



Microarchitectural Side Channels

BJÖRN RUYTENBERG
EINDHOVEN UNIVERSITY OF TECHNOLOGY

SPECIAL THANKS TO
YUVAL YAROM, THE UNIVERSITY OF ADELAIDE AND DATA61
FOR PROVIDING SUPPORT AND SELECTED CONTENT

Roadmap

Introduction to Side Channels

Microarchitectural Basics

What is Meltdown?

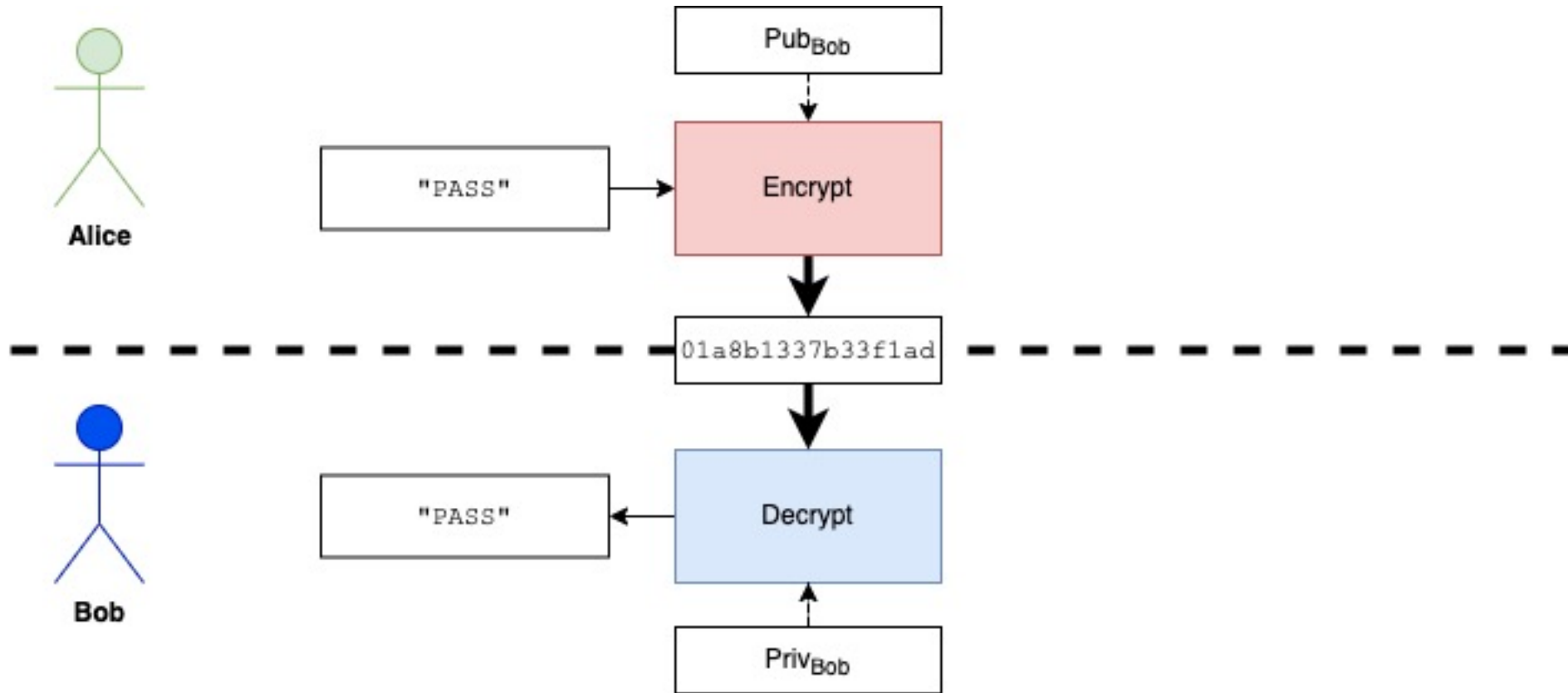
What is Spectre?

Exploitation Scenarios

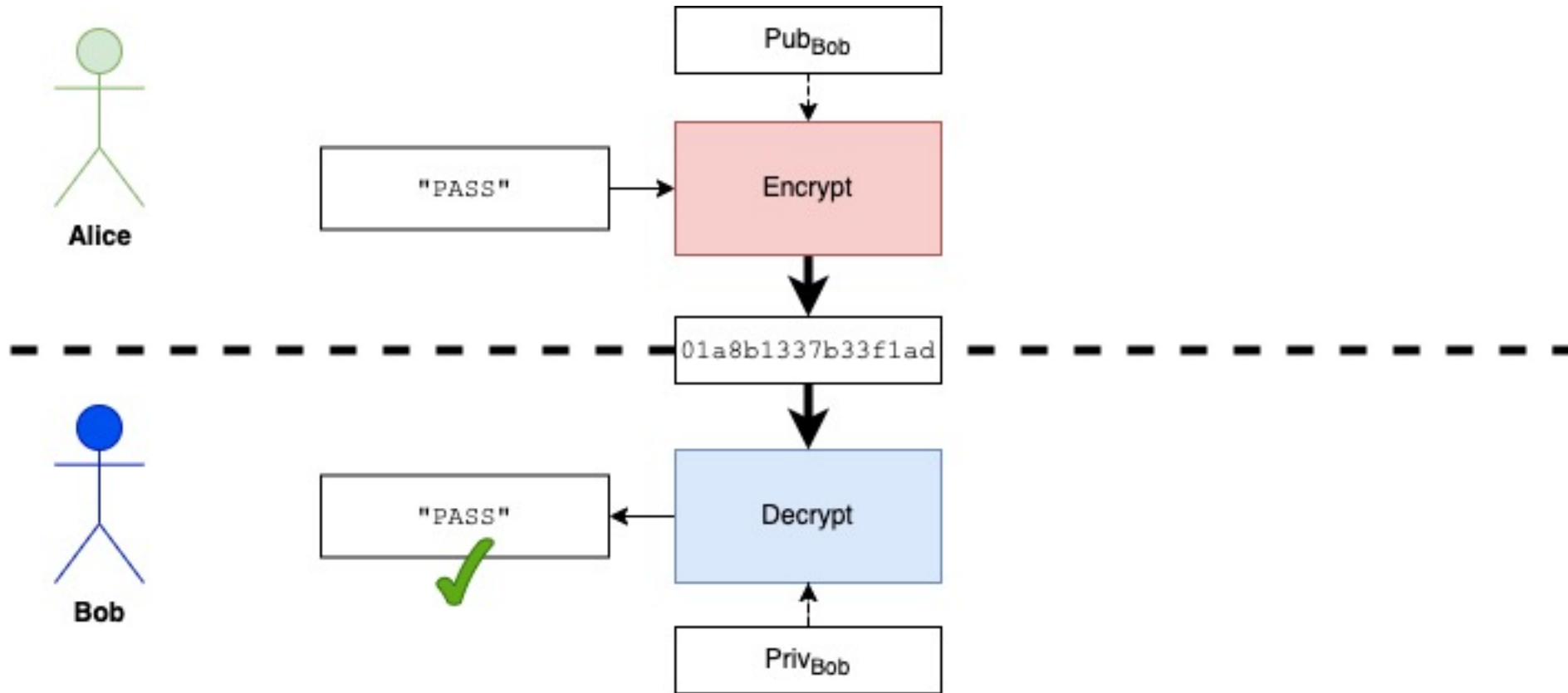
Mitigations

Attack Variants

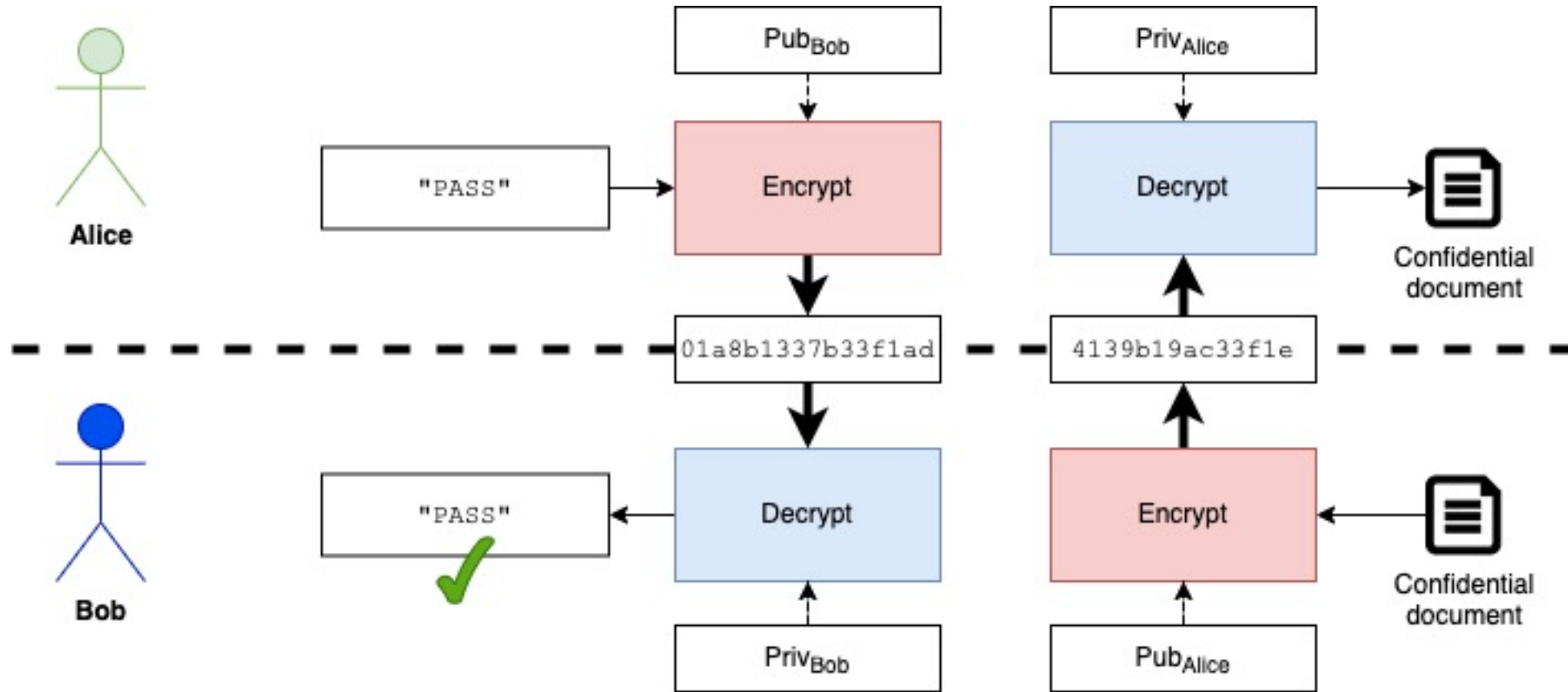
What is a Side Channel?



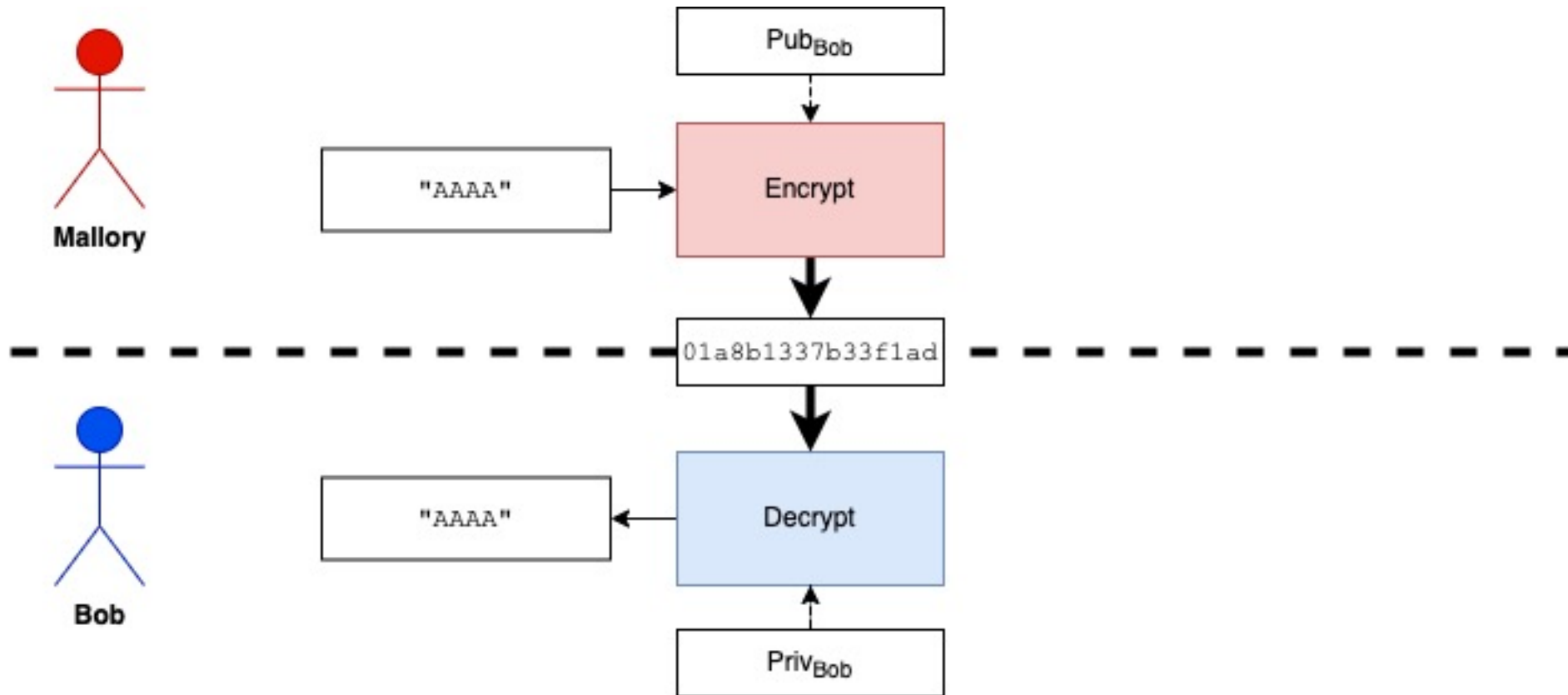
What is a Side Channel?



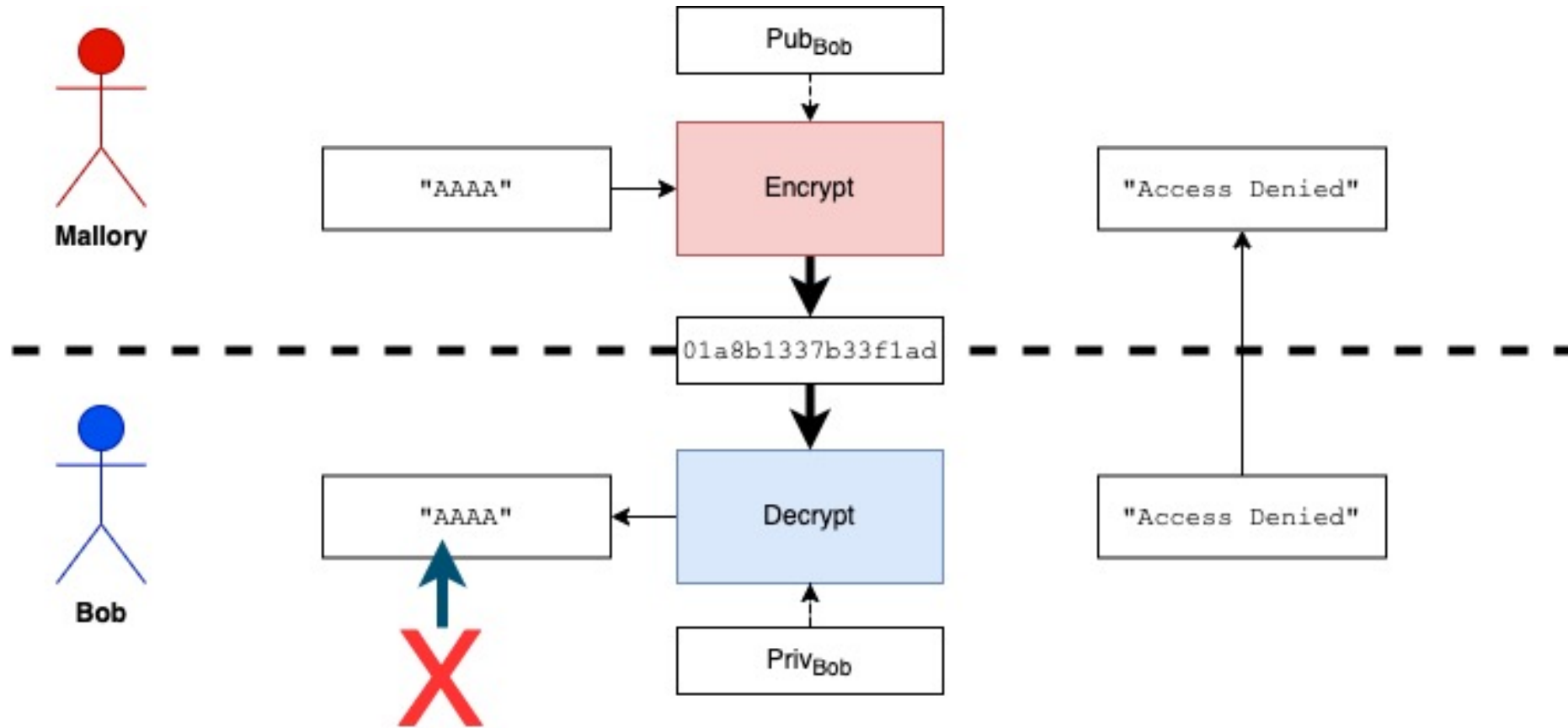
What is a Side Channel?



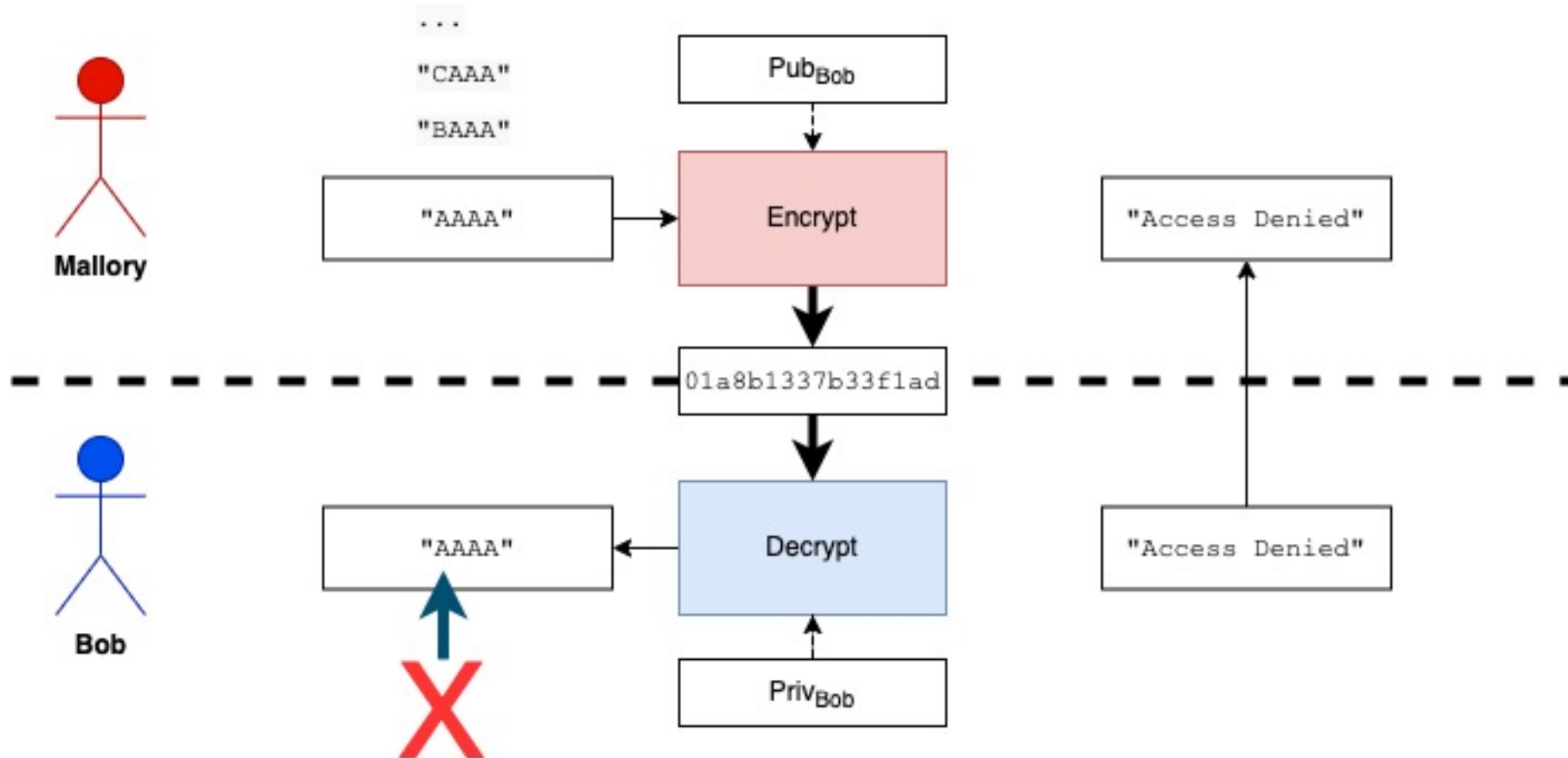
What is a Side Channel?



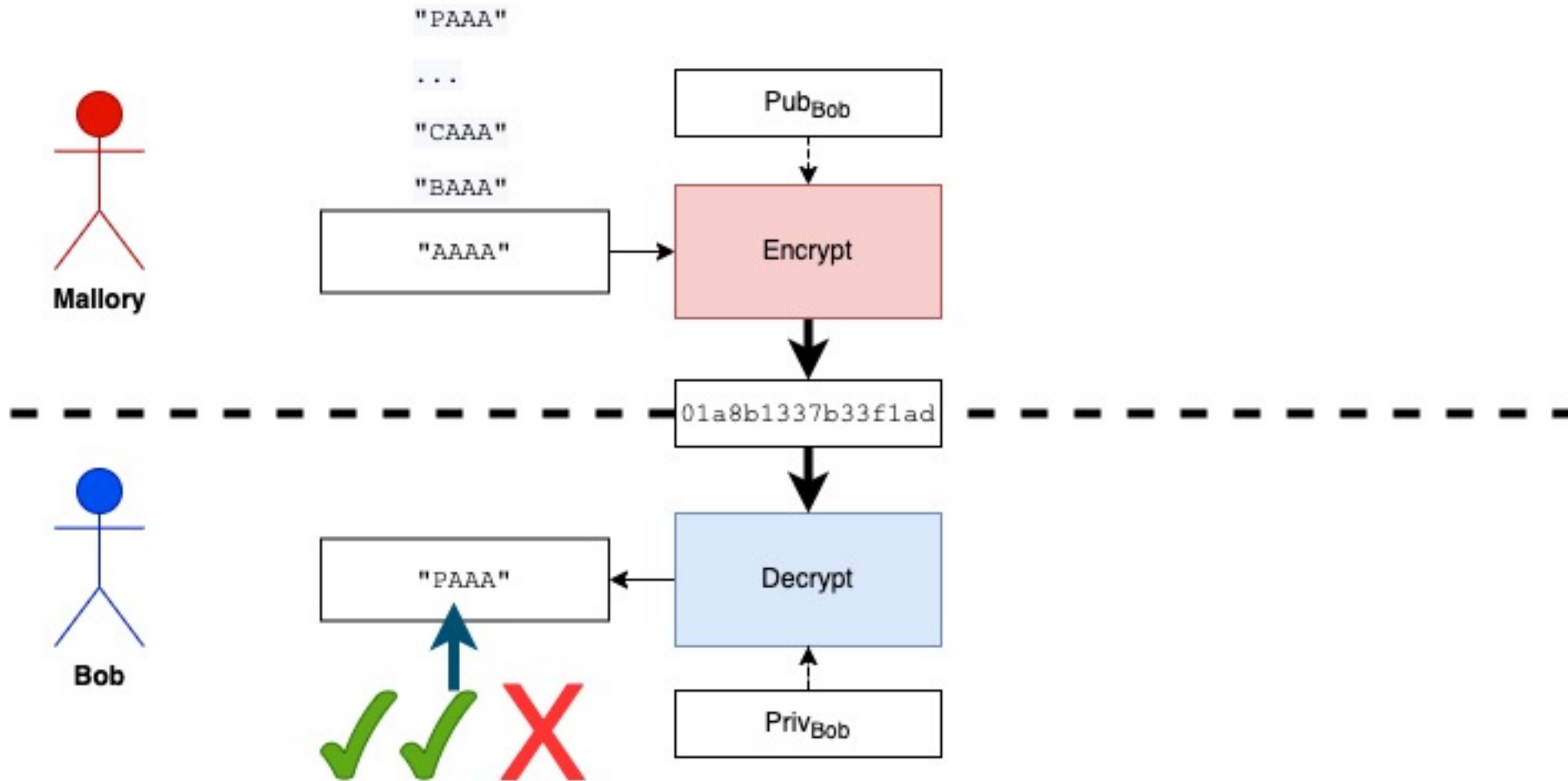
What is a Side Channel?



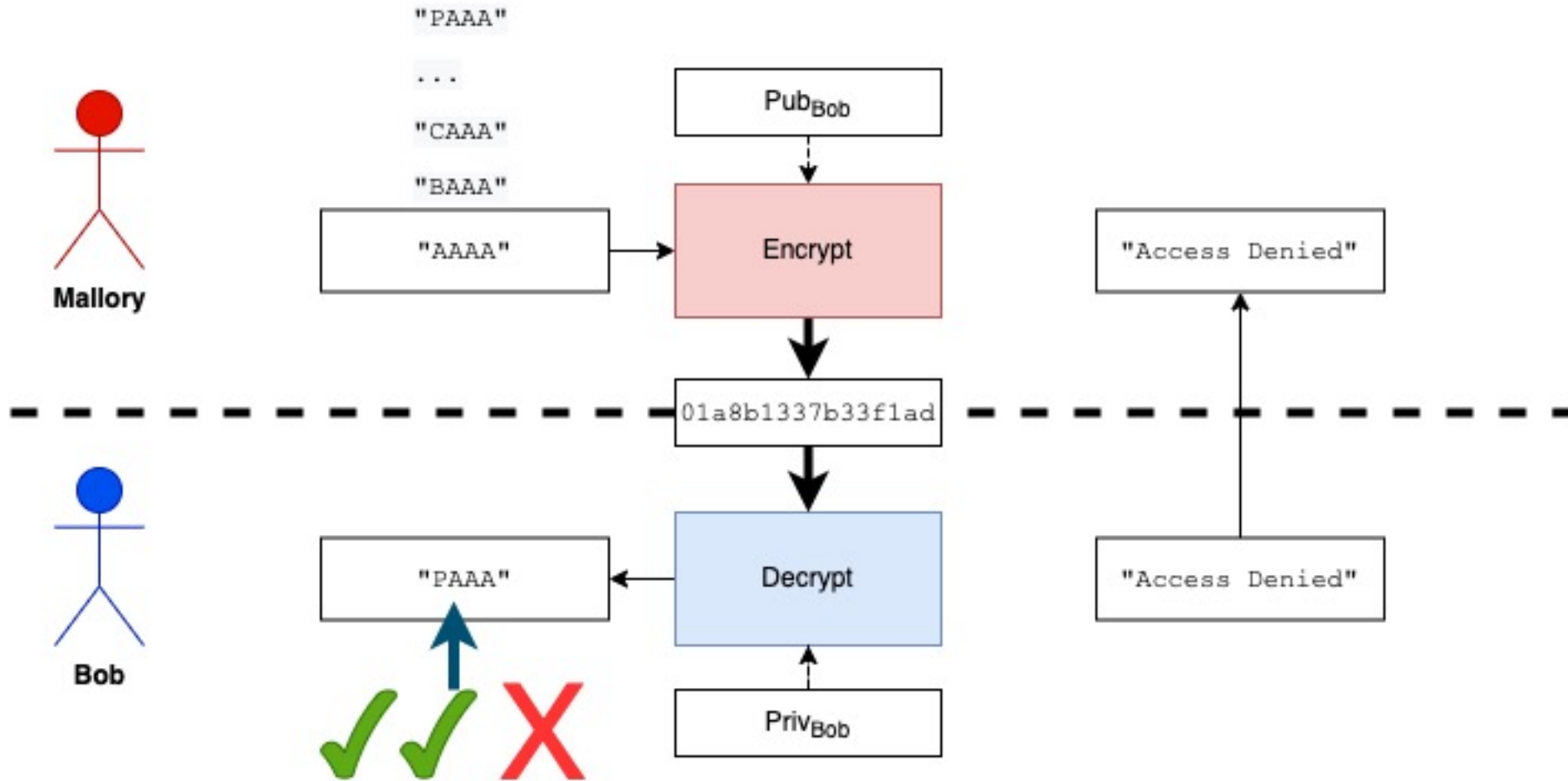
What is a Side Channel?



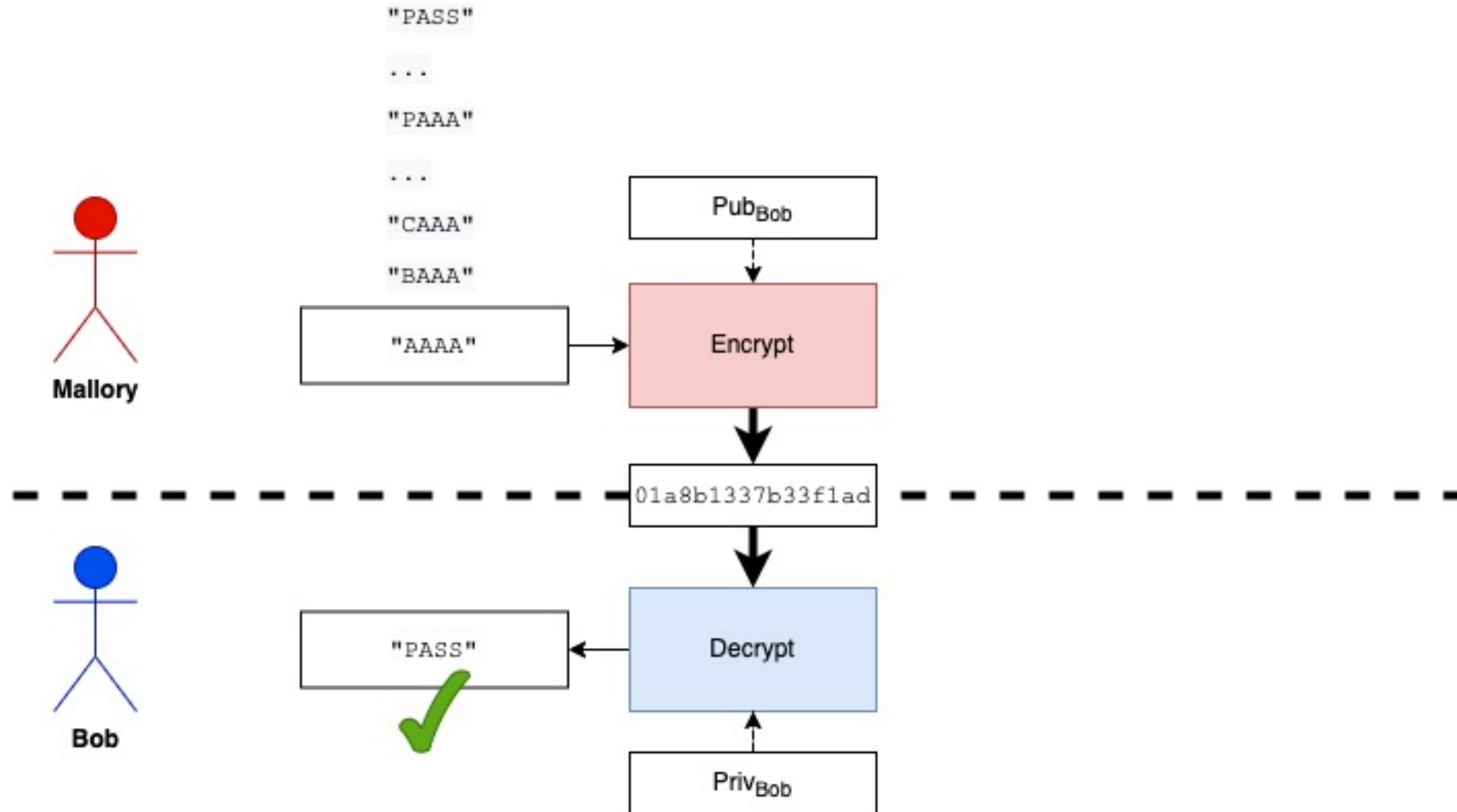
What is a Side Channel?



What is a Side Channel?



What is a Side Channel?



What is a Side Channel?

- Unintentional, usually covert communication channel leaking potentially sensitive information

Various types

- Power analysis
- EM radiation
- Sound/light
- Timing



Microarchitectural Side Channels

Part 1: Meltdown



Meltdown – Basic Outline

- Design flaw that affects most modern Intel CPUs (and some ARMs)
- Uses out-of-order execution to leak data through cache timing attack

- From an unprivileged process, an attacker can:
 - Bypass language-based security
 - Bypass sandboxes, containers/paravirtualization hypervisors
 - Read arbitrary memory, including kernel memory



Caches

- Fast processor but slower memory
- Cache utilizes locality to bridge the gap
 - Divides memory into *lines*
 - Stores recently used lines

Processor



Cache



Memory



Instruction Pipelining

- Nominally, the processor executes instructions one after the other
- Instruction execution consists of multiple steps
 - Each uses a different unit

Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
-------------------	--------------------	----------------	---------	------------

```

mulq $m0
add %rax,$A[0]
mov
8*2($np),%rax
lea 32($tp),$tp
adc \0,%rdx
mov %rdx,$A[1]
mulq $m1
add %rax,$N[0]
mov
8($a,$j),%rax
adc \0,%rdx
add $A[0],$N[0]
adc \0,%rdx
mov $N[0],-
24($tp)
mov %rdx,$N[1]
mulq $m0
add %rax,$A[1]
mov
8*1($np),%rax
adc \0,%rdx
mov %rdx,$A[0]
mulq $m1
add %rax,$N[1]
mov ($a,$j),%rax
mov
8($a,$j),%rax
adc \0,%rdx

```




Instruction Pipelining

- Nominally, the processor executes instructions one after the other
- Instruction execution consists of multiple steps
 - Each uses a different unit
- Pipelining increases utilization by executing steps of multiple instructions

Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back

$c = a / b;$

$d = c + 5;$

```

mulq $m0
add %rax,$A[0]
mov
8*2($np),%rax
lea 32($tp),$tp
adc \0,%rdx
mov %rdx,$A[1]
mulq $m1add
%rax,$N[0]
mov
8($a,$j),%rax
adc \0,%rdx
add $A[0],$N[0]
adc \0,%rdx
mov $N[0],-
24($tp)
mov %rdx,$N[1]
mulq $m0
add %rax,$A[1]
mov
8*1($np),%rax
adc \0,%rdx
mov %rdx,$A[0]
mulq $m1
add %rax,$N[1]


```

How to deal with dependencies?



Out-of-Order Execution (1)

- Execute instructions when data is available rather than by program order

IF	ID	AF	EX	WB
IF	ID		EX	WB
IF	ID	AF	EX	WB

$c = a / b;$

$d = c + 5;$


$e = f + g;$

- Completed instructions wait in the **reorder buffer** until all previous instructions are **retired**
- Why not retire immediately?



Out-of-Order Execution (2)

- Execute instructions when data is available rather than by program order

IF	ID	AF	EX	WB
IF	ID		EX	WB
IF	ID	AF	EX	WB

$c = a / b;$

$d = c + 5;$

$e = f + g;$

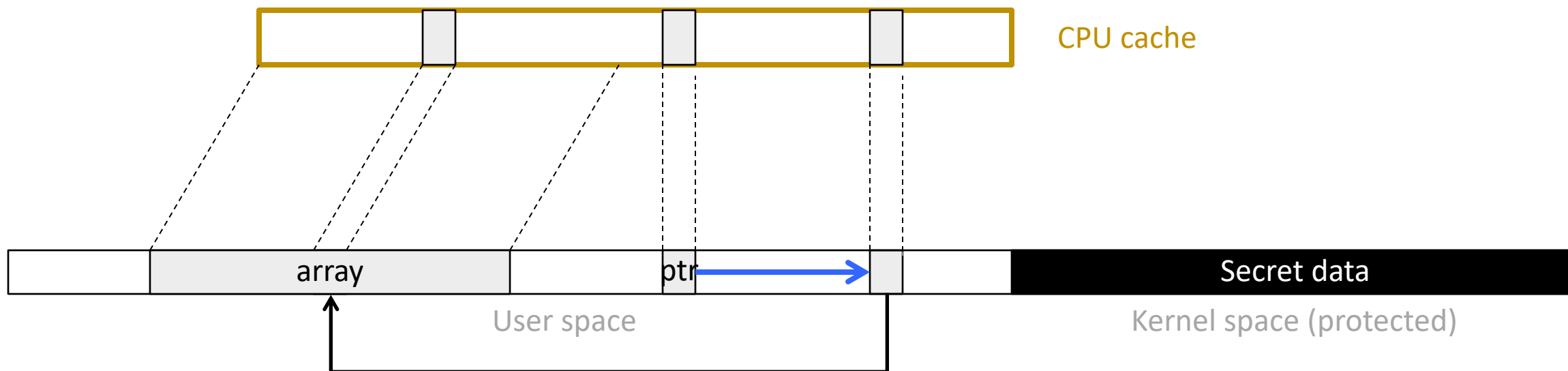
What if $b=0$?

- Completed instructions wait in the **reorder buffer** until all previous instructions are **retired**
- Why not retire immediately?
- Out-of-order execution is speculative!**
- Need to **abandon** instructions in the reorder buffer if never executed



Program Flow – Legitimate Behavior

```
i = *pointer;  
y = array[i * 256];
```





Attack Flow (1)

```

i = *pointer;
y = array[i * 256];

```

Step 1:

Set pointer to kernel space

Step 2:

Due to out-of-order processing, CPU fetches secret value from kernel space

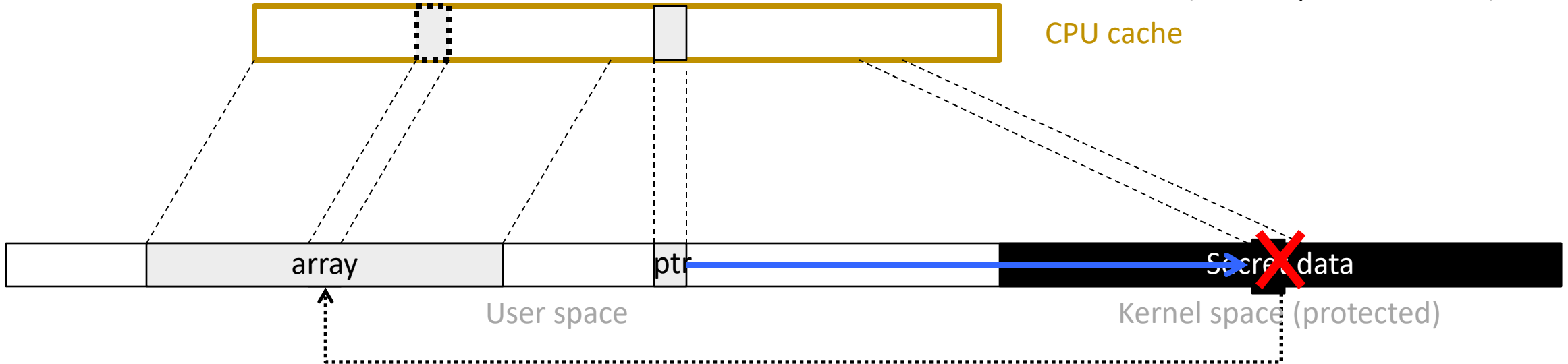
Step 3:

Secret value is used to index user space array

Exception triggered:

Results of out-of-order instructions discarded (*i* takes previous value)

CPU cache





Attack Flow (2)



```
i = *pointer;
y = array[i * 256];
```

Fast (cache hit)

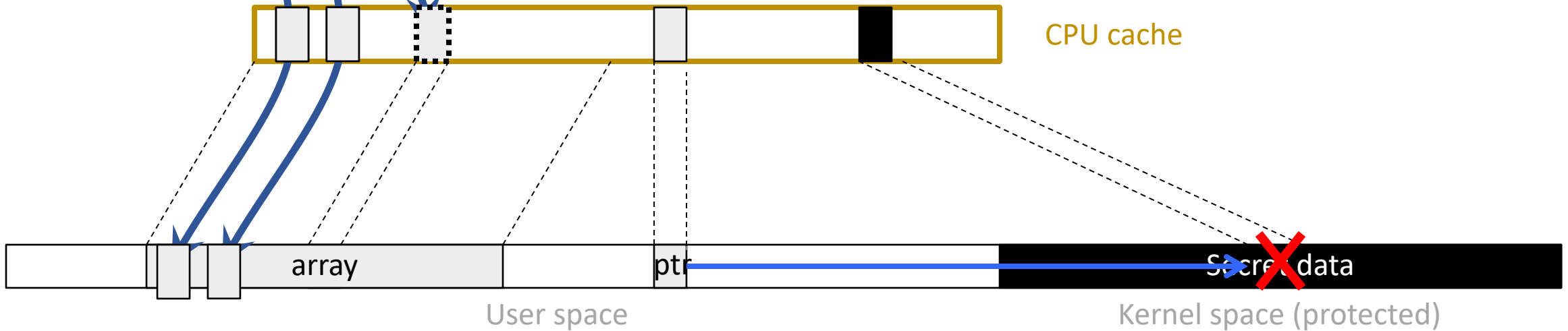
Slow Slow

Step 4:

Unprivileged process iterates through array elements

Step 5:

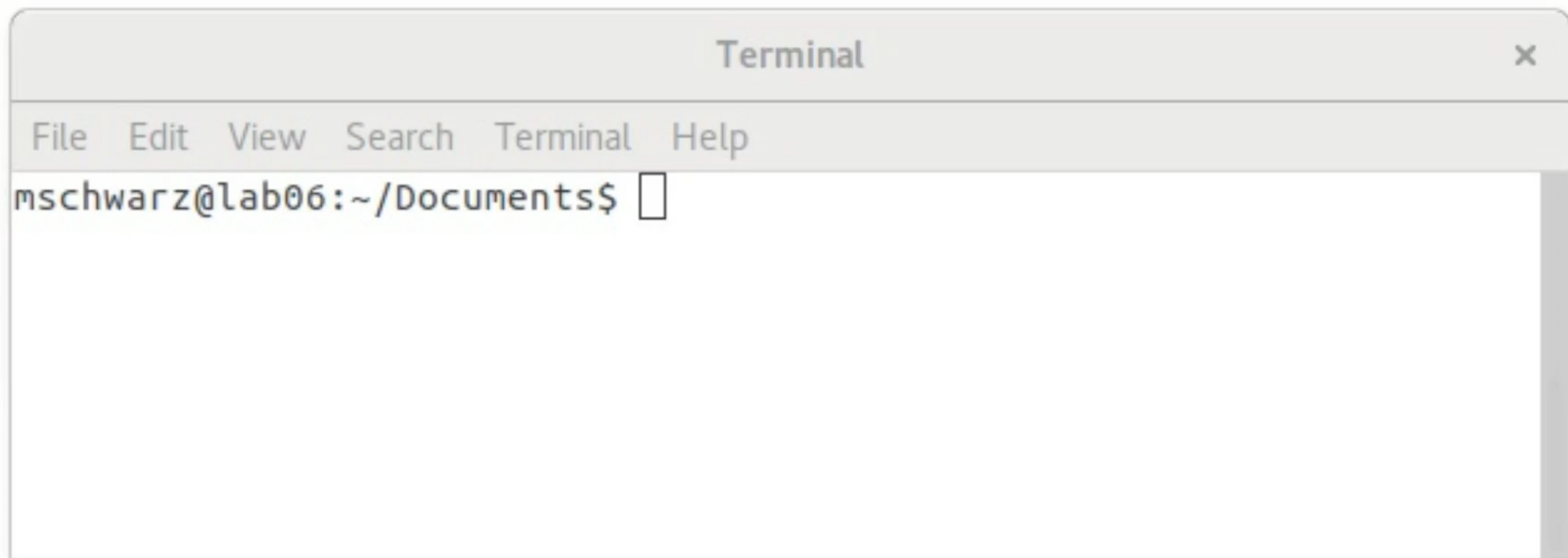
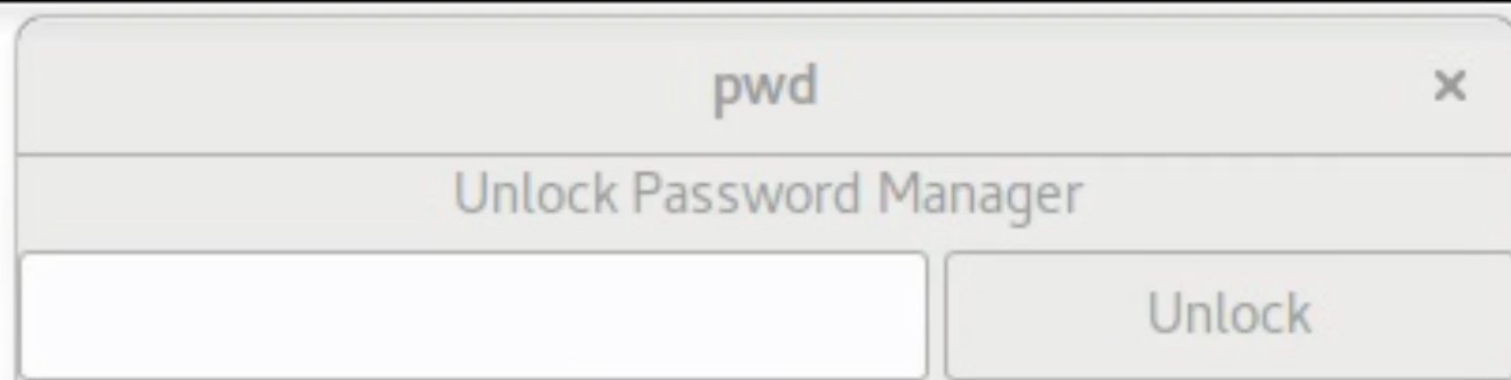
Cached element will return much faster: index indicates secret byte value



DEMO

Spying in Realtime on Password Input

[Source: [Schwarz \(2018\)](#)]





Meltdown – Mitigation

- **Kernel Page Table Isolation (KPTI)**
 - Linux kernel memory no longer mapped into user space processes
 - User space can no longer access kernel memory
- Approach seemingly solid, but...
 - On-going discussion about soundness
 - SMI handlers: parts of kernel memory must always be mapped into user space processes
 - Protects kernel, but user space programs still vulnerable
 - Introduces overhead when jumping from user mode to kernel mode
 - New capability introduced (CAP_DISABLE_PT), disables KPTI for “safe” processes¹

¹ [Re: \[RFC PATCH v2 6/6\] x86/entry/pti: don't switch PGD on when pti_disable is set \[LWN.net\]](#)



Meltdown - Intel-only? (1)

- Meltdown initially thought to be linked with Transactional Synchronization Extensions (TSX-NI)
 - Intel-only hardware atomic memory operations on Haswell and later
 - Enables Meltdown attack without triggering software exception handling
- TSX not a requirement for Meltdown
 - Does make attack virtually impossible to detect²

² *Detecting Attacks that Exploit Meltdown and Spectre with Performance Counters – Trend Micro, 2018*



Meltdown - Intel-only? (2)

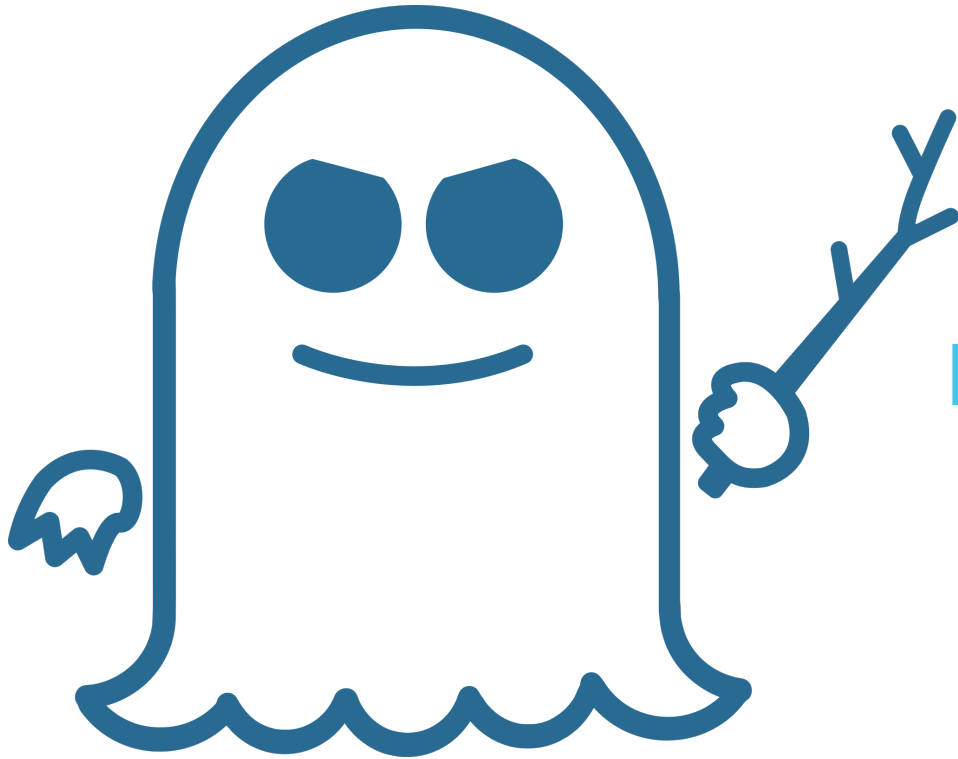
- Meltdown initially thought not to affect AMD processors³

```
From      Tom Lendacky <thomas.lendacky@amd.com>  
Subject   [PATCH] x86/cpu, x86/pti: Do not enable PTI on AMD processors  
Date      Tue, 26 Dec 2017 23:43:54 -0600
```

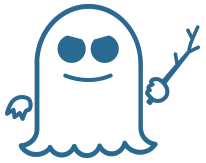
```
AMD processors are not subject to the types of attacks that the kernel  
page table isolation feature protects against.  The AMD microarchitecture  
does not allow memory references, including speculative references, that  
access higher privileged data when running in a lesser privileged mode  
when that access would result in a page fault.
```

- Meltdown paper release: AMD *is* likely vulnerable
- PoC confirms OoO execution occurs across security domains, practical exploitation therefore seems feasible

³ [LKML: Tom Lendacky: \[PATCH\] x86/cpu, x86/pti: Do not enable PTI on AMD processors](#)



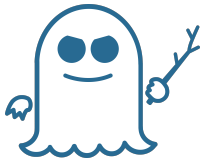
Microarchitectural Side Channels Part 2: Spectre



Spectre – Basic Outline

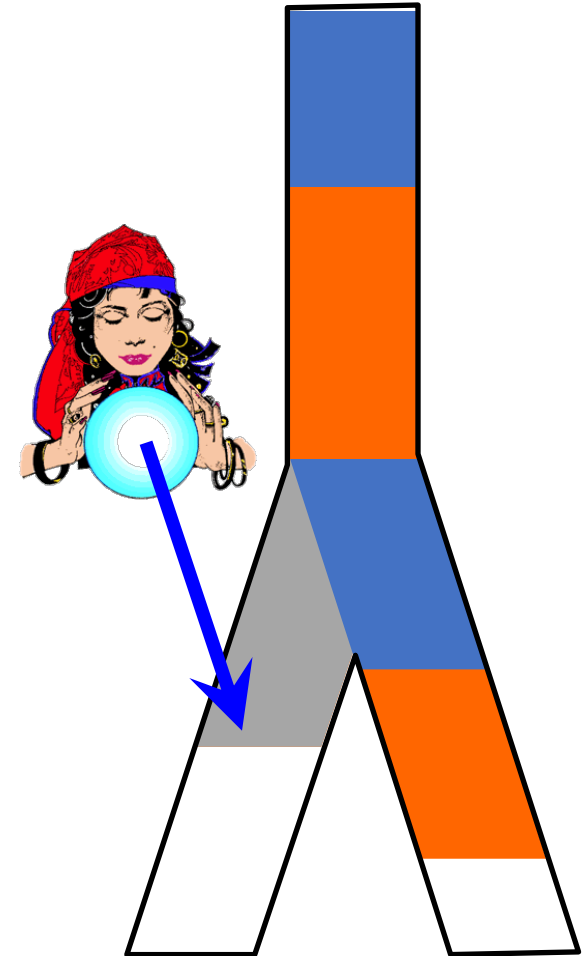
- Design flaw that affects all modern CPUs: Intel, AMD, ARM, PowerPC
- Branch prediction and speculative execution leave traces in cache
- Cache timing attack reveals data from different security domains

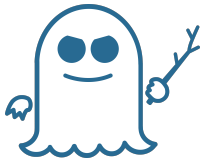
- Two variants:
 - **Spectre-v1**: Read from the current user space process
 - **Spectre-v2**: Read from other processes



Speculative Execution and Branches (1)

- When execution reaches a branch
- The processor predicts the outcome of the branch
- Execution proceeds (speculatively) along predicted branch
- **Correct prediction** → all is well
- **Misprediction** → abandon and resume



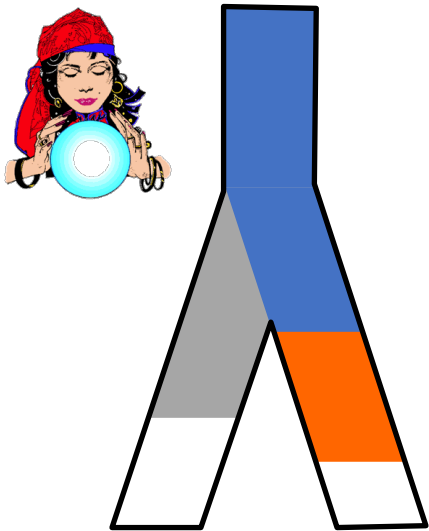


Speculative Execution and Branches (2)

- **Branch History Buffer (BHB)**

Outcome of conditional branches

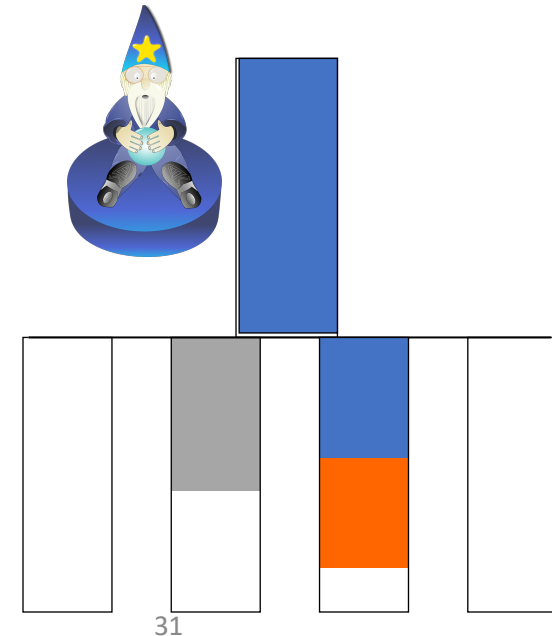
JGE 4006c9

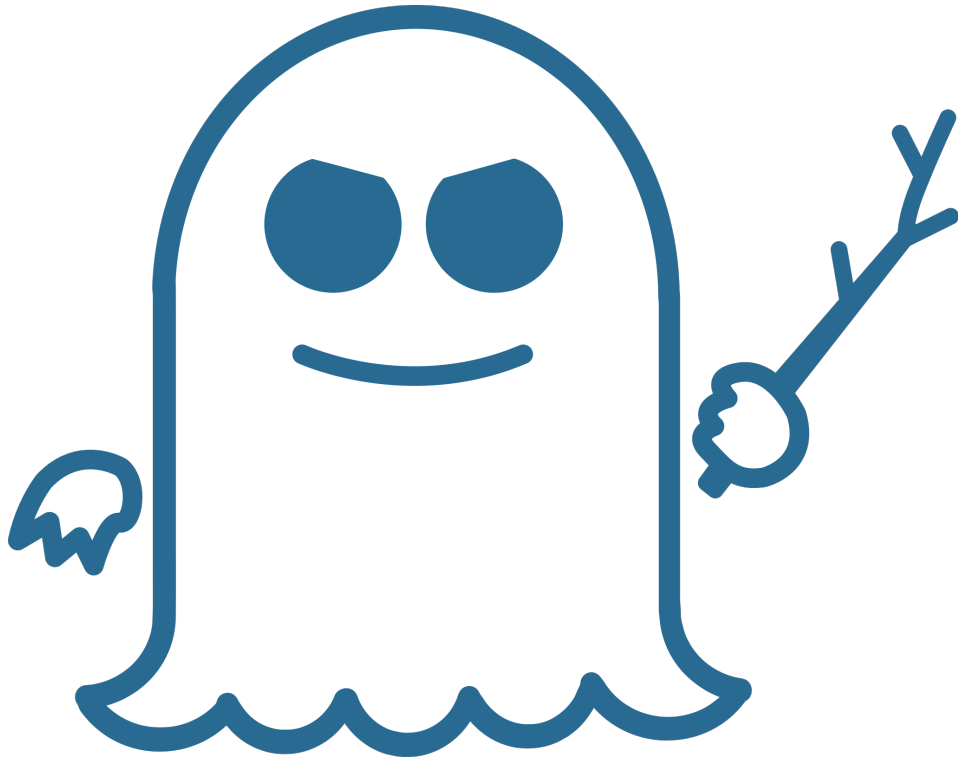


- **Branch Target Buffer (BTB)**

Target of indirect branches

JMP eax

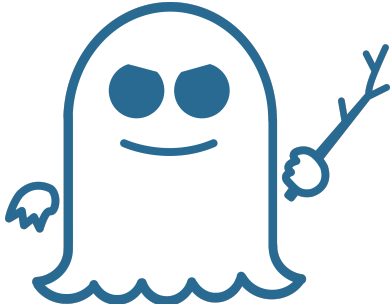




Spectre Variant 1 – Bounds Check Bypass

Victim

Spectre-v1



Attacker

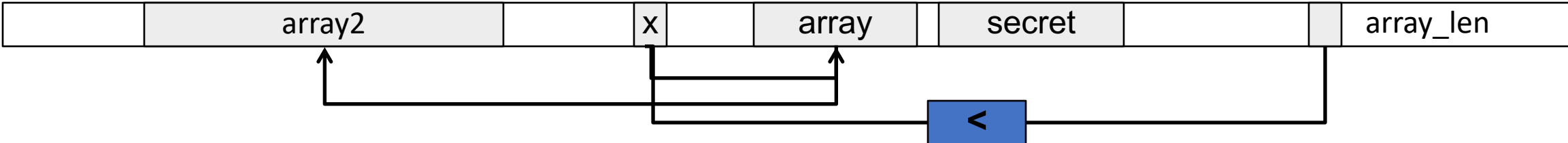


Branch taken!

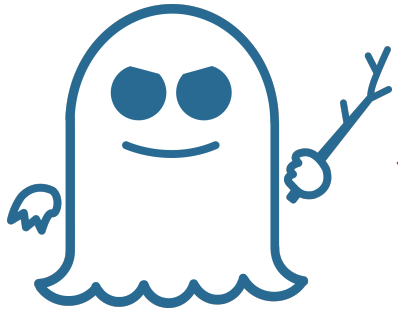
```

if (x < array_len) {
  i = array[x];
  y = array2[i * 256];
}

```



Spectre-v1



Attacker



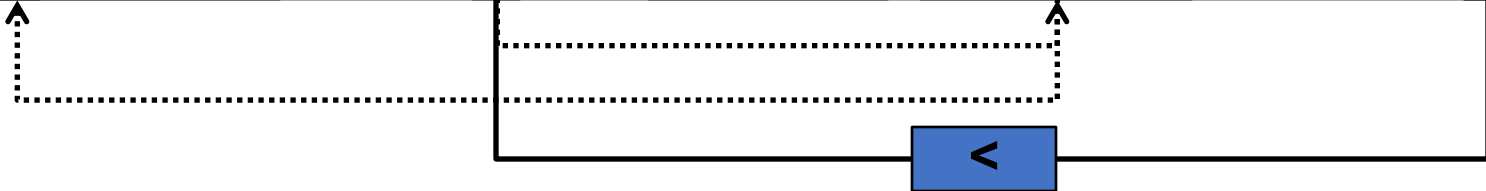
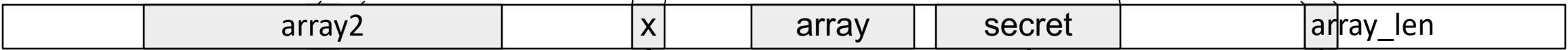
Branch taken!

Victim
X is large

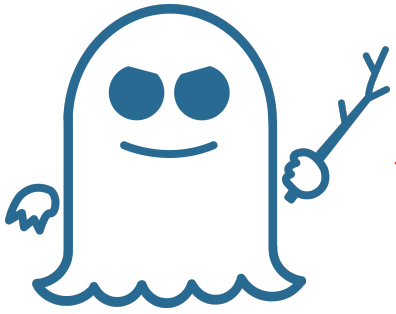
```
if (x < array_len) {  
    i = array[x];  
    y = array2[i * 256];  
}
```



Cache



Spectre-v1



Attacker



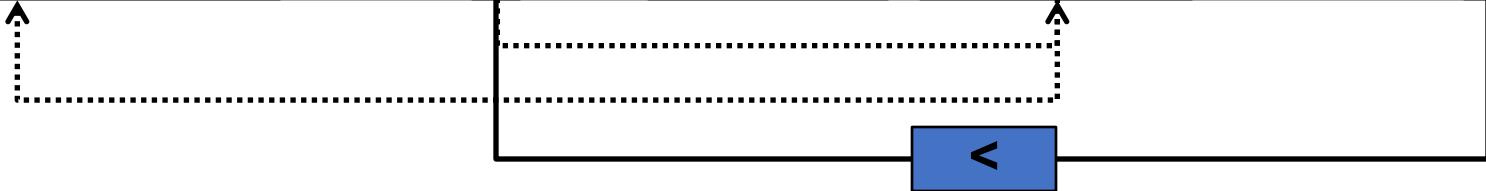
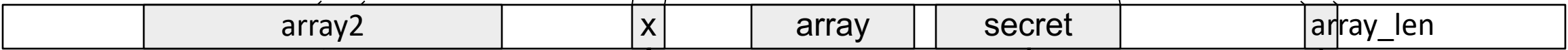
~~Branch
taker~~

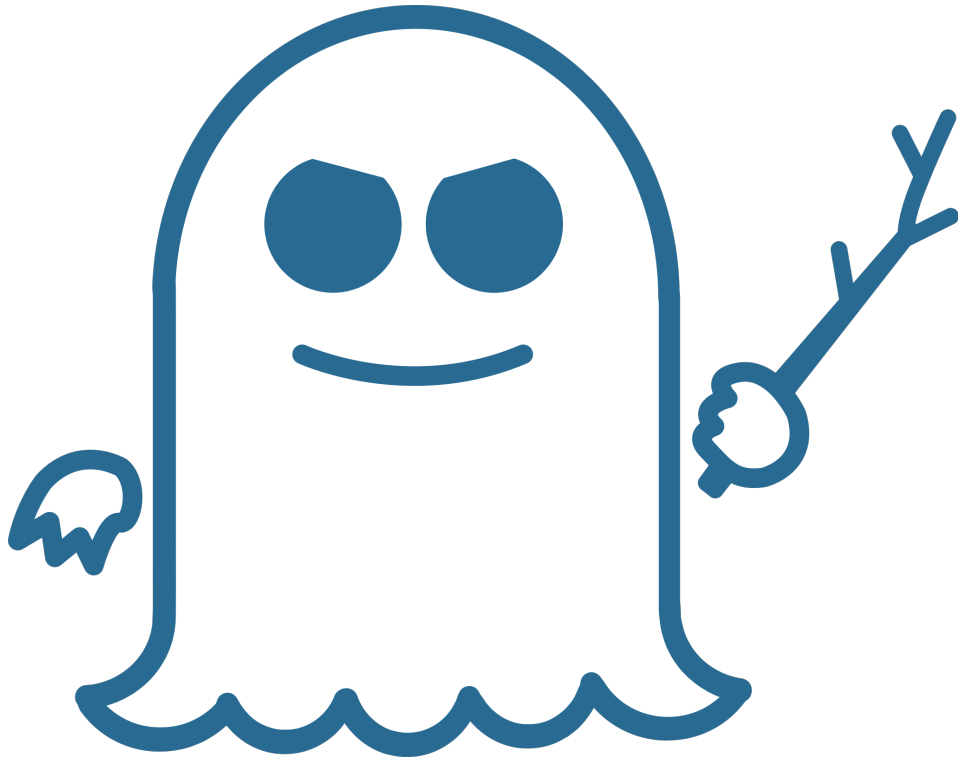
Mispredict

```
if (x < array_len) {  
    i = array[x];  
    y = array2[i * 256];  
}
```

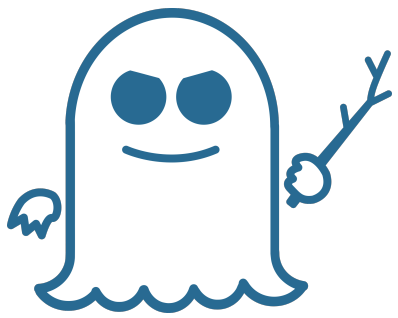


Cache





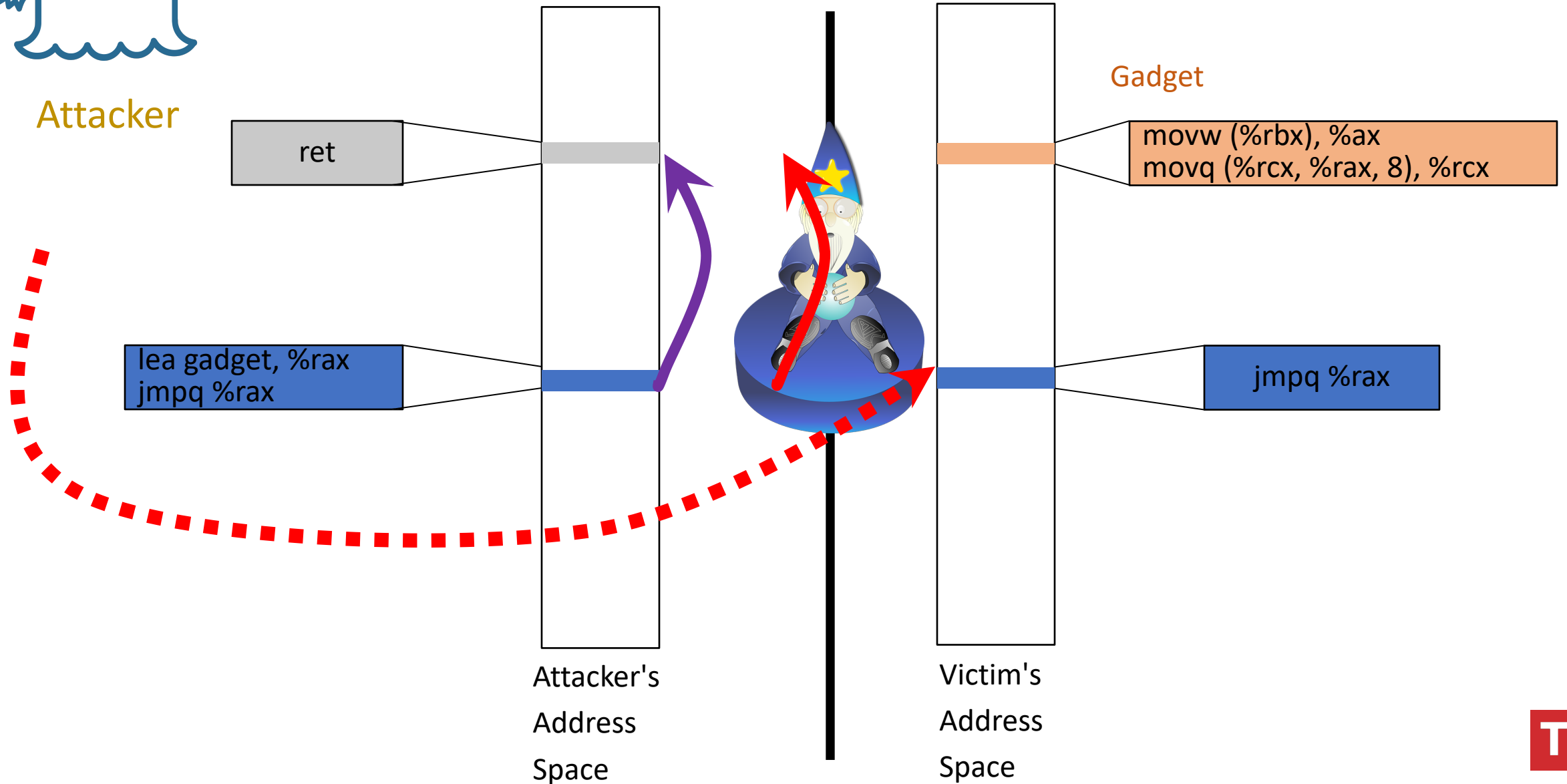
Spectre Variant 2 – Branch Target Injection

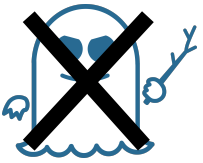


Attacker

Spectre-v2

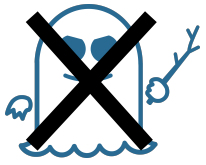
Victim





Spectre – Mitigations

- **Basic idea:** prevent speculative execution across branches
- **Three approaches:**
 - Spectre-v1: Explicitly prevent speculative execution across conditional branches by inserting blocking operation
 - Spectre-v2: Avoid training branch predictor by replacing branch instructions with semantic equivalents
 - Spectre-v2: Disable branch prediction across security domains



Spectre-v1 – Insert Blocking Operation

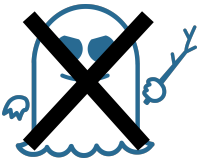
- **Approach:** Prevent speculative execution by inserting blocking operation
- LFENCE (serialize load operations), PAUSE (spin loop hint)

```
scanf("%d", &untrusted);  
if(untrusted < arrayLength)  
{  
    value = array[untrusted];  
    asm("lfence");  
    value2 = array2[value * 64];  
}
```

```
call    4004a0 <__isoc99_scanf@plt>  
mov     ecx,DWORD PTR [rbp-0xe4]  
cmp     ecx,DWORD PTR [rbp-0xe8]  
mov     DWORD PTR [rbp-0x114],eax  
jge    4006c9 <main+0x109>  
movsxd rax,DWORD PTR [rbp-0xe4]  
movsx  ecx,BYTE PTR [rbp+rax*1-0x70]  
mov     DWORD PTR [rbp-0xec],ecx  
lfence  
mov     ecx,DWORD PTR [rbp-0xec]  
shl    ecx,0x6
```

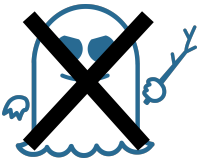


- Effective, but
 - Need to recompile code or patch binary
 - Significantly degrades performance – need static analysis to identify vulnerable code



Spectre-v2 – Retpoline (1)

- **Approach:** Avoid training branch predictor by replacing branch instructions with semantic equivalents
- **Return trampoline (retpoline)**
 - Indirect branch normally pulls return address off stack (“jump to this address”)
 - Replace with PUSH/RET
 - Push target address onto stack
 - Return to target address
 - BTB does not learn about branch due to pattern mismatch



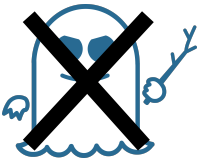
Spectre-v2 – Retpoline (2)

- Need to recompile code or patch binary
- Degrades performance
 - Somewhat mitigated by Return Stack Buffer (RSB)
- Not a perfect solution: **ineffective** on Skylake and later
 - RSB behavior different: when empty, falls back to BTB prediction
 - Addressed with RSB stuffing⁴; implemented by e.g. Linux kernel⁵ and LLVM compiler⁶

⁴ [Retpoline: A Branch Target Injection Mitigation - Intel, 2018](#)

⁵ [x86/retpoline: Avoid return buffer underflows on context switch - Patchwork](#)

⁶ [\[llvm-dev\] LLVM Release Schedules: 5.0.2, 6.0.1](#)



Spectre-v2 – Disable BTB Prediction

- **Approach:** Disable BTB prediction across security domains
- **Intel microcode update** ^{7,8}
 - Introduces new Model Specific Registers (MSRs) to control BTB
 - No learning across hyperthreads
 - Higher security levels do not learn from lower level activity
 - BTB clobbering, wiped on each context switch
 - Major performance impact

⁷ [Microcode Revision Guidance - Intel, 2018](#)

⁸ [Controlling the Performance Impact of Microcode and Security Patches - RedHat, 2018](#)

Beyond Spectre and Meltdown

- Various follow-up publications on variants, other CPU vulnerabilities (non-exhaustive):
 - SGXpectre (2018)
 - Foreshadow, Foreshadow-NG (2018)
 - Microarchitectural Data Sampling: Rogue In-Flight Data Load, Fallout, ZombieLoad (2019)
 - BlindSide (2020)
 - Load Value Injection (2020)
 - CacheOut, SGAxe (2020)
 - CrossTalk (2021)
 - Rage Against The Machine Clear (2021)
- Where available, mitigations usually comprise
 - Compiler- and kernel-based protections
 - Microcode updates for (then) in-market CPUs; partial silicon redesign for newer generations

References

Background reading

- “Meltdown: Reading Kernel Memory from User Space” (2018)
- “Spectre Attacks: Exploiting Speculative Execution” (2018)

Admin

- **Quiz**
 - Verify your understanding of material
- **Questions?**
 - Reach out via email