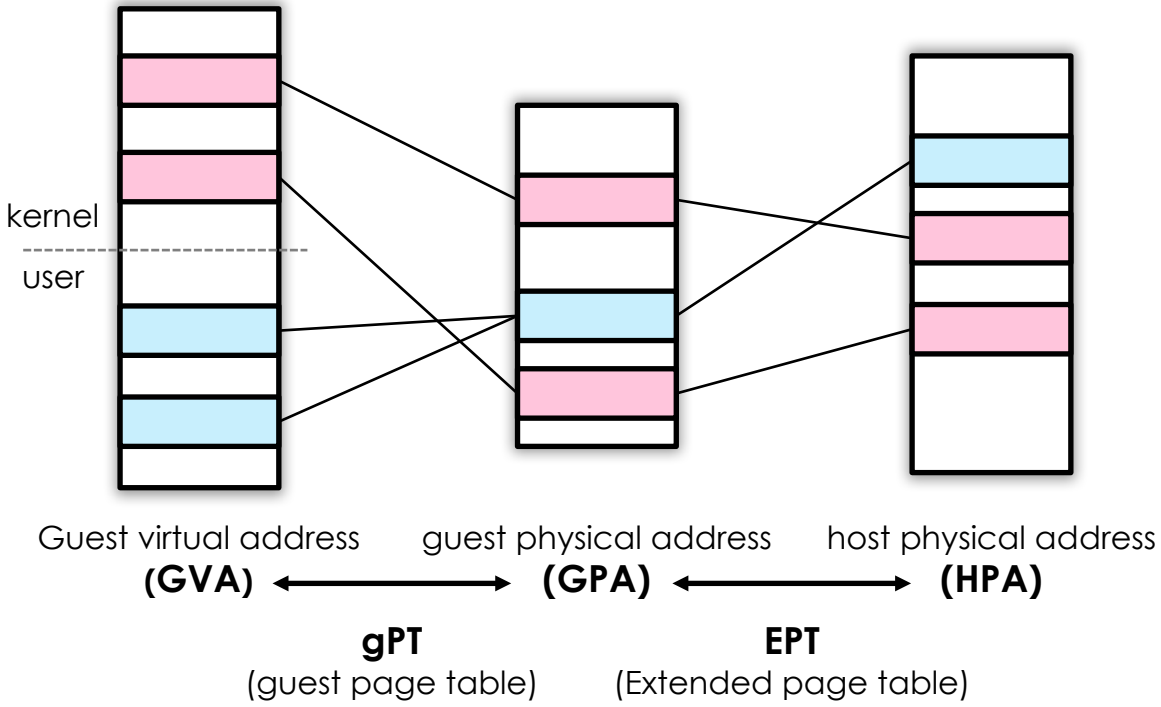
KPTI

	KPTI
Defend against Meltdown	**Yes**
Patch without manual effort	No
Independent on kernel version	No
Without rebooting	No
Performance overhead	Moderate

KPTI vs. EPTI (Our Solution)

	KPTI	EPTI
Defend against Meltdown	**Yes**	**Yes**
Patch without manual effort	No	**Yes**
Independent on kernel version	No	**Yes**
Without rebooting	No	**Yes**
Performance overhead	Moderate	**Low**

Address Translation in Cloud VMs



gPT translates GVA to GPA

- Controlled by the **guest**

EPT translates GPA to HPA

- Controlled by the **hypervisor**

EPT Switching

Switching EPT directly in the guest VM

- Hardware functionality provided by Intel (with *VMFUNC* instruction)
- Select an EPT from a list (configured by the hypervisor)
- No trap to the hypervisor during the switching

Performance characteristics of EPT switching

- No TLB flush during switching
- Low latency
  - About 160 cycles

EPT Switching

Switching EPT directly in the guest

- Hardware functionality provided by Intel (with *VMFUNC* instruction)

Use **two EPTs** to isolate the user and kernel space:
EPT-k for the kernel and **EPT-u** for the user

Performance characteristics of EPT switching

- No TLB flush during switching
- Low latency
  - About 160 cycles

Challenges

How to construct the EPT-k and EPT-u to isolate user and kernel space?

- Hypervisor knows limited semantics of the guest

How to achieve high performance?

- Getting guest semantics needs a lot of traps

How to provide seamless protection?

- Enable the protection without rebooting the guest

EPTI

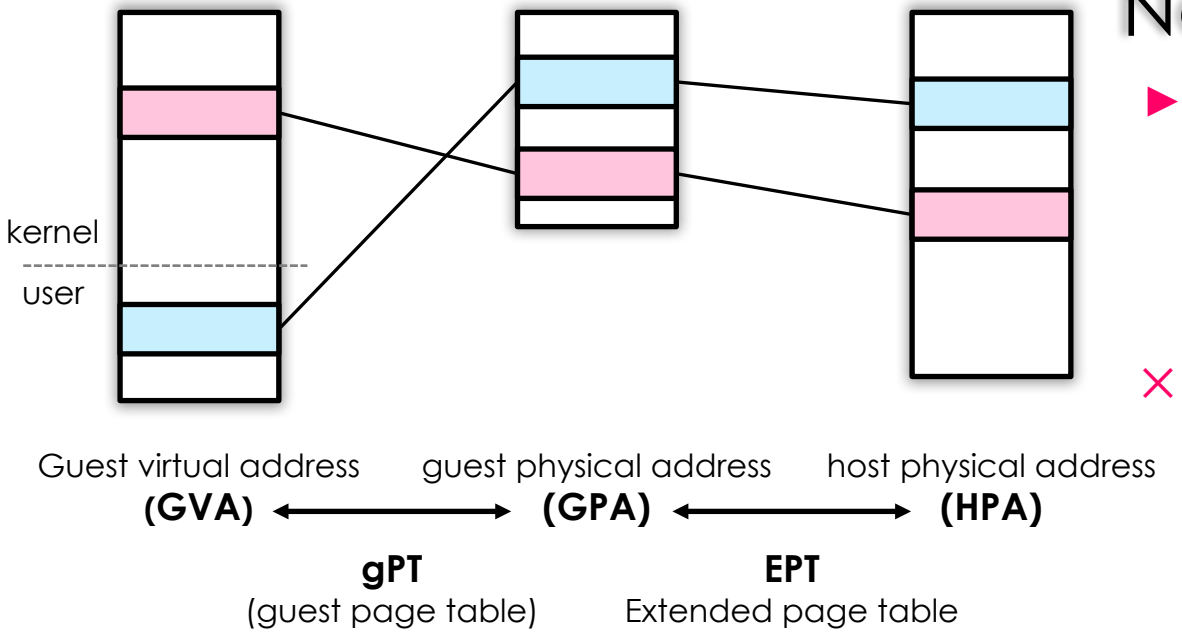
EPT-based kernel space isolation

Seamless protection

EPT-based Kernel Space Isolation

First try: directly remove kernel mapping in EPT-u

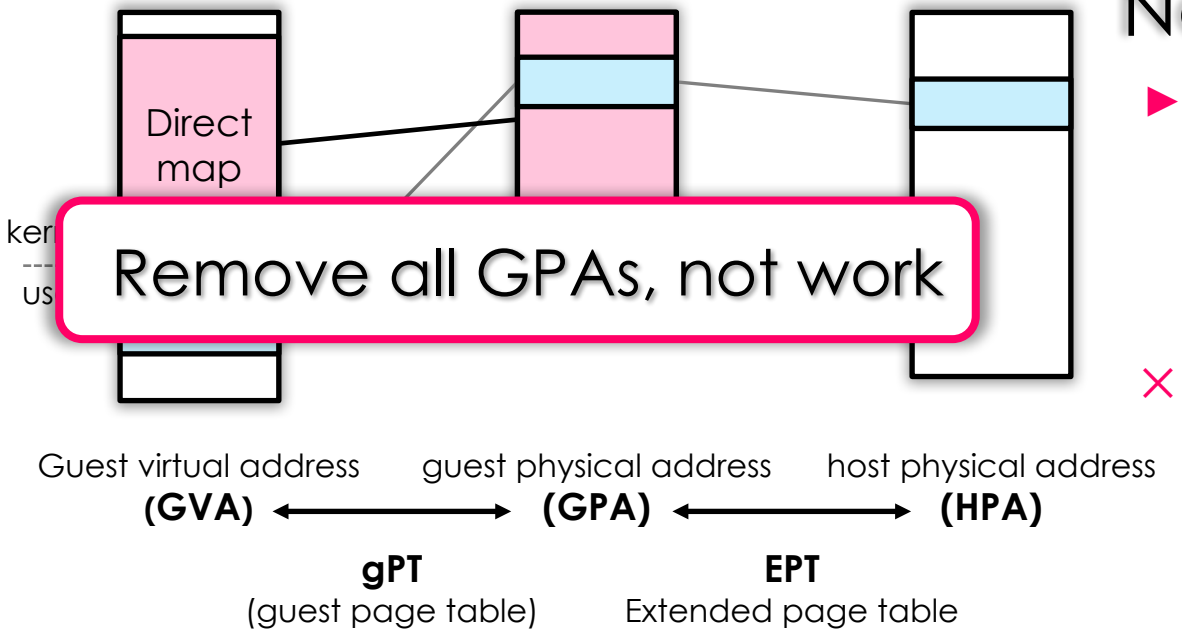
EPT-based Kernel Space Isolation



Naïve method

- ▶ Remove GPA-to-HPA mapping for kernel in EPT-u
- × Cannot distinguish kernel-used and user-used GPA

EPT-based Kernel Space Isolation



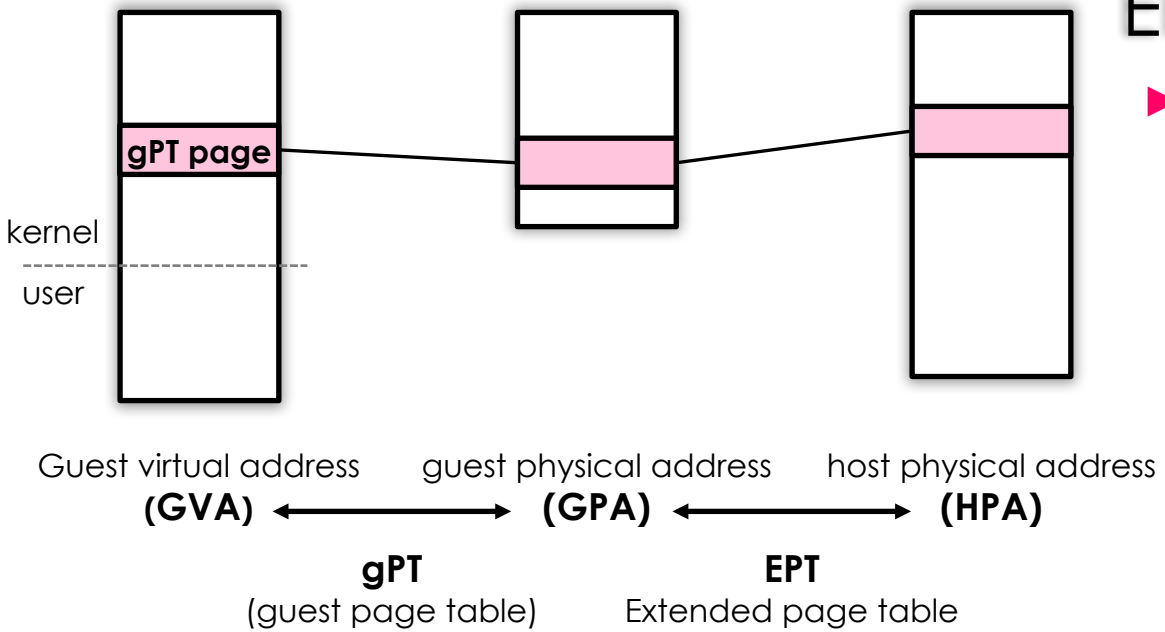
Naïve method

- ▶ Remove GPA-to-HPA mapping for kernel in EPT-u
- × Cannot distinguish kernel-used and user-used GPA

EPT-based Kernel Space Isolation

Second try: zero the guest page table for kernel space in EPT-u

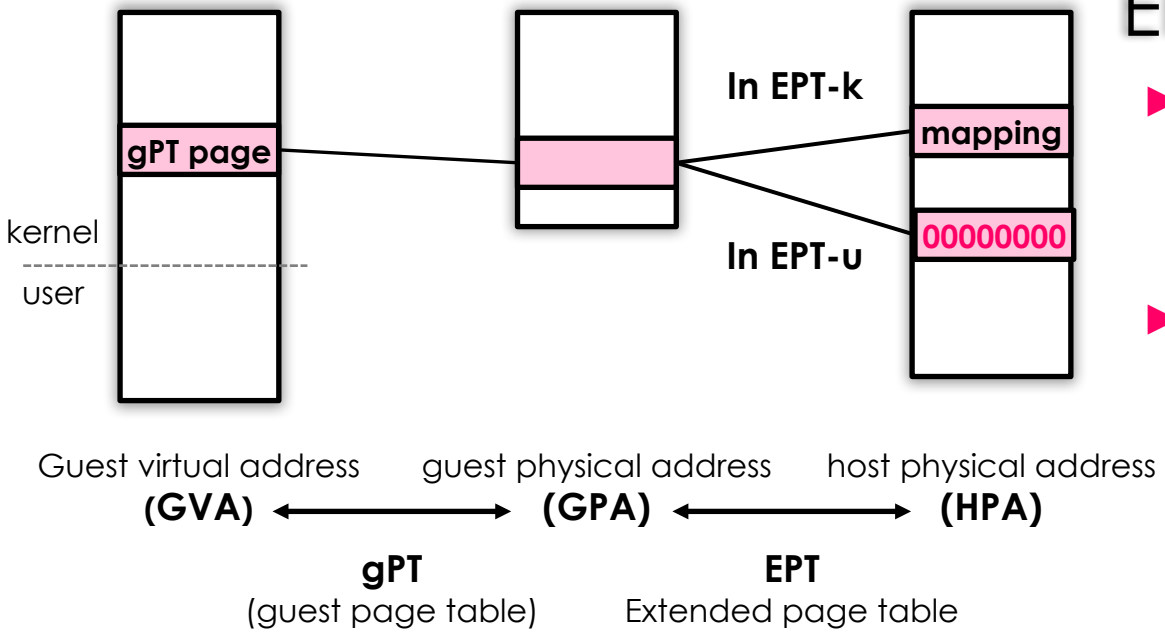
EPT-based Kernel Space Isolation



EPTI method

- Zero GVA-to-GPA mapping for kernel

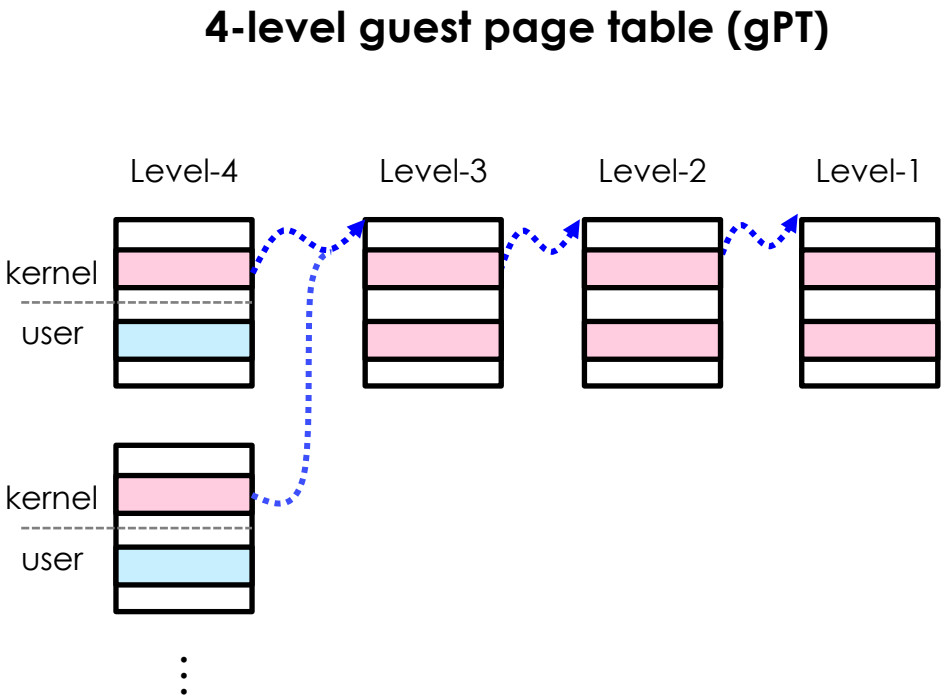
EPT-based Kernel Space Isolation



EPTI method

- Zero GVA-to-GPA mapping for kernel
- Remap gPT page which controls kernel mapping to a zeroed page in EPT-u

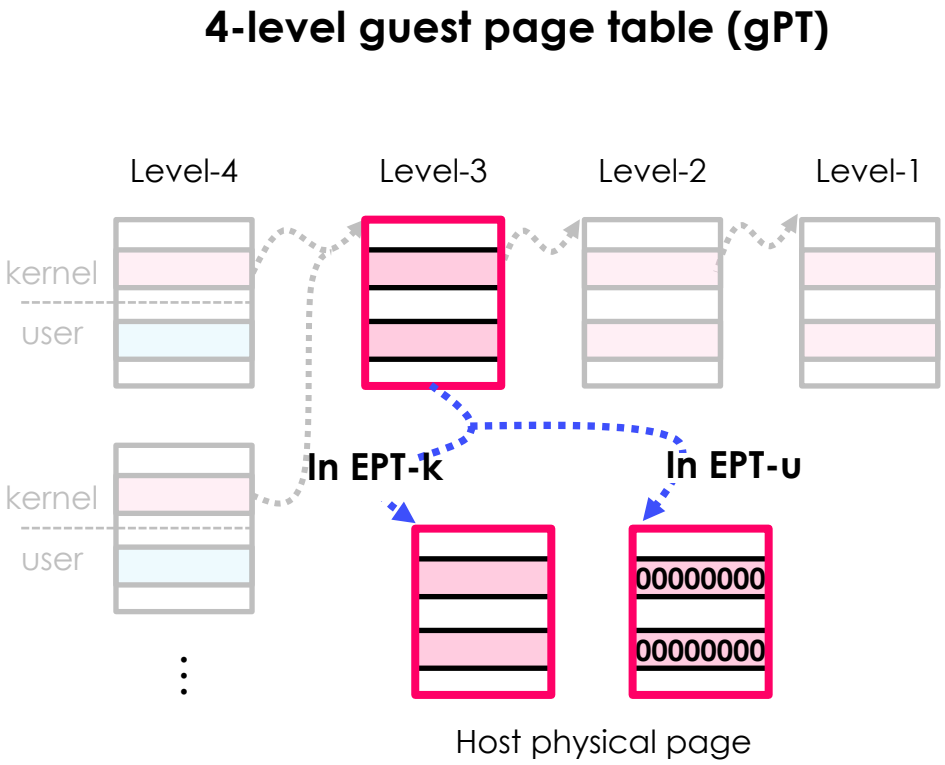
EPT-based Kernel Space Isolation



Remap guest level-3 page table page (gL3)

- All processes share the same level-3 page table page for kernel mapping (kernel gL3)

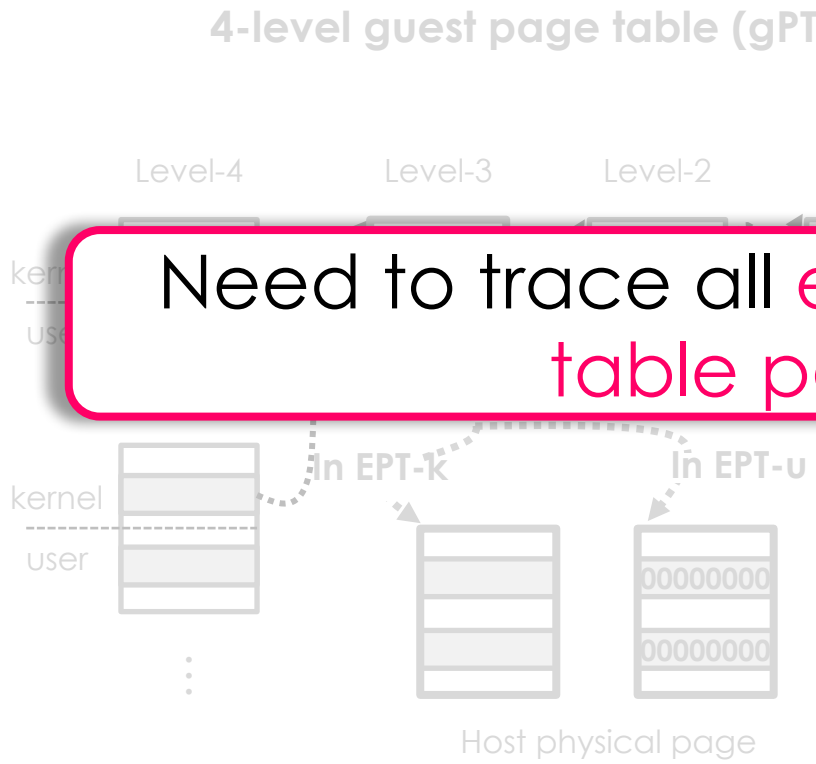
EPT-based Kernel Space Isolation



Remap guest level-3 page table page (gL3)

- All processes share the same level-3 page table page for kernel mapping (kernel gL3)
- Remap kernel gL3 to a zeroed host physical page in EPT-u

EPT-based Kernel Space Isolation



Need to trace all enabled kernel level-3 page table pages (kernel gL3)

Remap guest level-3 page table (gL3)

- All processes share the
- Remap kernel gL3 to a zeroed host physical page in EPT-u

Tracing Kernel gL3

Use trap to get kernel gL3 in hypervisor

- Trap guest load-CR3 operation
  - To get all guest level-4 page table pages
- Trap write operation of guest level-4 page table page
  - To get all enabled kernel guest level-3 page table pages (kernel gL3)

Too many traps will hurt the performance

Tracing Kernel gL3

Three optimizations to reducing traps

- Write protection method for access/dirty bit updating
- Selectively trap load-CR3 operation
- Trap modification on gL3 only

Access/Dirty Bits Updating

CPU updates access/dirty bit

- Each page table entry has access/dirty bits (A/D bits)
- Update when the entry is used to perform address translation

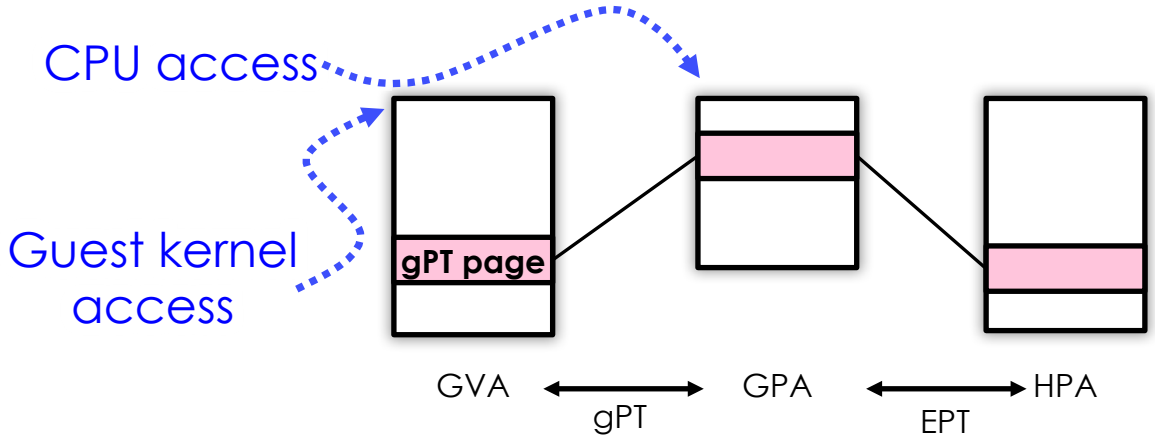
Need to trap kernel modifications on guest level-4 page table pages (gL4)

- Map gL4 as write protected in EPT
- All guest memory accesses update A/D bits in gL4 and cause a **trap**

EPTI Write-protection Method (Opt-1)

CPU and guest have different access paths

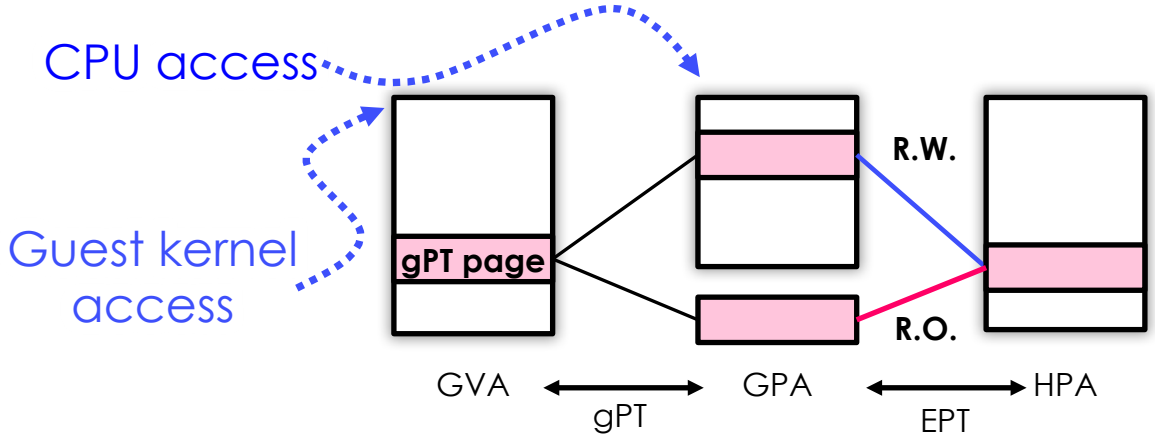
- CPU modifies guest page table (gPT) with GPA
  - Map GPA of target gPT page as R.W. in EPT
- Guest kernel modifies gPT with GVA



EPTI Write-protection Method (Opt-1)

CPU and guest have different access paths

- CPU modifies guest page table (gPT) with GPA
  - Map GPA of target gPT page as R.W. in EPT
- Guest kernel modifies gPT with GVA
  - Map GVA of target gPT page to a new GPA and map it as R.O. in EPT



Load-CR3 Operation in Guest VM

CR3 contains the guest page table pointer

- Change during the process switching

EPTI needs to trace the new enabled guest page table

- Trap load-CR3 operation
- Loading an old page table also causes a trap

Selectively Trap Load-CR3 (Opt-2)

Hardware feature *target_cr3_value*

- loading CR3 value same as the target_cr3_value will not cause trap
- Host can configure four target_cr3_values

Disable trap on loading frequently-used CR3 value

- Write the most frequently-used CR3 value into the target_cr3_value field

Modification on gL4

Guest level-4 page table (gL4) contains both user and kernel mapping

- Adding either kernel gL3 or user gL3 needs to modify the gL4

EPTI needs to trap "adding kernel gL3"

- Trap modification on gL4
- "Adding user gL3" also writes gL4 and causes a trap

Trap Modification on gL3 Only (Opt-3)

Kernel address space consists of different regions

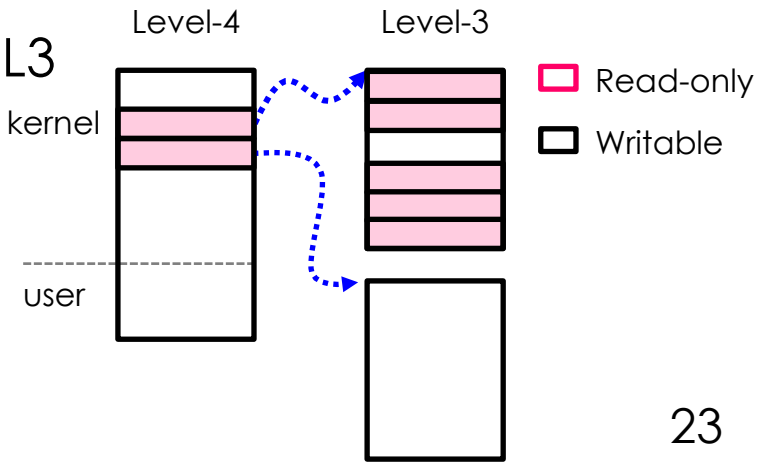
- E.g., direct_map region, text region, vmalloc region, …
- All the regions either have fixed length or increase continuously

A new kernel gL3 is added until the last entry of one existing kernel gL3 is used

Trap Modification on gL3 Only (Opt-3)

Trap gL3 only until detecting a new gL3 will be added

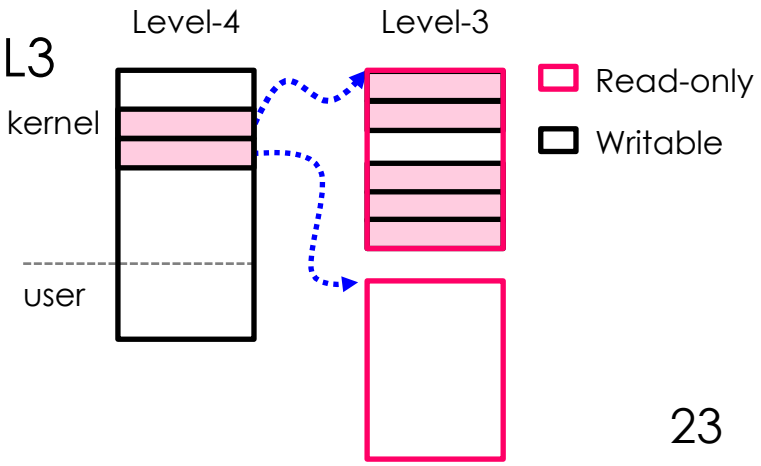
- Step-1: trap modification on gL3
- Step-2: when the last entry of gL3 is used, start to trap modification on gL4
- Go to step-1 when detect the new gL3



Trap Modification on gL3 Only (Opt-3)

Trap gL3 only until detecting a new gL3 will be added

- Step-1: trap modification on gL3
- Step-2: when the last entry of gL3 is used, start to trap modification on gL4
- Go to step-1 when detect the new gL3



Malicious EPT Switching

Intel allows the EPT switching to be performed in guest user mode

- Attacker can switch to EPT-k, which contains the kernel mapping, and perform Meltdown attack

Make EPT-k to be useless in user mode

- All GPAs are mapped as non-executable in EPT-k except the kernel code or kernel modules
- Switching to EPT-k in user mode causes trap

EPTI

EPT-based kernel space isolation

Seamless protection

Seamless Protection

Dynamically trampoline injecting

- Trampoline switches the EPT-k and EPT-u

Seamless protection method

- No need to reboot the guest by leveraging live migration

More details in the paper

Performance Evaluation

Hardware platform

- Intel Core i7-7700 (4 cores * 2 thread)
- 16GB memory

Software environments

- Linux 4.9.75 + KVM for host
- Linux 4.9.75 for guest
  - Other Linux versions are also tested (more results in the paper)

Guest configurations

- 4 vCPUs (each is pinned on one physical thread)
- 8GB memory

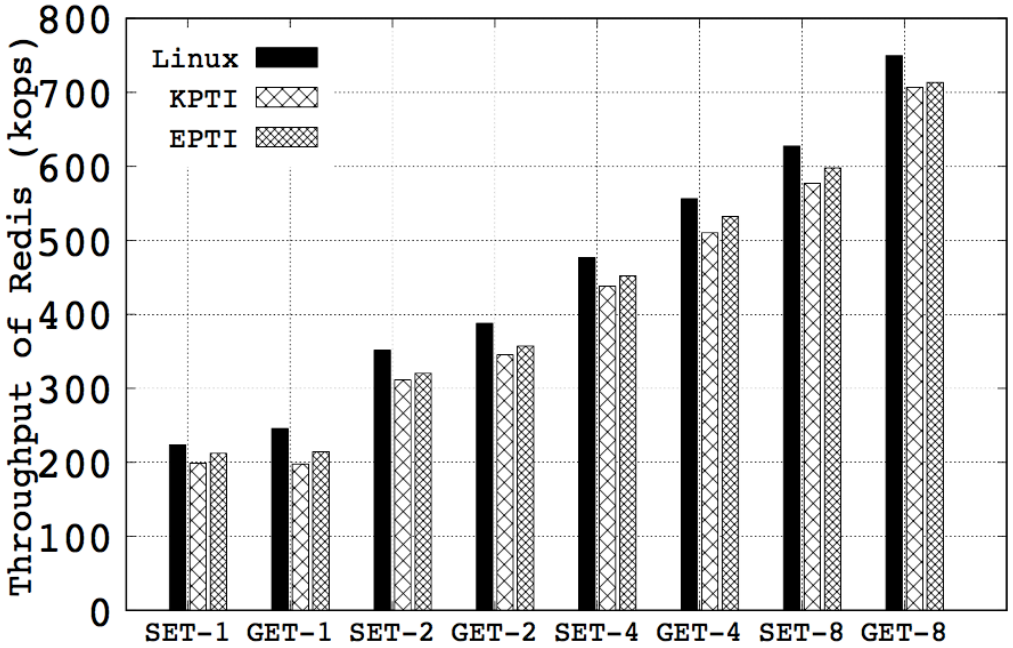
LMBench

Operation(µs)	Linux	KPTI	KPTI (normalized)	EPTI	EPTI (normalized)
Null syscall	0.04	0.16	4x	0.12	3x
Null I/O	0.07	0.2	2.86x	0.16	2.28x
Open/Close	0.70	0.93	1.33x	0.83	1.19x
Signal Handle	0.68	0.81	1.19x	0.76	1.12x
Fork syscall	72.9	79	1.08x	75	1.03x
E>					
C					

EPTI has lower overhead of syscall latency, which is the main overhead of KPTI

Redis

Test sets: get and set operations of redis-benchmark



KPTI has 12% overhead on average

EPTI has 7% overhead on average

Conclusion

EPTI provides a new Meltdown defense method in cloud

- Use two EPTs (EPT-k and EPT-u) to isolate user/kernel space

High usability

- Protect unpatched guest
- No dependence on kernel version
- No need to reboot the guest

Low performance overhead

Thanks

Institute of Parallel And Distributed Systems (IPADS)
Shanghai Jiao Tong University

http://ipads.se.sjtu.edu.cn