HOST 2019 Tutorial on:

Secure Processor Architectures in the Era of Spectre and Meltdown

Jakub Szefer

Wenjie Xiong

Shuwen Deng

Dept. Of Electrical Engineering
Yale University

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

HOST 2019 Tutorial on:

Secure Processor Architectures in the Era of Spectre and Meltdown

Slides and information at:

http://caslab.csl.yale.edu/tutorials/host2019/

WiFi Information:

Network: Hilton-Meeting, Password: HOST2019

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Tutorial Outline & Schedule

15:30 – 16:10	**Secure Processor Architectures**
16:10 – 16:20	**Break**
16:20 – 17:10	**Secure Processor Caches**
17:10 – 17:20	**Break**
17:20 – 18:00	**Transient Execution Attacks and Mitigations**
18:00	**Wrap Up**

Slides and information at:
http://caslab.csl.yale.edu/tutorials/host2019/

WiFi Information:
Network: Hilton-Meeting, Password: HOST2019

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Processor Architectures

Jakub Szefer Wenjie Xiong Shuwen Deng

Dept. Of Electrical Engineering
Yale University

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Processor Architectures

Memory Protection

Side-Channels Threats and Protections

Speculative or Transient Execution Threats

Principles of Secure Processor Architecture Design

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Principles of Computer Architecture

Traditional computer architecture has six principles regarding processor design:

- Caching
  - E.g. caching frequently used data in a small but fast memory helps hide data access latencies.
- Pipelining
  - E.g. break processing of an instruction into smaller chunks that can each be executed sequentially reduces critical path of logic and improves performance.
- Predicting
  - E.g. predict control flow direction or data values before they are actually computed allows code to execute speculatively.
- Parallelizing
  - E.g. processing multiple data in parallel allows for more computation to be done concurrently.
- Use of indirection
  - E.g. virtual to physical mapping abstracts away physical details of the system.
- Specialization
  - E.g. custom instructions use dedicated circuits to implement operations that otherwise would be slower using regular processor instructions.

What are principles for secure architectures?

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Processor Architectures

Secure Processor Architectures extend a processor with hardware (and related software) features for protection of software

- Protected pieces of code and data are now commonly called Enclaves
  - But can be also Trusted Software Modules, whole Operating Systems, or Virtual Machines
- Focus on the main processor in the system
  - Others focus on co-processors, cryptographic accelerators, or security monitors
- Add more features to isolate secure software from other, untrusted software
  - Includes untrusted Operating System or Virtual Machines
  - Many also consider physical attacks on memory
- Isolation *should* cover all types of possible ways for information leaks
  - Architectural state
  - Micro-architectural state
  - Due to spatial or temporal sharing of hardware

Most recent threats, i.e. Spectre, etc.

Side and covert channel threats

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

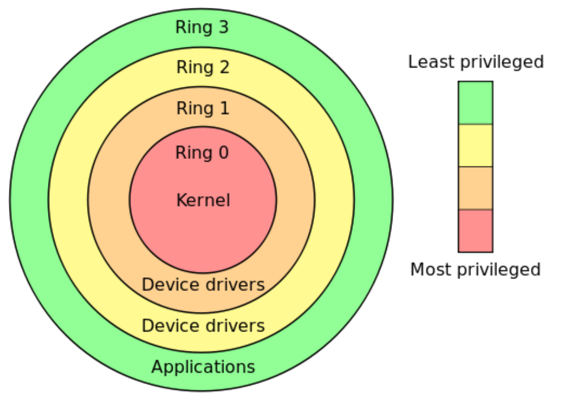
Baseline (Unsecure) Processor Hardware

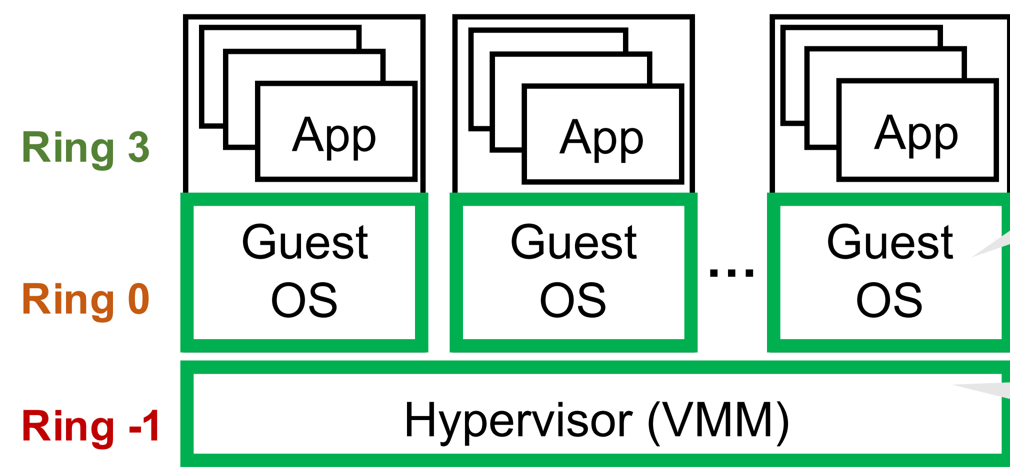
Typical computer system with no secure components nor secure processor architectures considers all the components as trusted:

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Baseline (Unsecure) Processor Software

Typical computer system uses ring-based protection scheme, which gives most privileges (and most trust) to the lowest levels of the system software:





Compromised or malicious **OS** can attack all the applications in the system.

Compromised or malicious **Hypervisor** can attack all the OSes in the system.

Image: https://commons.wikimedia.org/wiki/File:Priv_rings.svg

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Brief History of Secure Processor Architectures

Starting with a typical baseline processor, many secure architectures have been proposed

Starting in late 1990s or early 2000s, academics have shown increased interest in secure processor architectures:

XOM (2000), AEGIS (2003), Secret-Protecting (2005), Bastion (2010), NoHype (2010), HyperWall (2012), CHERI (2014), Sanctum (2016), Keystone (about 2017), MI6 (2018)

Commercial processor architectures have also included security features:

LPAR in IBM mainframes (1970s), Security Processor Vault in Cell Broadband Engine (2000s), ARM TrustZone (2000s), Intel TXT & TPM module (2000s), Intel SGX (mid 2010s), AMD SEV (late 2010s)

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Add New Privilege Levels

Modern computer systems define protections in terms of **privilege level** or protection rings, new privilege levels are defined to provide added protections.

Ring 3	Application code, least privileged.
Rings 2 and **1**	Device drivers and other semi-privileged code, although rarely used.
Ring 0	Operating system kernel.
Ring -1	Hypervisor or virtual machine monitor (VMM), most privileged mode that a typical system administrator has access to.
Ring -2	System management mode (SMM), typically locked down by processor manufacturer
Ring -3	Platform management engine, retroactively named “ring -3”, actually runs on a separate management processor.

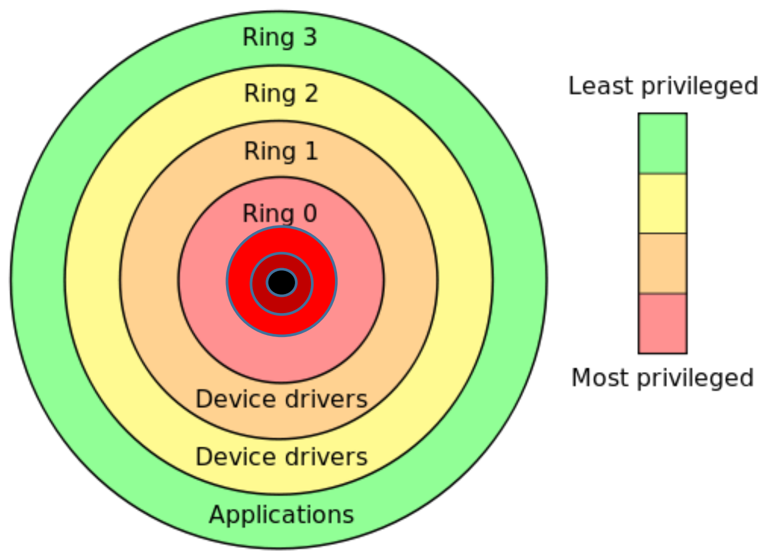


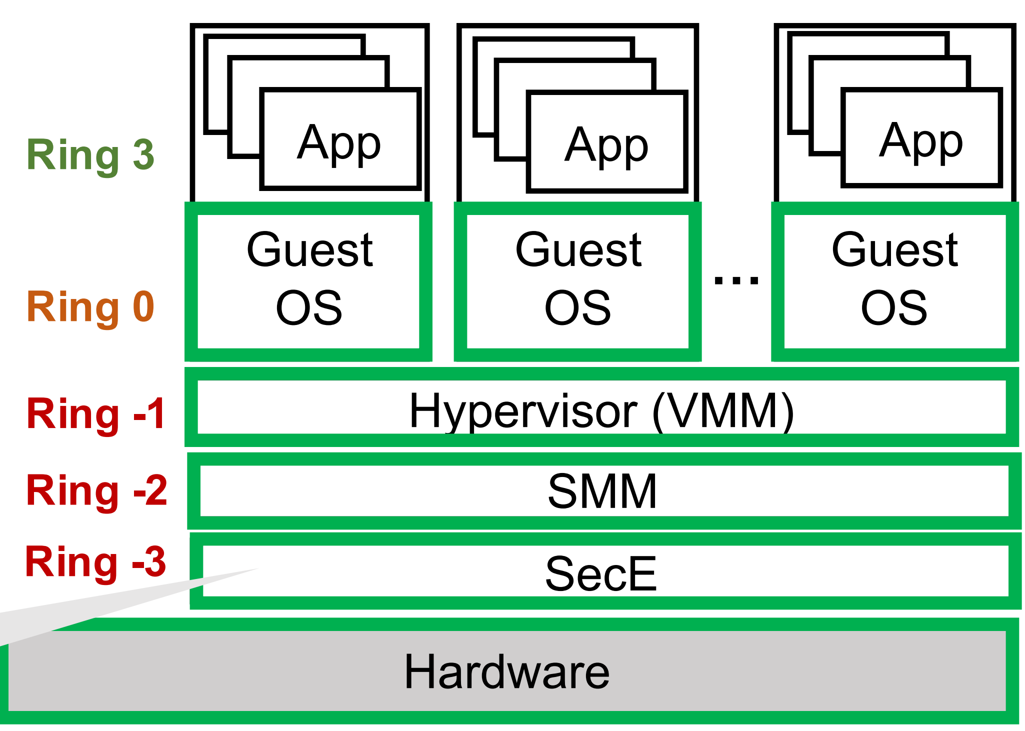
Image: https://commons.wikimedia.org/wiki/File:Priv_rings.svg

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Extend Linear Trust to the New Protection Levels

The hardware is most privileged as it is the lowest level in the system.

- There is a linear relationship between protection ring and privilege (lower ring is more privileged)
- Each component **trusts** all the software "below" it

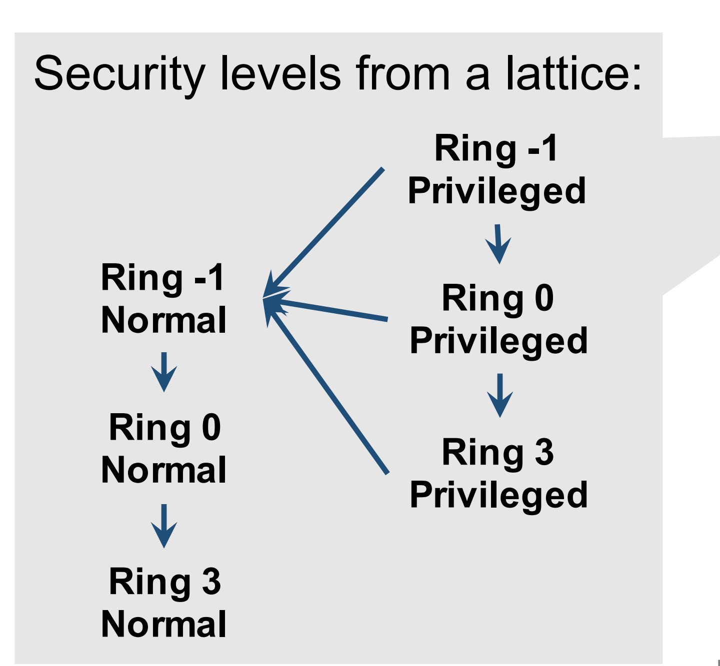


Security Engine (SecE) can be something like Intel's ME or AMD's PSP.

Add Horizontal Privilege Separation

New privileges can be made orthogonal to existing protection rings.

- E.g. ARM's TrustZone's "normal" and "secure" worlds
- Need privilege level (ring number) and normal / secure privilege



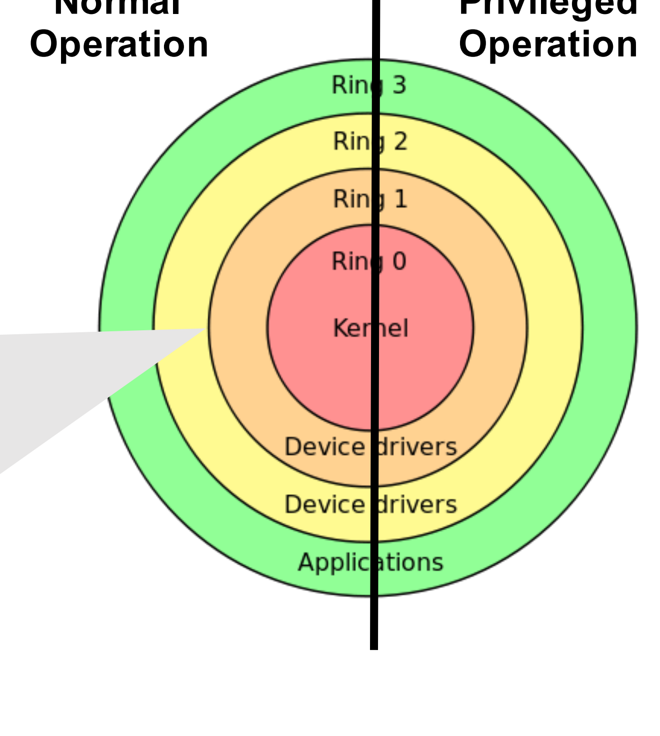


Image: https://commons.wikimedia.org/wiki/File:Priv_rings.svg

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Breaking Linear Hierarchy of Protection Rings

Examples of architectures that do and don't have a linear relationship between privileges and protection ring level:

Ring 3
App
Ring 0
Guest OS
Ring -1
HV
Ring -2
SMM
Ring -3
SecE
Hardware

Normal Computer

Ring 3
TSM
Ring 0
Guest OS
Ring -1
HV
Ring -2
SMM
Ring -3
SecE
Hardware

E.g. Bastion

Ring 3
Encl ave
Ring 0
Guest OS
Ring -1
HV
Ring -2
SMM
Ring -3
SecE
Hardware

E.g. SGX

Ring 3
App
Ring 0
Guest OS
Ring -1
HV
Ring -2
SMM
Ring -3
SecE
Hardware

E.g. SEV

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Providing Protections with a Trusted Processor Chip

Key to most secure processor architecture designs is the idea of **trusted processor chip** as the security wherein the protections are provided.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Limitations of the Trusted Processor Chip Assumption

Threats which are outside the scope of secure processor architectures:

- Bugs or Vulnerabilities in the TCB
- Hardware Trojans and Supply Chain Attacks
- Physical Probing and Invasive Attacks

TCB hardware and software is prone to bugs just like any hardware and software.

Modifications to the processor after the design phase can be sources of attacks.

At runtime hardware can be probed to extract information from the physical realization of the chip.

Threats which are underestimated when designing secure processor architectures:

- Side Channel Attacks

Information can leak through timing, power, or electromagnetic emanations from the implementation

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

TEE and TCB

The **Trusted Computing Base (TCB)** is the set of hardware and software that is responsible for realizing the TEE:

- TEE is created by a set of all the components in the TCB
- TCB is trusted to correctly implement the protections
- Vulnerability or successful attack on TCB nullifies TEE protections

- TCB is trusted
- TCB may not be trustworthy, if is not verified or is not bug free

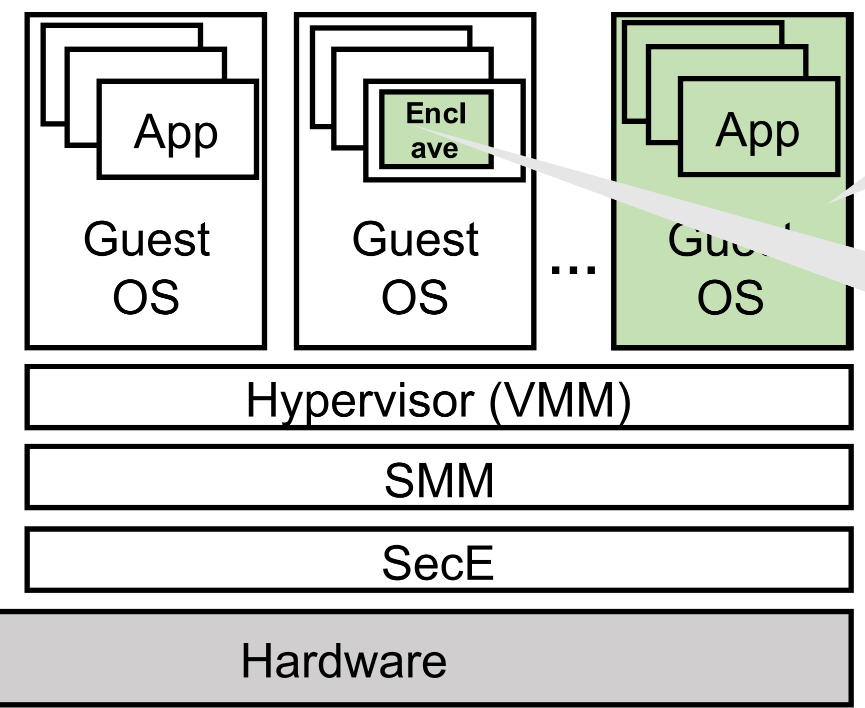
The goal of **Trusted Execution Environments (TEEs)** is to provide protections for a piece of code and data from a range of software and hardware attacks.

- Multiple mutually-untrusting pieces of protected code can run on a system at the same time

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

TEEs and Software They Protect

Different architectures mainly focus on **protecting Trusted Software Modules** (a.k.a. enclaves) or **whole Virtual Machines**.



Some TEEs have support for protecting whole virtual machines.

Other TEEs support Trusted Software Modules, a.k.a. enclaves

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Protections Offered by Secure Processor Architectures

Security properties for the TEEs that secure processor architectures aim to provide:

- Confidentiality
- Integrity

Confidentiality is the prevention of the disclosure of secret or sensitive information to unauthorized users or entities.

Integrity is the prevention of unauthorized modification of protected information without detection.

- Availability is usually not provided usually

Confidentiality and integrity protections are from attacks by other components (and hardware) not in the TCB. **There is typically no protection from malicious TCB.**

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Sample Protections Categorized by Architecture

Secure processor architectures break the linear relationship (where lower level protection ring is more trusted):

Privilege Levels	pre-protection rings	protection rings	Bastion	protection rings	HyperWall, AMD SEV	HyperWall, AMD SEV	AEGIX, XOM, SP, Intel SGX	not useful
Application	T	U	T	U	T	U	T	U
Operating System	T	T	U	U	T	T	U	U
Hypervisor	T	T	T	T	U	U	U	U
SMM								
SecE								

Trusted or Untrusted

Sample Architecture

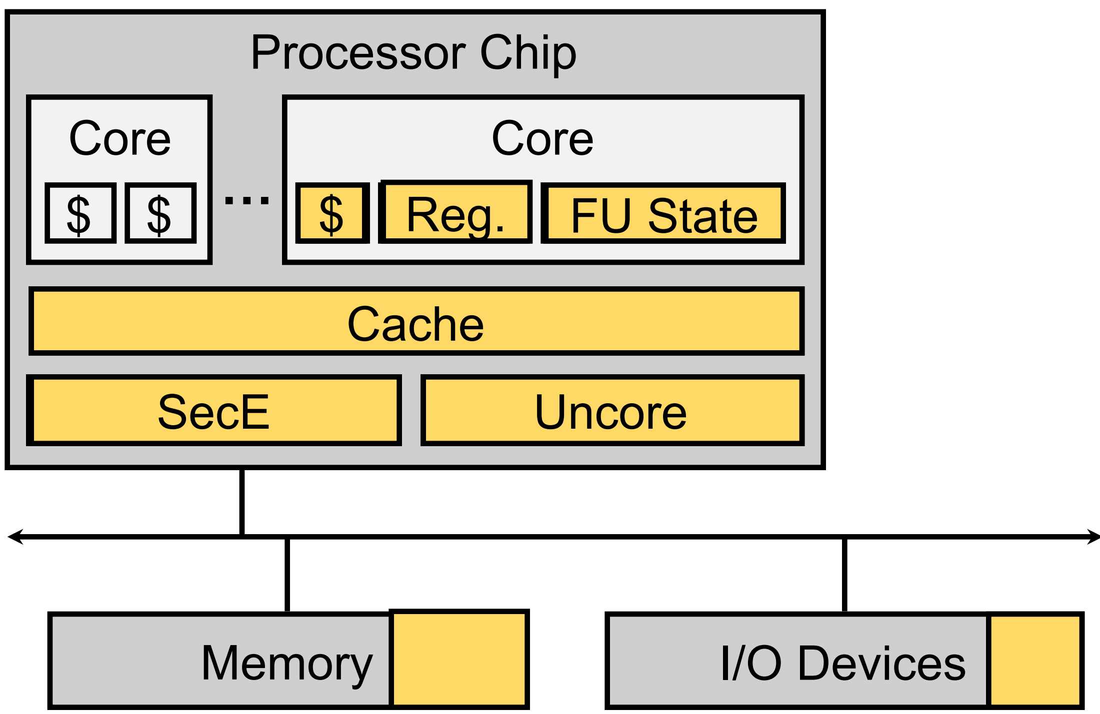
SMM and SecE are always trusted today, no architecture explores design where these levels are untrusted.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Protecting State of the Protected Software

Protected software's **state** is distributed throughout the processor. All of it needs to be protected from the untrusted components and other (untrusted) protected software.

- Protect memory through encryption and hashing with integrity trees
- Flush state, or isolate state, of functional units in side processor cores
- Isolate state in uncore and any security modules
- Isolate state in I/O and other subsystems

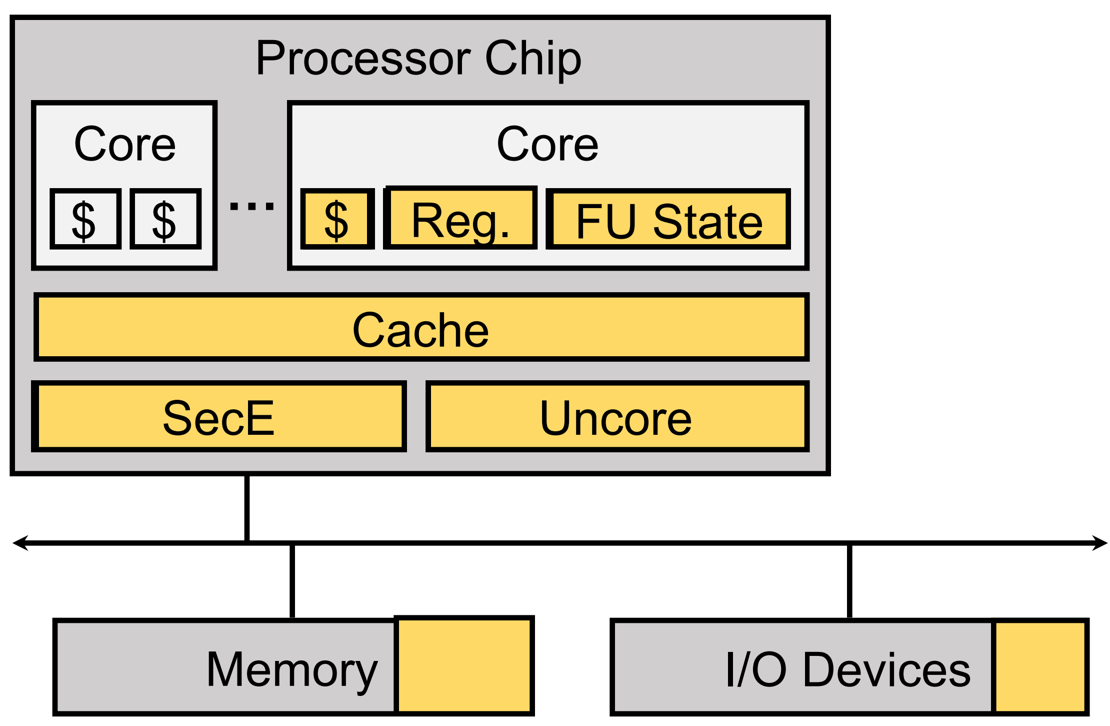


© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Ideal No Side-Effects Execution

Secure processor architectures ideally have **no side-effects which are visible to the untrusted components** whenever protected software is executing.

1. System is in some state before protected software runs
2. Protected software runs modifying system state
3. When protected software is interrupted or terminates the state modifications are erased



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

No Protections from Protected Software

The software (code and data) executing within TEE protections is assumed to be benign and not malicious:

- Goal of Secure Processor Architectures is to create minimal TCB that realizes a TEE within which the protected software resides and executes
- Secure Processor Architectures can not protect software if it is buggy or has vulnerabilities

Code bloat endangers invalidating assumptions about benign protected software.

Attacks from within protected software should be defended.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Hardware TCB as Circuits or Processors

Key parts of the hardware TCB can be implemented as dedicated circuits or as firmware or other code running on dedicated processor

- **Custom logic or hardware state machine:**
  - Most academic proposals
- **Code running on dedicated processor:**
  - Intel ME = ARC processor or Intel Quark processor
  - AMD PSP = ARM processor

Vulnerabilities in TCB "hardware" can lead to attacks that nullify the security protections offered by the system.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Ensuring Trustworthy TCB Execution

Trustworthiness of the TCB depends on the ability to monitor the TCB code (hardware and software) execution as the system runs.

TCB should be monitored to ensure it is trustworthy.

Monitoring of TCB requires mechanisms to:

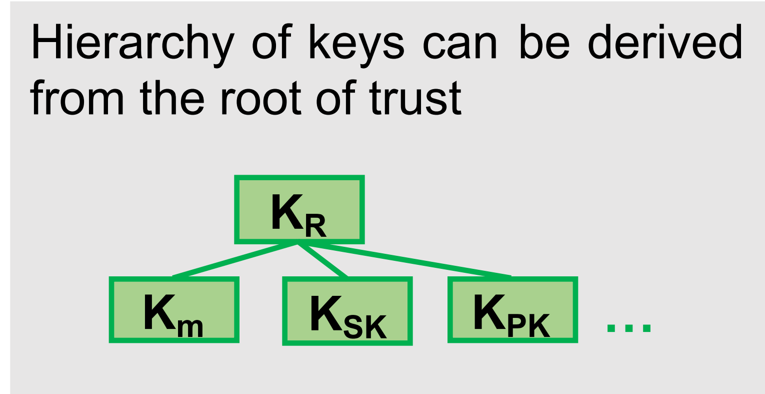
- Fingerprint and authenticate TCB code
- Monitor TCB execution
- Protect TCB code (on embedded security processor)
    - Virtual Memory, ASLR, …

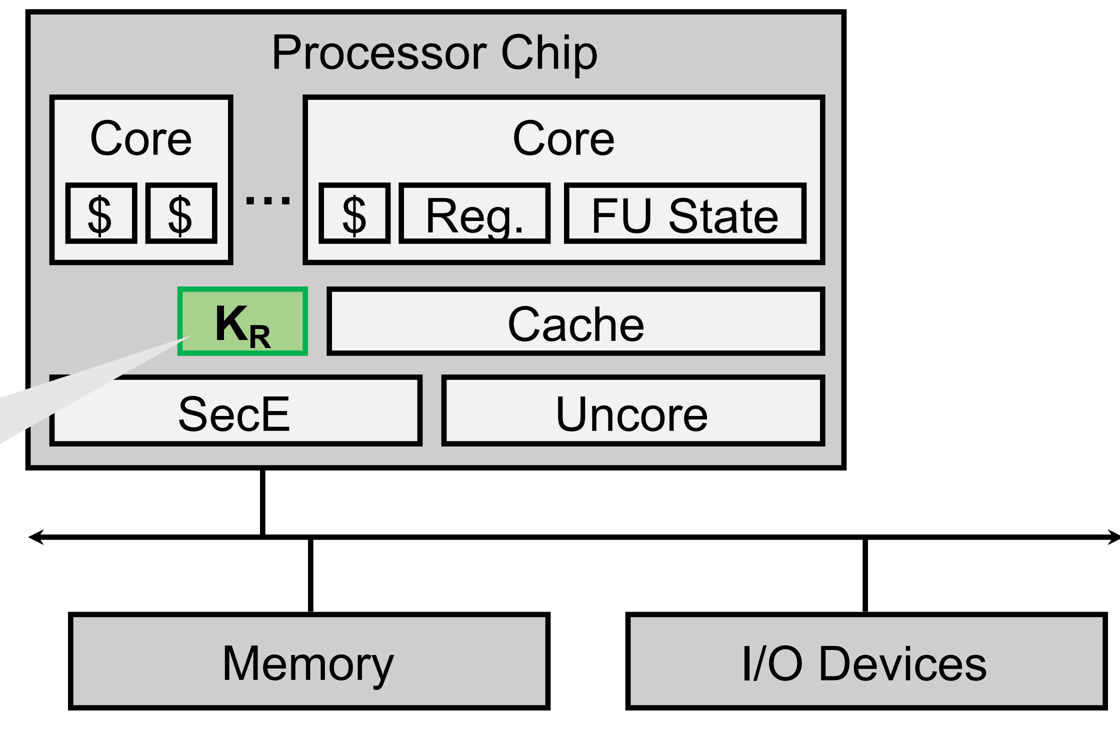
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Root of Trust for TCB

Security of the system is derived from a **root of trust**.

- A secret (cryptographic key) only accessible to TCB components
- Derive encryption and signing keys from the root of trust



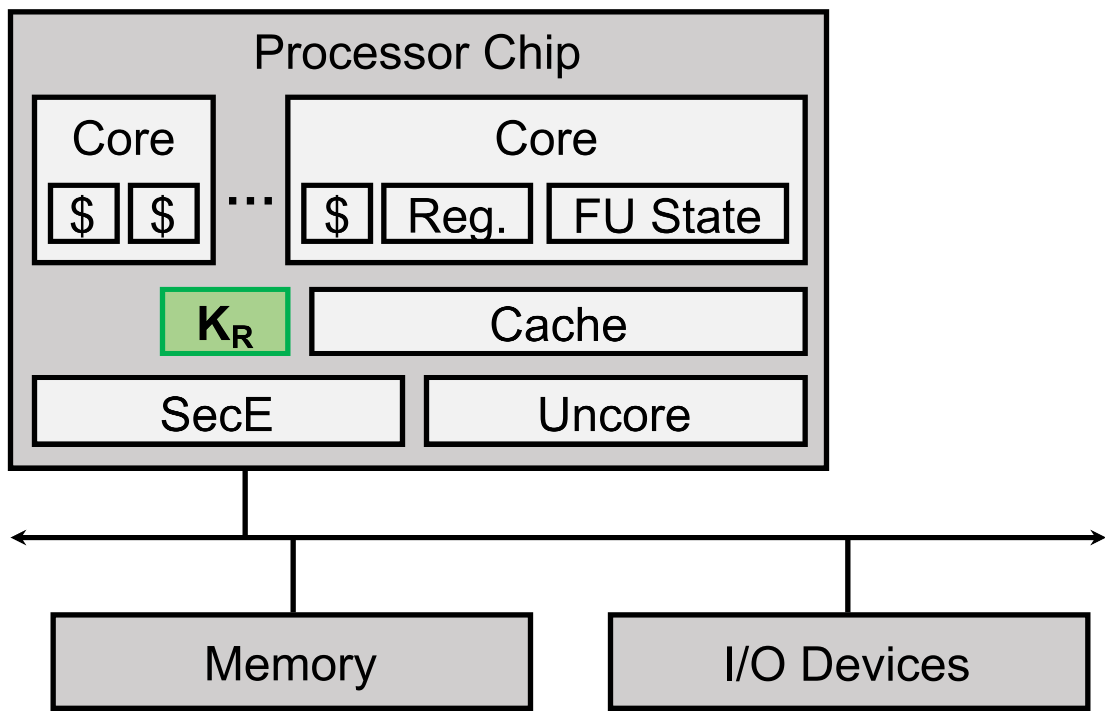


© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Root of Trust and Processor Key

Each processor requires a unique secret.

- **Burn in at the factory** by the manufacturer (but implies trust issues with manufacturer and the supply chain)
  - E.g. One-Time Programmable (OTP) fuses
- Use **Physically Uncloneable Functions** (but requires reliability)
  - Extra hardware to derive keys from PUF
  - Mechanisms to generate and distribute certificates for the key

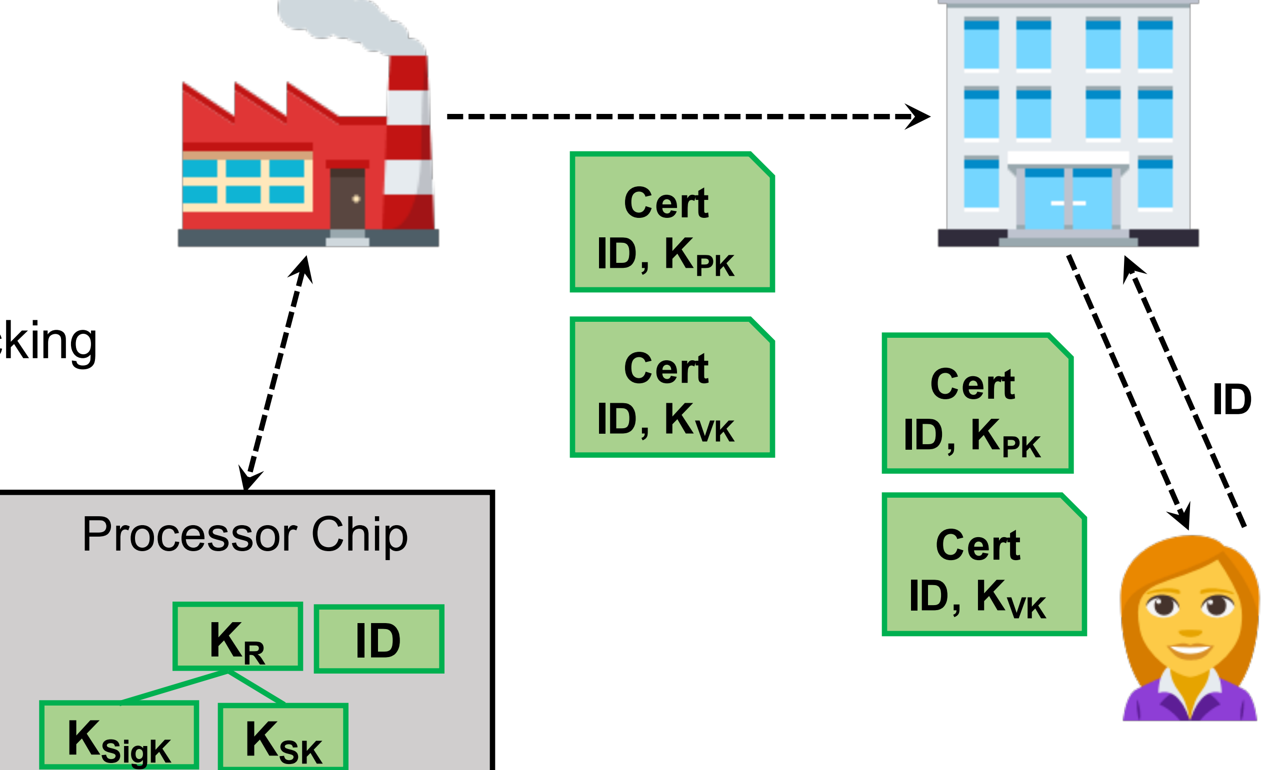


© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Derived Keys and Key Distribution

Derived form the root of trust are signing and verification keys.

- Public key, K_{PK}, for encrypting data to be sent to the processor
  - Data handled by the TCB
- Signature verification key, K_{VK}, for checking data signed by the processor
  - TCB can sign user keys
- Key distribution for PUF based designs will be different
- Need infrastructure!



Emoji Image:
https://www.emojione.com/emoji/1f3ed
https://www.emojione.com/emoji/1f469-1f4bc
https://www.emojione.com/emoji/1f3e2

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Software Measurement

With an embedded signing key, the software running in the TEE can be "measured" to attest to external users what code is running on the system.

App
Guest OS
App
Guest OS
…
App
Guest OS
Hypervisor (VMM)
SMM
SecE
Hardware
Measurement

When all levels are trusted, compute cryptographic hashes over code and data of each level.

Some architectures, e.g. SGX or SEV, "skip" untrusted layers when computing hashes

Emoji Image: https://www.emojione.com/emoji/1f469-1f4bc

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Using Software Measurement

Trusted / Secure / Authenticated Boot:

- Abort boot when wrong measurement is obtained
- Or, continue booting but do not decrypt secrets
- Legitimate software updates will change measurements, may prevent correct boot up

Remote attestation:

- Measure and digitally sign measurements that are sent to remove user

Data sealing (local or remote):

- Only unseal data if correct measurements are obtained

TOC-TOU attacks and measurements:

- Time-of-Check to Time-of-Use (TOC-TOU) attacks leverage the delay between when a measurement is taken, and when the component is used
- Cannot easily use hashes to prevent TOC-TOU attacks

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Need for Continuous Monitoring of Protected Software

Continuous monitoring is potential solution to TOC-TOU:

- Constantly measure the system, e.g. performance counters, and look for anomalies
- Requires knowing correct and expected behavior of system
- Can be used for continuous authentication

Attacker can “hide in the noise” if they change the execution of the software slightly and do not affect performance counters significantly.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Processor Architectures

Memory Protection

Side-Channels Threats and Protections

Speculative or Transient Execution Threats

Principles of Secure Processor Architecture Design

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Sources of Attacks on Memory

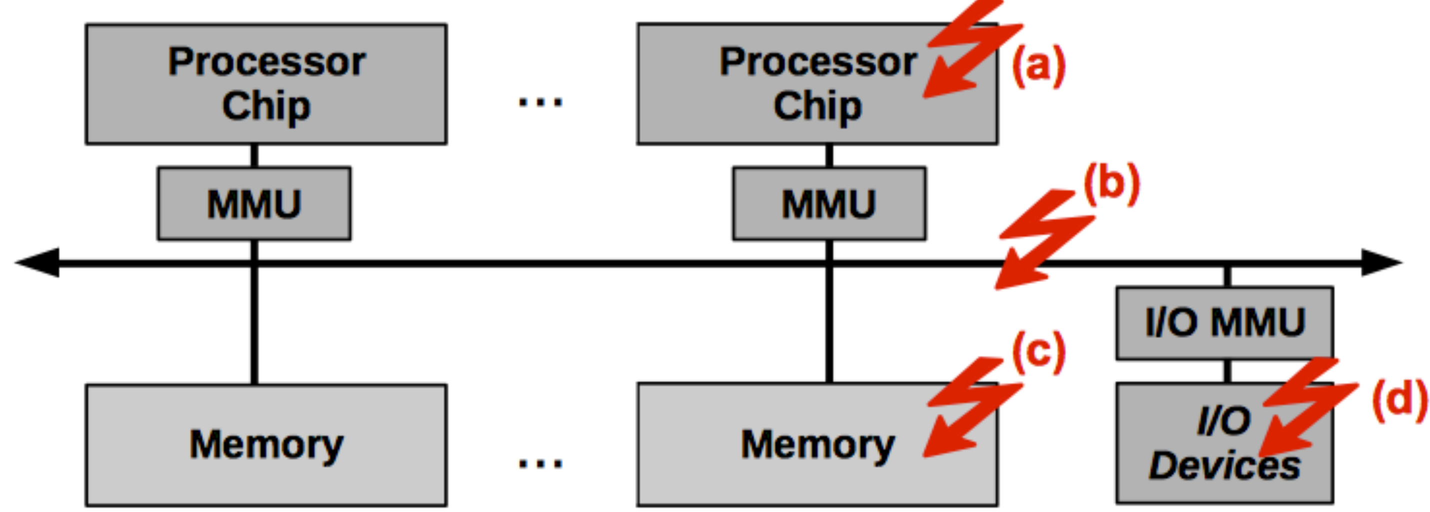
Memory is vulnerable to different types of attacks:

a) Untrusted software running no the processor

b) Physical attacks on the memory bus, other devices snooping on the bus, man-in-the-middle attacks with malicious device

c) Physical attacks on the memory (Coldboot, …)

d) Malicious devices using DMA or other attacks



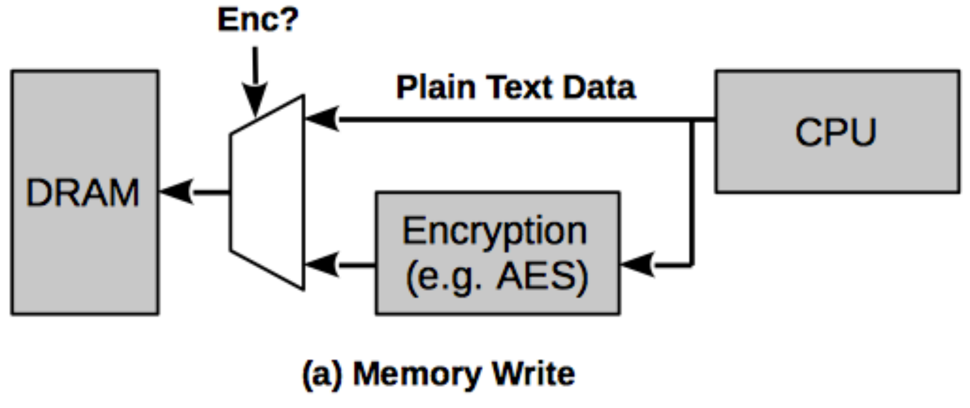
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Confidentiality Protection with Encryption

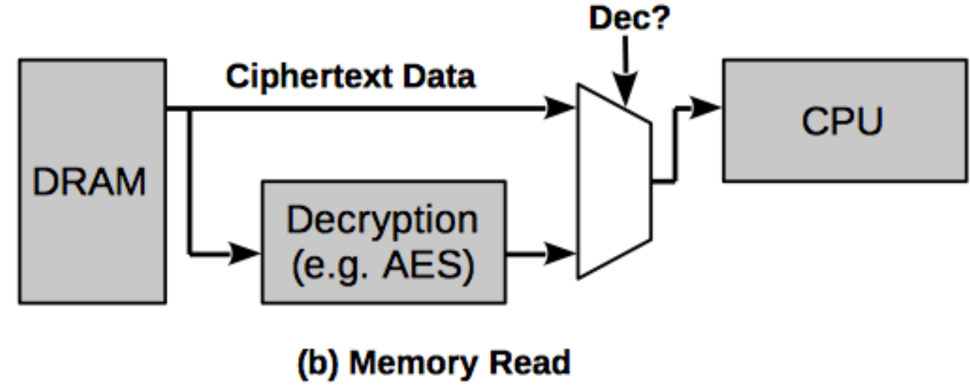
Contents of the memory can be protected with encryption. Data going out of the CPU is encrypted, data coming from memory is decrypted before being used by CPU.

a) Encryption engine (usually AES in CTR mode) encrypts data going out of processor chip
b) Decryption engine decrypts incoming data

Pre-compute encryption pads, then only need to do XOR; speed depends on how well counters are fetched / predicted.



(a) Memory Write

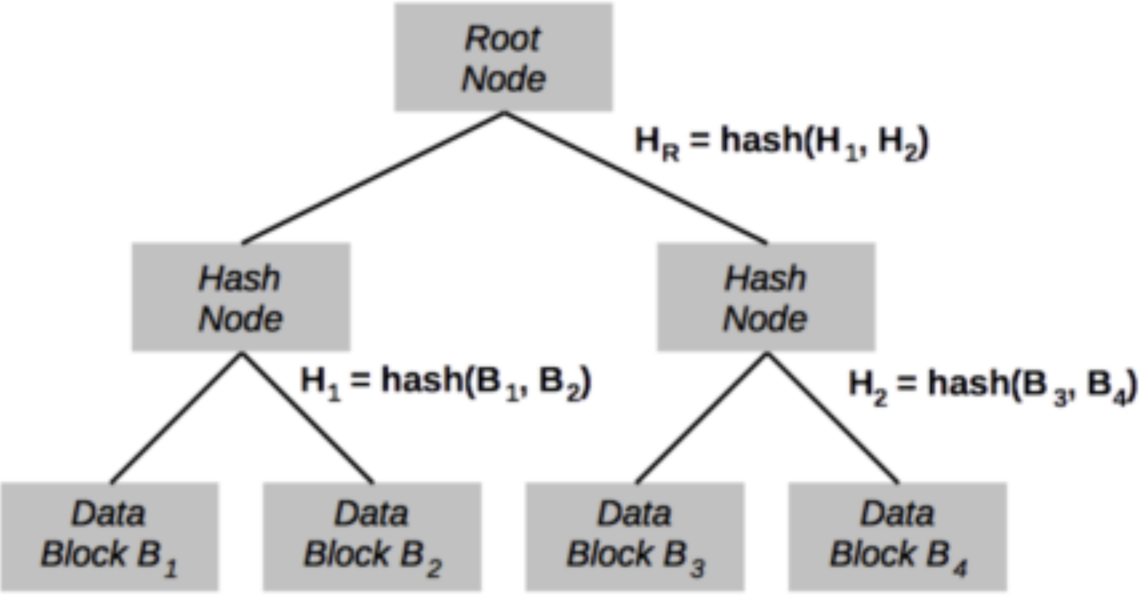


(b) Memory Read

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Integrity Protection with Hash Trees

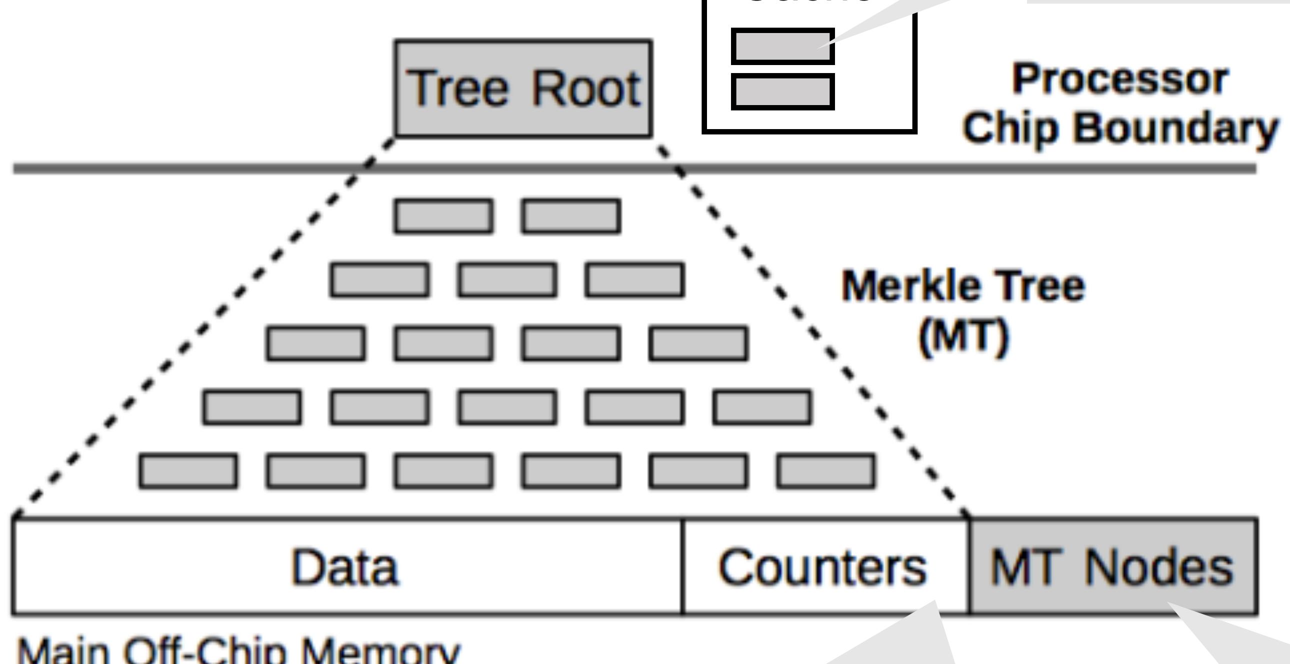
Hash tree (also called **Merkle Tree**) is a logical three structure, typically a binary tree, where two child nodes are hashed together to create parent node; the root node is a hash that depends on value of all the leaf nodes.



Integrity Protection with Hash Trees

Memory blocks can be the leaf nodes in a Merkle Tree, the tree root is a hash that depends on the contents of the memory.

On-chip (cached) nodes are assumed trusted, used to speed up verification.



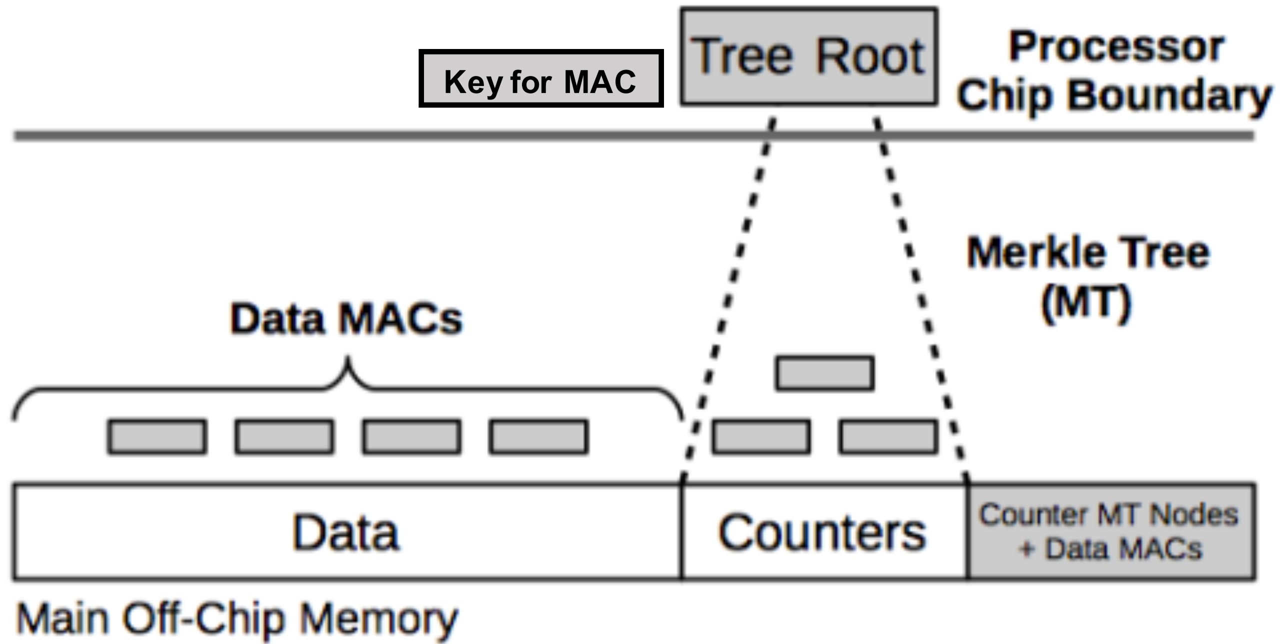
Counters are included in hashes for freshness.

Hash tree nodes are stored in (untrusted) main memory.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Integrity Protection with Bonsai Hash Trees

Message Authentication Codes (MACs) can be used instead of hashes, and a smaller "Bonsai" tree can be constructed.

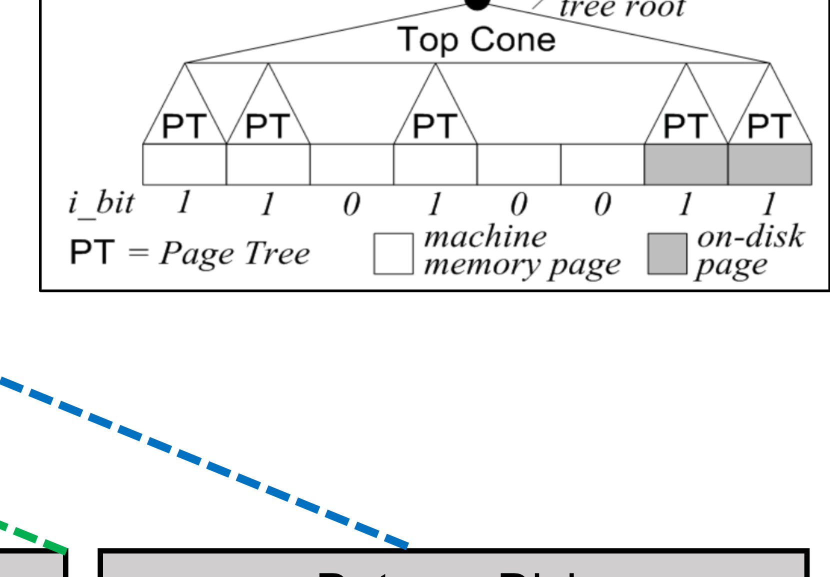


© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Integrity Protection of Selected Memory Regions

- For encryption, type of encryption does not typically depend on memory configuration
- For integrity, the integrity tree needs to consider:
  - Protect whole memory
  - Protect parts of memory (e.g. per application, per VM, etc.)
  - Protect external storage (e.g. data swapped to disk)

E.g., Bastion's memory integrity tree (Champagne, et al., HPCA '10)



Memory

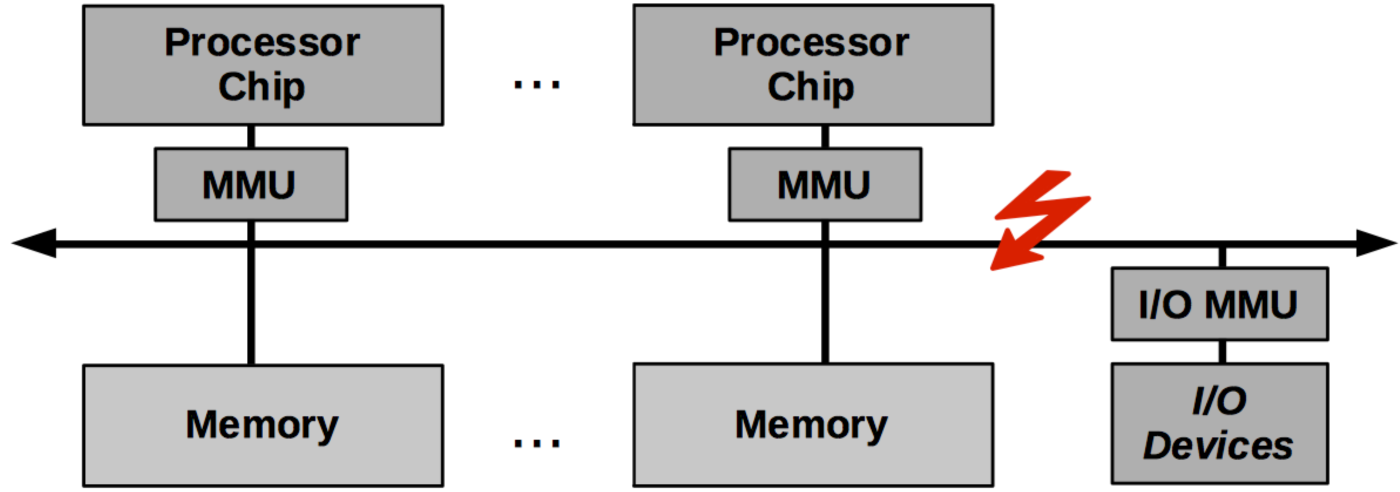
Data on Disk

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Memory Access Pattern Protection

Snooping attacks can target extracting data (protected with encryption) or **extracting access patterns** to learn what a program is doing.

- Easier in symmetric multiprocessing (SMP) due to shared bus
- Possible in other configuration if there are untrusted components



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Security of Non-Volatile Memories and NVRAMs

- Non-volatile memories (NVMs) can store data even when there is no power
- Non-volatile random-access memory (NVRAM) is a specific type of NVM that is suitable to serve as a computer system's main memory, and replace or augment DRAM

- Many types of NVRAMs:
  - ReRAM – based on memristors, stores data in resistance of a dialectric material
  - FeRAM – uses ferroelectric material instead of a dialectric material
  - MRAM – uses ferromagnetic materials and stores data in resistance of a storage cell
  - PCM – typically uses chalcogenide glass where different glass phases have different resistances

Security considerations

- Data remanence makes passive attacks easier (e.g. data extraction)
- Data is maintained after reboot or crash (security state also needs to be correctly restored after reboot or crash)

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Encrypted, Hashed, Oblivious Access Protected Memory

Off-chip memory is untrusted and the contents ideally should be protected from the snooping, spoofing, splicing, replay, and disturbance attacks:

- **Encryption** – snooping and spoofing protection
- **Hashing** – spoofing, splicing, replay (counters must be used), and disturbance protection
- **Oblivious Access** – snooping protection

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Processor Architectures

Memory Protections

Side Channel Threats and Protections

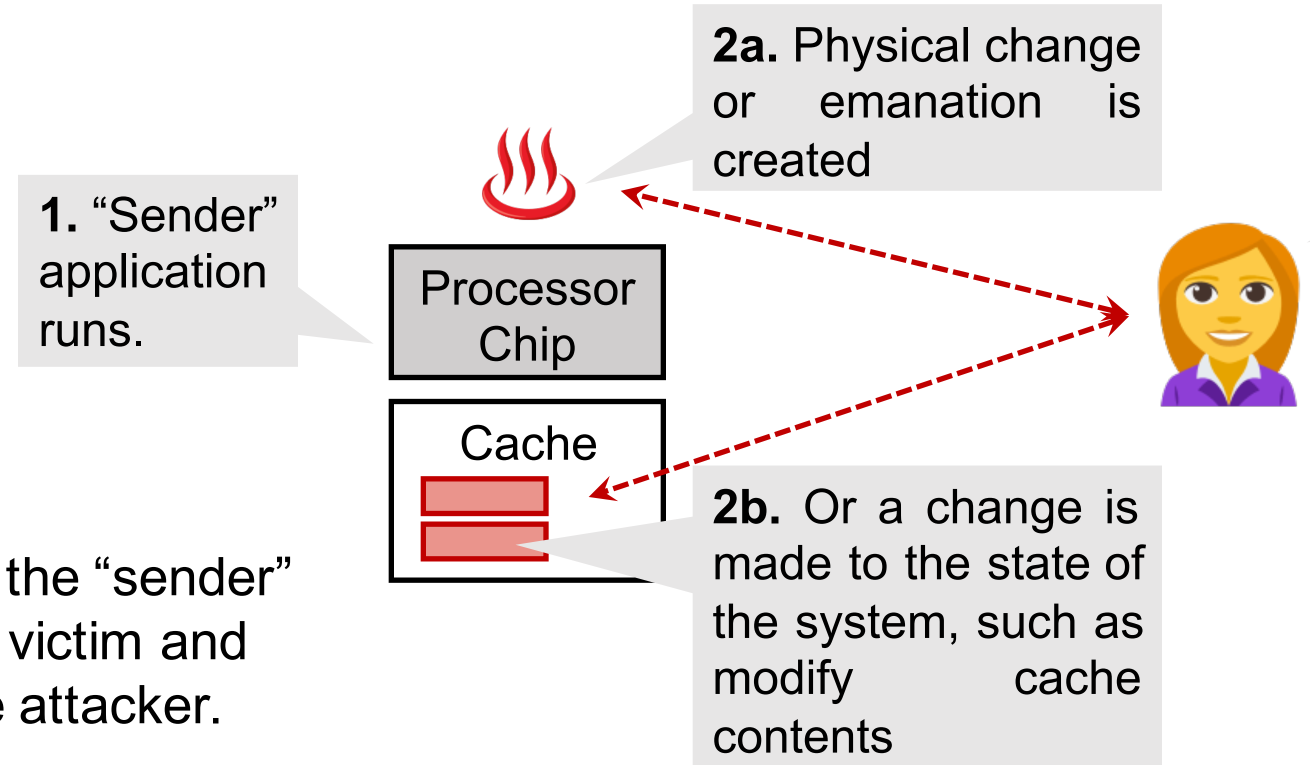
Speculative or Transient Execution Threats

Principles of Secure Processor Architecture Design

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Side and Covert Channels

A **covert channel** is an intentional communication between a sender and a receiver via a medium not designed to be a communication channel.



In a **side channel**, the "sender" in an unsuspecting victim and the "receiver" is the attacker.

Emoji Image:
https://www.emojione.com/emoji/2668
https://www.emojione.com/emoji/1f469-1f4bc

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Side and Covert Channels

Covert Channel – a communication channel that was not intended or designed to transfer information, typically leverage unusual methods for communication of information, never intended by the system's designers

Side Channel – is similar to a covert channel, but the sender does not intend to communicate information to the receiver, rather sending (i.e. leaking) of information is a side effect of the implementation and the way the computer hardware or software is used.

Means for transmitting information: **Timing**, **Power**, **Thermal emanations**, **Electro-magnetic (EM) emanations**, **Acoustic emanations**

- Covert channel is easier to establish, a precursor to side-channel attack
- Differentiate side channel from covert channel depending on who controls the "sender"

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Timing Side Channels Inside a Processor

Many components of a modern processor pipeline can contribute to side channels.

Emoji Image: https://www.emojione.com/emoji/2668

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Sources of Timing Side Channels

Four sources of side channels that can lead to attacks:

1. **Variable Instruction Execution Timing** – Execution of different instructions takes different amount of time
2. **Functional Unit Contention** – Sharing of hardware leads to contention, whether a program can use some hardware leaks information about other programs
3. **Stateful Functional Units** – Program's behavior can affect state of the functional units, and other programs can observe the output (which depends on the state)
4. **Memory Hierarchy** – Data caching creates fast and slow execution paths, leading to timing differences depending on whether data is in the cache or not

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Variable Instruction Execution Timing

Computer architecture principles of **pipelining** and **making common case fast** drive processor designs where certain operations take more time than others – program execution timing may reveal which instruction was used.

- Multi-cycle floating point vs. single cycle addition
- Memory access hitting in the cache vs. memory access going to DRAM

Constant time software implementations can choose instructions to try to make software run in constant time

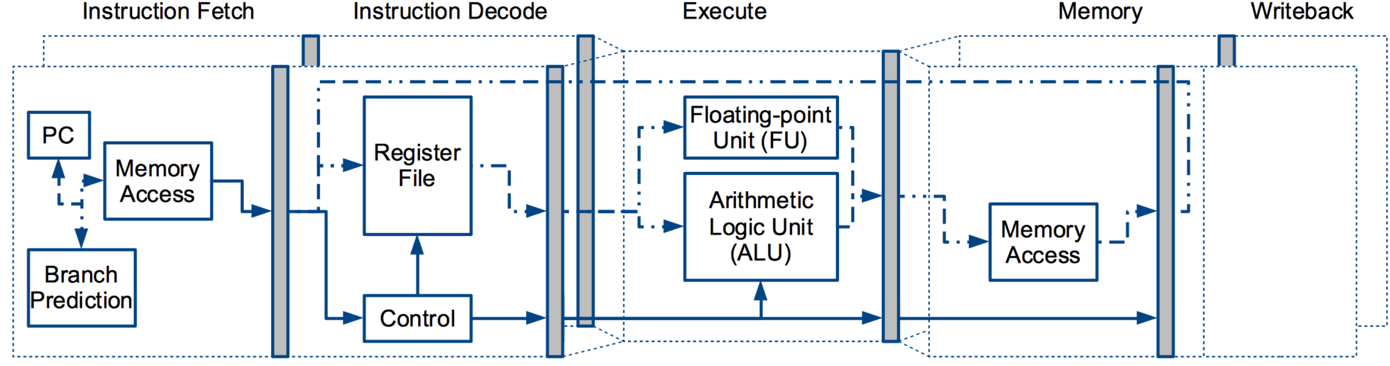
- Arithmetic is easiest to deal with
- Caches may need to be flushed to get constant memory instruction timing
- No way to flush state of functional units such as branch predictor

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Functional Unit Contention

Functional units within processor are re-used or shared to save on area and cost of the processor resulting in varying program execution.

- Contention for functional units causes execution time differences



Spatial or Temporal Multiplexing allows to dedicate part of the processor for exclusive use by an application

- Negative performance impact or need to duplicate hardware

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Stateful Functional Units

Many functional units inside the processor keep some history of past execution and use the information for prediction purposes.

- Execution time or other output may depend on the state of the functional unit
- If functional unit is shared, other programs can guess the state (and thus the history)
- E.g. caches, branch predator, prefetcher, etc.

Flushing state can erase the history.

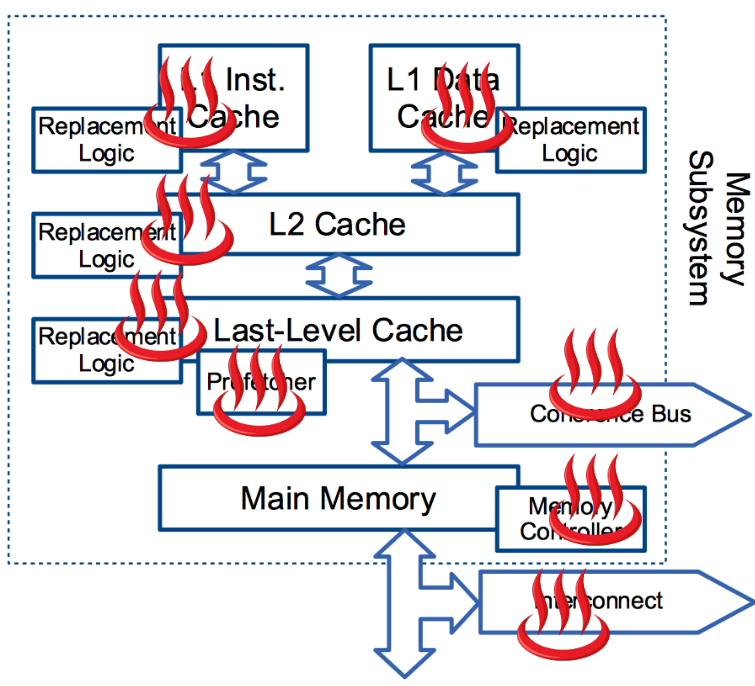
- Not really supported today
- Will have negative performance impact

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Timing Side Channels in Memory Hierarchy

Memory hierarchy aims to improve system performance by hiding memory access latency (creating fast and slow executions paths); and parts of the hierarchy area a shared resource.

- Cache replacement logic
  - Inclusive caches
  - Non-inclusive caches
  - Exclusive caches
- Prefetcher logic
  - Also speculative instruction fetching from processor core
- Memory controller
- Interconnect
- Coherence bus



Emoji Image: https://www.emojione.com/emoji/2668

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Classical vs. Speculative Side-Channels

Side channels can now be classified into two categories:

- **Classical** – which do not require speculative execution
- **Speculative** – which are based on speculative execution

Difference is victim is not fully in control of instructions they execute (i.e. some instructions are executed speculatively)

State of functional unit is modified by victim and it can be observed by the attacker via timing changes

Root cause of the attacks remains the same

Focusing only on speculative attacks does not mean classical attacks are prevented, e.g. defenses for cache-based attacks

Defending classical attacks defends speculative attacks as well, but not the other way around

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Timing Side Channel Bandwidths

The Orange Book, also called the Trusted Computer System Evaluation Criteria (TCSEC), specifies that a channel bandwidth exceeding a rate of **100 bps is a high bandwidth channel**.

Bandwidth (bps)

10^8
10^6
10^4
10^2
10^0
10^{-2}

2004 2006 2008 2010 2012 2014 2016 2018

Year

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Side Channels as Attack Detectors

Side channels can be used to detect or observe system operation.

- Measure timing, power, EM, etc. to detect unusual behavior
- Similar to using performance counters, but attacker doesn't know measurement is going on

Tension between **side channels as attack vectors vs. detection tools**

- Side channels are mostly used for attack today

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Side Channels due to Physical Emanations

Side-channels can be also observed from outside of the computer system, notably through physical emanations.

- Thermal
- Electromagnetic
- Acoustic

Require measuring temperature. Thermal channels possible in data centers without physical presence.

Require measuring EM radiation. Today need dedicated equipment.

Require measuring sound. Today need dedicated equipment.

See Part 2 of the tutorial for more on side channels.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Processor Architectures

Memory Protections

Side Channel Threats and Protections

Speculative or Transient Execution Threats

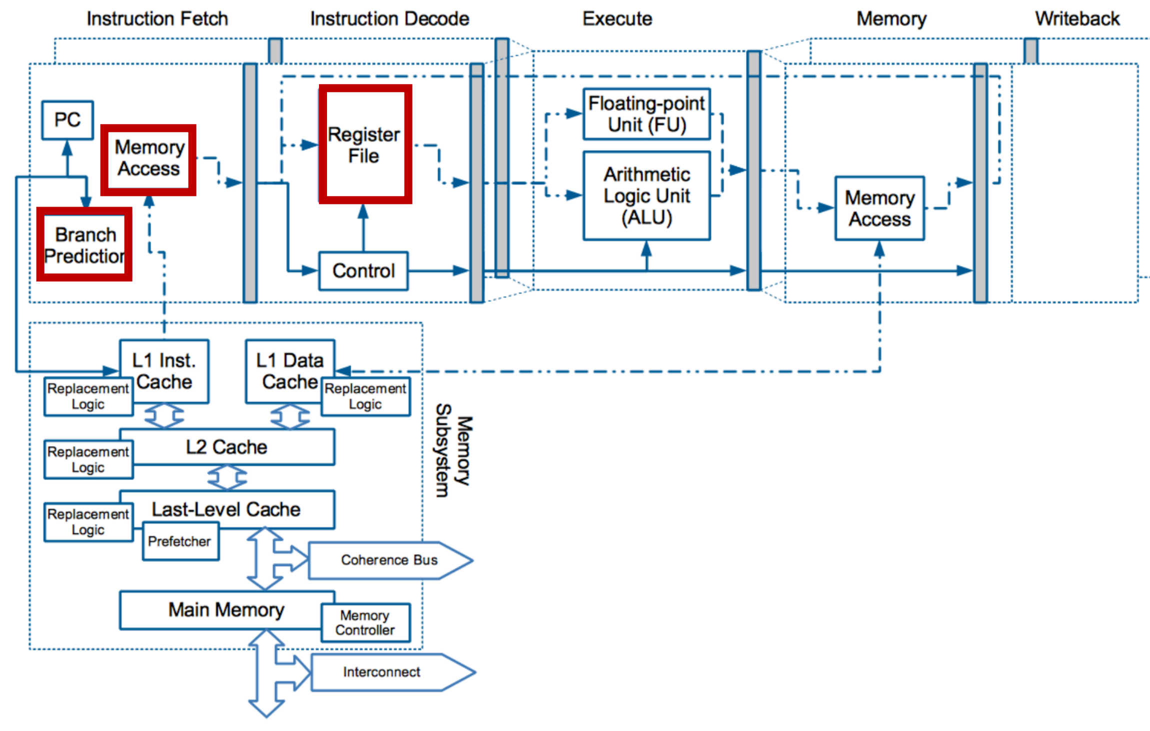
Principles of Secure Processor Architecture Design

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Prediction and Speculation in Modern CPUs

Prediction is one of the six key features of modern processor

- Instructions in a processor pipeline have dependencies on prior instructions which are in the pipeline and may not have finished yet
- To keep pipeline as full as possible, prediction is needed if results of prior instruction are not known yet
- Prediction can be done for:
  - Control flow
  - Data dependencies
  - Actual data (also called value prediction)
- Not just branch prediction: prefetcher, memory disambiguation, …



Speculative or Transient Execution Threats

Speculation causes transient execution to exist in modern processors

- During transient execution, processor state is modified
- If state (architectural or micro-architectural) is not properly cleaned up when mispredicted instructions are squashed, sensitive data can be leaked out

Attacks based on transient execution have two parts:

1. Leverage speculation to execute some code transiently, which modifies processor state based on some secret value
2. Use a side-channel to extract the information from the processor state

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Mitigation Techniques for Attacks due to Speculation

1. **Prevent or disable speculative execution** – addresses Speculation Primitives
   - Overheads are not clear, application specific
   - Today there is no user interface for fine grain control of speculation
2. **Limit attackers ability to influence predictor state** – addresses Speculation Primitives
   - Some proposals exist to add new instructions to minimize ability to affect branch predictor state, etc.
3. **Minimize attack window** – addresses Windowing Gadgets
   - Ultimately would have to improve performance of memory accesses, etc.
   - Not clear how to get exhaustive list of all possible windowing gadget types
4. **Track sensitive information** (information flow tracking) – addresses Disclosure Gadgets
   - Stop transient speculation and execution if sensitive data is touched
   - Users must define sensitive data
5. **Prevent timing channels** – addresses Disclosure Primitives
   - Add secure caches

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Mitigation Techniques for Attacks due to Faults

1. **Evaluate fault conditions sooner**
    - Will impact performance, not always possible
2. **Limit access condition check races**
    - Don't allow accesses to proceed until relevant access checks are finished

See Part 3 of the tutorial for more on speculative execution.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Processor Architectures

Memory Protections

Side Channel Threats and Protections

Speculative or Transient Execution Threats

Principles of Secure Processor Architecture Design

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Principles of Secure Processor Architecture Design

Four principles for secure processor architecture design based on existing designs and also on ideas about what ideal design should look like are:

- Architectural state
- Micro-architectural state
- Due to spatial or temporal sharing of hardware

1. **Protect Off-chip Communication and Memory**
2. **Isolate Processor State among TEE Execution and other Software**
3. **Allow TCB Introspection**
4. **Authenticate and Continuously Monitor TEE and TCB**

Additional design suggestions:

- Avoid code bloat
- Minimize TCB
- Ensure hardware security (Trojan prevention, supply chain issues, etc.)
- Use formal verification

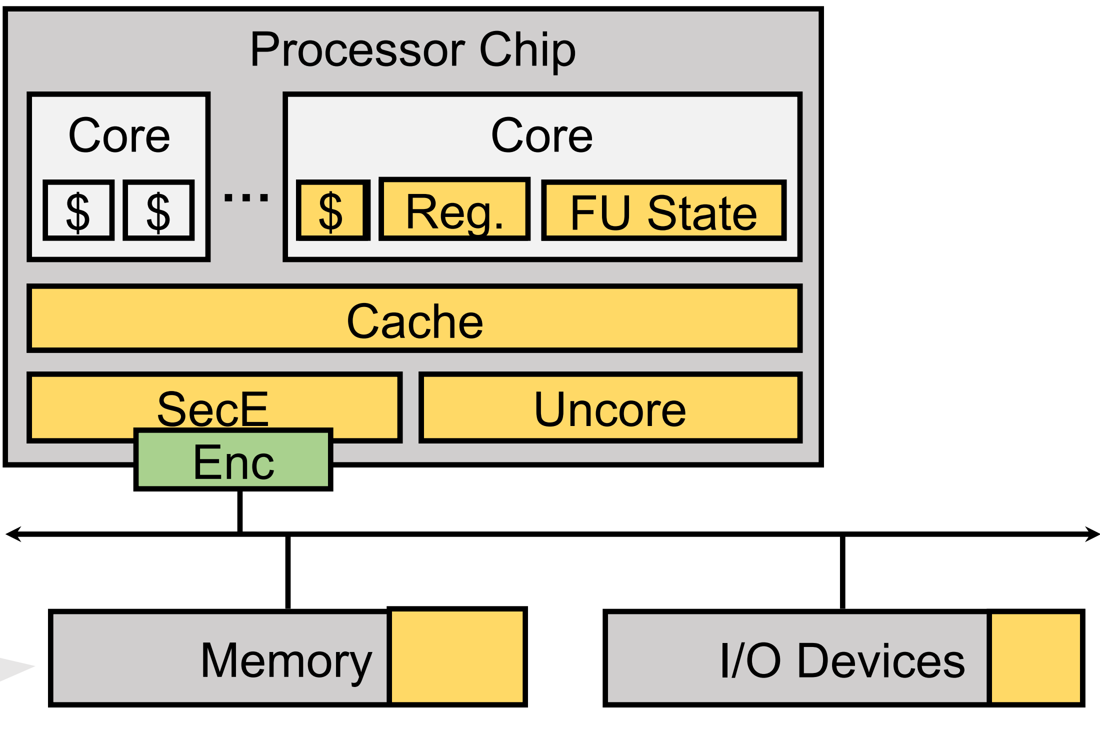
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Protect Off-chip Communication and Memory

Off-chip components and communication are untrusted, need protection with **encryption**, **hashing**, **access pattern protection**.

Open research challenges:

- Performance
- Key distribution



E.g. encryption defends Cold boot style attacks on main memory.

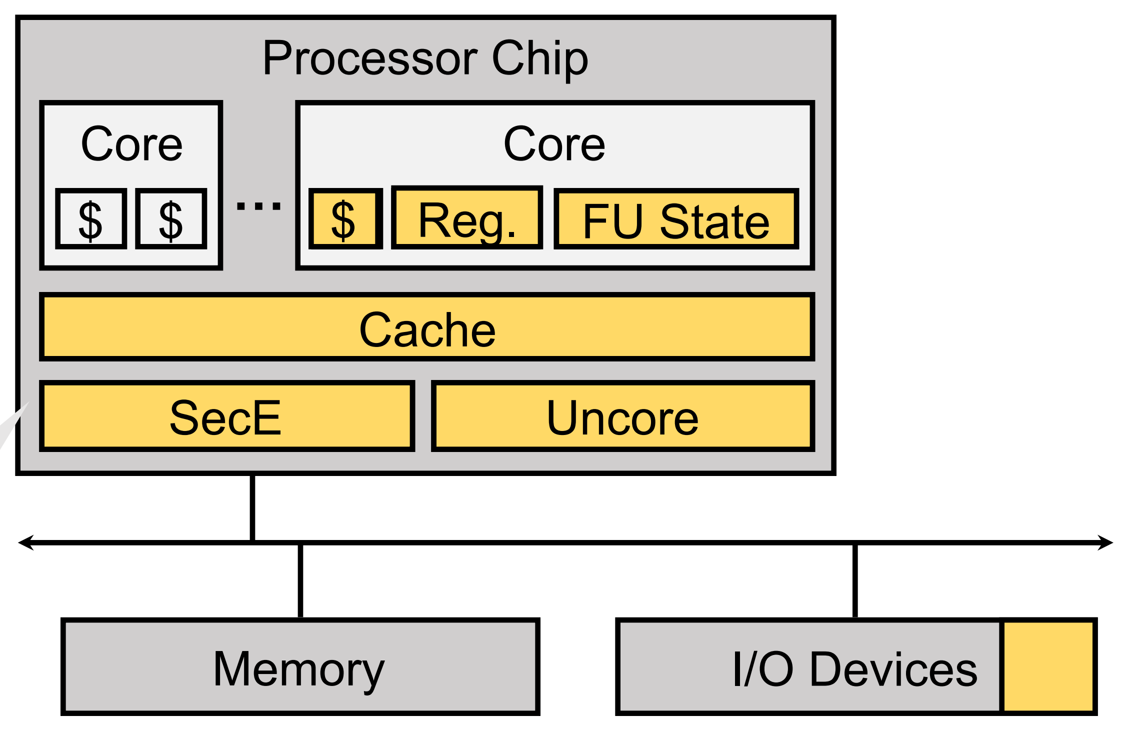
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Isolate Processor State among TEE Execution

When switching among protected software and other software or other protected software, need to flush the state, or save and restore it, to prevent one software influencing another.

Open research challenges:

- Performance
- Finding all the state to flush or clean
- Isolate state during concurrent execution
- ISA interface to allow state flushing



E.g. flushing state helps defend Spectre and Meltdown type attacks.

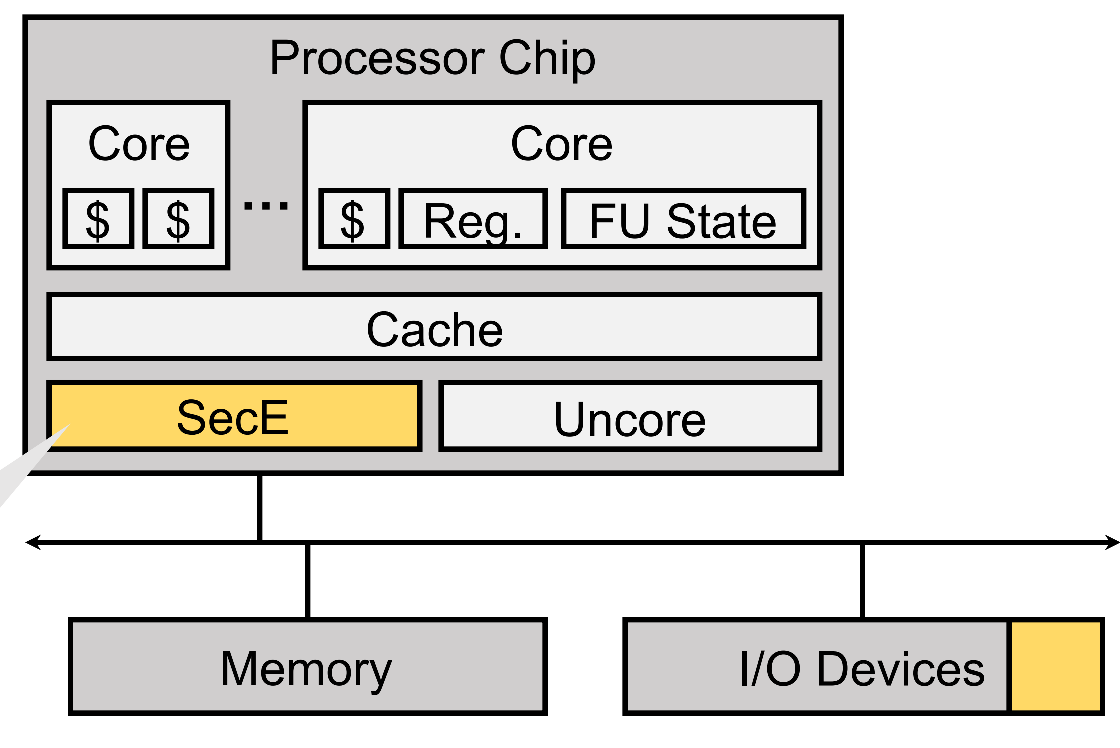
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Allow TCB Introspection

Need to ensure correct execution of TCB, through **open access to TCB design**, **monitoring**, **fingerprinting**, and **authentication**.

Open research challenges:

- ISA interface to introspect TCB
- Area, energy, performance costs due extra features for introspection
- Leaking information about TCB or TEE



E.g. open TCB design can minimize attacks on ME or PSP security engines

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Authenticate and Continuously Monitor TEE and TCB

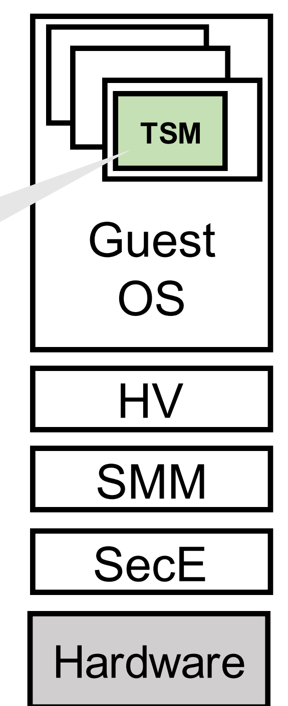
Monitoring of software running inside TEE, e.g. TSMs or Enclaves, gives assurances about the state of the protected software.

Likewise monitoring TCB ensures protections are still in place.

Open research challenges:

- Interface design for monitoring
- Leaking information about TEE

E.g. continuous monitoring of a TEE can help prevent TOC-TOU attacks.



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Pitfalls and Fallacies

- Pitfall: Security by Obscurity
- Fallacy: Hardware Is Immutable
- Pitfall: Wrong Threat Model
- Pitfall: Fixed Threat Model
- Pitfall: Use of Outdated or Custom Crypto
- Pitfall: Not Addressing Side Channels
- Pitfall: Requiring Zero-Overhead Security
- Pitfall: Code Bloat
- Pitfall: Incorrect Abstraction

E.g. recent attacks on industry processors.

Most actually realized architectures use a security processor (e.g. ME or PSP).

E.g. original SGX did not claim side channel protection, but researchers attacked it.

Most designs are one-size-fits all solutions.

E.g. today's devices will be in the field for many years, but do not use post-quantum crypto.

Most architectures underestimate side channels.

Performance-, area-, or energy-only focused designs ignore security.

E.g. rather than partition a problem, large code pieces are ran instead TEEs; also TCB gets bigger and bigger leading to bugs.

Abstraction (e.g. ISA assumptions) does not match how device or hardware really behaves.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Pitfalls and Fallacies

- Pitfall: Focus Only on Speculative Attacks

Defending only speculative attacks does not ensure classical attacks are also protected

- …

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Tutorial Outline & Schedule

15:30 – 16:10	Secure Processor Architectures
16:10 – 16:20	Break
16:20 – 17:10	Secure Processor Caches
17:10 – 17:20	Break
17:20 – 18:00	Transient Execution Attacks and Mitigations
18:00	Wrap Up

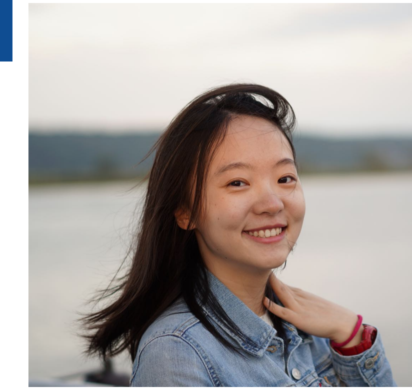
Slides and information at:
http://caslab.csl.yale.edu/tutorials/host2019/

WiFi Information:
Network: Hilton-Meeting, Password: HOST2019

Secure Processor Caches

Jakub Szefer

Wenjie Xiong


Shuwen Deng

Dept. Of Electrical Engineering
Yale University

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Cache Timing Side-Channel Attacks

Secure Cache Techniques

Secure Cache Architectures

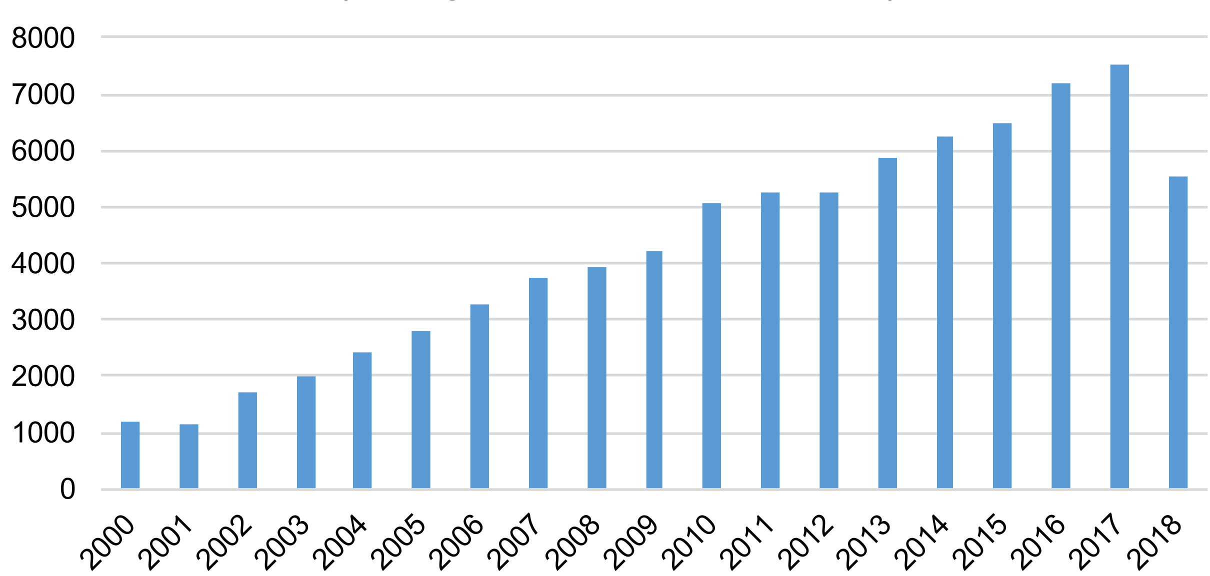
Cache Timing Attacks Continue to Raise Concerns

- Cache timing attacks have a long history, but the research on attacks and defenses is still a very active field

- Timing attacks using caches, and other cache-like structures, often target cryptographic software

- Very difficult to write “constant time” software, so attacks are still potent

- Attacks can achieve quite high bandwidth in idealized settings, about 1Mbps or more

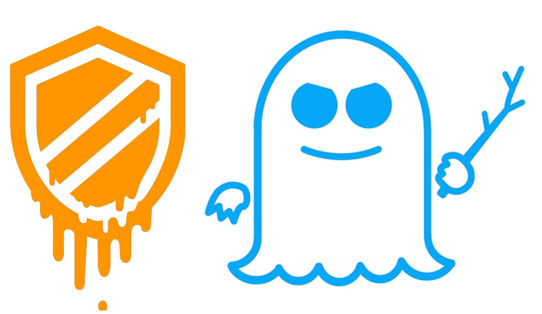


© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Cache Timing Attacks Continue to Raise Concerns

- There is renewed interest in timing attacks due to Transient Execution Attacks (i.e. Spectre, Meltdown, and their various variants)
- Most of them use **transient executions and** leverage **cache timing attacks**
- Variants using cache timing attacks (side or covert channels):

Variant 1:	Bounds Check Bypass (BCB)	Spectre
Variant 1.1:	Bounds Check Bypass Store (BCBS)	Spectre-NG
Variant 1.2:	Read-only protection bypass (RPB)	Spectre
Variant 2:	Branch Target Injection (BTI)	Spectre
Variant 3:	Rogue Data Cache Load (RDCL)	Meltdown
Variant 3a:	Rogue System Register Read (RSRR)	Spectre-NG
Variant 4:	Speculative Store Bypass (SSB)	Spectre-NG
(none)	LazyFP State Restore	Spectre-NG 3
Variant 5:	Return Mispredict	SpectreRSB



NetSpectre, Foreshadow, SGXSpectre, or SGXPectre
SpectrePrime and MeltdownPrime (both use Prime+Probe instead of original Flush+Reload cache attack)

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Cache Timing Attacks

- **Attacker and Victim**
  - Victim (holds security critical data)
  - Attacker (attempts to learn the data)
- **Attack requirement**
  - Attacker has ability to monitor timing of cache operations made by the victim or by self
  - Can control or trigger victim to do some operations using sensitive data
- **Operations having timing differences**
  - Memory accesses: load, store
  - Data invalidation: different flushes (clflush, etc.), cache coherence
- **Side-Channel vs. Covert-Channel Attack**
  - Side channel: victim is not cooperating
  - Covert channel: victim (sender) works with attacker – easier to realize and higher bandwidth
- **Many Known Attacks:** Prime+Probe, Flush+Reload, Evict+Time, or Cache Collision Attack

App, OS, VM, etc.
Victim

Side Channels

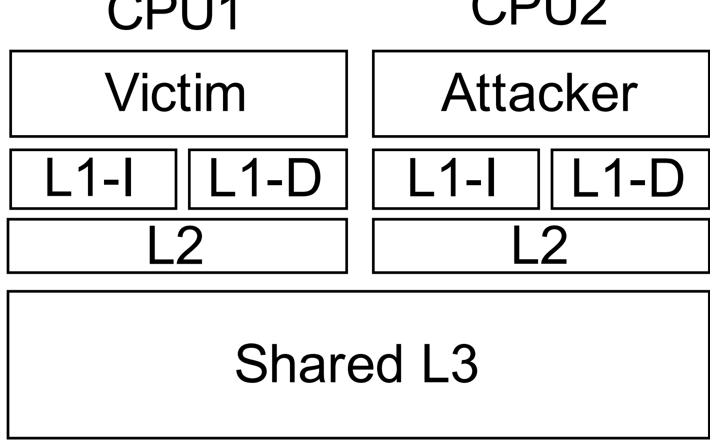
App, OS, VM, etc.
Attacker

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Prime-Probe Attacks

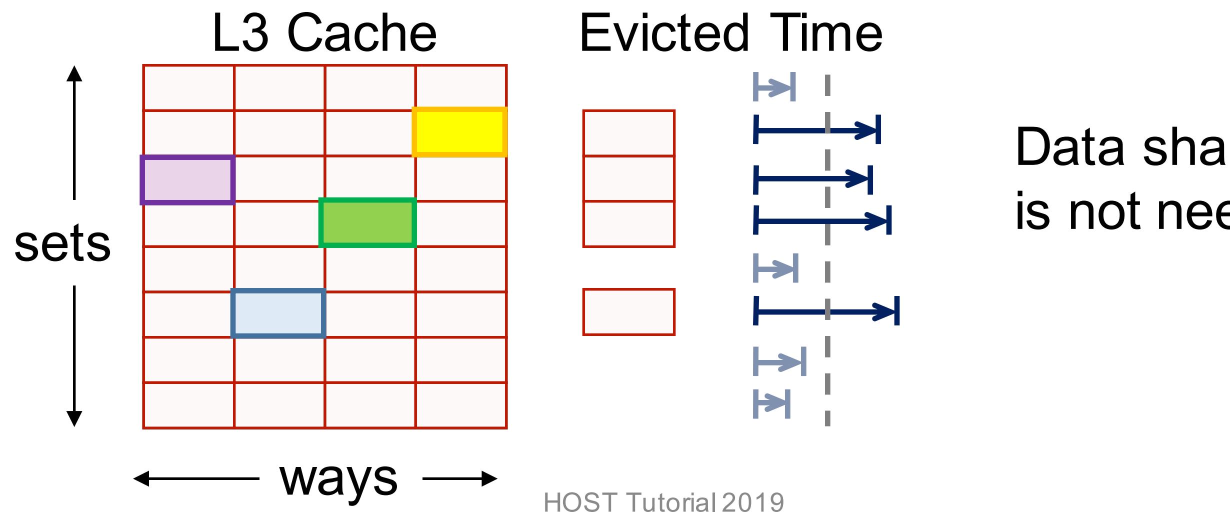
Osvik, D. A., Shamir, A., & Tromer, E, “Cache attacks and countermeasures: the case of AES”. 2006.

2- Victim accesses critical data



1- Attacker primes each cache set

3- Attacker probes each cache set (measure time)



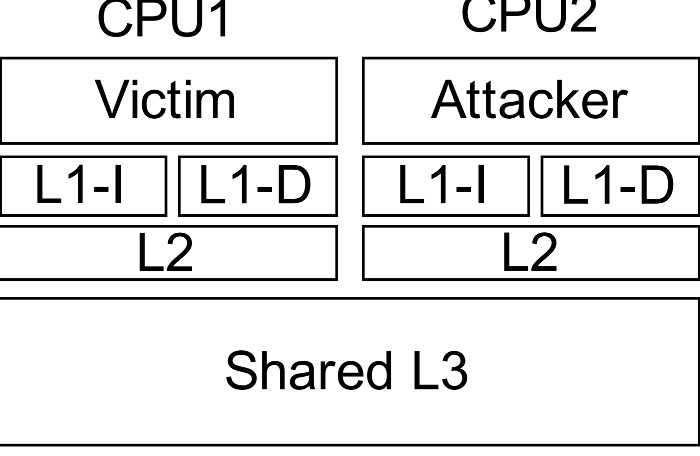
Data sharing is not needed

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Flush-Reload Attack

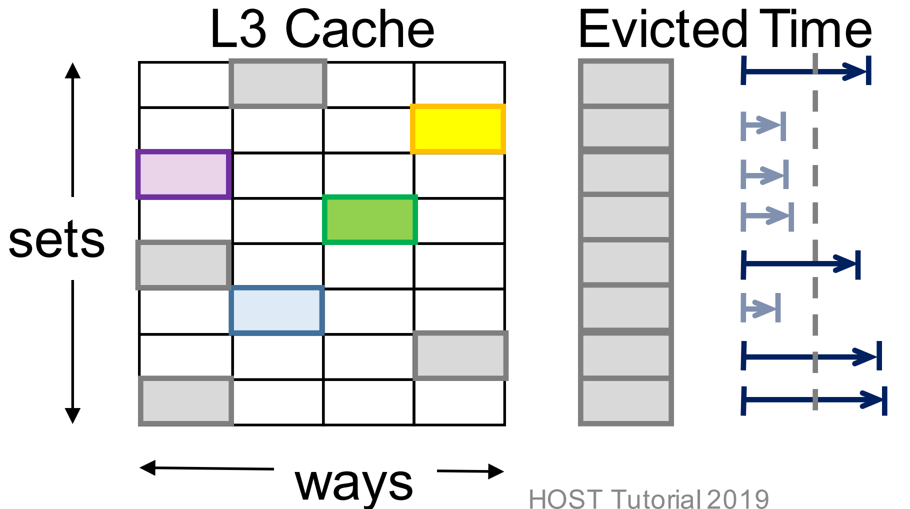
Yarom, Y., & Falkner, K. "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack", 2014.

2- Victim accesses critical data



1- Attacker flushes each line in the cache

3- Attacker reloads critical data by running specific process (measure time)



Data sharing is needed

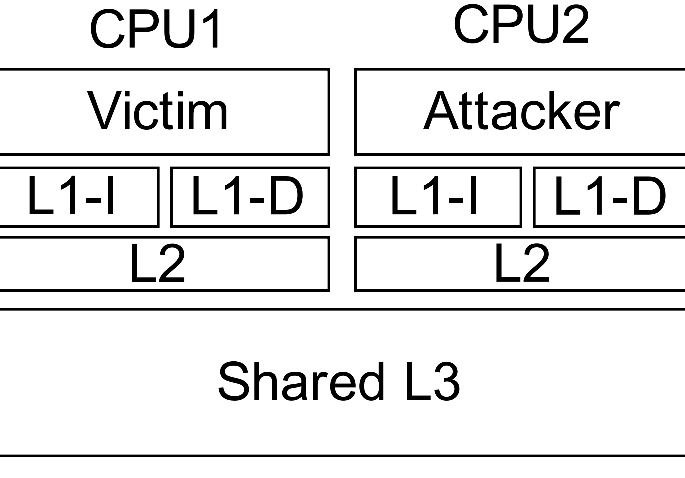
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Evict-Time Attack

Osvik, D. A., Shamir, A., & Tromer, E, “Cache attacks and countermeasures: the case of AES”. 2006.

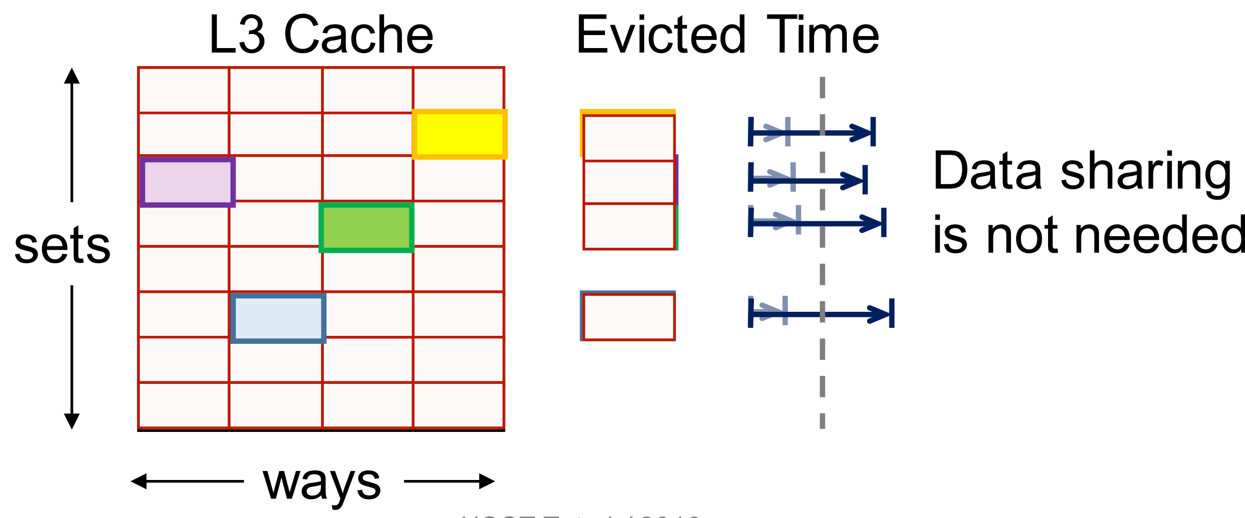
1- Victim has some critical data

3- Victim accesses critical data



2- Attacker evicts cache set and fill its own data (can evict set by set)

(Attacker measures time)



Data sharing is not needed

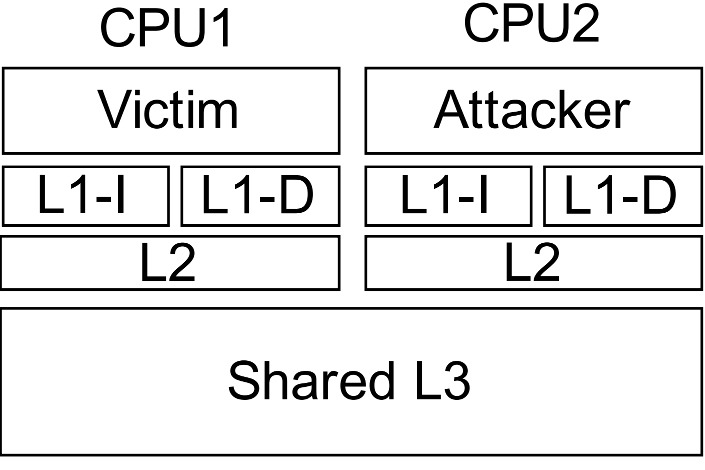
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Cache Collision Attack

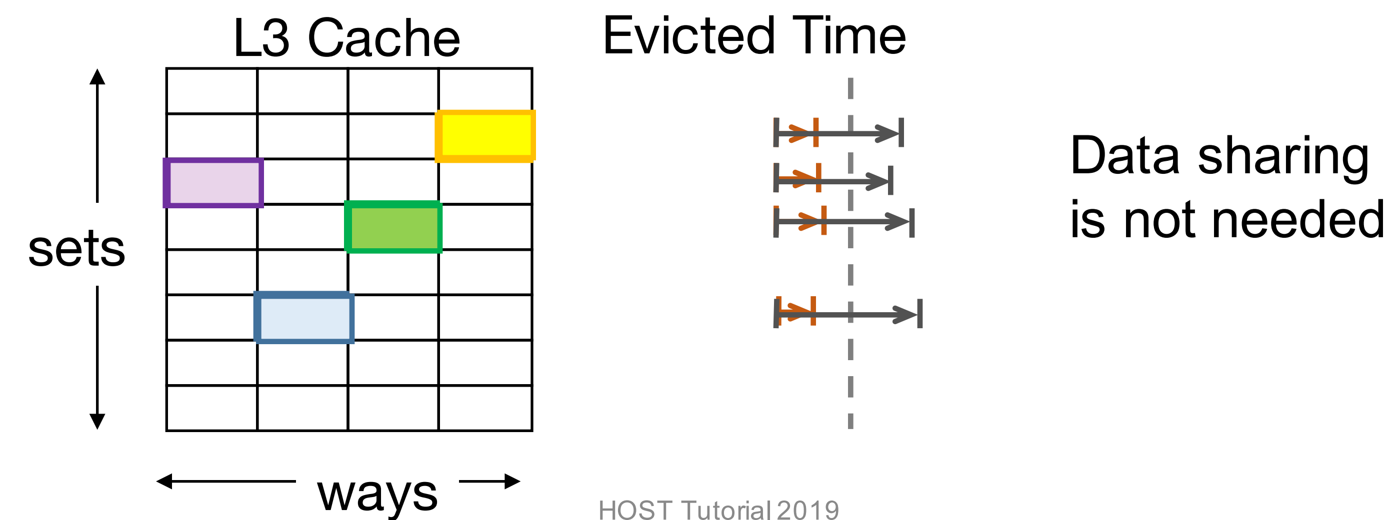
Bonneau, J., & Mironov, I. "Cache-collision timing attacks against AES", 2006.

1- Victim has some critical data

2- Victim reuses critical data



(Attacker measures time)

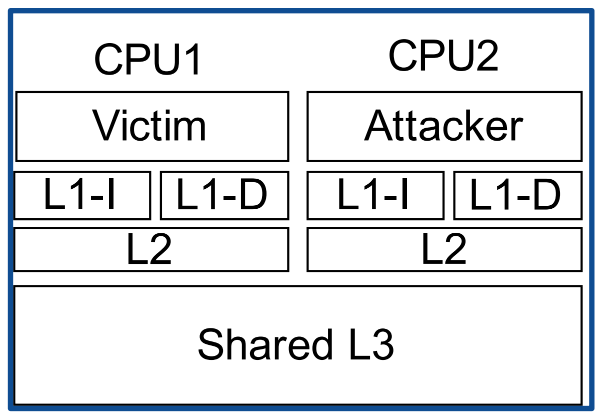


Data sharing is not needed

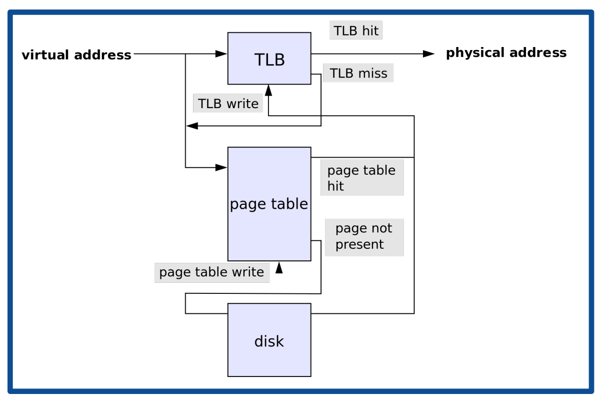
Similar Attacks on Cache-Like Structures

Timing attacks do not only leverage caches, but any cache-like structure with varying timing (due to hits or missies in the structure) can be vulnerable to timing attacks

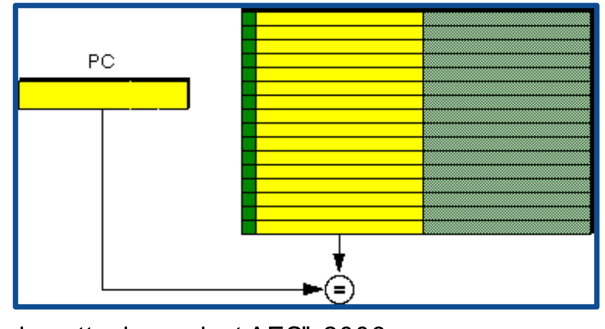
Instruction or Data Cache



Translation Look-aside Buffer (TLB)



Branch Target Buffer (BTB)



Return Stack Buffer (RSB)

code
Software call stack
main:
…
call F1
…
ret
… 0x00006210:
local variables
ebp old value
0x00008250
arguments
F2 call frame
local variables
EBP old value
0x00006210
arguments
F1 call frame
…
F1:
…
call F2
…
ret
… 0x00008250:
…
F2:
…
ret
RSB
0x00008250
F2 return address
0x00006210
F1 return address

Typical attacks:
Cache: Bonneau, J., & Mironov, I, “Cache-collision timing attacks against AES”, 2006
TLB: Gras, B., Razavi, K., Bos, H., & Giuffrida, C, “Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with {TLB} Attacks”, 2018
BTB: Evtyushkin, D., Riley, R., Abu-Ghazaleh, N. C., & Ponomarev, D, “Branchscope: A new side-channel attack on directional branch predictor”, 2018
RSB: Koruyeh, E. M., Khasawneh, K. N., Song, C., & Abu-Ghazaleh, N., “Spectre returns! speculation attacks using the return stack buffer”, 2018

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Understanding All Possible Timing Attacks

- The Prime+Probe, Flush+Reload, Evict+Time, or Cache Collision attacks are just some of the possible timing attacks
- **Defenders need to understand all possible types of attacks**, as attacker just needs to find out that works – but defenders need to protect all types of attacks
- A recent **3-step model can be used to understand timing attacks**…

A Three-Step Model for Cache Attack Modeling

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Observation:

- All the existing cache timing attacks within three memory operations → three-step model
- Cache replacement policy the same to each cache block → focus on one cache block

The Three-Step Single-Cache-Block-Access Model

$Step1$	⇝	$Step2$	⇝	$Step3$ (fast/slow)
The initial state of the cache block set by a memory operation		Memory operation alters the state of the cache		Final memory operations and timing observation (fast/slow)

- Analyzed possible states of the cache block + used cache three-step simulator and reduction rules derive all the effective vulnerabilities

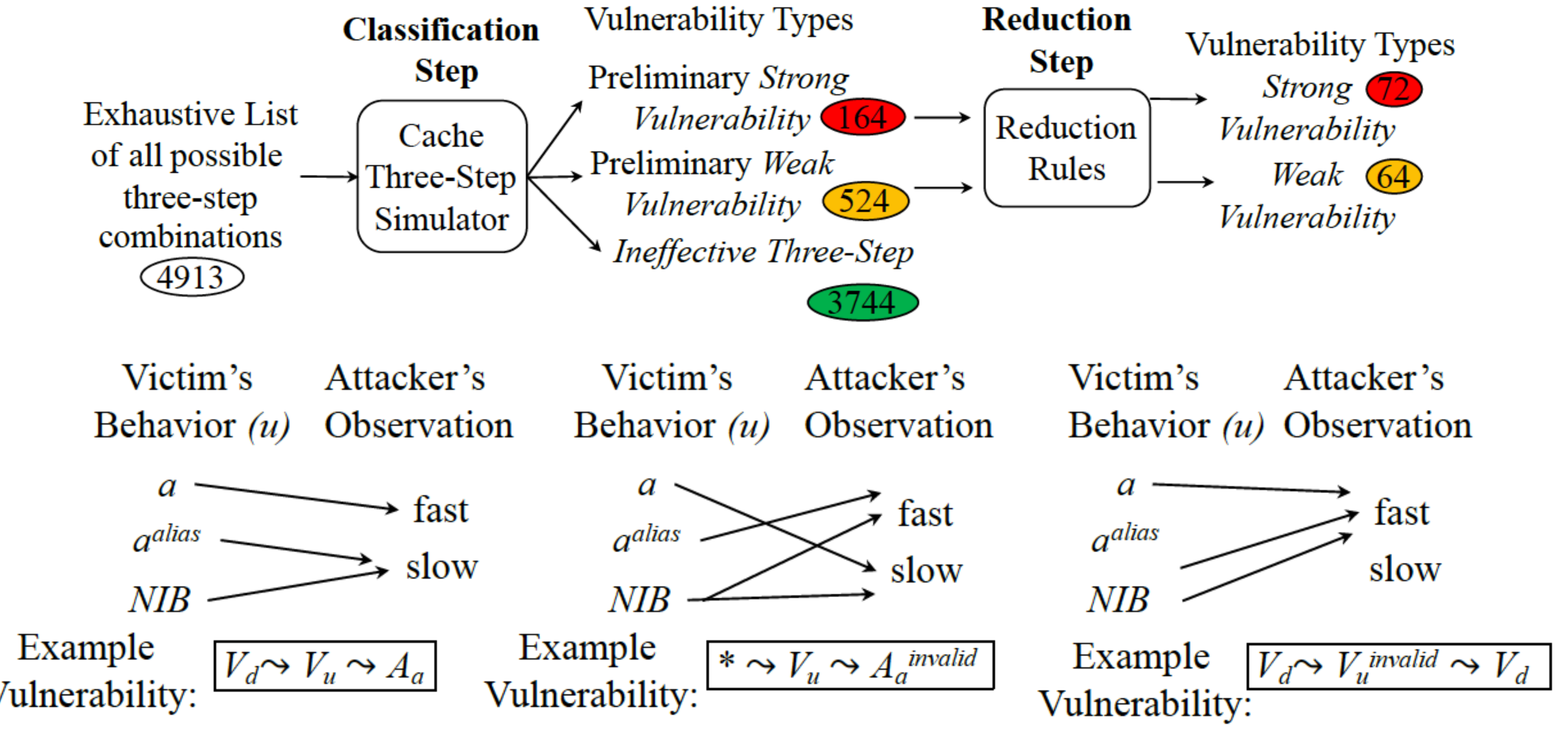
- **There are 72 possible cache timing attack types**

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Three-Step Single-Cache-Block-Access Model (cont'd)

Deng, Shuwen, Xiong, Wenjie, Szefer, Jakub, "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019
Deng, Shuwen, Xiong, Wenjie, Szefer, Jakub, "Cache Timing Side-Channel Vulnerability Checking with Computation Tree Logic", 2018

- There are in total 17 possible states for each of the steps
- Exhaustively listed all 17 (step1) * 17 (step2) * 17 (step3) = 4913 three-step patterns
- Used cache three-step simulator and reduction rules to find all the strong effective vulnerabilities
- In total 72 strong effective vulnerabilities were derived and presented



Exhaustive List of Cache Timing Side- Channel Attacks

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Cache Collision

Flush+ Reload

Evict+ Time

Prime+ Probe

Attack Strategy	Vulnerability Type			Macro Type	Attack
	Step 1	*Step 2*	*Step 3*		
Cache Internal Collision	A	V_u	V_a (fast)	IH	(2)
	V^{inv}	V_u	V_a (fast)	IH	(2)
	A_d	V_u	V_a (fast)	IH	(2)
	V_d	V_u	V_a (fast)	IH	(2)
	$A_{a alias}$	V_u	V_a (fast)	IH	(2)
	$V_{a alias}$	V_u	V_a (fast)	IH	(2)
	A_a^{inv}	V_u	V_a (fast)	IH	(2)
	V_a^{inv}	V_u	V_a (fast)	IH	(2)
Flush + Reload	A_a^{inv}	V_u	A_a (fast)	EH	(5)
	V_a^{inv}	V_u	A_a (fast)	EH	(5)
	A^{inv}	V_u	A_a (fast)	EH	(5)
	V^{inv}	V_u	A_a (fast)	EH	(5)
	A_d	V_u	A_a (fast)	EH	(5)
	V_d	V_u	A_a (fast)	EH	(5)
	$A_{a alias}$	V_u	A_a (fast)	EH	(5)
	$V_{a alias}$	V_u	A_a (fast)	EH	(5)
Reload + Time	V_u	A_a	V_u (fast)	EH	new
	V_u^{inv}	V_a	V_u (fast)	IH	new
Flush + Probe	A_a	V_u^{inv}	A_a (slow)	EM	(6)
	A_a	V_u^{inv}	V_a (slow)	IM	new
	V_a	V_u^{inv}	A_a (slow)	EM	new
	V_a	V_u^{inv}	V_a (slow)	IM	new
Evict + Time	V_u	A_d	V_u (slow)	EM	(1)
	V_u	A_a	V_u (slow)	EM	(1)
Prime + Probe	A_d	V_u	A_d (slow)	EM	(4)
	A_a	V_u	A_a (slow)	EM	(4)
Bernstein's Attack	V_u	V_a	V_u (slow)	IM	(3)
	V_u	V_d	V_u (slow)	IM	(3)
	V_d	V_u	V_d (slow)	IM	(3)
	V_a	V_u	V_a (slow)	IM	(3)
Evict + Probe	V_d	V_u	A_d (slow)	EM	new
	V_a	V_u	A_a (slow)	EM	new
Prime + Time	A_d	V_u	V_d (slow)	IM	new
	A_a	V_u	V_a (slow)	IM	new
Flush + Time	V_u	A_a^{inv}	V_u (slow)	EM	new
	V_u	V_a^{inv}	V_u (slow)	IM	new

(1) Evict + Time attack [31].
(2) Cache Internal Collision attack [4].
(3) Bernstein's attack [2].
(4) Prime + Probe attack [31,33], Alias-driven attack [16].
(5) Flush + Reload attack [50,49], Evict + Reload attack [15].
(6) SpectrePrime, MeltdownPrime attack [41].

Attack Strategy	Vulnerability Type			Macro Type	Attack
	Step 1	*Step 2*	*Step 3*		
Cache Internal Collision Invalidation	A^{inv}	V_u	V_a^{inv} (slow)	IH	new
	V^{inv}	V_u	V_a^{inv} (slow)	IH	new
	A_d	V_u	V_a^{inv} (slow)	IH	new
	V_d	V_u	V_a^{inv} (slow)	IH	new
	$A_{a alias}$	V_u	V_a^{inv} (slow)	IH	new
	$V_{a alias}$	V_u	V_a^{inv} (slow)	IH	new
Flush + Flush	A_a^{inv}	V_u	V_a^{inv} (slow)	IH	(1)
	V_a^{inv}	V_u	V_a^{inv} (slow)	IH	(1)
	A_a^{inv}	V_u	A_a^{inv} (slow)	EH	(1)
	V_a^{inv}	V_u	A_a^{inv} (slow)	EH	(1)
Flush + Reload Invalidation	A^{inv}	V_u	A_a^{inv} (slow)	EH	new
	V^{inv}	V_u	A_a^{inv} (slow)	EH	new
	A_d	V_u	A_a^{inv} (slow)	EH	new
	V_d	V_u	A_a^{inv} (slow)	EH	new
	$A_{a alias}$	V_u	A_a^{inv} (slow)	EH	new
	$V_{a alias}$	V_u	A_a^{inv} (slow)	EH	new
Reload + Time Invalidation	V_u^{inv}	A_a	V_u^{inv} (slow)	EH	new
	V_u^{inv}	V_a	V_u^{inv} (slow)	IH	new
Flush + Probe Invalidation	A_a	V_u^{inv}	A_a^{inv} (fast)	EM	new
	A_a	V_u^{inv}	V_a^{inv} (fast)	IM	new
	V_a	V_u^{inv}	A_a^{inv} (fast)	EM	new
	V_a	V_u^{inv}	V_a^{inv} (fast)	IM	new
Evict + Time Invalidation	V_u	A_d	V_u^{inv} (fast)	EM	new
	V_u	A_a	V_u^{inv} (fast)	EM	new
Prime + Probe Invalidation	A_d	V_u	A_d^{inv} (fast)	EM	new
	A_a	V_u	A_a^{inv} (fast)	EM	new
Bernstein's Invalidation Attack	V_u	V_a	V_u^{inv} (fast)	IM	new
	V_u	V_d	V_u^{inv} (fast)	IM	new
	V_d	V_u	V_d^{inv} (fast)	IM	new
	V_a	V_u	V_a^{inv} (fast)	IM	new
Evict + Probe Invalidation	V_d	V_u	A_d^{inv} (fast)	EM	new
	V_a	V_u	A_a^{inv} (fast)	EM	new
Prime + Time Invalidation	A_d	V_u	V_d^{inv} (fast)	IM	new
	A_a	V_u	V_a^{inv} (fast)	IM	new
Flush + Time Invalidation	V_u	A_a^{inv}	V_u^{inv} (fast)	EM	new
	V_u	V_a^{inv}	V_u^{inv} (fast)	IM	new

(1) Flush + Flush attack [14].

Understanding All Possible Timing Attacks

- The Prime+Probe, Flush+Reload, Evict+Time, or Cache Collision attacks are just some of the possible timing attacks

- **Defenders need to understand all possible types of attacks**, as attacker just needs to find out that works – but defenders need to protect all types of attacks

- A recent **3-step model can be used to understand timing attacks**…

…most attacks have been known in literature under various names, but:

- **Possible new, untested attacks exist**

- Systematic approach to checking for attacks is necessary, not just for caches, but TLBs and other cache-like structures.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Cache Timing Side-Channel Attacks

Secure Cache Techniques

Secure Cache Architectures

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Motivation for Design of Secure Caches

- Software defenses are possible (e.g. page coloring or "constant time" software)

- Require software writers to consider timing attacks, and to consider all possible attacks, if new attack is demonstrated (e.g. from the 3-step model) previously written secure software may no longer be secure

- **Root cause of timing attacks are caches themselves**
  - Correctly functioning caches can leak critical secrets like encryption keys when the cache is shared between victim and attacker
  - Need to care about different levels for the cache hierarchy, different kinds of caches and cache-like structures

- Secure processor architectures also are affected by timing attacks on caches
  - E.g., Intel SGX is vulnerable to some Spectre variants
  - E.g., Cache timing side-channel attacks are possible in ARM TrustZone
  - Secure processors must have secure caches

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Cache Techniques

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

- Numerous academic proposals have presented different secure cache architectures that aim to defend against different cache-based side channels.
- To-date there are 17 secure cache proposals
- They share many similar, key techniques

Secure Cache Techniques:

- **Partitioning** – isolates attacker and victim
- **Randomization** – randomizes address mapping or data brought into the cache
- **Differentiating Sensitive Data** – allows fine-grain control of secure data

Goal of all secure caches is to minimize interference between victim and attacker or victim themselves

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Different Types of Interference Between Cache Accesses

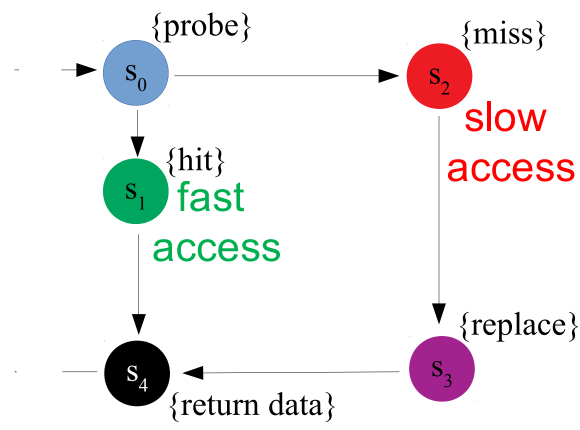
Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Where the interference happens

- External-interference vulnerabilities
  - Main interference (e.g., eviction of one party's data from the cache or observing hit of one party's data) happens between the attacker and the victim
- Internal-interference vulnerabilities
  - Main interference happens within the victim's process itself

Memory reuse conditions

- Hit-based vulnerabilities
  - Cache hit (fast)
  - Invalidation of the data when the data is in the cache (slow)
    - More operation needed (e.g., write back the dirty data)
- Miss-based vulnerabilities
  - Cache miss (slow)
  - Invalidation of the data when the data is in the cache (fast)



Partitioning

- **Goal:** limit the victim and the attacker to be able to only access a limited set of cache blocks
- **Partition among security levels:** High (higher security level) and Low (lower security level) or even more partitions
- **Type:** Static partitioning v.s. dynamic partitioning
- **Partitioning based on:**
  - Whether the memory access is victim's or attacker's
  - Where the access is to (e.g., to a sensitive or not memory region)
  - Whether the access is due to speculation or out-of-order load or store, or it is a normal operations
- **Partitioning granularity:**
  - Cache sets
  - Cache lines
  - Cache ways

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Partitioning (cont'd)

- **Partitioning usually targets external interference**, but are weak at internal interference
  - Interference between the attack and the victim partition becomes impossible
    → attacks based on these types of external interference will fail
  - Interference within victim itself is still possible
- **Wasteful in terms of cache space and degrades system performance**
  - Dynamic partitioning can help limit the negative performance and space impacts
    - At a cost of revealing some side-channel information when adjusting the partitioning size for each part
    - Does not help with internal interference
- **Partitioning in hardware or software**
  - Hardware partitioning
  - Software partitioning
    - E.g. page-coloring

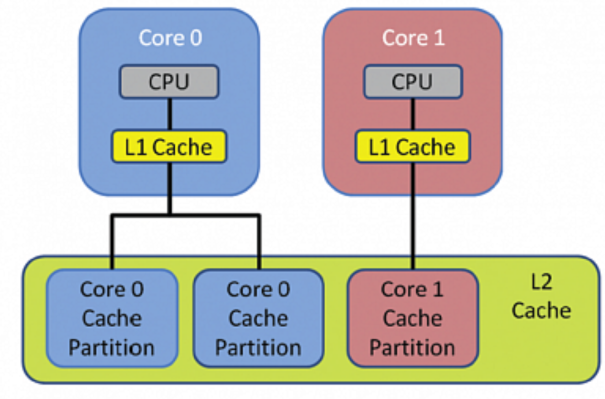


Image: https://www.aerodefensetech.com/component/content/article/adt/features/articles/20339

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Partitioning Summary

Feature	Partition										
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data
		flush adjusted	partial flush								
L1 Cache											
Lower (LLC)											
TLB											

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Randomization

- Inherently de-correlate the relationship among:

Information of victim's security critical data's normal or speculatively-brought address

Observed timing from cache hit or miss

Observed timing of flush or cache coherence operations

  - Randomize the address to cache set mapping
  - Random fill
  - Random eviction
  - Random delay
    - The mutual information from the observed timing could be reduced to 0
- Can be used to reduce miss-based internal interference vulnerabilities
  - May still suffer from hit-based vulnerabilities
- Require a fast and secure random number generator
- Mostly cache-line-based and can be combined with differentiating sensitive data for efficiency

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

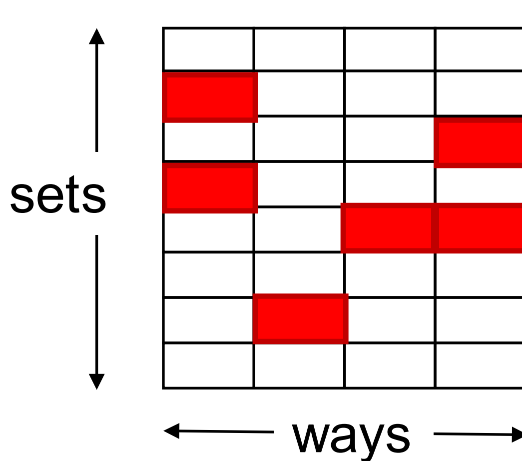
Randomization Summary

Feature	Randomization					
	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay
L1 Cache						
Lower (LLC)						
TLB						

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Differentiating Sensitive Data

- **Allows the victim or attacker software or management software to explicitly label a certain range of the data of victim which they think are sensitive**
- Can **use new cache-specific instructions** to protect the data and limit internal interference between victim's own data
  - E.g., it is possible to disable victim's own flushing of victim's labeled data, and therefore prevent vulnerabilities that leverage flushing
- Allows the designer to **have stronger control over security critical data**
  - How to identify sensitive data and whether this identification process is reliable are open research questions
- **Independent of whether a cache uses partitioning or randomization**
- Has advantage in preventing internal interference



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Differentiating Sensitive Data Summary

Feature	Differentiating sensitive data					
	secure or non-secure	speculative or not	tag hit	tag + id hit	lock bit	protection bit
L1 Cache						
Lower (LLC)						
TLB						

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Cache Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Cache Timing Side-Channel Attacks

Secure Cache Techniques

Secure Cache Architectures

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Caches

Deng, Shuwen, Xiong, Wenjie, Szefer, Jakub, “Analysis of Secure Caches and Timing-Based Side-Channel Attacks”, 2019

17 different secure caches exist in literature:

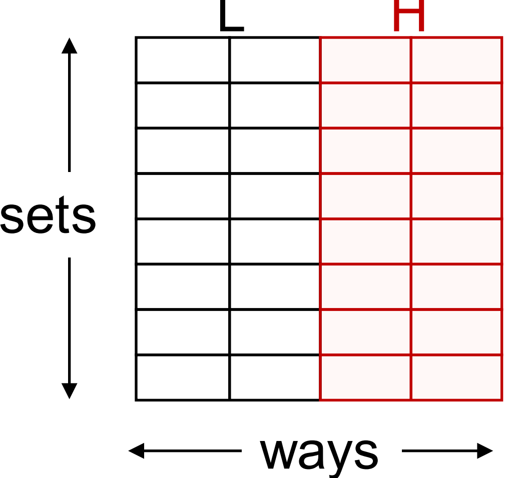
- **Partitioning-based caches**
  - Static Partition (SP) cache, SecVerilog cache, SecDCP cache, Non-Monopolizable (NoMo) cache, SHARP cache, Sanctum cache, MI6 cache, Invisispec cache, CATalyst cache, DAWG cache, RIC cache, Partition Locked (PL) cache
- **Randomization-based caches**
  - SHARP cache, Random Permutation (RP) cache, Newcache, Random Fill (RF) cache, CEASER cache, Non-deterministic cache
- **Differentiating sensitive data**
  - CATalyst cache, Partition Locked (PL) cache, Random Permutation (RP) cache, Newcache, Random Fill (RF) cache, CEASER cache, Non-deterministic cache

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Static Partition (SP) Cache

He, Z., and Lee, R.. "How secure is your cache against side-channel attacks?", 2017.
Lee, R., et al. "Architecture for protecting critical secrets in microprocessors." In ACM SIGARCH Computer Architecture News, vol. 33, no. 2, pp. 2-13. IEEE Computer Society, 2005.

- Basic Design for partition based caches
  - Statically partition the cache for victim and attacker
  - Victim and attacker have different cache ways (or sets)
  - No eviction of the cache line between different processes is allowed
  - Data reuse can be allowed between processes
  - Performance is degraded



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

SP Cache Secure Feature Summary

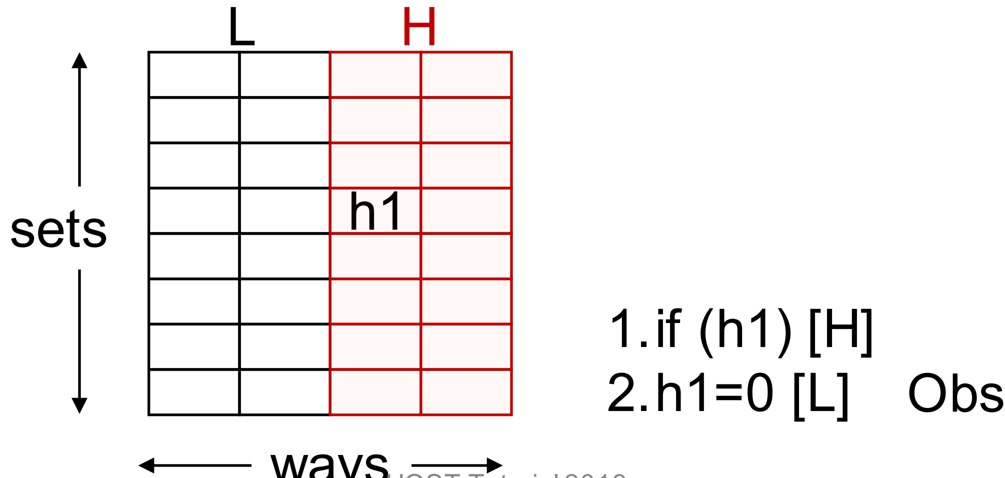
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

SecVerilog Cache

Zhang, D., Askarov, A., & Myers. "Language-based control and mitigation of timing channels", 2012.

- Statically partitioned but allow specific information sharing
  - Statically partitioned to different regions (H (High) and L (Low) security)
    - by different ways
  - Different instructions are tagged with different labels (H and L)
    - H instruction can read H and L partition
    - L instruction can only read L partition
    - Read/write miss, H and L instruction can only modify their own partition (except data will be moved from H to L partition for L miss)



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

SecVerilog Cache Secure Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

SecDCP Cache

Wang, Y., et al. "SecDCP: secure dynamic cache partitioning for efficient timing channel protection", 2016.

- Build on the SecVerilog cache
- Dynamically partitioned
  - Security classes H (High) and L (Low) security, or more
    - by different ways
  - Adjust the ways assigned to L
    - Percentage of cache misses for L instructions ⤋ L's partition size ⤊
  - When adjusting ways
    - Change from L's to H's
      - Cache line is flushed before reallocating
    - Change from H's to L's
      - H lines remain unmodified
    - Reduce extra performance overhead and protect the confidentiality
    - May leak timing information when changing from H's to L's

SecDCP Cache Secure Feature Summary

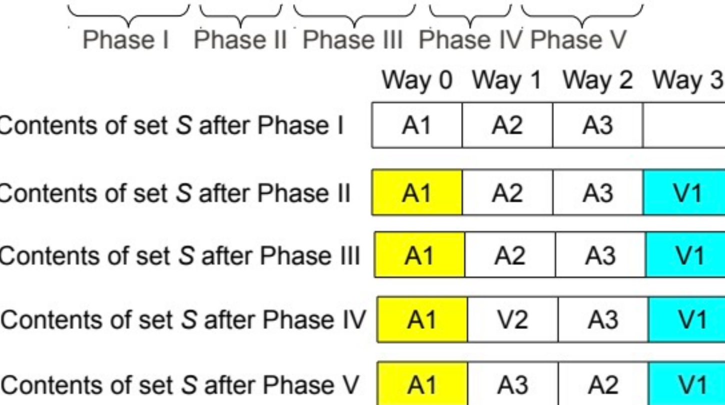
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Non-Monopolizable (NoMo) Cache

Domnitser, L., et al. "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks", *2012.*

- Dynamically partitioned
  - Process-reserved ways and unreserved ways
  - N : number of ways, M : number of SMT threads, Y each thread's exclusively reserved blocks, $Y \in [0, floor(\frac{N}{M})]$. E.g.,
    - NoMo-0: traditional set associative cache
    - NoMo- $floor(\frac{N}{M})$: partitions evenly for the different threads and no non-reserved ways
    - NoMo-1:



  - When adjusting number of blocks assigned to each thread, Y blocks are invalidated

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

NoMo Cache Secure Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
		Dynamic																						
	Static	flush adjusted	partial flush	cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Partitioning-Based Secure Caches vs. Attacks (a)

Deng, S., Xiong, W., Szefer, J., “Analysis of Secure Caches and Timing-Based Side-Channel Attacks”, 2019

	SP cache	SecVerilog cache	SecDCP cache	NoMo cache
external miss-based attacks				
internal miss-based attacks				
external hit-based attacks				
internal hit-based attacks				

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

SHARP Cache

Yan, M., et al. "Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel attacks", 2017.

- Use both partitioning and randomization scheme
- Mainly designed to prevent eviction based attacks
- Cache block augmented with the core valid bits (CVB, similar to process ID)

cache line	CVB

- Replacement policy
  - Cache hit is allowed among different processes
  - Cache misses and data to be evicted following the order:
    - (1) Data not belonging to any current processes
    - (2) Data belonging to the same process
    - (3) Random data in the cache set + an interrupt generated to the OS
  - Eviction between different processes becomes random
- Disallow flush (*clflush*) in the R or X model
  - Invalidation using cache coherence is still possible

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

SHARP Cache Secure Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
		Dynamic																						
	Static	flush adjusted	partial flush	cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Sanctum Cache

Costan, V., Ilia L., and Srinivas D., "Sanctum: Minimal hardware extensions for strong software isolation", 2016.

- Sanctum
  - Open-source minimal secure processor
  - Provide strong provable isolation of software modules running concurrently and sharing resources
  - Isolate enclaves (Trusted Software Module equivalent) from each other and OS
- Cover L1 cache, TLB and last-level cache (LLC)
  - L1 cache and TLB
    - Security monitor (software) flushes core-private cache lines to achieve isolation
  - LLC
    - Page-coloring-based cache partitioning ensure per-core isolation between OS and enclaves
    - Assign each enclave or OS to different DRAM address regions

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Sanctum Cache Secure Feature Summary

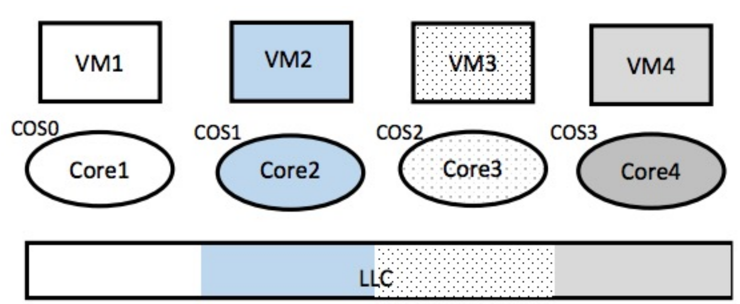
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

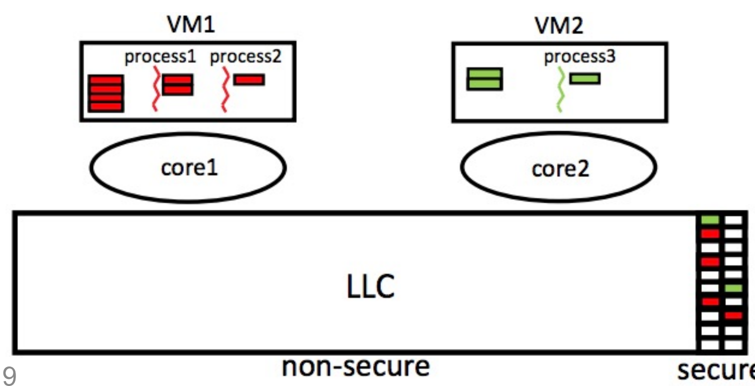
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

CATalyst Cache

Liu, F., et al, "Catalyst: Defeating last-level cache side channel attacks in cloud computing", 2016.

- Targets at LLC
- Use Cache Allocation Technology (CAT) to do coarse partition
  - Available for some Intel processors
  - Allocates up to 4 different Classes of Services (CoS) for separate cache ways
  - Replacement of cache blocks is only allowed within a certain CoS.
  - Partition the cache into secure and non-secure parts
- Use software to do fine partition
  - Secure pages not shared by more than one VM
  - Pesudo-locking mechanism pins certain page frames (immediately bring back after eviction)
    - Malicious code cannot evict secure pages





© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

CATalyst Cache Secure Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

CATalyst Cache

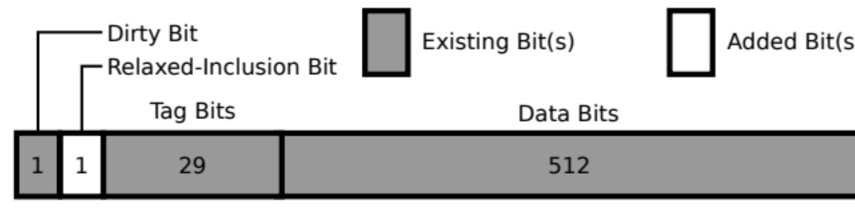
Liu, Fangfei, et al. "Catalyst: Defeating last-level cache side channel attacks in cloud computing", 2016.

- Implicitly performs preloading
  - Remap security-critical code or data to secure pages
- Flushes can only be done within each VM
  - Secure pages are reloaded immediately after the flush
    - Done by virtual machine monitor
    - Make sure all the secure pages are pinned in the secure partition
    - Assume security critical data is able to fit within the secure partition of the cache
- Cache coherence
  - Assign secure pages to only one processor and not sharing pages among VMs

Relaxed Inclusion Caches (RIC)

Kayaalp, M., et al, "RIC: relaxed inclusion caches for mitigating LLC side-channel attacks", 2017.

- Defend against eviction-based timing-based attacks
- Targets on LLC
- Cache replacement of inclusive cache
  - For normal cache
    - Eviction of data in the LLC will cause the same data in L1 cache to be invalidated
    - Eviction-based attacks in the higher level cache possible
      - Attacker is able to evict victim's security critical cache line
  - RIC cache
    - Single relaxed-inclusion bit set
      - Corresponding LLC line eviction will not cause the same line in the higher-level cache to be invalidated
      - Two kinds of data with the bit set
        - Read-only data
        - Threat private data
        - Above two should cover all the critical data for ciphers



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

RIC Cache Secure Feature Summary

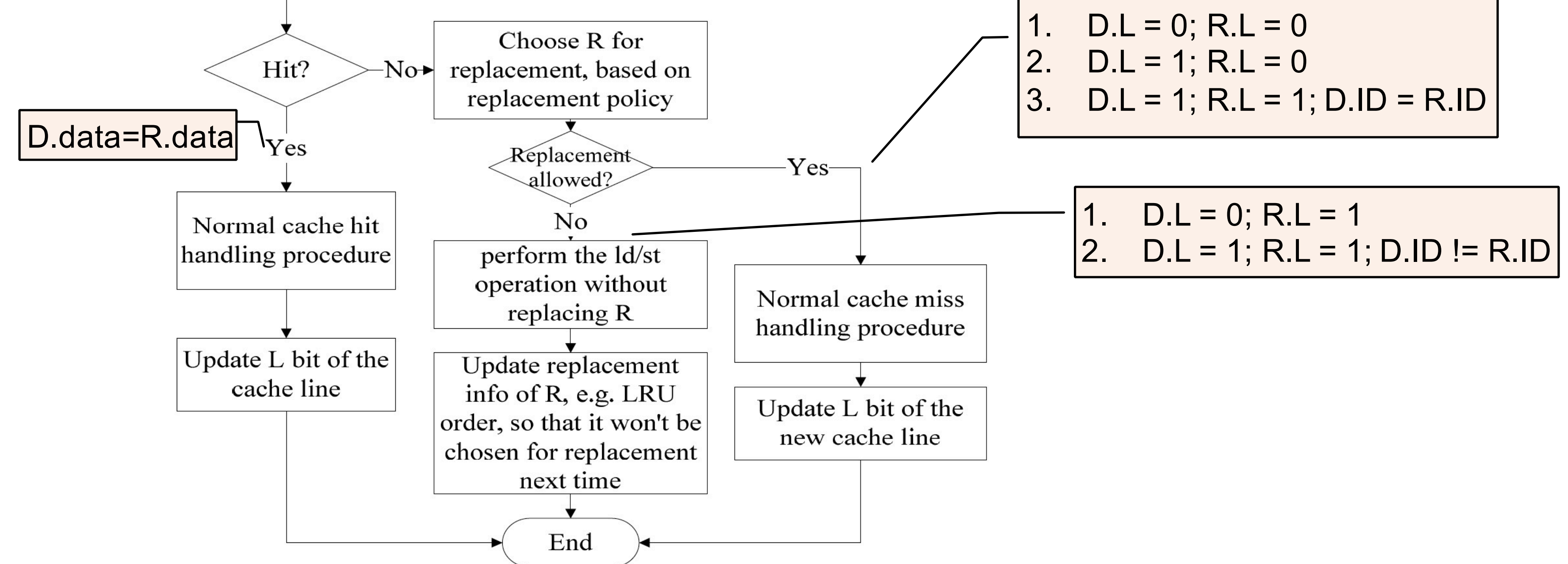
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
		Dynamic																						
	Static	flush adjusted	partial flush	cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Partition Locked (PL) Cache

Wang, Z., and Lee, R.B., "New cache designs for thwarting software cache-based side channel attacks", 2007.

- Dynamically partitioned each cache lines
  - Cache line extended with process identifier (ID) and a locking bit (L)
  - ID and L are controlled by extending load/store instruction
    - ld.lock/ld.unlock & st.lock/st.unlock
  - Replacement policy (D: brought in; R: replaced)



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

PL Cache Secure Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Partitioning-Based Secure Caches vs. Attacks (b)

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

	SHARP cache	Sanctum cache	CATalyst cache	RIC cache	PL cache
external miss-based attacks					
internal miss-based attacks					
external hit-based attacks					
internal hit-based attacks					

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

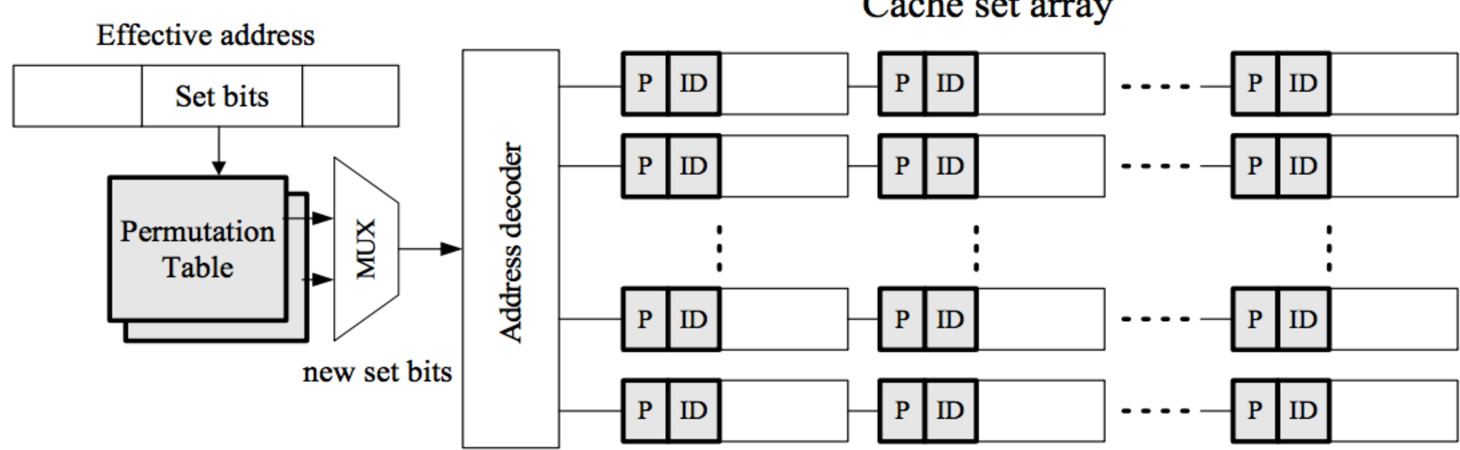
Random Permutation (RP) Cache

Wang, Z., and Lee, R.B., "New cache designs for thwarting software cache-based side channel attacks", 2007.

- Use randomization
  - De-correlate the memory address accessing and timing of the cache
- Process ID and protection bit (P) extended for each line

P	ID	Original cache line

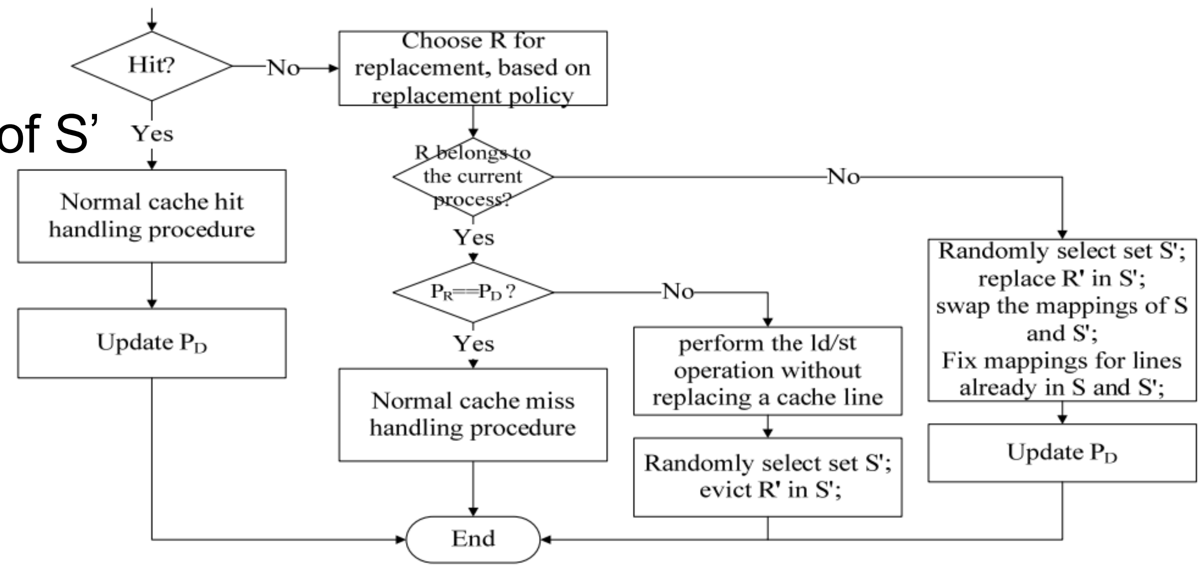
- A permutation table (PT) maintained
  - Store each cache set's pre-computed permuted set number
    - Number of tables depends on the number of protected processes



Random Permutation (RP) Cache

Wang, Z., and Lee, R.B., "New cache designs for thwarting software cache-based side channel attacks", 2007.

- Replacement policy
  - Cache hits
    - When both process ID and the address are the same
  - Cache misses (D: brought in; R: replaced)
    - D and R in the same process, have different protection bits
      - Arbitrary data of a random cache set S' is evcted
      - D is accessed without caching
    - D and R in the different processes
      - D is stored in an evicted cache block of S'
      - Mapping of S and S' is swapped
    - Other cases
      - Normal replacement policy is used



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

RP Cache Secure Feature Summary

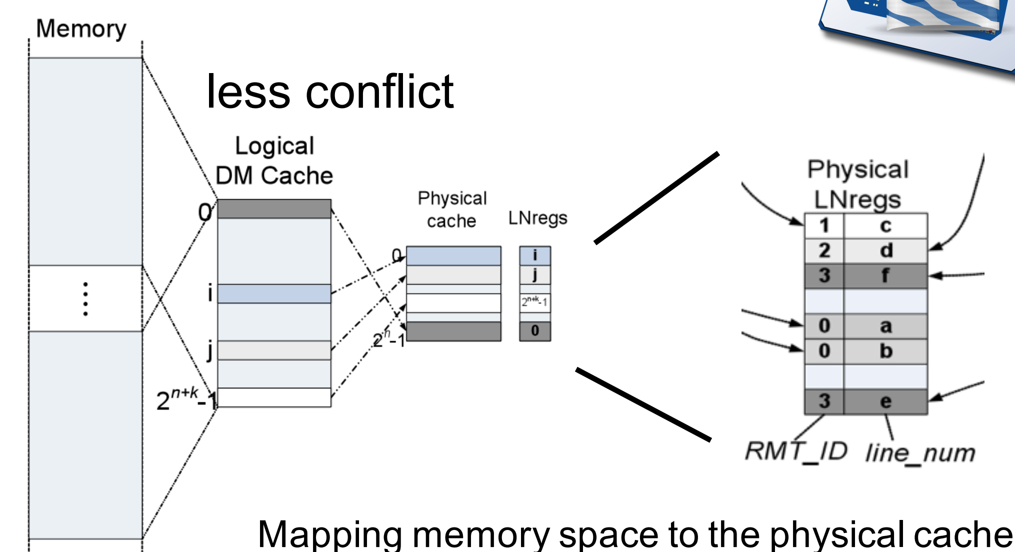
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

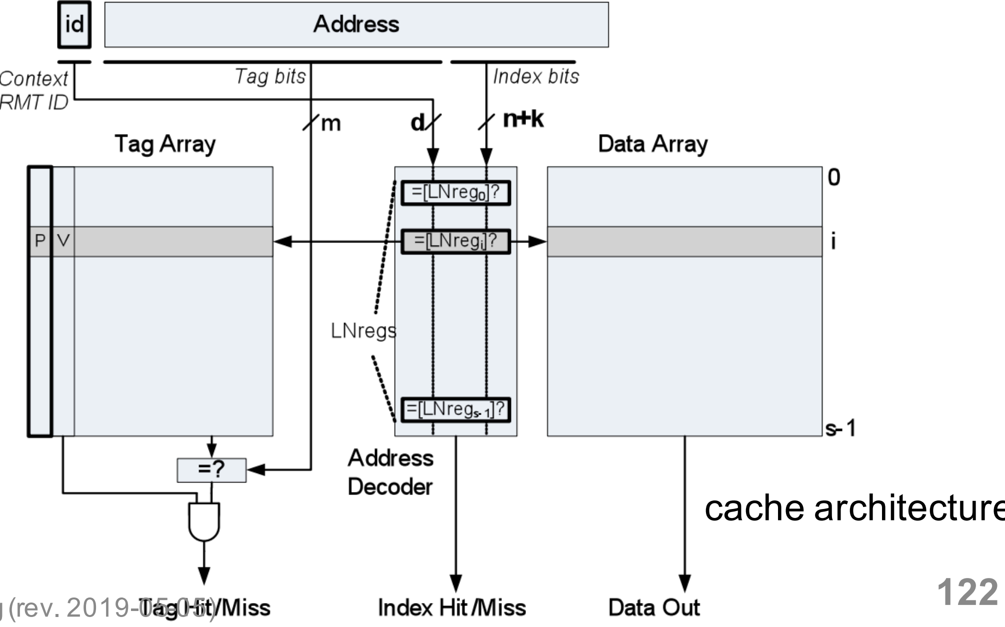
Newcache Cache

Wang, Z., and Lee, R.B., "A novel cache architecture with enhanced performance and security", 2008.

- Dynamically randomize memory-to-cache mapping
- Maintain a ReMapping Table (RMT)
  - Mapping between memory address and RMT
    - As direct mapped
  - logic-index bits of memory address used to look up entries in the RMT
- Each cache line has RMT ID and a protection bit (P)
- Cache Access
  - Index miss
    - Context RMT ID and index bit match
  - Tag miss
    - Tag matches
  - Replacement policy similar to RP cache
    - Except no normal replacement for any protected-data-related replacing



Mapping memory space to the physical cache



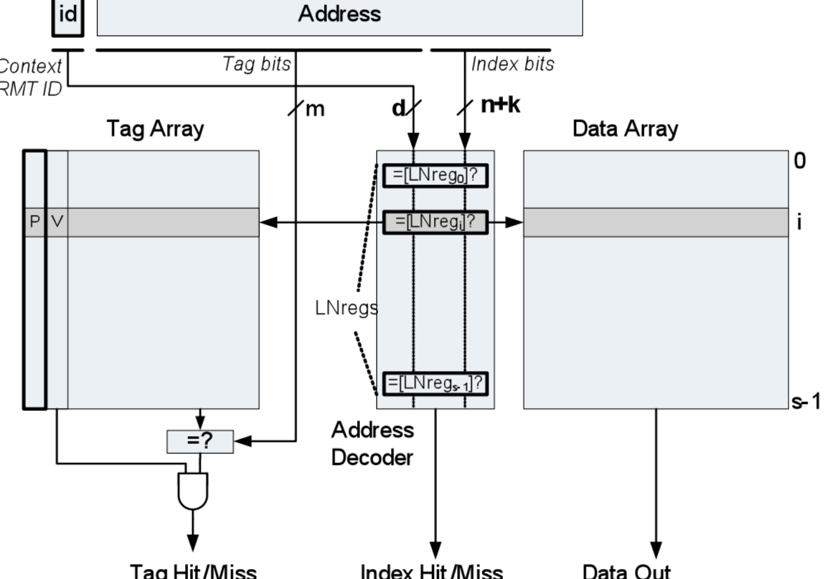
cache architecture

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Newcache Cache

Wang, Z., and Lee, R.B., "A novel cache architecture with enhanced performance and security", 2008.

- Each line is associated with the RMT ID and a protection bit (P)
- Index bits of memory address used to look up entries in the RMT
- Index stored in RMT combined with the process ID is used to look up the actual cache



- Cache replacement policy
  - Similar to RP cache with small differences
    - Any one of the D and R in the same process and having P bit set will cause random eviction without caching triggered

Newcache Secure Feature Summary

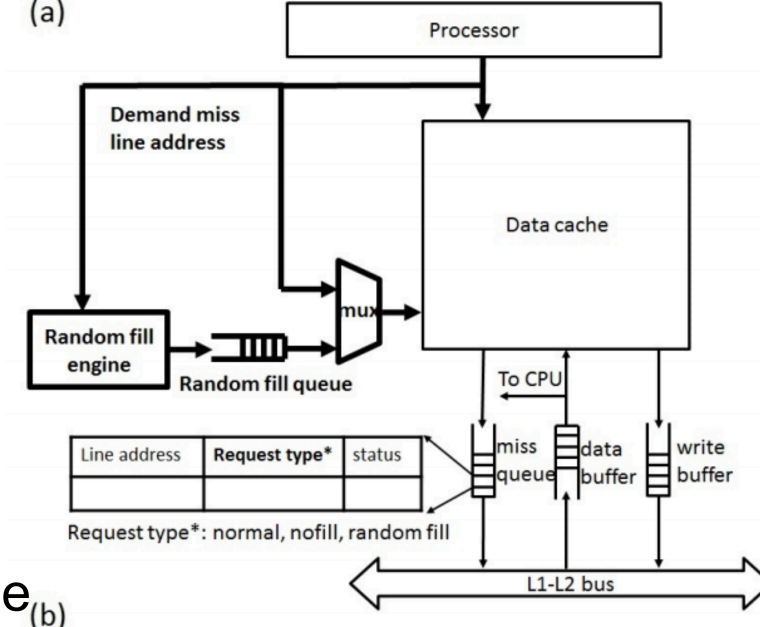
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

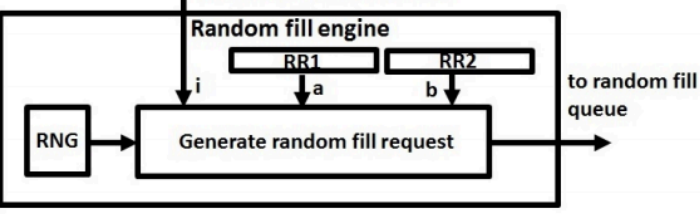
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Random Fill (RF) Cache

Liu, F., and Lee, R.B., "Random fill cache architecture.", 2014.

- De-correlate cache fills with the memory access
- Targets on hit-based attacks
- Multiple types of requests
  - Normal data: “normal fill”
  - Demand request: “nofill”
  - Random fill request
    - Look up the cache
    - Get forwarded to miss queue on a miss
    - “random fill” the address calculated by the random fill engine
- Random Fill Engine
  - Generate an access within a neighborhood
  - Two range registers (RR1 and RR2)
    - (LowerBound, Range) or (LowerBound, UpperBound)
  - Window size can be customized





a) block diagram
b) random fill engine

RF Secure Feature Summary

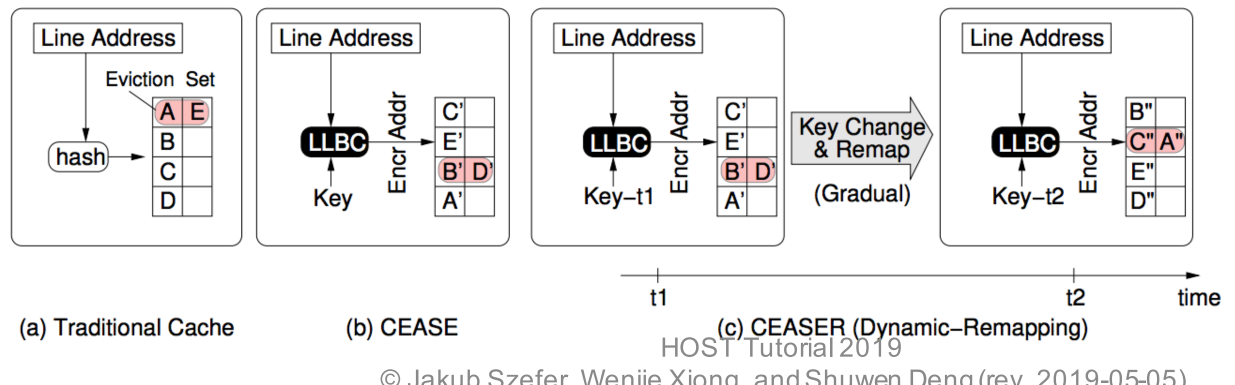
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

CEASER Cache

Qureshi, M. K, "CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping", 2018.

- Mitigating conflict-based cache attacks
- When memory access tries to modify the cache state
  - The address is encrypted using Low-Latency BlockCipher (LLBC)
    - Randomize the cache set it maps
    - Scatters the original, possible ordered addresses to different cache sets
      - Decrease rate of conflict misses
    - Encryption and decryption can be done within 2 cycles using LLBC
- Encryption key will be periodically changed to avoid key reconstruction
  - Dynamically change the address remapping
  - Improved work to be appeared @ISCA 2019



(a) Traditional Cache (b) CEASE (c) CEASER (Dynamic–Remapping)

CEASER Secure Feature Summary

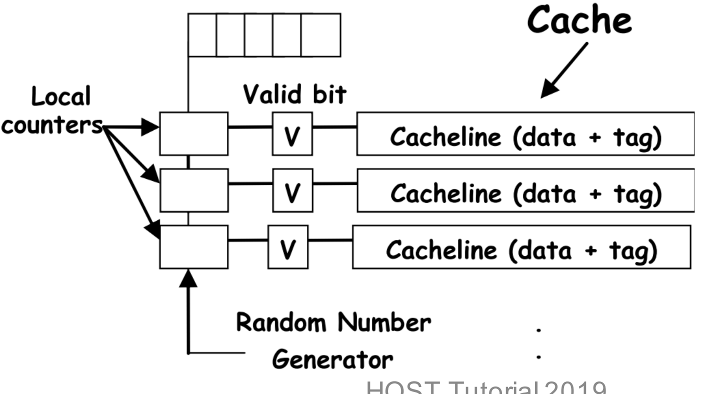
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Non Deterministic Cache

Keramidas, G., et al. "Non deterministic caches: A simple and effective defense against side channel attacks", 2008.

- Use cache access decay to randomize the relation between accessing and timing
- Counters control the decay of a cache block
  - Local counter records the interval of its data activeness
    - Increased on each global counter clock tick
    - When reaching a predefined value
      - Corresponding cache line is invalidated
- Non deterministic cache randomly sets local counter's initial value
  - Can lead to different cache hit and miss statistics
  - May have larger performance degradation compared with other data-targeted secure caches



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Non Deterministic Secure Feature Summary

Feature	Partition										Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																				
L1 Cache																							
Lower (LLC)																							
TLB																							

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Randomization-Based Secure Caches vs. Attacks

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

	RP cache	Newcache	RF cache	CEASER cache	Non deterministic cache
external miss-based attacks					
internal miss-based attacks					
external hit-based attacks					
internal hit-based attacks					

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

MI6 Cache

Bourgeat, T., et al. "MI6: Secure Enclaves in a Speculative Out-of-Order Processor",2018.

- Speculation-related cache
- MI6
  - Secure Enclaves in a Speculative Out-of-Order Processor
  - Isolation of enclaves (Trusted Software Module equivalent) from each other and OS
- Combination of:
  - Sanctum cache's security feature
  - Disabling speculation during the speculative execution of memory related operations

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

MI6 Cache Secure Feature Summary

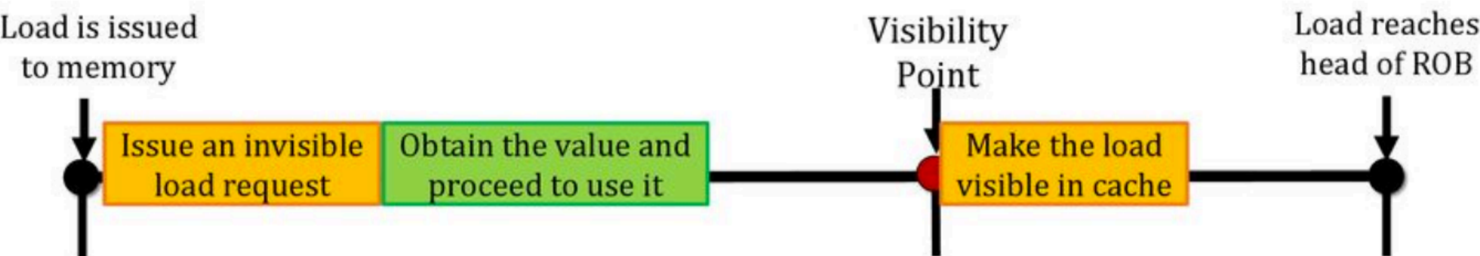
Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
		Dynamic																						
	Static	flush adjusted	partial flush	cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

InvisiSpec Cache

Yan, M., et al. "Invisispec: Making speculative execution invisible in the cache hierarchy",2018.

- Speculation-related cache
- A speculative buffer (SB) will store the unsafe speculative loads (USL) before modifying the cache states
  - Mismatch of data in the SB and the up-to-date values in the cache
    - Squashed
  - The core receives possible invalidation from the OS before checking of memory consistency model
    - No comparison is needed
- Target on Spectre-like attacks



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

InvisiSpec Cache Secure Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Dynamically Allocated Way Guard (DAWG) Cache

Kiriansky, V., et al. "DAWG: A defense against cache timing attacks in speculative execution processors", 2018.

- Use partitioning schemes
- Provide full isolation for hits, misses and metadata between the attacker and the victim
- Cache hits
  - When both the cache address tag and *domain_id (process ID) associated* are the same
  - Allows read-only cache lines to be replicated across different domains
- Cache misses
  - Victim can only be chosen within the ways belonging to the same *domain_id*
  - Replacement policy's bits and metadata is updated within the domain selection
- Noninterference property
  - Orthogonal to speculative execution
  - Existing attacks such as Spectre Variant 1 and 2 will not work on a system equipped with DAWG

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

DAWG Cache Secure Feature Summary

Feature	Partition											Randomization						Differentiating Sensitive Data						User Program edit needed
	Static	Dynamic		cache set	cache way	cache line	page coloring	software monitor	CAT	relax inclusive	sensitive data	randomize cache set mapping	random fill	random evict	encrypt address	key reconstruction	random delay	secure or non-secure	speculative or not	tag hit	tag +id hit	lock bit	protection bit	
		flush adjusted	partial flush																					
L1 Cache																								
Lower (LLC)																								
TLB																								

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Speculation-Related Secure Caches vs. Attacks

Deng, S., Xiong, W., Szefer, J., “Analysis of Secure Caches and Timing-Based Side-Channel Attacks”, 2019

	MI6 cache		InivisiSpec cache		DAWG cache	
	Normal	Specu-lative	Normal	Specu-lative	Normal	Specu-lative
external miss-based attacks						
internal miss-based attacks						
external hit-based attacks						
internal hit-based attacks						

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Cache Configuration

Deng, S., Xiong, W., Szefer, J., “Analysis of Secure Caches and Timing-Based Side-Channel Attacks”, 2019

Metric		SP* Cache	SecVerilog Cache	SecDCP Cache	NoMo Cache	SHARP Cache	Sanctum Cache	MI6 Cache	InvisiSpec Cache	CATalyst Cache	DAWG Cache	RIC	PL Cache	RP Cache	Newcache	Random Fill Cache	CEASER Cache	Non Deterministic Cache
Cache Configuration	L1 Cache	-	4-way 32KB	private 2-way 32KB D/I	8-way 32KB D/I	private 4-way 32KB D/I	8-way 32KB D/I	8-way 32KB D/I	private 8-way 64KB D, 4-way 32KB I	-	private 8-way 32KB	4-way 32KB D/I	direct-mapped, 2-way and 4-way 4KB to 32KB	2-way 4-way 16KB 32KB	2-way, 4-way or 8-way	4-way 32KB	private 8-way 32KB	2-way 2KB D/I
	L2 Cache	-	-	shared 8/16-way 1/2 MB	unified 8-way 256KB	private 8-way 256KB	256KB 8-way L2	1MB, 16-way, max 16 requests	-	-	private 8-way 256 KB	8-way 256 KB	-	-	-	8-way 2 MB	private 8-way 256KB	shared 4-way 128 KB
	L3 Cache	-	-	16-way 2MB	shared 16-way 2MB	shared 16-way 2MB	8MB 16-way	coherent with I and D	shared 16-way 2MB	20-way 20 MB	shared 8× 16-way 2 MB	shared 16-way 2 MB/512 KB	-	-	-	-	shared 16-way 8MB	-

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Cache Implementation

Deng, S., Xiong, W., Szefer, J., “Analysis of Secure Caches and Timing-Based Side-Channel Attacks”, 2019

Metric	SP* Cache	SecVerilog Cache	SecDCP Cache	NoMo Cache	SHARP Cache	Sanctum Cache	MI6 Cache	InvisiSpec Cache	CATalyst Cache	DAWG Cache	RIC	PL Cache	RP Cache	Newcache	Random Fill Cache	CEASER Cache	Non Deterministic Cache
Benchmark	RSA, AES and MD5	MiBench, ciphers and hash functions of OpenSSL	SPEC 2006	SPEC 2006	SPEC INT2006, SPEC FP2006 and PARSEC	SPEC INT2006	SPEC INT2006	SPEC INT2006, SPEC FP2006 and PARSEC	SPEC 2006 and PARSEC	PARSEC and GAP Benchmark Suite (GAPBS)	SPEC 2006	AES, SPEC 2000	AES, SPEC 2000	SPEC 2000	SPEC 2006	SPEC 2006 and GAP	AES cryptographic algorithm
Implementation	-	MIPS processor	Gem5 simulator	Pin based trace-driven x86 simulator	MARSS cycle-level full-system simulator	Rocket Chip Generator	Riscy OO processor+ Xilinx FPGA	Gem5 simulator + CACTI 5	Intel Xeon E5 2618L v3 processors	zsim x86-64 simulator, Haswell hardware	Cacti version 6.5	M-Sim v2.0	M-Sim v2.0	CACTI 5.0	Gem5 simulator	Pin-based x86 simulator	HotLeakage simulator

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Secure Cache Performance

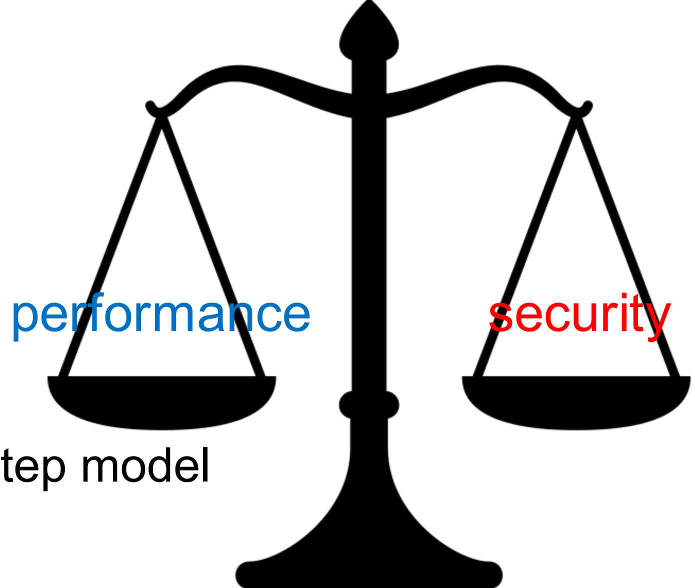
Deng, S., Xiong, W., Szefer, J., “Analysis of Secure Caches and Timing-Based Side-Channel Attacks”, 2019

Metric	SP* Cache	SecVerilog Cache	SecDCP Cache	NoMo Cache	SHARP Cache	Sanctum Cache	MI6 Cache	InvisiSpec Cache	CATalyst Cache	DAWG Cache	RIC	PL Cache	RP Cache	Newcache	Random Fill Cache	CEASER Cache	Non Deterministic Cache
Performance Overhead	1%	-	12.5% better over SP cache	1.2% average, 5% worst	3%-4%	-	-	reduce slowdown of Spectre from 74% to 21%	average slowdown of 0.7% for SPEC and 0.5% for PARSEC	L1 and L2 most 4%-7%	improves 10%	12%	0.3%, 1.2% worst	within the 10% range of the real miss rate	3.5%, 9% if setting the window size to be largest	1% for performance optimization	7% with simple benchmarks
Power	-	-	-	-	-	-	-	L1 0.56 mW, LLC 0.61 mW	-	-	-	-	average 1.5nj	<5% power	-	-	-
Area Overhead	-	-	-	-	-	-	-	L1-SB LLC-SB Area (mm2) 0.0174 0.0176	-	-	0.176 %	-	-	-	-	-	-

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Research Challenges

- Balance tradeoff between performance and security
  - Curse of quantitative computer architecture: focus on performance, area, power numbers, but no easy metric for security – designers focus on performance, area, power numbers since they are easy to show ”better” design, there is no clear metric to say deign is “more secure” than another design
- Running on simulation vs. real machines
  - Simulation workloads may not represent real systems, performance impact of security features is unclear
  - Real systems (hardware) can be modified to add new features and test security
- Embedded with commercial processors and secure processors
  - Many designs exist, but not in commercial processors
- Formal verification of the secure feature implementations
  - Still limited work on truly showing design is secure
  - Also, need more work on modelling all possible attacks, a la the 3-step model



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Tutorial Outline & Schedule

15:30 – 16:10	Secure Processor Architectures
16:10 – 16:20	Break
16:20 – 17:10	Secure Processor Caches
17:10 – 17:20	**Break**
17:20 – 18:00	**Transient Execution Attacks and Mitigations**
18:00	**Wrap Up**

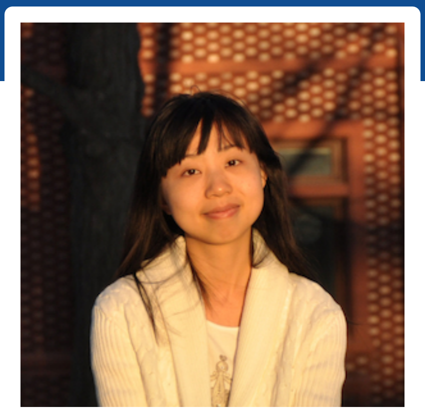
Slides and information at:
http://caslab.csl.yale.edu/tutorials/host2019/

WiFi Information:
Network: Hilton-Meeting, Password: HOST2019

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Transient Execution Attacks and Mitigations

Jakub Szefer


Wenjie Xiong

Shuwen Deng

Dept. Of Electrical Engineering
Yale University

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Outline

- **Taxonomy of Transient Execution Attacks**
  - What are transient execution attacks?
  - How dangerous are the attacks?
  - How the attacks happen?

- **Mitigations in Micro-Architecture**
  - How to mitigate the transient execution attacks in micro-architecture designs?

- **Secure Architectures and Transient Execution Attacks**
  - How the transient execution attacks affect secure architectures?
  - How to protect secure architecture from transient execution attacks?

- **Timing Side Channels which use Speculation**

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Taxonomy of Transient Execution Attacks

Mitigations in Micro-Architecture

Secure Architectures and Transient Execution Attacks

Timing Side Channels which use Speculation

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Spectre, Meltdown, and More…

- Spectre, Meltdown, and their variants exploit fundamental design flaws existing in nearly all processors.
- So far, fixes in software are expensive

	Meltdown	Spectre
Affected CPU Types	Intel, Apple	Intel, Apple, ARM, AM
Attack Vector	Execute Code on the System	Execut on the
Method	Intel Privilege Escalation & Speculative Execution *(CVE-2017-5754)*	Branch Pr Speculativ *(CVE-2017*
Exploit Path	Read Kernel Memory from User Space	Read Memo
Remediation	Software Patches	

https://www.alienvault.com/blogs/security-essentials/improve your-readiness-to-defeat-meltdown-spectre

Spectre and Meltdown vulnerabilities show haste makes waste

When the Meltdown and Spectre vulnerabilities c find a fix. As a result, the patching process has b

Ed Tittel

IT professionals, especially those wit unpleasant surprise when the Spectr light.

Security

Meltdown/Spectre fixes made AWS CPUs cry, says SolarWinds

CPU utilization up, throughput down, but a second

27 SHARE ▼

Meltdown and Spectre, one year on: Feared CPU slowdown never really materialized

John Leyden 31 January 2019 at 12:01 UTC

Hardware | Performance | Secure Development

NEWS | FEATURES | PHOTOS | CRN PIPELINE | CRN FAST50 | CHANNEL CHIEFS | MAGAZINE | RESOURCES | LOG IN | SUBSCRIBE

15 months after Spectre and Meltdown, the fixes are still flowing

By Simon Sharwood
Apr 8 2019
11:16AM

The Spectre and Meltdown CPU design flaw bugs that emerged in early January 2018 are still creating work for users.

Cisco last week issued a Field Notice to users of its Content Delivery

How Dangerous are these Attacks?

- **Can read all memory in the victim's address space**
  - privileged data — e.g., Meltdown [M. Lipp et al., 2018]
  - across virtual machine — e.g., Foreshadow-NG [J. Van Bulck, 2018]
  - data in SGX enclave — e.g., SGXpectre [G. Chen et al., 2018], Foreshadow [J. Van Bulck, 2018]
  - memory-protection keys (PKU) — e.g., Meltdown-PK [C. Canella et al., 2018]
  - sandbox in JavaScript — e.g., Spectre [P. Kocher et al, 2018]

- **Can read stale data**
  - e.g., Lazy FPU [J. Stecklina et al., 2018], Spectre v4

- **Some of the variants can attack across CPU cores**
  - e.g., Spectre v1 Flush+Reload [P. Kocher et al, 2018]

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Transient Execution Attacks

- Spectre, Meltdown, etc. leverage the instructions that are **executed transiently**:
  1. These transient instructions execute for a short time (e.g. due to mis-speculation),
  2. until processor computes that they are not needed, and
  3. the pipeline flush occurs and it **should discard any architectural effects** of these instructions so
  4. architectural state remain as if they never executed, but …

These attacks exploit transient execution to encode secrets through **microarchitectural side effects** that can later be recovered by an attacker through a (most often timing based) observation at the architectural level

Transient Execution Attacks = Transient Execution + Covert or Side Channel

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Example: Spectre (v1) – Bounds Check Bypass

Example of Spectre variant 1 attack:

Victim code:

```
struct array *arr1 = ...;
struct array *arr2 = ...;
unsigned long offset = ...;
if (offset < arr1->len) {
  unsigned char value = arr1->data[offset];
  unsigned long index = value;
  unsigned char value2 = arr2->data[index];
...
```


The attacker can then check if arr2[X] is in the cache. If so, secret = X

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Attack Components

Microsoft, https://blogs.technet.microsoft.com/srd/2018/03/15/mitigating-speculative-execution-side-channel-hardware-vulnerabilities/

Attacks leveraging transient execution have 4 components:

```
e.g. if (offset < arr1->len) {
       unsigned char value = arr1->data[offset];
       unsigned long index = value;
       unsigned char value2 = arr2->data[index];
   ...
```

- `if (offset < arr1->len)` → Speculation Primitive; `arr1->len` is not in cache → Windowing Gadget
- `value`, `index`, `value2` lines → Disclosure Gadget; cache Flush+Reload covert channel → Disclosure Primitive

1. Speculation Primitive
"provides the means for entering transient execution down a non-architectural path"

2. Windowing Gadget
"provides a sufficient amount of time for speculative execution to convey information through a side channel"

3. Disclosure Gadget
"provides the means for communicating information through a side channel during speculative execution"

4. Disclosure Primitive
"provides the means for reading the information that was communicated by the disclosure gadget"

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Speculation Primitives

C. Canella, et al., "A Systematic Evaluation of Transient Execution Attacks and Defenses", 2018

1. Speculation Primitive

- **Spectre-type**: transient execution after a prediction
  - Branch prediction
    - Pattern History Table (PHT) — Bounds Check bypass (V1)
    - Branch Target Buffer (BTB) — Branch Target injection (V2)
    - Return Stack Buffer (RSB) — SpectreRSB (V5)
  - Memory disambiguation prediction — Speculative Store Bypass (V4)
- **Meltdown-type**: transient execution following a CPU exception

	Exception Type				Permission Bit					
Attack	#GP	#NM	#BR	#PF	U/S	P	R/W	RSVD	XD	PK
Variant 3a [10]	●	○	○	○						
Lazy FP [83]	○	●	○	○						
Meltdown-BR	○	○	●	○						
Meltdown [59]	○	○	○	●	●	○	○	○	○	○
Foreshadow [90]	○	○	○	●	○	●	○	●	○	○
Foreshadow-NG [93]	○	○	○	●	○	●	○	●	○	○
Meltdown-RW [50]	○	○	○	●	○	○	●	○	○	○
Meltdown-PK	○	○	○	●	○	○	○	○	○	●

GP: general protection fault
NM: device not available
BR: bound range exceeded
PF: page fault
U/S: user / supervisor
P: present
R/W: read / write
RSVD: reserved bit
XD: execute disable
PK: memory-protection keys (PKU)

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Speculation Primitives – Sample Code

- **Spectre-type:** transient execution after a prediction
  - **Branch prediction**
    - Pattern History Table (PHT) -- Bounds Check bypass (V1)
    - Branch Target Buffer (BTB) -- Branch Target injection (V2)
    - Return Stack Buffer (RSB) -- SpectreRSB (V5)
  - **Memory disambiguation prediction** -- Speculative Store Bypass (V4)

Spectre Variant 1

```
struct array *arr1 = ...;
struct array *arr2 = ...;
unsigned long offset = ...;
if (offset < arr1->len) {
  unsigned char value = arr1->data[offset];
  unsigned long index = value;
  unsigned char value2 = arr2->data[index];
...
```

Spectre Variant 2

```
        (Attacker trains the BTB
        to jump to GADGET)

        jmp LEGITIMATE_TRGT
        ...
GADGET: mov r8, QWORD PTR[r15]
        lea rdi, [r8]
        ...
```

Spectre Variant 5

```
        (Attacker pollutes the RSB)
main:   Call F1
        ...
F1:     ...
        ret
        ...
GADGET: mov r8, QWORD PTR[r15]
        lea rdi, [r8]
        ...
```

Spectre Variant 4

```
char sec[16] = ...;
char pub[16] = ...;
char arr2[0x200000] = ...;
char * ptr = sec;
char **slow_ptr = *ptr;
clflush(slow_ptr)
*slow_ptr = pub;
```

Store “slowly”

```
value2 = arr2[(*ptr)<<12];
```

Load the value at the same memory location "quickly”. “ptr” will get a stale value.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Speculation Primitives – Sample Code

C. Canella, et al., "A Systematic Evaluation of Transient Execution Attacks and Defenses", 2018

Meltdown-type: transient execution following a CPU exception

Attack	#GP	#NM	#BR	#PF	U/S	P	R/W	RSVD	XD	PK
	Exception Type				Permission Bit					
Variant 3a [10]	●	○	○	○						
Lazy FP [83]	○	●	○	○						
Meltdown-BR	○	○	●	○						
Meltdown [59]	○	○	○	●	●	○	○	○	○	○
Foreshadow [90]	○	○	○	●	○	●	○	●	○	○
Foreshadow-NG [93]	○	○	○	●	○	●	○	●	○	○
Meltdown-RW [50]	○	○	○	●	○	○	●	○	○	○
Meltdown-PK	○	○	○	●	○	○	○	○	○	●

GP: general protection fault
NM: device not available
BR: bound range exceeded
PF: page fault

U/S: user/surpervisor
P: present
R/W: read/write
RSVD: reserved bit
XD: execute disable
PK: memory-protection keys (PKU)

```
(rcx = address lead to exception)
(rbx = probe array)
Retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
Mov rbx, qword [rbx + rax]
```

[M. Lipp et al., 2018]

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Windowing Gadget

2. Windowing Gadget

Windowing gadget is used to create a “window” of time for transient instructions to execute while the processor resolves prediction or exception:

- Loads from main memory
- Chains of dependent instructions, e.g., floating point operations, AES

Memory access time determines how long it takes to resolve the branch

E.g.: Spectre v1 :

```
if (offset < arr1->len) {
  unsigned char value = arr1->data[offset];
  unsigned long index = value;
  unsigned char value2 = arr2->data[index];
  ...
}
```

Necessary (but not sufficient) success condition:
speculative window size > disclosure gadget’s triger latency

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Disclosure Gadget

3. Disclosure Gadget

1. Load the secret to register
2. Encode the secret into channel

} Transient execution

The code pointed by the arrows is the disclosure gadget:

Spectre Variant1 (Bounds check)
Cache side channel

```
struct array *arr1 = ...;
struct array *arr2 = ...;
unsigned long offset = ...;
if (offset < arr1->len) {
➡  unsigned char value = arr1->data[offset];
   unsigned long index = value;
➡  unsigned char value2 = arr2->data[index];
...
```

AVX side channel

```
if(x < bitstream_length){
➡   if(bitstream[x])
➡       _mm256_instruction();
}
```

Spectre Variant2 (Branch Poisoning)
Cache side channel

```
        (Attacker trains the BTB
        to jump to GADGET)

        jmp LEGITIMATE_TRGT
        ...
GADGET: mov r8, QWORD PTR[r15]
➡       lea rdi, [r8]
        ...
```


© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Disclosure Primitives

4. Disclosure Primitive

Two types of disclosure primitives:

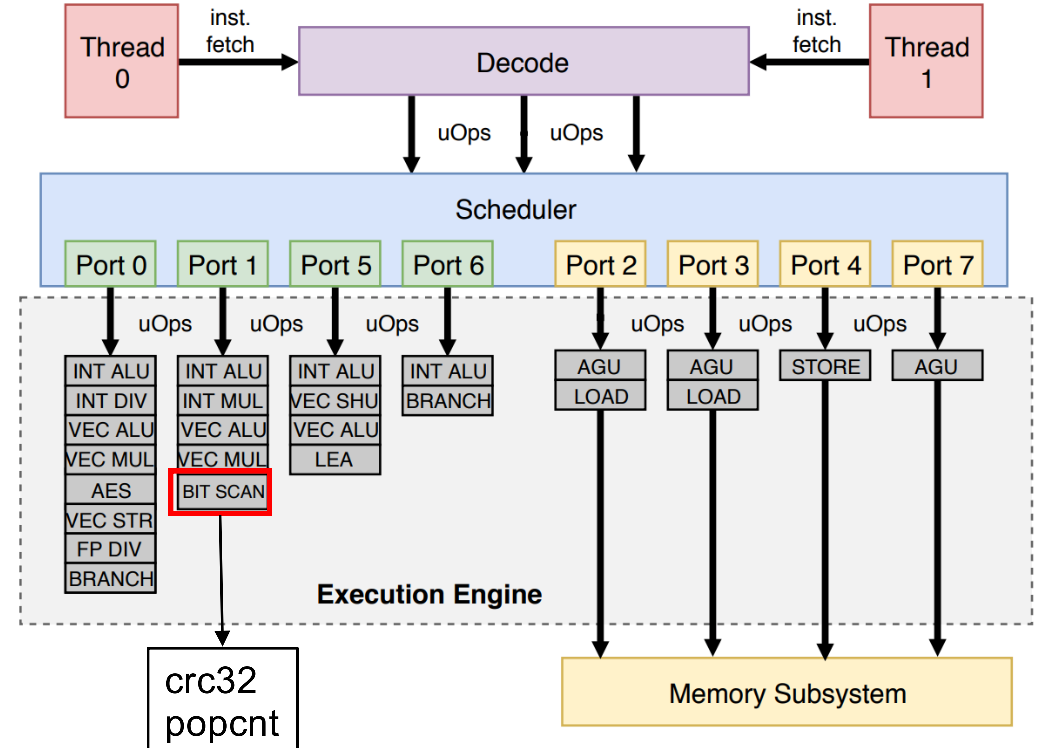
- **Transient channel** (hyper-threading / multi-core scenario):
  1. Share resource on the fly (e.g., bus, port, cache bank)
  2. or state change within speculative window (e.g., speculative buffer)
- **Permanent channel**:
  - Change the state of micro-architecture
  - The change remains even after the speculative window
  - Micro-architecture components to use:
    - D-Cache (L1, L2, L3) (Tag, replacement policy state, Coherence State, Directory), I-cache; TLB, AVX (power on/off), DRAM Rowbuffer, …
  - Encoding method:
    - Contention (e.g., cache Prime+Probe)
    - Reuse (e.g., cache Flush+Reload)

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Disclosure Primitives – Port Contention

A. Bhattacharyya, et al., “SMoTherSpectre: exploiting speculative execution through port contention”, 2019
A. C. Aldaya, et al., “Port Contention for Fun and Profit”, 2018

- Execution units and ports are shared between hyper-threads on the same core
- Port contention affect the timing of execution



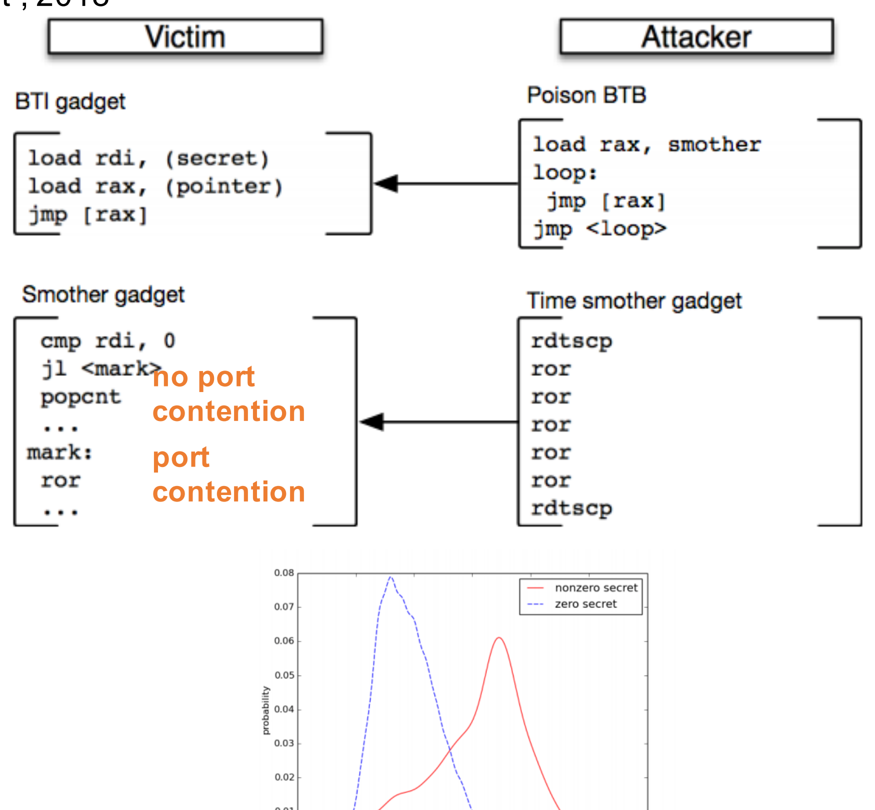


Fig. Probability density function for the timing of an attacker measuring ***crc32*** operations when running concurrently with a victim process that speculatively executes a branch which is conditional to the (secret) value of a register being zero.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Disclosure Primitives – Cache Coherence State

C. Trippel, et al., "MeltdownPrime and SpectrePrime: Automatically-Synthesized Attacks Exploiting Invalidation-Based Coherence Protocols", 2018
F. Yao, et al., "Are Coherence Protocol States Vulnerable to Information Leakage?", 2018

- The coherence protocol may invalidate cache lines in sharer cores as a result of a speculative write access request even if the operation is eventually squashed

Gadget:

```
void victim_function(size_t x) {
        if (x < array1_size) {
                array2[array1[x] * 512] = 1;
        }
}
```

-- a write on the remote core makes the cache coherence state to be exclusive on the remote core.

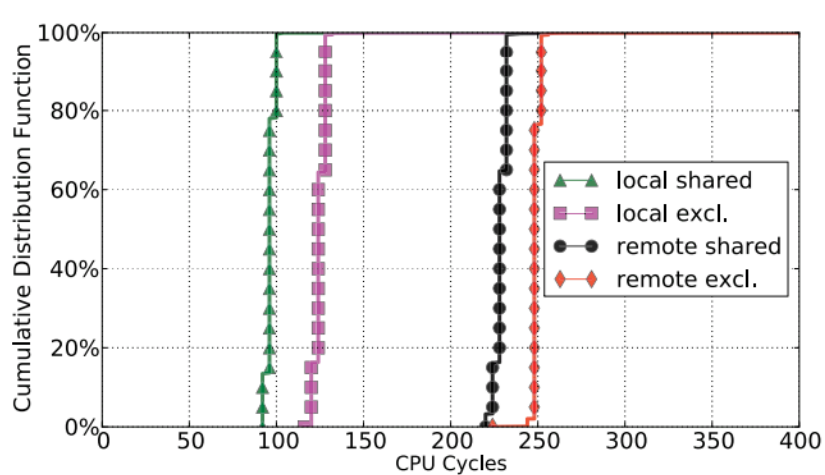


Fig. 2: Load operation latency in various (location, coherence state) combinations.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Disclosure Primitives – Directory in Non-Inclusive Cache

M. Yan, et al. "Attack Directories, Not Caches: Side-Channel Attacks in a Non-Inclusive World", S&P 2019

- Similar to the caches, the directory structure in can be used as covert channel

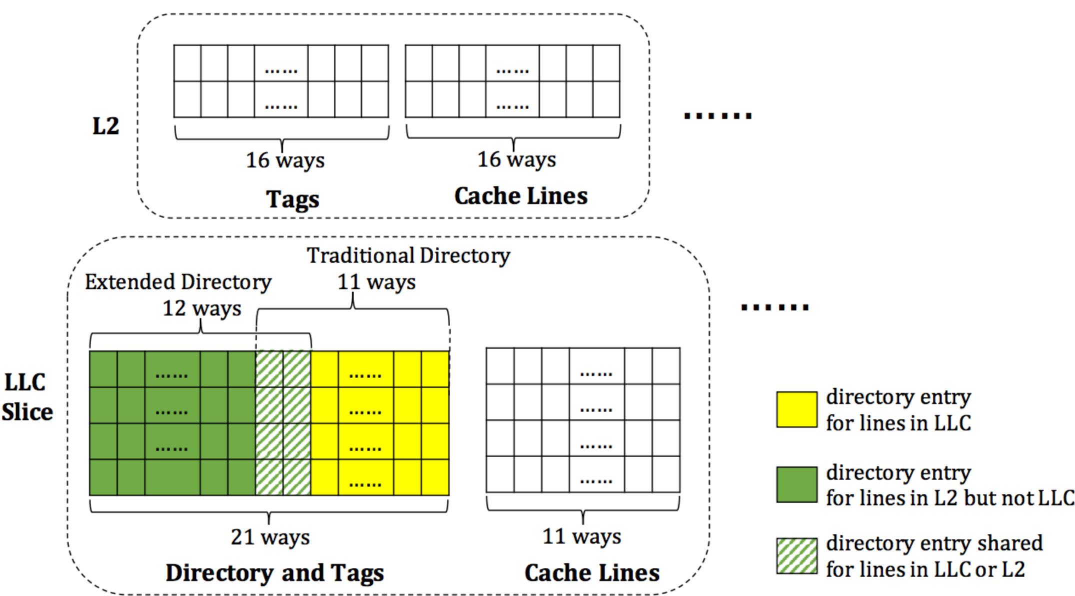


Fig. 9. Reverse engineered directory structure.

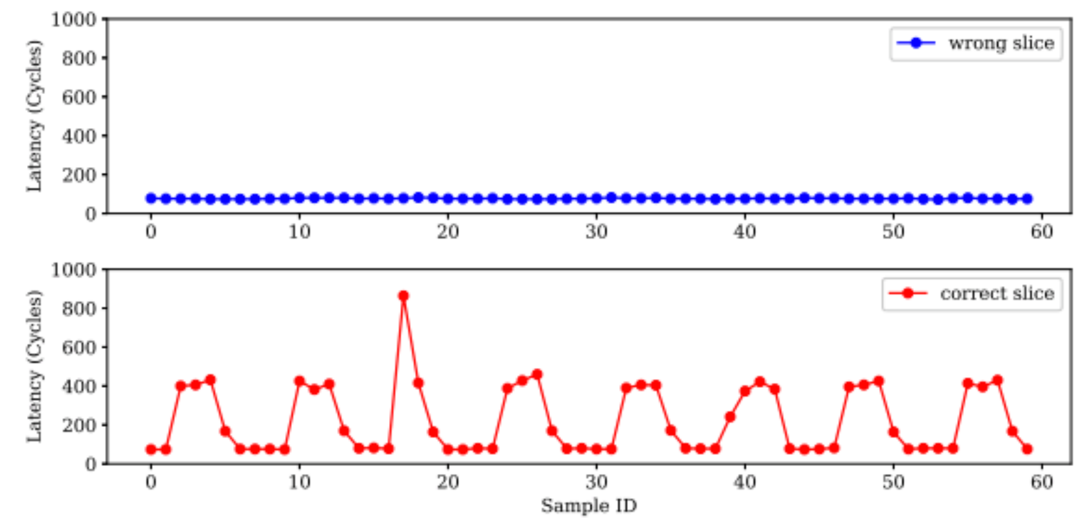


Fig. 13. The upper plot shows receiver's access latencies on a slice not being used for the covert channel, while the lower one shows the one used in the covert channel. Sender transmits sequence "101010...".

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Disclosure Primitives - AVX Unit States

M. Schwarz, et al., “NetSpectre: Read Arbitrary Memory over Network”, 2018

- To save power, the CPU can power down the upper half of the AVX2 unit which is used to perform operations on 256-bit registers
- The upper half of the unit is powered up as soon as an instruction is executed which uses 256-bit values
- If the unit is not used for more than 1 ms, it is powered down again

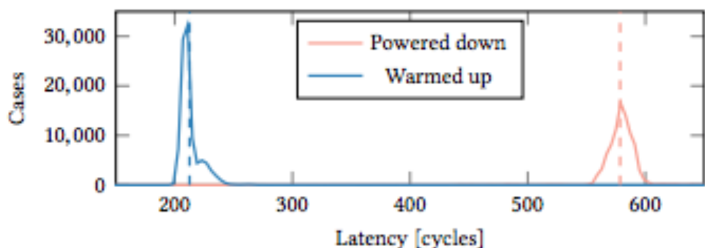


Figure 5: Differences in the execution time for AVX2 instructions (Intel i5-6200U). If the AVX2 unit is inactive (powered down), executing a 256-bit instruction takes on average 366 cycles longer than on an active AVX2 unit. The average values are shown as dashed vertical lines.

Gadget:

```
if(x < bitstream_length){
    if(bitstream[x])
        _mm256_instruction();
}
```

Disclosure Primitives – Metrics

- **Observability by the attacker**
  Time resolution of attacker’s clock. e.g., 10 cycles or 100 cycles
- **Retention time of the state**
  How long the channel will keep the secret. e.g., AVX channel, 0.5~1ms
- **Encoding time**
  Required speculation window
- **Bandwidth of the channel**
  How fast data can be transmitted
- **Cross-core or not**
  Some covert channel is only between threads in SMT settings. e.g., port contention, L1 and L2 cache

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Taxonomy of Transient Execution Attacks

- Each entry in the table could be an attack
- Defender needs to cover the whole table

		Covert Channel															
		Ports	L1 (tag)		L2 (tag)		L3 (tag)		DTLB (tag)		STLB (tag)		Cache coherence	Directory		AVX	…
			F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	P+P	F+R	P+P		…
Cause of Transient Execution	Conditional branch	**SMoTher Spectre**	**V1**										**Spectre prime**			**Net Spectre**	
	Indirect branch BTB		**V2**														
	Return branch RSB		**V5**														
	Mem-disambiguation		**V4**														
	Exception		**V3; L1TF; etc.**														

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Taxonomy of Transient Execution Attacks

Mitigations in Micro-Architecture

Secure Architectures and Transient Execution Attacks

Timing Side Channels which use Speculation

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Mitigations in Micro-architecture: Principle

Transient Execution Attacks = Transient Execution + Covert or Side Channel

- No transient execution (not practical)
  - No load of secrets in transient execution

    What is secret? E.g., Meltdown-type attacks.
  - Limit attackers ability to influence prediction unit (PHT, BTB, RSB)
  - Transient execution should not result in observable changes
    - State changes (Permanent channels)
      - Cache InvisiSpec [M. Yan, et al., 2018]; SafeSpec [K. N. Khasawneh, et al., 2018]
    - Resource contentions (Transient channels)

- Mitigate covert channels (across security domains)
  - Cache covert channel DAWG [V. Kiriansky, et al., 2018]
  - …

Mitigations in Micro-architecture: InvisiSpec

M. Yan, et al., "InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy", 2018

- Focus on transient loads in disclosure gadgets
- Unsafe speculative load (USL)
  - The load is speculative and may be squashed
  - Which should not cause any micro-architecture state changes visible to the attackers
  - Speculative Buffer: a USL loads data into the speculative buffer (for performance), not into the local cache
- Visibility point of a load
  - After which the load can cause micro-architecture state changes visible to attackers
- Validation or Exposure:
  - Validation: the data in the speculative buffer might not be the latest, a validation is required to maintain memory consistency.
  - Exposure: some loads will not violate the memory consistency.
- Limitations: only for covert channels in caches

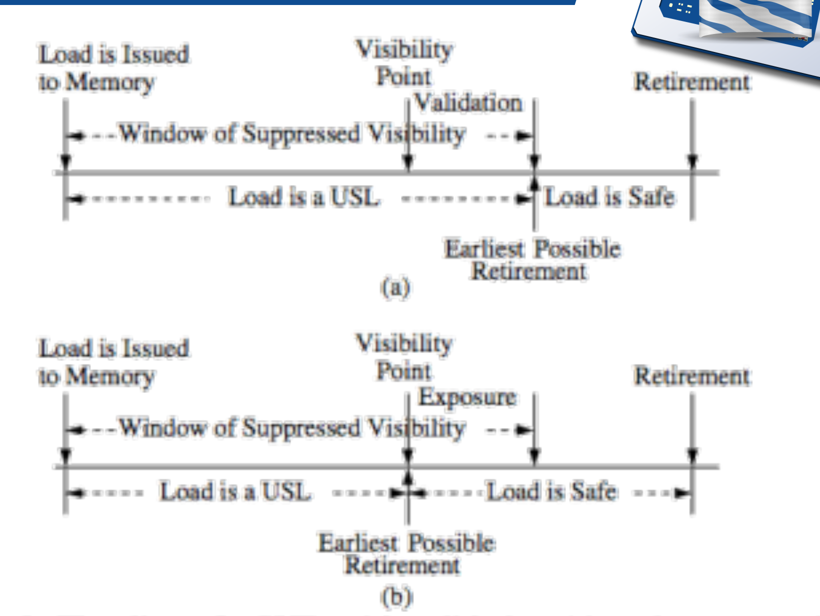
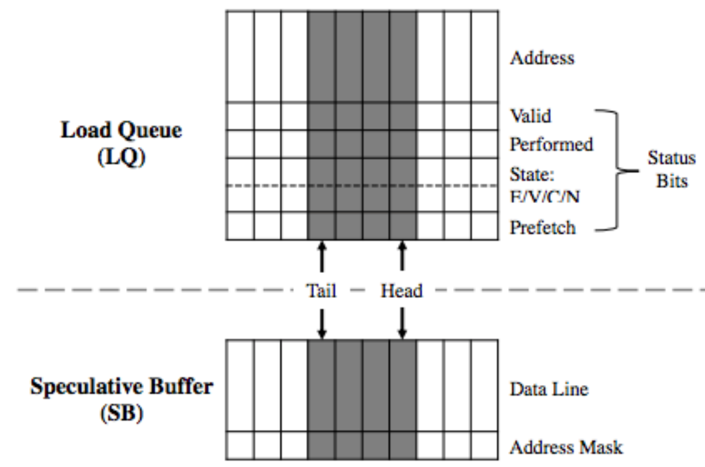


Fig. 2: Timeline of a USL with validation (a) and exposure (b).



Taxonomy of Transient Execution Attacks

- InvisiSpec can defend transient execution attacks using cache covert channels

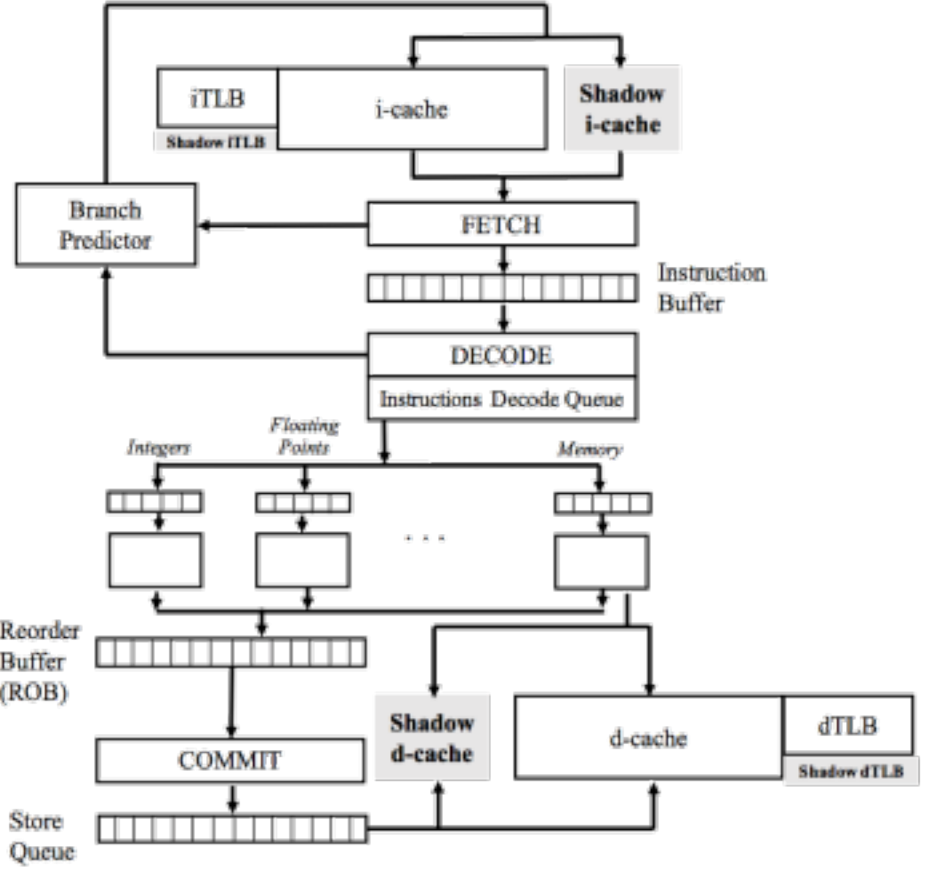
Cause of Transient Execution		Covert Channel															
		Ports	L1 (tag)		L2 (tag)		L3 (tag)		DTLB (tag)		STLB (tag)		Cache coherence	Directory		AVX	…
			F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	P+P	F+R	P+P		…
	Conditional branch	SMoTher Spectre	V1										Spectre prime			Net Spectre	
	Indirect branch BTB		V2														
	Return branch RSB		V5														
	Mem-disambiguation		V4														
	Exception		V3; L1TF; etc.														

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Mitigations in Micro-architecture: SafeSpec

K. N. Khasawneh, et al., "SafeSpec: Banishing the Spectre of a Meltdown with Leakage-Free Speculation", 2018

- Similar to Invisispec, shadow caches and TLBs are proposed to store the micro-architecture changes by speculative loads temporarily



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Taxonomy of Transient Execution Attacks

- SafeSpec can defend transient execution attacks using cache and TLB covert channels

		Covert Channel															
		Ports	L1 (tag)		L2 (tag)		L3 (tag)		DTLB (tag)		STLB (tag)		Cache coherence	Directory		AVX	…
			F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	P+P	F+R	P+P		…
Cause of Transient Execution	Conditional branch	SMoTher Spectre	V1										Spectre prime			Net Spectre	
	Indirect branch BTB		V2														
	Return branch RSB		V5														
	Mem-disambiguation		V4														
	Exception		V3; L1TF; etc.														

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Mitigations in Micro-architecture: DAWG

V. Kiriansky, et al., “DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors”, 2018

- Mitigate the cache covert channels by partitioning the cache
- Dynamically Allocated Way Guard (DAWG) fully isolates hits, misses, and metadata updates across protection domains

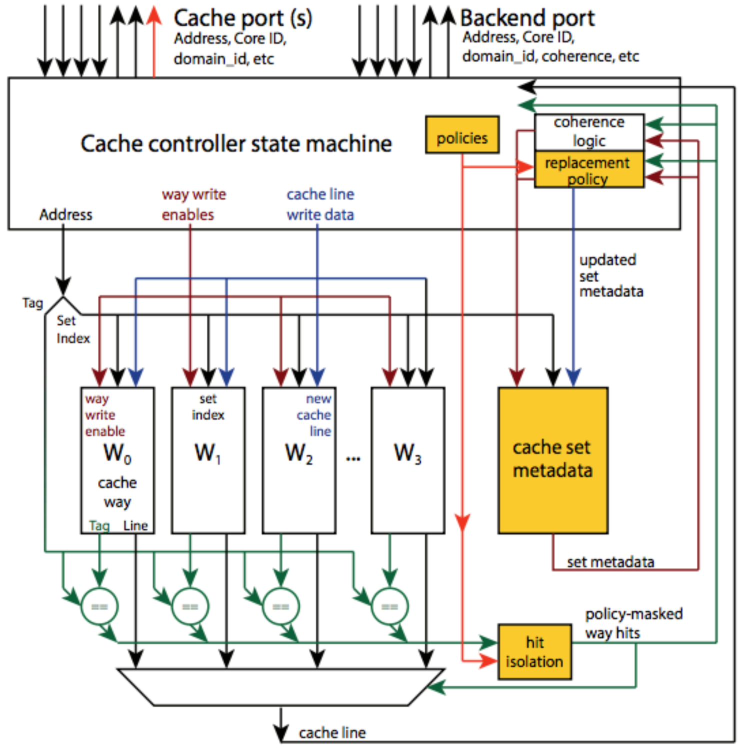


Fig. 4. A Set-Associative Cache structure with DAWG.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Taxonomy of Transient Execution Attacks

- DAWG can defend transient execution attacks using cache covert channels

		Covert Channel															
		Ports	L1 (tag)		L2 (tag)		L3 (tag)		DTLB (tag)		STLB (tag)		Cache coherence	Directory		AVX	...
			F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	F+R	P+P	P+P	F+R	P+P		...
Cause of Transient Execution	Conditional branch	SMoTher Spectre	V1										Spectre prime			Net Spectre	
	Indirect branch BTB		V2														
	Return branch RSB		V5														
	Mem-disambiguation		V4														
	Exception		V3; L1TF; etc.														

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Mitigations in Micro-architecture: Performance

- Performance overhead of hardware mitigations of Spectre Attacks at micro-architecture level

	Performance Loss	Benchmark
Fence after each branch (software)	88%	SPEC2006
InvisiSpec [M. Yan, et al., 2018]	22%	SPEC2006
SafeSpec [K. N. Khasawneh, et al., 2018]	3% improvement (due to larger effective cache size)	SPEC2017
DAWG [V. Kiriansky, et al., 2018]	1-15%	PARSEC, GAPBS

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Taxonomy of Transient Execution Attacks

Mitigations in Micro-Architecture

Secure Architectures and Transient Execution Attacks

Timing Side Channels which use Speculation

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Transient Execution Attacks on SGX: SgxPectre

G. Chen, et al., “SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution”, 2018

- Spectre can attack current secure architectures!
- E.g., Spectre v2 on SGX

1. Poison BTB (Speculation Primitive)
2. Flush the victim’s branch target address and deplete the RSB (Windowing Gadget)
3. Set secret address and probe array address
4. Execute victim code (Disclosure Gadget)
5. Obtain secret from covert channel (Disclosure Primitive)

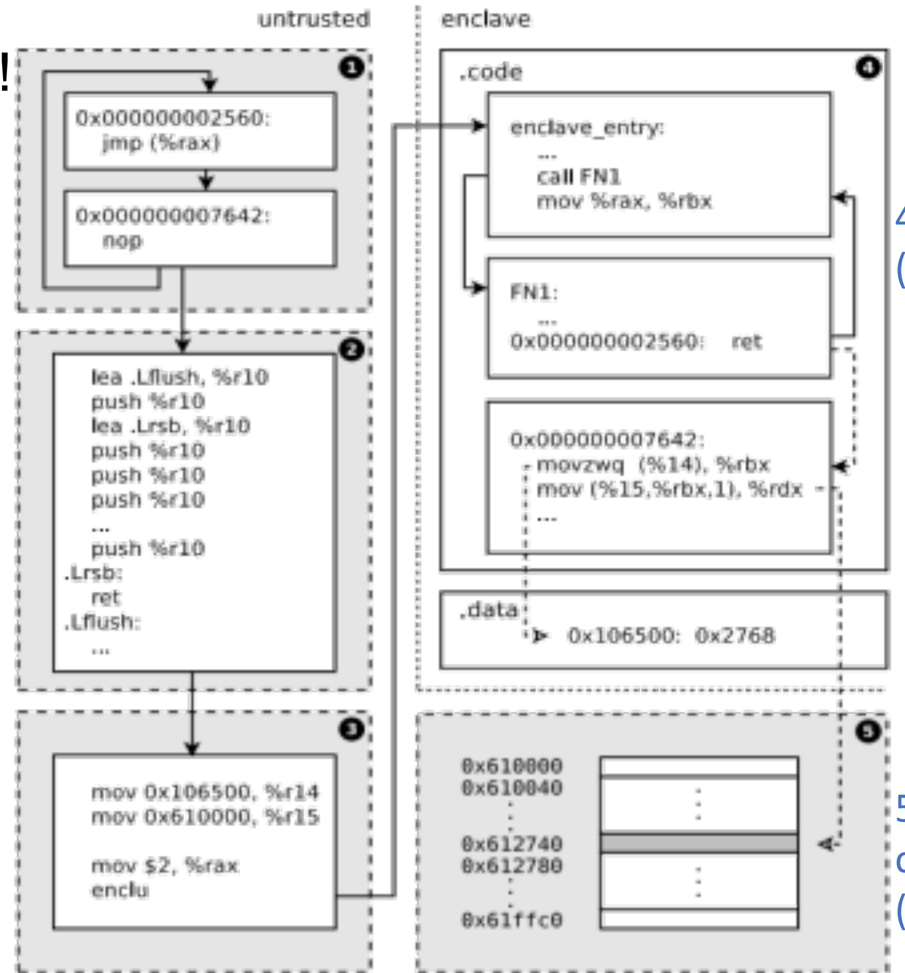


Figure 1: A simple example of SgxPectre Attacks. The gray blocks represent code or data outside the enclave. The white blocks represent enclave code or data.

Transient Execution Attacks on SGX: Foreshadow

J. Van Bulck, et al., “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”, 2018

- Meltdown-type attack can attack current secure architectures!

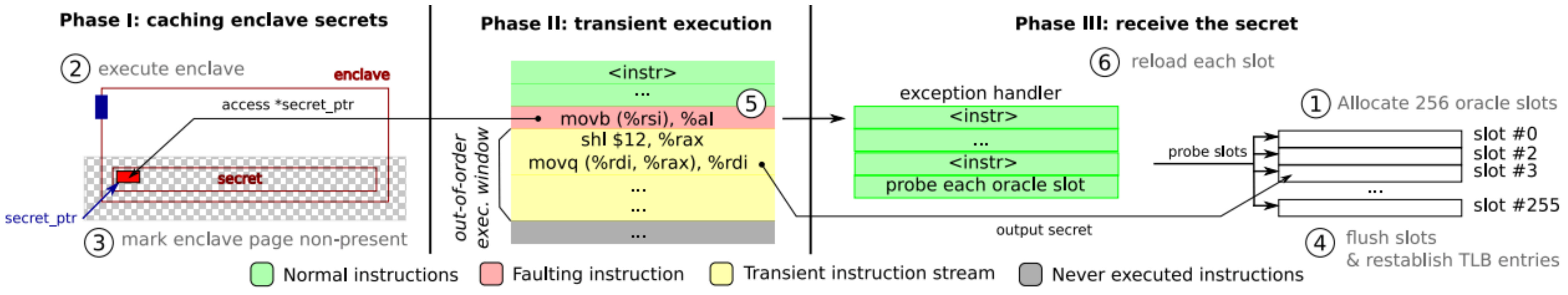


Figure 2: Basic overview of the Foreshadow attack to extract a single byte from an SGX enclave.

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Mitigating Transient Execution Attacks

Transient execution attacks need to be mitigated in secure processors

Transient Execution Attacks = Transient Execution + Covert Channel

- No transient execution
    - No load of secrets in transient execution

      What is secret? Data in secure domains in secure architectures.
    - Transient execution should not result in observable changes
        - State changes (Permanent channels) Flush states on interrupts, and other exits.
            - Cache
        - Resource contentions (Transient channels) Disable SMT during secure execution.

- Mitigate covert channels (across security domains)
    - Cache covert channel: partitioning cache; randomizing cache
    - …

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Taxonomy of Transient Execution Attacks

Mitigations in Micro-architecture

Secure Architectures and Transient Execution Attacks

Timing Side Channels which use Speculation

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Timing Side Channels Using Speculation

Transient Execution Attacks = Transient Execution + Covert Channel

- Cache (tag)
- Cache coherence
- TLB
- AVX
- **Prediction units**
- …

- This section demonstrates how the prediction units can be leveraged to built a covert channel

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Timing Side Channels Using Speculation

- Modern computer architectures gain performance by making predictions:
  - Success prediction -> fast execution
  - Mis-prediction -> slow execution
- Mis-predictions are also counted by performance counters

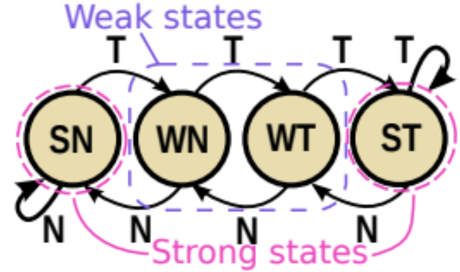
- The prediction units (e.g., Branch prediction, Memory Disambiguation prediction) usually make prediction based on some history stored
- The prediction units are often shared between threads running on the same core
- Victim's execution history can affect the prediction of the attacker thread, and the attacker observe the timing difference

- **This type of side channels are different from the transient executions attacks!**

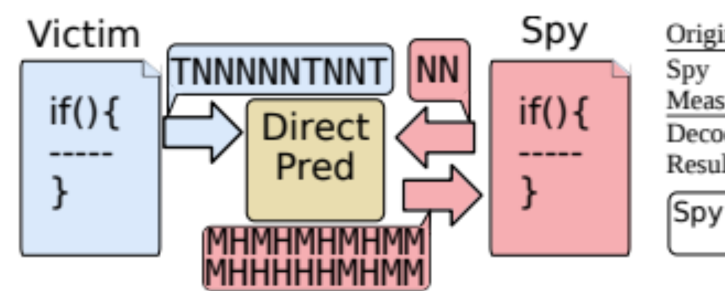
Pattern History Table (PHT) : BranchScope

D. Evtyushkin, et al., “BranchScope: A New Side-Channel Attack on Directional Branch Predictor”, 2018
D. Evtyushkin, et al., “Covert Channels Through Branch Predictors: A Feasibility Study”, 2015

- PHT is shared among all processes on core, and is not flushed on context switches
- The branch predictor stores its history in the form of a 2-bit saturating counter in a pattern history table (PHT)



- The PHT entry used is a simple function of the branch address
- Prime+Probe Strategy
- Attacks:
  - Covert channels
  - Attack SGX enclave code




```
int sec_data[]
= {1,0,1,1,..};
i = 0;
void victim_f(){
 //Victim Branch
 if(sec_data[i])
  asm("nop;nop");
 i++;
}
```

Code 1. Pseudo-code of the victim.

```
int probe_array [2] = {1, 1};//Not-taken
int main(){
  for(int i = 0; i < N_BITS; i++){
    randomize_pht();//(1)
    usleep(SLEEP_TIME);//Wait for victim
    spy_function(probe_arr); } }
void spy_function(int array [2]){
  for(int i = 0; i < 2; i++){
    a = read_branch_mispred_counter();
    if(array[i])// <- Spy branch
      asm("nop; nop; nop;");
    b = read_branch_mispred_counter();
    store_branch_mispred_data(b - a); } }
```

Code 2. Pseudo-code of the attacker.

Branch Target Buffer (BTB): Jump Over ASLR

D. Evtyushkin, et al., “Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR”, 2016

- The BTB stores target addresses of recently executed branch instructions, so that those addresses can be obtained directly from a BTB lookup

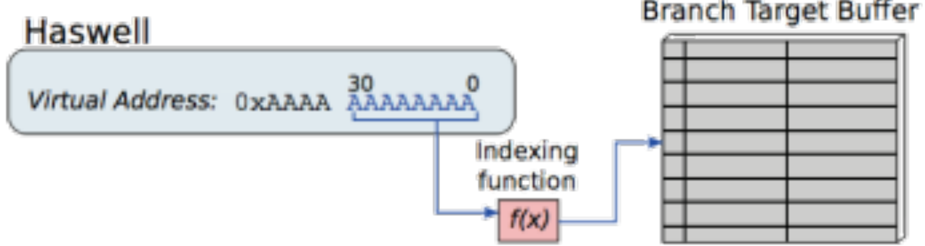


Fig. 4: BTB addressing scheme in Haswell processor

- Same-Domain Collisions (SDC)
- BTB collisions between two processes executing in the same protection domain

- Attacks:

  attack KASLR (Kernel address space layout randomization)

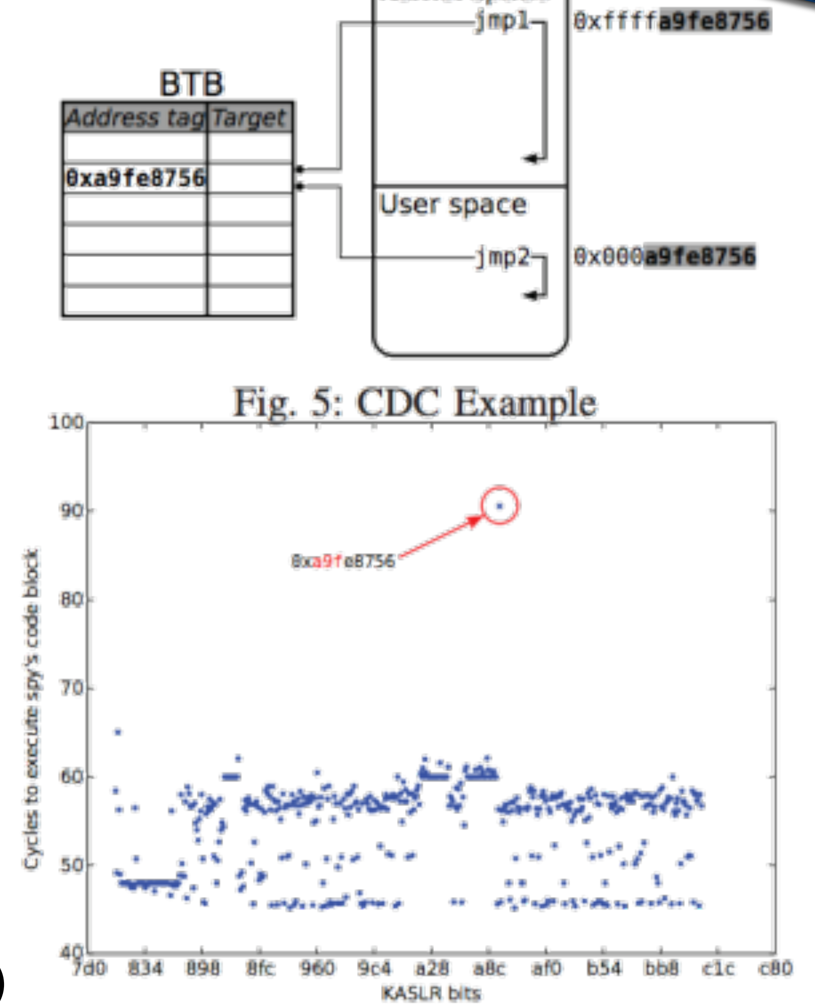


Fig. 5: CDC Example

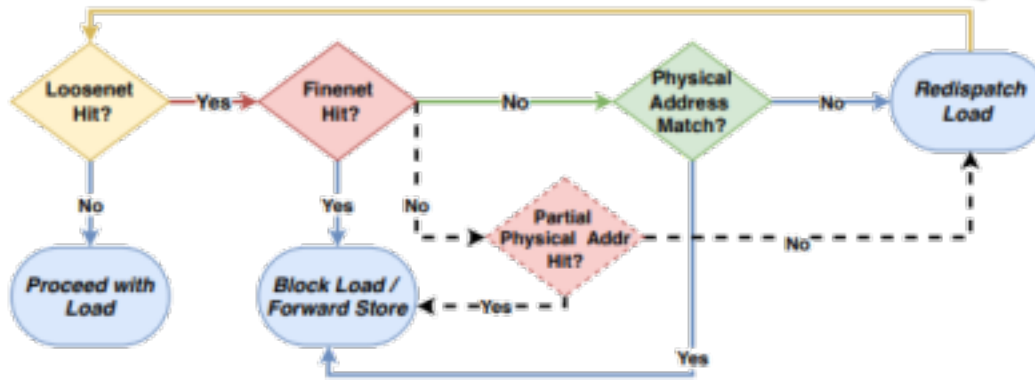
Fig. 7: Results of the BTB-based Attack on KASLR

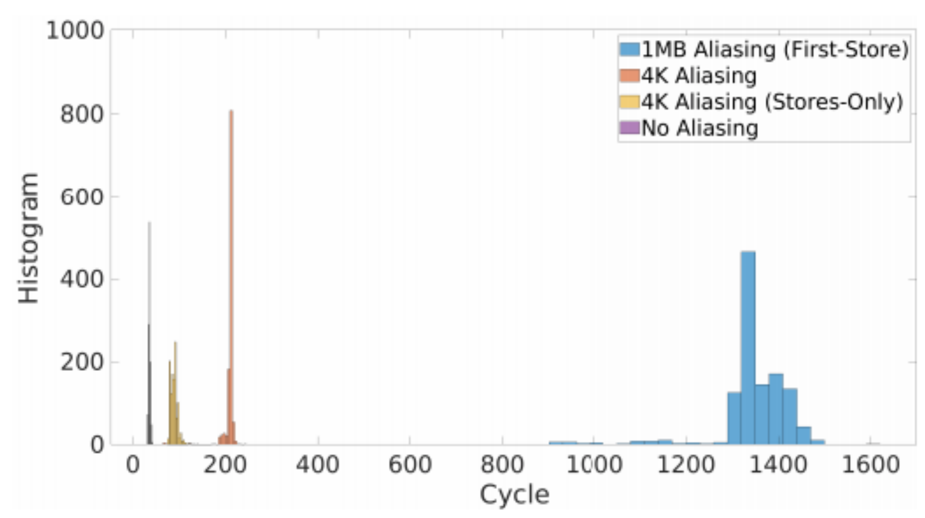
© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Memory Disambiguation: Spoiler Attack

S. Islam, et al., "SPOILER: Speculative Load Hazards Boost Rowhammer and Cache Attacks", 2019

- The processor executes the load speculatively before the stores, and forwards the data of a preceding store to the load if there is a potential dependency
- The finenet check may be implemented based on checking the partial physical address bits
- 1MB aliasing in Intel processors
- Attacks: Leakage of the Physical Address Mapping
  - Efficient eviction set finding for Prime+Probe attacks in LLC
  - Helps to conduct DRAM row conflicts





© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Open-Source Attack PoC Code

- Number of proof-of-concept (PoC) codes are available for the attacks, but they do not cover all possible attacks
- Please contact jakub.szefer@yale.edu to report more proof-of-concept or sample codes available for testing

PoC codes:

- Spectre v1: https://spectreattack.com/spectre.pdf
  https://github.com/crozone/SpectrePoC
- Spectre v4: https://bugs.chromium.org/p/project-zero/issues/detail?id=1528
- Meltdown: https://github.com/IAIK/meltdown
- Foreshadow: https://foreshadowattack.eu/
- For Windows: https://github.com/msmania/microarchitectural-attack
- Spectre (v1, v2) on RISC-V: https://github.com/riscv-boom/boom-attacks

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Tutorial Outline & Schedule

15:30 – 16:10	Secure Processor Architectures
16:10 – 16:20	Break
16:20 – 17:10	Secure Processor Caches
17:10 – 17:20	Break
17:20 – 18:00	Transient Execution Attacks and Mitigations
18:00	**Wrap Up**

Slides and information at:
http://caslab.csl.yale.edu/tutorials/host2019/

WiFi Information:
Network: Hilton-Meeting, Password: HOST2019

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

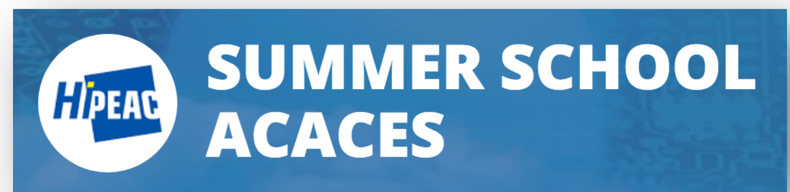
Summer Course on Processor Architecture Security

Who: Jakub Szefer

What: Summer Course on **Processor Architecture Security**

Where: at the 15th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES), in Rome, Italy

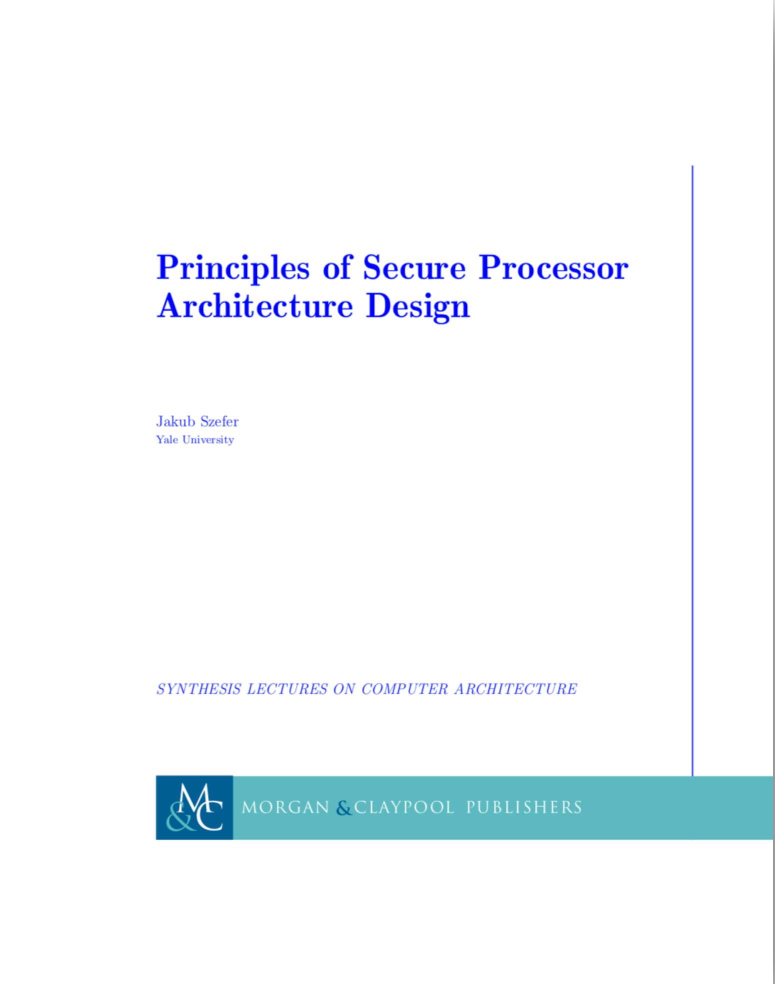
When: Sunday evening July 14th, 2019 until Friday evening July 19th, 2019



Secure Processor Architectures Book

Jakub Szefer, "Principles of Secure Processor Architecture Design," in Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, October 2018.

http://caslab.csl.yale.edu/books/



© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)

Thank You!

© Jakub Szefer, Wenjie Xiong, and Shuwen Deng (rev. 2019-05-05)