Rethinking Instruction Set Guarantees for Software-Driven Hardware Security

Ashay Rane Calvin Lin Mohit Tiwari

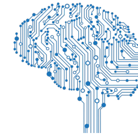
The University of Texas at Austin

Secret Information in Various Applications

Applications in several domains operate on private or confidential information.

Cryptography

Web Browsers

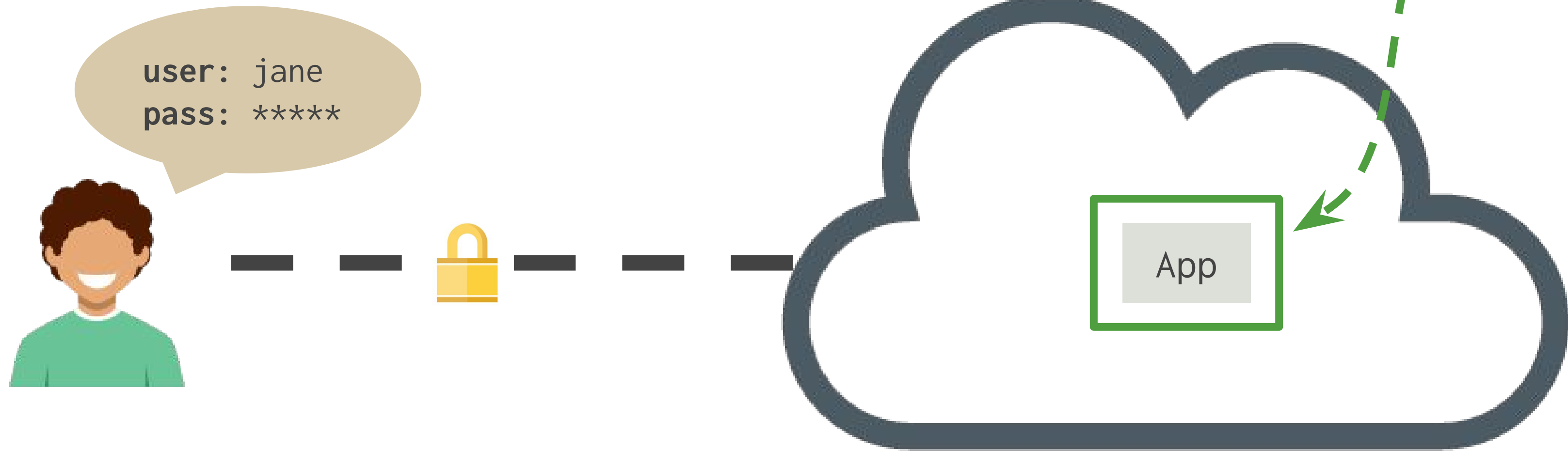


Machine Learning

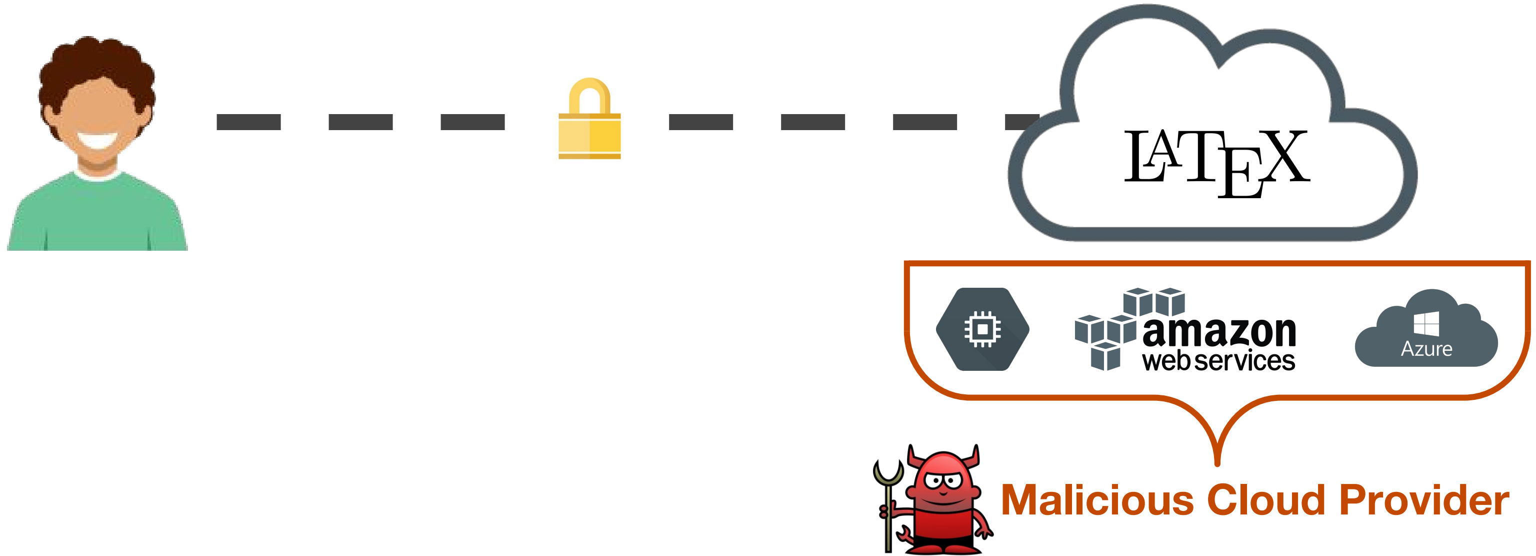
We want to prevent leakage of secret information

Many Techniques for Preventing Information Leakage

Virtual Machines and **Containers** for isolating applications from each other.



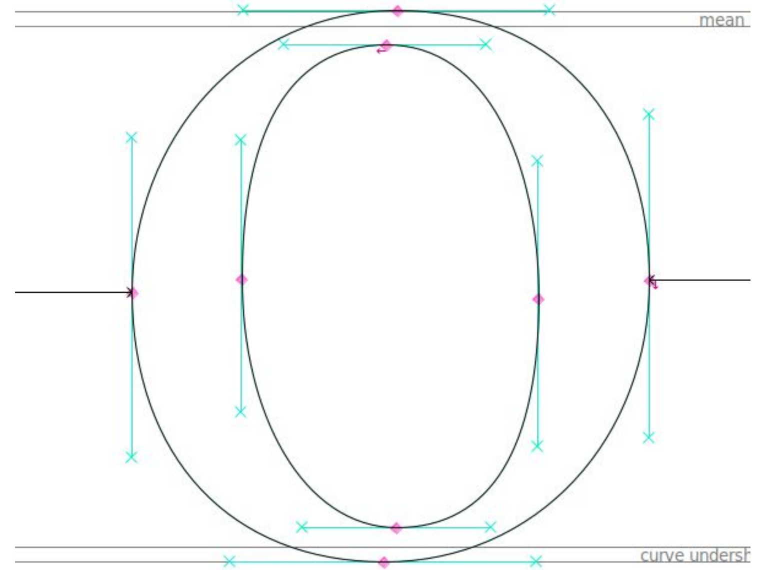
Example of Side Channel Information Leakage



Example of Side Channel: Rendering Characters

o

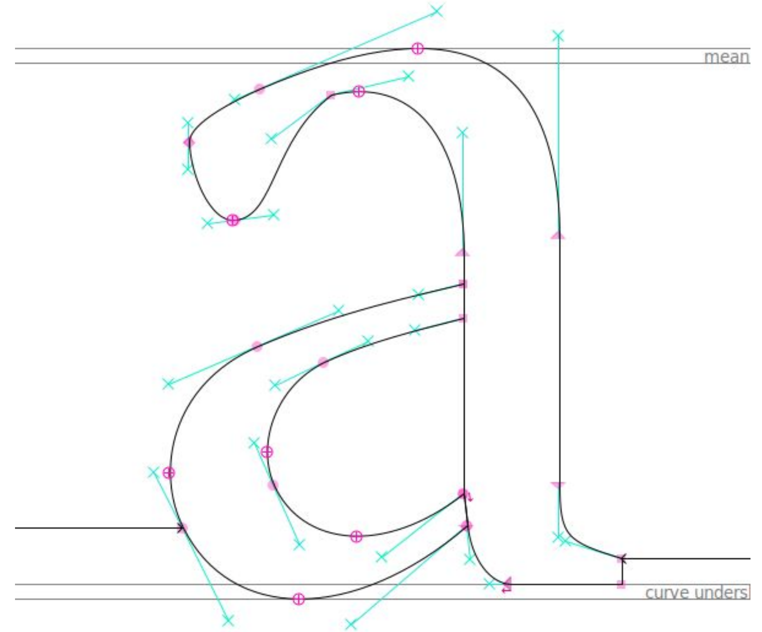
Rendered using
lines and curves



Example of Side Channel: Rendering Characters

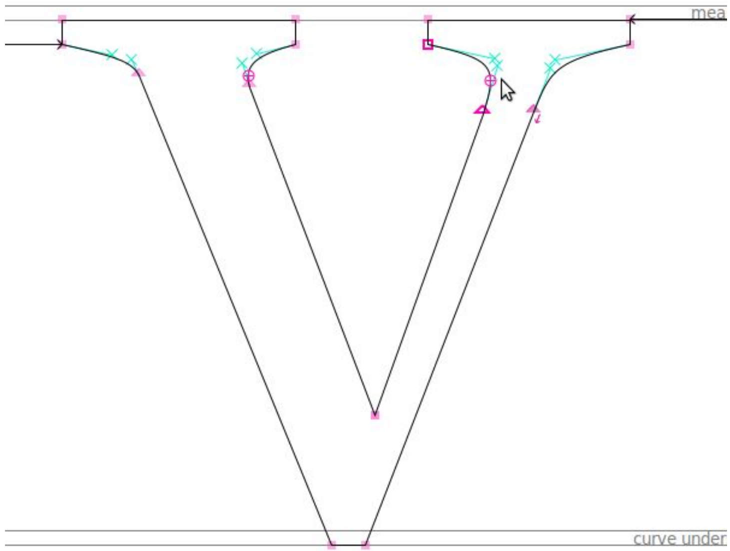
a

Rendered using
lines and curves

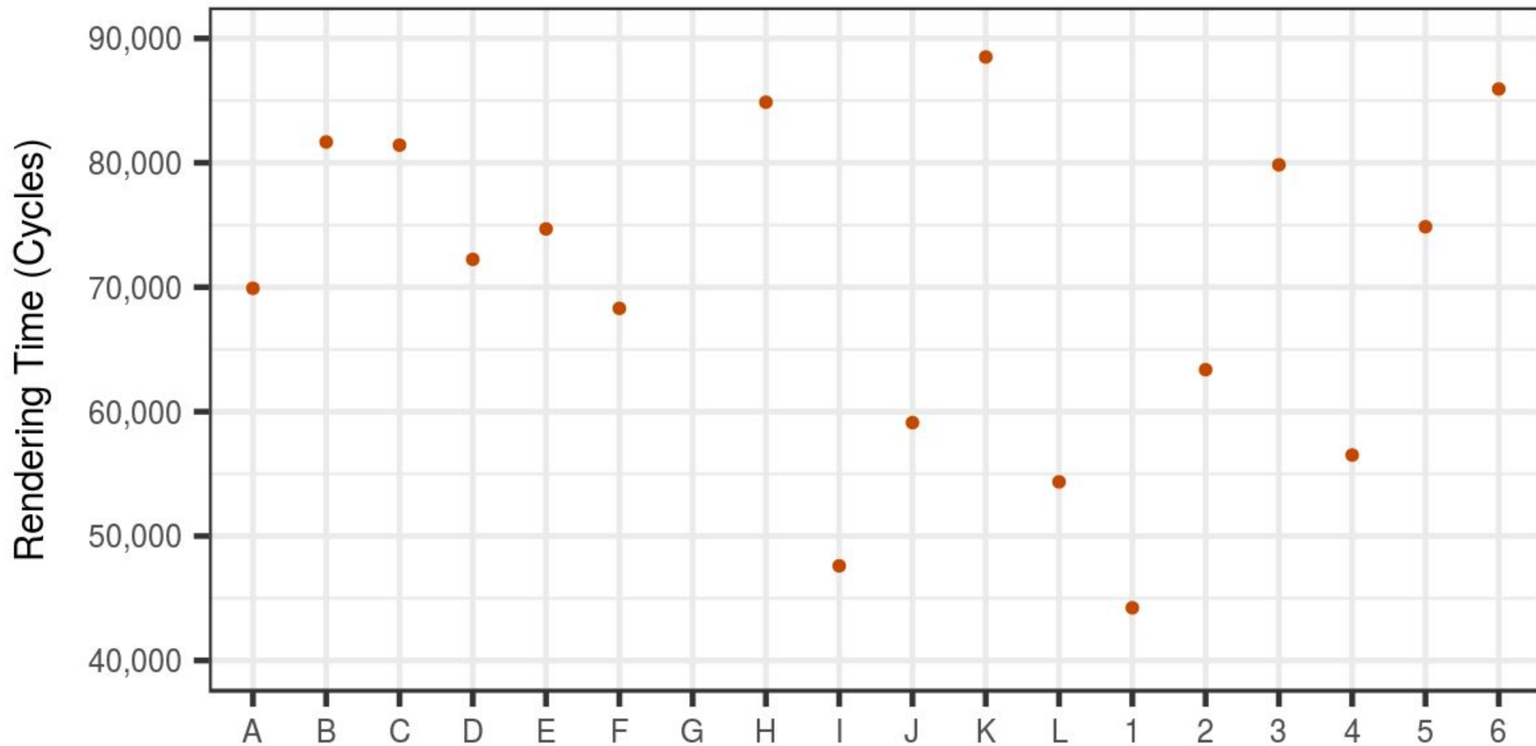


Example of Side Channel: Rendering Characters

Rendered using
lines and curves



Execution Time for Rendering Characters



Execution Time for Rendering Words

Words of same length take different times to render

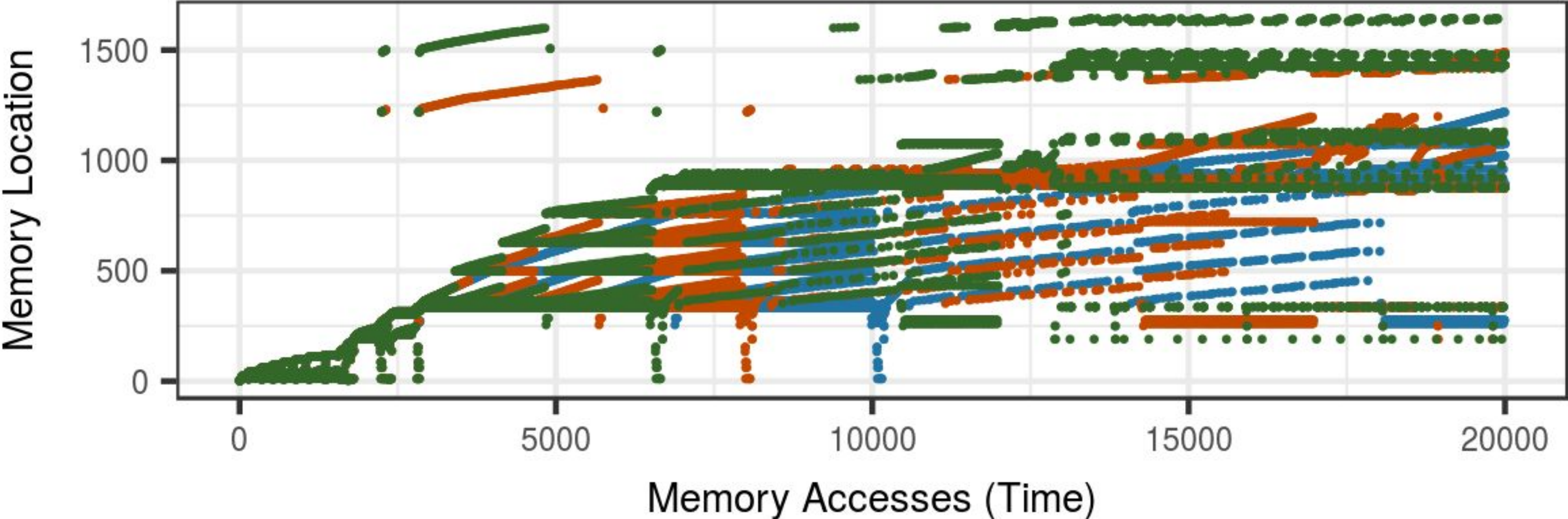
Word	Rendering Time
`tranquilizer`	734×10^3 cycles
`convincingly`	809×10^3 cycles
`experiencing`	853×10^3 cycles
`demagnetized`	943×10^3 cycles

Execution time of rendering process

can reveal information about document

even if the document is encrypted in transit.

Memory Address Trace while Rendering Characters



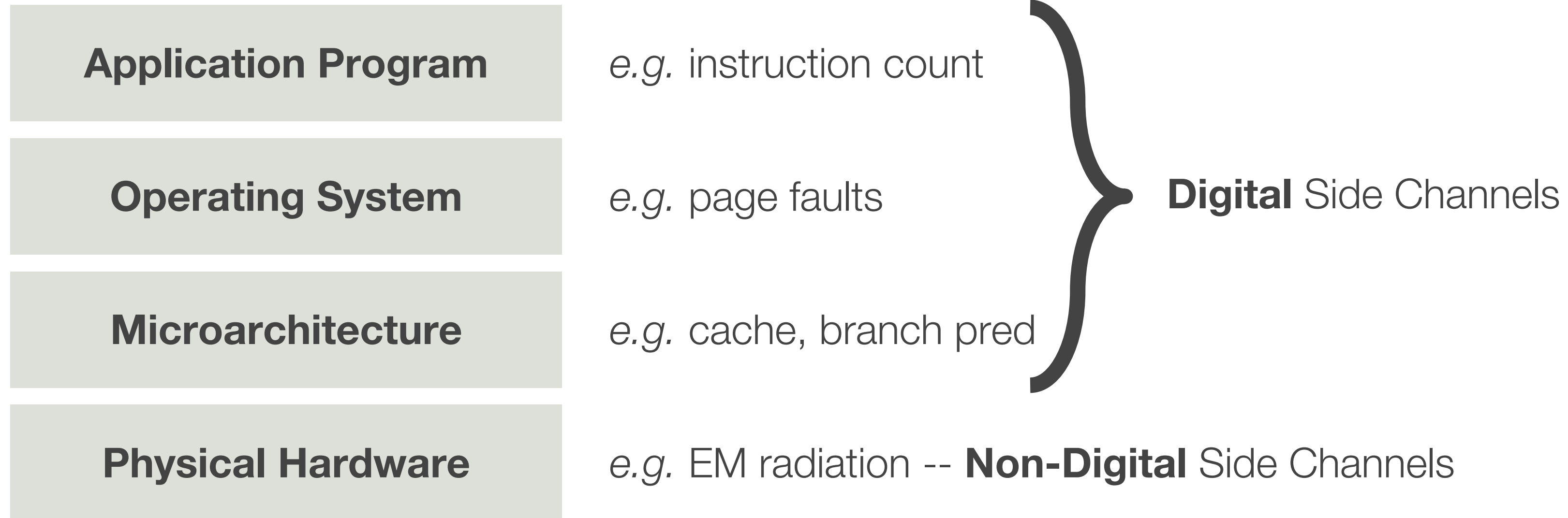
Secrets may Leak Through **Many Side Channels**

Secret data can be inferred using many side channels

Application Program	*e.g.* instruction count
Operating System	*e.g.* page faults
Microarchitecture	*e.g.* cache, branch predictor
Physical Hardware	*e.g.* EM radiation

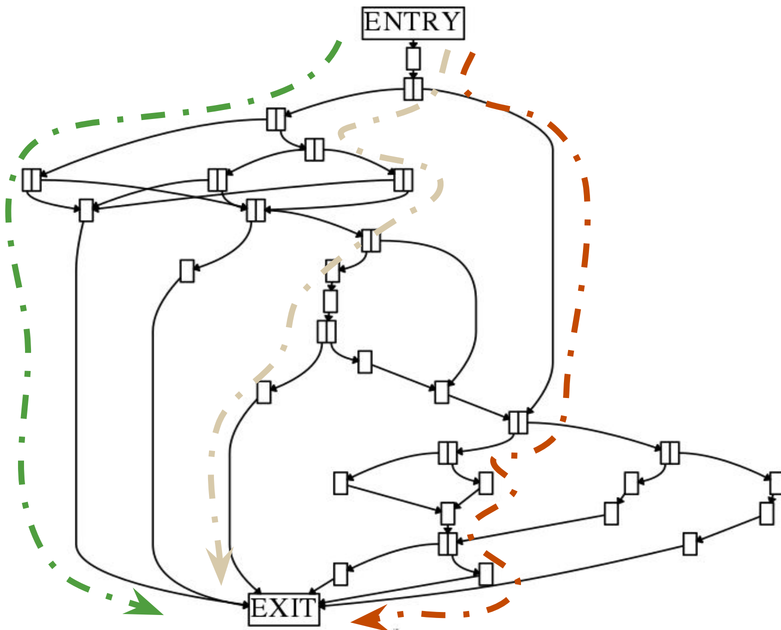
Secrets may Leak Through **Many Side Channels**

Secret data can be inferred using many side channels



What is the **Core Vulnerability**?

The code executes different sequences of instructions for rendering each character.



Different input values execute different program paths, thus causing variation in execution.

Prior Digital Side Channel Defenses

Are point solutions, since they focus on symptoms and not the root cause

Application Program	*e.g.* number of instructions [ASPLOS15], [ICISC05], [ICISC10]
Operating System	*e.g.* page faults
Microarchitecture	*e.g.* branch predictor, cache, PC, DRAM addresses

Prior Digital Side Channel Defenses

Are point solutions, since they focus on symptoms and not the root cause

Application Program	*e.g.* number of instructions
Operating System	*e.g.* page faults
Microarchitecture	*e.g.* branch predictor, cache, PC, DRAM addresses

[ISCA07], [ISCA08], [HPCA09], [NDSS15], [CCS13a]

Prior Digital Side Channel Defenses

Are point solutions, since they focus on symptoms and not the root cause

Application Program	*e.g.* number of instructions
Operating System	*e.g.* page faults
Microarchitecture	*e.g.* branch predictor, cache, PC, DRAM addresses

[ISCA13], [CCS13b], [CCS13c], [ASIACRYPT11]

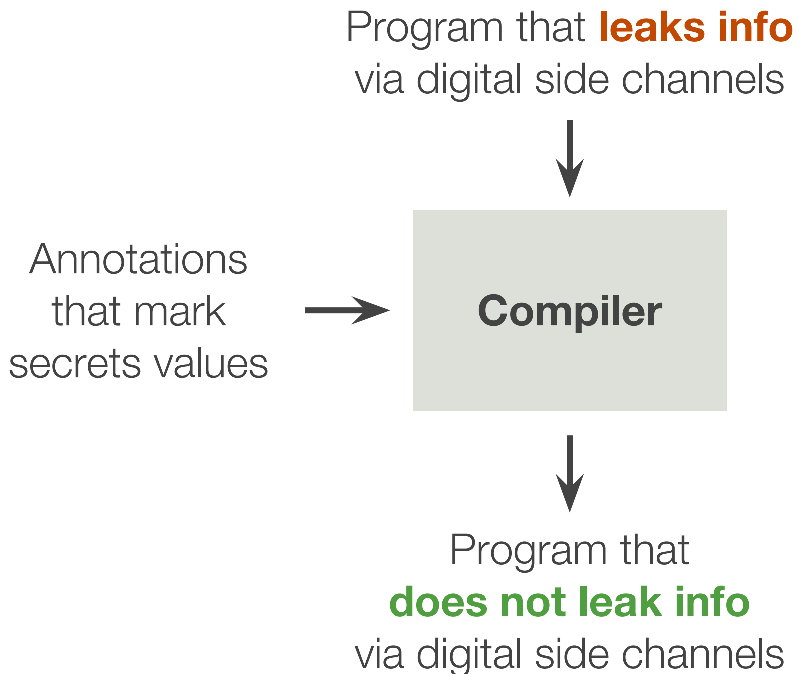
Our Solution

Simultaneously defends from many side channels using a single solution

Compatible with most optimizations in the compiler and in the microarchitecture (e.g. caches, prefetchers)

Built into a compiler, making our solution automated and tailored to application programs

Our Solution

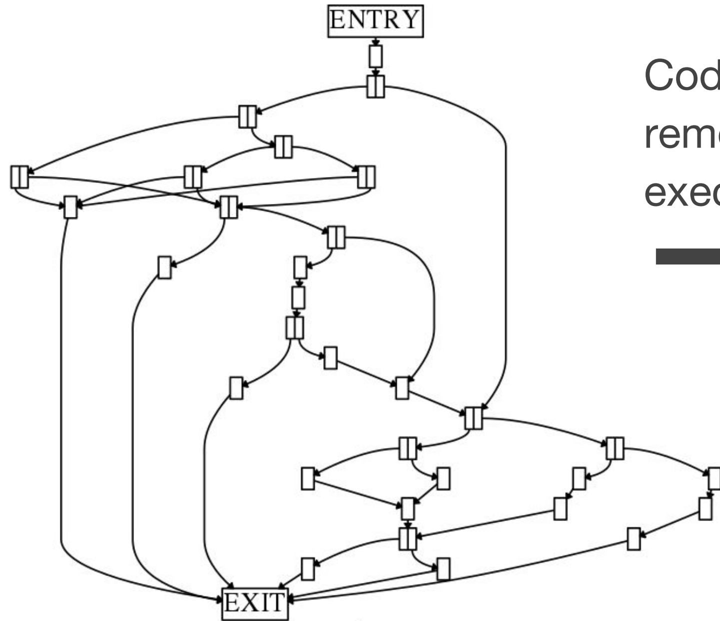


Step 1 [Code Analysis]:
Identify portions of the program that need security.

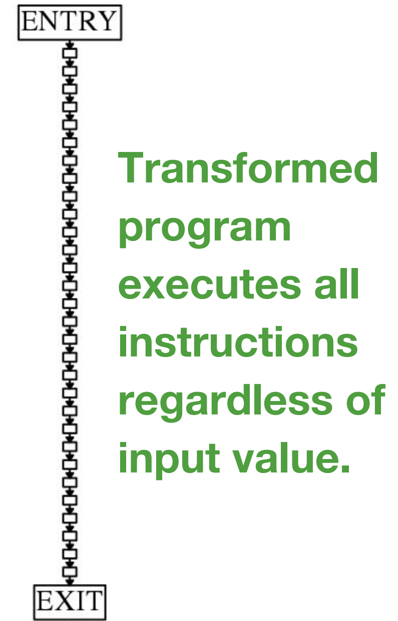
Step 2 [Code Transformation]:
Change relevant portions of the program, so that they don’t leak secrets through side channels.

CFG of Program *P* that **Leaks** Information via **Program Counter**

Program *Q* that **Does Not Leak** Information via **Program Counter**



Code transformation removes variations in executed basic blocks



Transformed program executes all instructions regardless of input value.

Challenges in Eliminating Variations

Naively executing all instructions will produce invalid results.

Incorrect Output	Crashing Execution	Stuck Program

Incorrect Transformation

Original Program

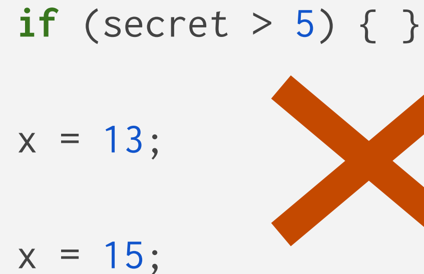
```
if (secret > 5) {
  x = 13;
} else {
  x = 15;
}
```

Incorrect Transformation

```
if (secret > 5) { }

x = 13;

x = 15;
```



Ensuring **Correctness**

Original Program

```
if (secret > 5) {
  x = 13;
} else {
  x = 15;
}
```

Correct Transformation

```
(secret > 5)    x = 13;
(secret <= 5)   x = 15;
```

Key Building Block: **Predicated Write Operation**

```
mov   a -> output   // Set destination
test  cond, cond    // Check if non-zero
cmovz b -> output   // Conditional update
test  a, a          // Overwrite flags
```

Ensuring **Correctness**

Original Program

```
if (secret > 5) {
  x = 13;
} else {
  x = 15;
}
```

Correct Transformation

```
pred = secret > 5;
x = pred_write(pred,  13, x);
x = pred_write(!pred, 15, x);
```

Key Building Block: **Predicated Write Operation**

The **`pred_write()`** function:

- ➔ Has same sequence of instructions.
- ➔ Accesses zero memory locations.
- ➔ Consumes same number of processor cycles (verified empirically).

`pred_write()` conditionally updates a memory location without leaking the predicate through side channels.

Key Building Block: **Predicated Write Operation**

The `pred_write()` function:

- ➔ Has same sequence of instructions.
- ➔ Accesses zero memory locations.
- ➔ Consumes same number of processor cycles (verified empirically).

`pred_write()` conditionally updates a memory location without leaking the predicate through side channels.

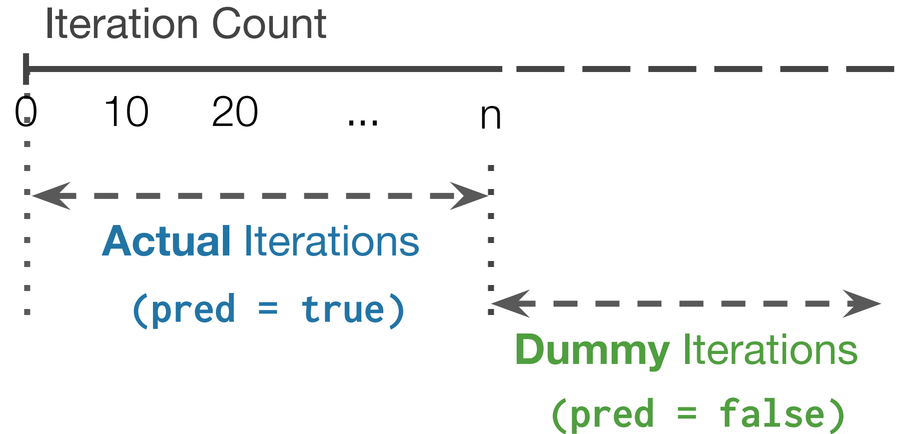
We can now **execute arbitrary* instructions**, but we allow them to update memory contents only if the instruction is part of the correct execution path.

* System calls and library function calls are outside the scope of our compiler's transformations, since the callee's code cannot be changed.

Transforming Loops

```
loop i :: 0 to n
    x = x * y;
    i = i + 1;
```

Assume **n** is secret.
Transformation should hide the number of executed iterations.



Memory Address Trace

```
result = table[secret];
```

```
addr := base(table) + secret
result := read addr
```

An adversary that can observe address, can also derive secret.

```
secret = addr - base(table)
```

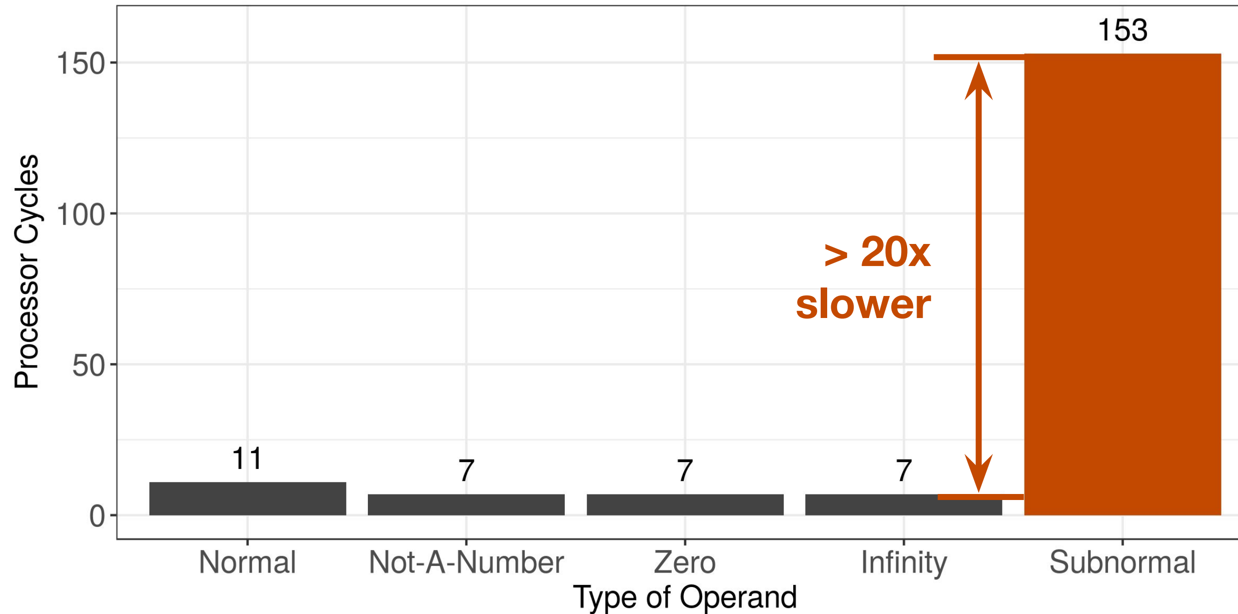
Solution #1: Array Streaming

Accesses the entire array to read one element of the array.

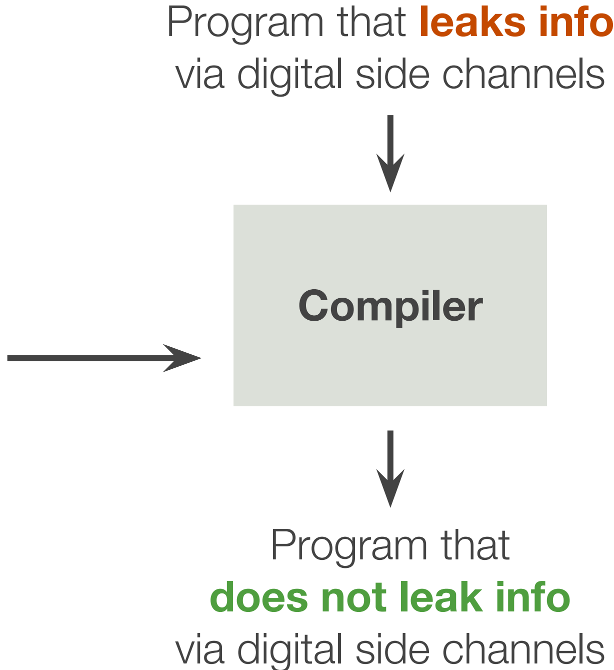
Solution #2: Software ORAM

Software version of Path ORAM [CCS'13], which shuffles memory to hide location of data.

Variable-Latency Floating-Point Instructions



Our Approach: **Model-Driven Compilation**



Constructing a Model of Digital Side Channel Leakage

Model of Microarchitectural Side Channel Leakage

- ➔ Model constructed using code analysis (information flow analysis) of the ISA definition.
- ➔ `Sources`: Register and memory operands.
  `Sinks`: Digital side channel observations (e.g. instruction pointer, address trace, etc.).
- ➔ Any information flow between `sources` and `sinks` indicates the existence of a digital side channel.
- ➔ In our implementation, we analyze x86 and ARMv8 ISAs to construct the model.

Benefits of the Model-Driven Approach

1. **Compilers can generate code that is free from digital side channels**

   Enables automated flexible defenses since the compiler can selectively apply mitigation techniques only where needed.

   We have successfully transformed graph kernels, floating-point libraries, and end-user applications (like machine-learning classifiers) [Rane15, Rane16].

2. **Allows formal verification of the absence of digital side channels**

   We formally verified a handful of common cryptographic implementations to be free from digital side channels [Bond17].

   Eliminates the compiler from the trusted computing base.

Adding Security Contract to ISA

Existing ISAs provide a functional contract: **`add a,b,r`** ⇒ **`r = (a+b) mod`** 2^{32}

We want to add **security properties** to ISA, e.g. **`add`** should consume fixed latency regardless of the operand values (a and b), thus closing the timing side channel.

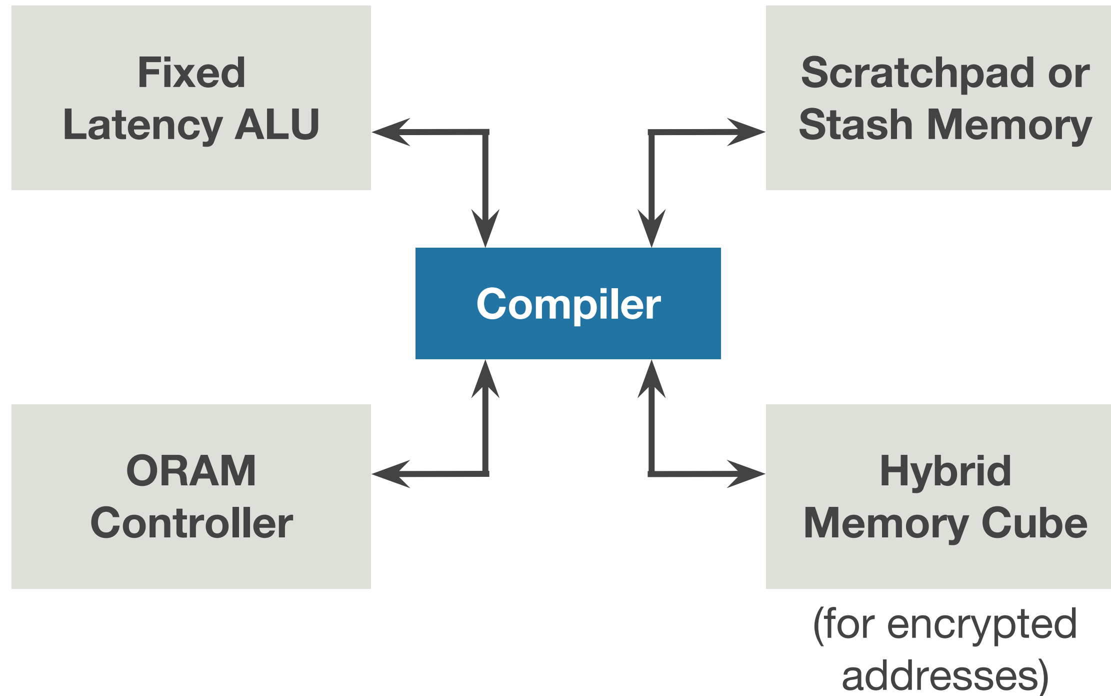
[ARM v8.4 allows setting a bit which ensures fixed timing of certain instructions]

Other security properties in ISA:

- Cache that exposes a portion of its space for a user-controlled scratchpad
- Memory controller that selectively enables ORAM accesses

Compiler to Orchestrate Security

Our goal is to let the compiler orchestrate the use of hardware security components



Conclusion

1. Side channels are an important problem and they are hard to close.

2. The key to effective side channel defenses is a model of the microarchitectural side channel leakage.

3. We can build high-assurance and high-performance defenses by augmenting the ISA with security properties.