Broad-Based Side-Channel Defenses for Modern Processor Architectures

Ashay Rane
Advisors: Calvin Lin and Mohit Tiwari
The University of Texas at Austin
Meltdown and Spectre: ‘worst ever’ CPU bugs affect virtually all computers

The Washington Post
Huge security flaws revealed – and tech companies can barely keep up

chip maker faces 35 lawsuits over the attacks
Spectre, Meltdown, ...
Side Channels

Spectre, Meltdown, ...
Private Information in Various Applications

- Cryptography
- Web Browsers
- Machine Learning
Private Information in Various Applications

- Cryptography
- Web Browsers
- Machine Learning

We want to prevent leakage of private information
Techniques to Maintain Privacy of Information
Techniques to Maintain Privacy of Information

Encryption
Techniques to Maintain Privacy of Information

- User: jane
- Pass: *****
- 2FA: *****

Encryption
Authentication + Authorization
Techniques to Maintain Privacy of Information

**User: Jane**
**Pass: ******
**2FA: ******

Virtual Machine

- Encryption
- Authentication + Authorization
- Sandboxing
Example Scenario

bank-stmt.pdf
Example Scenario

Account Number: 012345
Current Balance: $5,582
Rendering Characters Using Lines and Curves
Rendering Characters Using Lines and Curves

O

Rendered using lines and curves

O
Rendering Characters Using Lines and Curves

a  Rendered using lines and curves

a
Rendering Characters Using Lines and Curves

V

Rendered using lines and curves
Execution Time for Rendering Characters

Rendering Time (Cycles)

A B C D E F G H I J K L 1 2 3 4 5 6
Proof of Concept on a Font Renderer

Original Text:

hello world my social security number is 123 45 6789
Proof of Concept on a Font Renderer

Original Text:

hello world my social security number is 123 45 6789

Recovered Text:
Proof of Concept on a Font Renderer

Original Text:
hello world my social security number is 123 45 6789

Recovered Text:
wello would my socqal secuuqtk kumweu it 1r3 45 6789
Proof of Concept on a Font Renderer

Original Text:
hello world my social security number is 123 45 6789

Recovered Text:
wello would my socqal secuuqtk kumweu it 1r3 45 6789

41 out of 52 characters correctly guessed
Attacker can measure *execution time* to steal sensitive document contents
Real-World Attack on FreeType Renderer
[Xu et al., Oakland-2015]

Application converts document into an image.
Real-World Attack on FreeType Renderer
[Xu et al., Oakland-2015]

Application converts document into an image
Application runs inside an SGX-like enclave
Real-World Attack on FreeType Renderer
[Xu et al., Oakland-2015]

Application converts document into an image
Application runs inside an SGX-like enclave
Malicious OS observes page faults
Real-World Attack on FreeType Renderer
[Xu et al., Oakland-2015]

Application converts document into an image
Application runs inside an SGX-like enclave
Malicious OS observes page faults
100% text recovered by OS
Memory Address Trace While Rendering Characters

Rendered Character:  
- X
- Y
- Z

Memory Location

Memory Accesses (Time)
Instruction Trace While Rendering Characters

Rendered Character:  

- Blue: X
- Red: Y
- Green: Z

Graph showing basic block ID on the y-axis and basic block execution (time) on the x-axis. The graph displays the execution timeline for X, Y, and Z characters.
Information May Leak Through Many Side Channels

- **Application Program**: e.g. execution time
- **Instruction Set Arch**: e.g. page faults
- **Microarchitecture**: e.g. branch predictor, cache, PC, DRAM addresses
- **Physical Hardware**: e.g. power consumption, EM radiation
What is the **Core Vulnerability**?
Different input values execute different paths, thus causing variations, which create side channels.
Prior Side Channel Defenses

Focus on symptoms, thus providing point solutions

- **Application Program**: e.g. execution time
  - [ISCA12], [ASPLOS15], [CHES00], [ICISC03], [ICISC05], [ICISC10]
- **Instruction Set Arch**: e.g. page faults
- **Microarchitecture**: e.g. branch predictor, cache, PC, DRAM addresses
- **Physical Hardware**: e.g. power consumption, EM radiation
**Prior Side Channel Defenses**

Focus on symptoms, thus providing point solutions

- **Application Program**
  - *e.g.* execution time

- **Instruction Set Architecture**
  - *e.g.* page faults

- **Microarchitecture**
  - *e.g.* branch predictor, **cache**, PC, DRAM addresses
  
  [ISCA07], [ISCA08], [HPCA09], [NDSS15], [CCS13a]

- **Physical Hardware**
  - *e.g.* power consumption, EM radiation
Prior Side Channel Defenses

Focus on symptoms, thus providing point solutions

- Application Program: e.g. execution time
- Instruction Set Arch: e.g. page faults
- Microarchitecture: e.g. branch predictor, cache, PC, **DRAM addresses** [ISCA13, CCS13b, CCS13c, ASIACRYPT11]
- Physical Hardware: e.g. power consumption, EM radiation
Drawbacks of Point Solutions
Drawbacks of Point Solutions

1. Focused on the symptoms not the root cause
   Requires a completely redesigned solution for every side channel
Drawbacks of Point Solutions

1. Focused on the symptoms not the root cause
   Requires a completely redesigned solution for every side channel

2. Difficult to ensure end-to-end or comprehensive security
   One point solution may negate the security guarantees of another
Drawbacks of Point Solutions

1. Focused on the symptoms not the root cause
   Requires a completely redesigned solution for every side channel

2. Difficult to ensure end-to-end or comprehensive security
   One point solution may negate the security guarantees of another

3. Require disabling of optimizations in the compiler and thus, require redesigning the processor for each side channel
   Since optimizations may break security guarantees
Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Original Program

```plaintext
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
}
```
Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Original Program

```plaintext
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
}
```

Ensure equal load instructions of each path

Transformed Code

```plaintext
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
    d <- load dummy
}
```
Drawbacks of Point Solutions

Original Program

```plaintext
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
}
```

Transformed Code

```plaintext
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
    d <- load dummy
}
```

Ensure equal load instructions of each path

Dead Code Elimination

GhostRider [ASPLOS-15]
### Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Optimizing compilers may break the security guarantee

<table>
<thead>
<tr>
<th>Original Program</th>
<th>Transformed Code</th>
</tr>
</thead>
</table>
| if (secret == 0) {
  x <- load ptr_1
  y <- load ptr_2
} else {
  z <- load ptr_3
} | if (secret == 0) {
  x <- load ptr_1
  y <- load ptr_2
} else {
  z <- load ptr_3
  d <- load dummy
} |

Ensure equal load instructions of each path

Dead Code Elimination
**Drawbacks of Point Solutions**

**GhostRider** [ASPLOS-15]

Original Program:

```java
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
}
```

Transformed Code:

```java
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
    d <- load dummy
}
```

Ensure equal load instructions of each path.
**Drawbacks of Point Solutions**

GhostRider [ASPLOS-15]

---

**Original Program**

```c
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
}
```

**Transformed Code**

```c
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
d <- load dummy
}
```

Ensure equal load instructions of each path

- Cache hit
- Cache hit
- Cache hit
- Miss!
Drawbacks of Point Solutions
GhostRider [ASPLOS-15]

Caches and prefetchers may break the security guarantee

Original Program
```python
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
}
```

Transformed Code
```python
if (secret == 0) {
    x <- load ptr_1
    y <- load ptr_2
} else {
    z <- load ptr_3
    d <- load dummy
}
```

Ensure equal load instructions of each path

Cache hit Cache hit Cache hit Miss!
Performance Impact of Using Point Solution

Disabled optimizations result in significant performance overhead

![Bar chart showing slow down for different operations: Matrix-Mult, Heap-Add, Heap-Pop, Binary-Search, Histogram, Map, Dijkstra-SSSP, and GEO-MEAN. The values are as follows: Matrix-Mult 20, Heap-Add 26, Heap-Pop 81, Binary-Search 112, Histogram 320, Map 495, Dijkstra-SSSP 1987, GEO-MEAN 147.]}
Prior Side Channel Defenses
Prior Side Channel Defenses

Are point solutions, since they focus on symptoms and not the root cause, and they may not compose well.
Prior Side Channel Defenses

Are point solutions, since they focus on symptoms and not the root cause, and they may not compose well.

Many require redesigned hardware, since the solution is forced to disable optimizations in compiler and microarch.
Prior Side Channel Defenses

Are point solutions, since they focus on symptoms and not the root cause, and they may not compose well.

Many require redesigned hardware, since the solution is forced to disable optimizations in compiler and microarch.

Many are inflexible because they cannot be tailored to the program or to portions of the program.
My Solutions

Closes a Broad Class of Side Channels
My Solutions

Closes a Broad Class of Side Channels

Executes on Modern Microprocessors
My Solutions

- Closes a Broad Class of Side Channels
- Executes on Modern Microprocessors
- Protects a Diverse Set of Applications
My Research Contributions

Input program → Compiler → Equivalent program that does not leak info through side channels
My Research **Contributions**

Input program → Compiler → Equivalent program that does not leak info through side channels

Lightweight annotations
My Research Contributions

Input program → Compiler → Equivalent program that does not leak info through side channels

Lightweight annotations

ISA
Microarchitecture
Physical Hardware
My Research Contributions

Input program → Compiler → Equivalent program that does not leak info through side channels

- Verified Side-Channel Leakage Analyzer
- Lightweight annotations
- ISA
- Microarchitecture
- Physical Hardware
My Research Contributions

<table>
<thead>
<tr>
<th>Raccoon</th>
<th>Escort</th>
<th>Vale</th>
<th>Vantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raccoon</td>
<td>Escort</td>
<td>Vale</td>
<td>Vantage</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>-----------------------</td>
<td>------------------------</td>
</tr>
</tbody>
</table>

Compilers for closing all digital side channels
**My Research Contributions**

<table>
<thead>
<tr>
<th>Raccoon</th>
<th>Escort</th>
<th>Vale</th>
<th>Vantage</th>
</tr>
</thead>
</table>

- Compilers for closing all **digital side channels**
- e.g. cache, address trace, branch predictor, TLB, etc.
My Research Contributions

Raccoon
USENIX Security Symposium 2015

Escort
USENIX Security Symposium 2016

Vale
USENIX Security Symposium 2017

Vantage
Work In Progress

Verified side channel leakage analyzer
<table>
<thead>
<tr>
<th>Raccoon</th>
<th>Escort</th>
<th>Vale</th>
<th>Vantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symposium 2015</td>
<td>Symposium 2016</td>
<td>Symposium 2017</td>
<td></td>
</tr>
</tbody>
</table>

Compiler that mitigates power side channel attacks in diverse programs and microarchitectures
Outline

Our Solution's Design
Outline

- Our Solution's Design
- Core Principles that Enable Generalization
Outline

- Our Solution's Design
- Core Principles that Enable Generalization
- Performance Comparison
Outline

- Our Solution's Design
- Core Principles that Enable Generalization
- Performance Comparison
- Future Work
Key Insight Behind Our Solutions
Key Insight Behind Our Solutions

A broad range of side channels arise due to variations in source-level behavior.
Key Insight Behind Our Solutions

A broad range of side channels arise due to variations in source-level behavior.

- Branch predictor side channel is caused by program path
- Memory trace channel is caused by pointer dereferences and program path
- Instruction count is caused by program path
Key Insight Behind Our Solutions

Source-Level Behavior causes Different Side Channels
Key Insight Behind Our Solutions

- Control Flow and Data Flow
  - causes
  - Source-Level Behavior
  - causes
  - Different Side Channels
Key Insight Behind Our Solutions

Sensitive Values → cause → Control Flow and Data Flow → causes → Source-Level Behavior → causes → Different Side Channels
Key Insight Behind Our Solutions

To close a broad class of side channels, make control flow and data flow independent of sensitive information.

Sensitive Values

Control Flow and Data Flow

causes

Source-Level Behavior

causes

Different Side Channels
Key Insight Behind Our Solutions

To close a broad class of side channels, make control flow and data flow independent of sensitive information.
Solution: **Execute All Paths**

```plaintext
if (secret_bit == 1) {
    z = (msg * z * z) mod n;
} else {
    z = (z * z) mod n;
}
```
Solution: Execute All Paths

```java
if (secret_bit == 1) {
    z = (msg * z * z) mod n;
} else {
    z = (z * z) mod n;
}
```

Adversary sees `secret_bit = 1`
Solution: **Execute All Paths**

```java
if (secret_bit == 1) {
    z = (msg * z * z) mod n;
} else {
    z = (z * z) mod n;
}

Adversary sees secret_bit = 1 and secret_bit != 1.
```
Predication to Execute All Paths

Original Program

```java
if (secret_bit == 1) {
    z = (msg * z * z) mod n;
} else {
    z = (z * z) mod n;
}
```
Predication to Execute All Paths

Original Program

```java
if (secret_bit == 1) {
    z = (msg * z * z) mod n;
} else {
    z = (z * z) mod n;
}
```

Transformed Program

```java
p = (secret_bit == 1)
p : z = (msg * z * z) mod n;
-p: z = (z * z) mod n;
```
Key Building Block: Software Predication
Key Building Block: **Software Predication**

- **new** → Predicated Write Operation
- **old** → Predicated Write Operation
Key Building Block: **Software Predication**

- **cond**
- **new**
- **old**

- **Predicated Write Operation**
Key Building Block: **Software Predication**

- **cond**
- **new**
- **old**

**Predicated Write Operation**

**output**

\{ 
new if cond = TRUE  
old otherwise 
\}
Key Building Block: **Software Predication**

- **cond**
- **new**
- **old**

```
Predicated Write Operation
```

- **output**
  - `new` if **cond** = TRUE
  - **old** otherwise

Implementation in x64 assembly:

```
mov old -> output  // Set destination
test cond, cond   // Check if non-zero
cmovz new -> output // Conditional update
test 0, 0          // Overwrite flags
```

Also implemented using ARM v7, ARM v8, and RISC-V assembly instructions.
Key Building Block: **Software Predication**

1. Straight-line control flow

```
mov    old -> output    // Set destination
test   cond, cond      // Check if non-zero
cmovz  new -> output   // Conditional update
test   0, 0             // Overwrite flags
```
Key Building Block: **Software Predication**

1. Straight-line control flow
2. All data in registers; no pointer dereferences

```
mov old -> output       // Set destination
test cond, cond         // Check if non-zero
cmovz new -> output     // Conditional update
test 0, 0               // Overwrite flags
```
Key Building Block: **Software Predication**

1. Straight-line control flow
2. All data in registers; no pointer dereferences
3. Fixed execution time

```assembly
mov    old -> output       // Set destination
test   cond, cond          // Check if non-zero
cmovz  new -> output       // Conditional update
test   0, 0                 // Overwrite flags
```
Software Predication to Execute All Paths

Original Program

```java
if (secret_bit == 1) {
    z = (msg * z * z) mod n
} else {
    z = (z * z) mod n
}
```

Transformed Program

```java
p = (secret_bit == 1)
z1 = (msg * z * z) mod n
z2 = (z * z) mod n
z = pred_write(p, z1, z)
z = pred_write(!p, z2, z)
```
Software Predication to Execute All Paths

**Original Program**

```plaintext
if (secret_bit == 1) {
    z = (msg * z * z) mod n
} else {
    z = (z * z) mod n
}
```

**Transformed Program**

```plaintext
p = (secret_bit == 1)
z1 = (msg * z * z) mod n
z2 = (z * z) mod n
z = pred_write(p, z1, z)
z = pred_write(!p, z2, z)
```
Software Predication to Execute All Paths

Original Program:

```java
if (secret_bit == 1) {
    z = (msg * z * z) mod n
} else {
    z = (z * z) mod n
}
```

Transformed Program:

```java
p = (secret_bit == 1)
z1 = (msg * z * z) mod n
z2 = (z * z) mod n
z = pred_write(p, z1, z)
z = pred_write(!p, z2, z)
```
Software Predication to Execute All Paths

Original Program

```java
if (secret_bit == 1) {
    z = (msg * z * z) mod n
} else {
    z = (z * z) mod n
}
```

Transformed Program

```java
p = (secret_bit == 1)
z1 = (msg * z * z) mod n
z2 = (z * z) mod n
z = pred_write(p, z1, z)
z = pred_write(!p, z2, z)
```
Software Predication to Execute All Paths

Original Program

```c
if (secret_bit == 1) {
    z = (msg * z * z) mod n
} else {
    z = (z * z) mod n
}
```

Transformed Program

```c
p = (secret_bit == 1)
z1 = (msg * z * z) mod n
z2 = (z * z) mod n
z = pred_write(p, z1, z)
z = pred_write(!p, z2, z)
```
But Predication **May Crash the Program**

```
Original Program

v = 0;
if (secret) {
    v = 10;
    y = x / v;
}
```
But Predication **May Crash the Program**

Original Program

```c
v = 0;
if (secret) {
    v = 10;
    y = x / v;
}
```

Transformed Program

```c
v = 0;
v = pred_write(secret, 10, v);
y = pred_write(secret, x/v, y);
```
But Predication **May Crash the Program**

**Original Program**

```plaintext
v = 0;
if (secret) {
    v = 10;
    y = x / v;
}
```

**Transformed Program**

```plaintext
v = 0;
if (secret) {
    v = pred_write(secret, 10, v);
    y = pred_write(secret, x/v, y);
}
```
But Predication **May Crash the Program**

**Original Program**

```plaintext
v = 0;
if (secret) {
    v = 10;
    y = x / v;
}
```

**Transformed Program**

```plaintext
v = 0;
v = pred_write(secret, 10, v);
y = pred_write(secret, x/v, y);
```

If `secret` is false, `v` is not updated, hence `v` remains 0.
But Predication **May Crash the Program**

**Original** Program

```plaintext
v = 0;
if (secret) {
  v = 10;
  y = x / v;
}
```

**Transformed** Program

```plaintext
v = 0;
v = pred_write(secret, 10, v);
y = pred_write(secret, x/v, y);
```

If `secret` is false, `v` is not updated, hence `v` remains 0.

Division by zero exception causes program to terminate.
But Predication **May Crash the Program**

**Original** Program

```python
v = 0;
if (secret) {
    v = 10;
    y = x / v;
}
```

**Transformed** Program

```python
v = 0;
v = pred_write(secret, 10, v);
y = pred_write(secret, x/v, y);
```

```python
v = 0;
v = pred_write(secret, 10, v);
\textcolor{red}{t = pred_write(v == 0, 1, v)};
y = pred_write(secret, x/t, y);
```
But Predication **May Crash the Program**

**Original Program**
```plaintext
v = 0;
if (secret) {
    v = 10;
    y = x / v;
}
```

**Transformed Program**
```plaintext
v = 0;
v = pred_write(secret, 10, v);
y = pred_write(secret, x/v, y);
```

Our solution masks exceptions by covertly changing divisor value.
But Predication **May Crash the Program**

**Original Program**

```plaintext
v = 0;
if (secret) {
  v = 10;
  y = x / v;
}
```

**Transformed Program**

```plaintext
v = 0;
v = pred_write(secret, 10, v);
y = pred_write(secret, x/v, y);
```

Our solution assumes that the pre-transformation program does not throw arch exceptions.

```plaintext
v = 0;
v = pred_write(secret, 10, v);
t = pred_write(v == 0, 1, v);
y = pred_write(secret, x/t, y);
```
But Predication **May Cause Infinite Loops**
But Predication May Cause Infinite Loops
But Predication **May Cause Infinite Loops**

Loops require a different transformation.
Transforming Loops

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

Assume \( n \) is secret. Transformation should hide the number of executed iterations.
Assume \( n \) is secret. Transformation should hide the number of executed iterations.
Transforming Loops

Original Program

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

Transforming Loops

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

**Transformed Program**
```
i = 0

loop ctr :: 0 to C
  i = i + 1;

ctr = ctr + 1;
```
Transforming Loops

Original Program:

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

Transformed Program:

```
i = 0

loop ctr :: 0 to C
  p: x = x * y;
  p: i = i + 1;

ctr = ctr + 1;
```
Transforming Loops

Original Program

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

Transformed Program

```
i = 0
p = TRUE
loop ctr :: 0 to C
  p: x = x * y;
  p: i = i + 1;

ctr = ctr + 1;
```
Transforming Loops

Original Program

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

Transformed Program

```
i = 0
p = TRUE
loop ctr :: 0 to C
  p: x = x * y;
  p: i = i + 1;

i == n: p = FALSE;
ctr = ctr + 1;
```

Turn predicate OFF to run dummy iterations.
Transforming Loops

Original Program:

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

Transformed Program:

```plaintext
i = 0
p = TRUE
loop ctr :: 0 to C
  p: x = x * y;
p: i = i + 1;
```

Annotated by user, for example:
`__loop_count(1024)`

`i == n: p = FALSE;
ctr = ctr + 1;`
Variations In Program Behavior

Control Flow

Data Flow
Variations in **Data Flow**

```python
result = table[secret];
```
Variations in Data Flow

\[
\text{result} = \text{table}[\text{secret}]; \quad \quad \quad \text{addr} \leftarrow \text{base(}\text{table}\text{)} + \text{secret} \\
\text{result} \leftarrow \text{read addr}
\]
Variations in **Data Flow**

```latex
result = \text{table}[\text{secret}];
```

```latex
\text{addr} \leftarrow \text{base(\text{table})} + \text{secret}
\text{result} \leftarrow \text{read \text{addr}}
```

An adversary that can observe address, can also derive \text{secret}.

```latex
\text{secret} = \text{addr} - \text{base(\text{table})}
```
Eliminating Variations in Data Flow

Solution #1: Array Streaming

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

Expensive to access entire array, but vector instructions, caches, and prefetchers reduce latency.
Eliminating Variations in Data Flow

Solution #1: Array Streaming
Accesses the entire array to read one element of the array.
Expensive to access entire array, but vector instructions, caches, and prefetchers reduce latency.

Solution #2: Software ORAM
Software version of Path ORAM [CCS'13], which shuffles memory to hide location of data.
Variations In Program Behavior

Control Flow  |  Data Flow
Variations in Program Behavior

- Control Flow
- Data Flow
- ISA Instructions
Example #1: Latency of Integer Division

Cleemput et al., "Compiler Mitigations for Time Attacks on Modern x86 Processors", ACM TACO 2012.
Example #1: Latency of Integer Division

Colors indicate different execution times

Cleemput et al., "Compiler Mitigations for Time Attacks on Modern x86 Processors", ACM TACO 2012.
Example #1: Latency of Integer Division

Colors indicate different execution times.

Execution time depends on the operand values, creating timing side channel.

Cleemput et al., "Compiler Mitigations for Time Attacks on Modern x86 Processors", ACM TACO 2012.
**Example #1: Latency of Integer Division**

Colors indicate different execution times

Execution time depends on the operand values, creating timing side channel

**Solution:** Rewrite division using bitwise arithmetic

---

Cleemput et al., "Compiler Mitigations for Time Attacks on Modern x86 Processors", ACM TACO 2012.
Example #2: Power Consumption of Comparisons

C Program

result = (secret != 3)

Assembly Program

cmp src, #3
movw dst <- #0
movne dst <- #1
Example #2: Power Consumption of Comparisons

C Program

result = (secret != 3)

Assembly Program

cmp src, #3
movw dst <- #0
movne dst <- #1

Compares secret with the literal constant 3
Example #2: Power Consumption of Comparisons

C Program

result = (secret != 3)

Assembly Program

cmp src, #3
movw dst <- #0
movne dst <- #1

Preemptively copies literal 0 into the destination
Example #2: Power Consumption of Comparisons

C Program

result = (secret != 3)

Assembly Program

cmp src, #3
movw dst <- #0
movne dst <- #1

**Conditionally** copies literal 1 into the destination
Example #2: Power Consumption of Comparisons

C Program

```c
result = (secret != 3)
```

Assembly Program

```assembly
cmp src, #3
movw dst <- #0
movne dst <- #1
```
Example #2: Power Consumption of Comparisons

C Program

result = (secret != 3)

Assembly Program

cmp src, #3
movw dst <- #0
movne dst <- #1

Register Write Ops affect Power Consumption
Example #2: Power Consumption of Comparisons

C Program

\[
\text{result} = (\text{secret} \neq 3)
\]

Assembly Program

\[
\begin{align*}
\text{cmp} & \quad \text{src}, \quad \#3 \\
\text{movw} & \quad \text{dst} <- \quad \#0 \\
\text{movne} & \quad \text{dst} <- \quad \#1
\end{align*}
\]

Register Write Ops \hspace{2cm} \text{affect} \hspace{2cm} \text{Power Consumption}

Power Model Analysis
Analysis of **Power Models**

- **Open Source Power Model**
  - for example, McPAT
Analysis of **Power Models**

**Open Source Power Model**
for example, McPAT

We apply **backward slicing** to identify the microarchitectural metrics that affect dynamic power consumption.
Analysis of **Power Models**

- **Open Source Power Model**
  - for example, McPAT

- **Closed Source Power Model**
  - for example, Intel RAPL

We apply **backward slicing** to identify the microarchitectural metrics that affect dynamic power consumption.
Analysis of Power Models

**Open Source Power Model**
- for example, McPAT

**Closed Source Power Model**
- for example, Intel RAPL

We apply **backward slicing** to identify the microarchitectural metrics that affect dynamic power consumption.

We fit a **regression model** to the RAPL power model, and we identify the inputs that have non-zero regression coefficients.
Superoptimizers for Alternate Code Sequences

Original Assembly Program

```
cmp src1,src2
movw dst <- #0
movne dst <- #1
```

Transformed Assembly Program

```
sub tmp <- src1,src2
sub dst <- src2,src1
orr dst <- dst , tmp
lsr dst <- dst , #31
```
Outline

- Our Solution’s Design
- Core Principles that Enable Generalization
- Performance Comparison
- Future Work
Mapping between Instructions and Leakage

LLVM Instructions → Side-Channel Information Leakage
Mapping between Instructions and Leakage

LLVM Instructions

\[ \text{div numerator denominator} \]

Side-Channel Information Leakage

\[ \text{numerator denominator} \]
Mapping between Instructions and Leakage

LLVM Instructions → Side-Channel Information Leakage

div numerator denominator → numerator denominator
Mapping between Instructions and Leakage

LLVM Instructions  $\alpha$  Side-Channel Information Leakage

ISA  Microarch.
Mapping between Instructions and Leakage

- LLVM Instructions
- Side-Channel Information Leakage
  - ISA
  - Microarch.
  - Automated Analysis
  - Manual Analysis
$f : I \rightarrow O$

computation     secret input data     output results
\[ f : I \rightarrow O \]

computation

secret input data

output results

\[ \alpha \]
\[
f : \text{computation} \
\alpha \
l : \text{information leakage} \\
\text{secret input data} \quad \text{secret input data} \quad \text{side channel observations}
\]
\[ f : I \rightarrow O \]
computation  secret input data  output results

\[ \alpha \]

\[ l : I \rightarrow S \]
information leakage  secret input data  side channel observations
Original Program

\[ f(x) \]
Initial Approach to Close Side Channel

Original Program

\[ f(x) \]
Initial Approach to Close Side Channel

Original Program

\[ f(x) \]

Transformed Program

\[ \forall i \in I \ f(i) \]
Initial Approach to Close Side Channel

Original Program: $f(x)$

Transformed Program: $\forall \ i \in I \ f(i)$

Inefficient if set of all possible inputs is large.
Approach #2: Slight Optimization
Approach #2: Slight Optimization

I
inputs

S
side channel observations
Approach #2: Slight Optimization

I
inputs

S
side channel observations
Approach #2: Slight Optimization

Use one representative input from each partition.
Approach #2: Slight Optimization

Original Program: $f(x)$

Transformed Program:

\[
\forall i \in I \quad f(i)
\]
\[
\forall i \in I/\sim \quad f(i)
\]
Approach #3: Further Optimization

\[ f : I \to O, \quad l : I \to S \]
Approach #3: Further Optimization

\[ f : I \rightarrow O, \quad l : I \rightarrow S \]
\[ f' : I \rightarrow O, \quad l' : I \rightarrow S' \quad \text{with} \quad |S'| = 1 \]
Approach #3: Further Optimization

\[ f' : I \rightarrow O, \quad l' : I \rightarrow S' \quad \text{with} \quad |S'| = 1 \]

Original Program

\[ f(x) \]

Transformed Program

\[ \forall i \in I \quad f(i) \]

\[ \forall i \in I \sim f(i) \]

\[ f'(x) \]
Implementation

- Close a Broad Class of Side Channels
- Execute on Modern Microprocessors
- Protect a Diverse Set of Applications

Digital Side Channels
- e.g. address trace, cache, branch predictor, etc.
Implementation

- Close a Broad Class of Side Channels
- Execute on Modern Microprocessors
- Protect a Diverse Set of Applications

- Digital Side Channels
- Non-Digital Side Channels
  - e.g., instruction-level power consumption
Implementation

- Close a Broad Class of Side Channels
- Execute on Modern Microprocessors
- Protect a Diverse Set of Applications

Microarchitectural Implementations of x64, 32-bit ARM, and 64-bit ARM ISAs
Implementation

Close a Broad Class of Side Channels

Execute on Modern Microprocessors

Protect a Diverse Set of Applications

Not just Cryptographic Implementations, but also Graph Kernels, Machine-Learning Libraries, etc.
Outline

- Our Solution’s Design
- Core Principles that Enable Generalization
- Performance Comparison
- Future Work
Comparison With **GhostRider**

- **Binary Search**: 112 (GhostRider), 65 (Our Solution)
- **Dijkstra SSSP**: 1987 (GhostRider), 6 (Our Solution)
- **Find Max**: 1294 (GhostRider), 2 (Our Solution)
- **Heap Add**: 2 (GhostRider), 26 (Our Solution)
- **Heap Pop**: 81 (GhostRider), 311 (Our Solution)
- **Histogram**: 495 (GhostRider), 50 (Our Solution)
- **Map**: 20 (GhostRider), 19 (Our Solution)
- **Matrix Mult**: 193 (GhostRider), 19 (Our Solution)
- **Geo. Mean**: 19 (GhostRider), 19 (Our Solution)
Evaluation Programs

- Cryptography: Lattice Crypto, Curve-25519, Poly-1305
- GhostRider: Binary Search, Heap Add, Matrix Mult
- Graph Kernels: Top-K Search, Bellman Ford, PageRank
- Machine Learning: Motion Tracking, SVM Classifier
- Utility Programs: Font Renderer, Hash Table, Bloom Filter
Hardware-Only Solution Evaluation

<table>
<thead>
<tr>
<th>Function</th>
<th>Slowdown (X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice Crypto</td>
<td>169</td>
</tr>
<tr>
<td>Curve 25519</td>
<td>375</td>
</tr>
<tr>
<td>Poly 1305</td>
<td>99</td>
</tr>
<tr>
<td>Binary Search</td>
<td>62</td>
</tr>
<tr>
<td>Heap Add</td>
<td>143</td>
</tr>
<tr>
<td>Matrix Mult</td>
<td>94</td>
</tr>
<tr>
<td>Top-K Search</td>
<td>177</td>
</tr>
<tr>
<td>Bellman Ford</td>
<td>102</td>
</tr>
<tr>
<td>Pagerank</td>
<td>67</td>
</tr>
<tr>
<td>Tax Solver</td>
<td>277</td>
</tr>
<tr>
<td>Hash Table</td>
<td>208</td>
</tr>
<tr>
<td>Bloom Filter</td>
<td>103</td>
</tr>
<tr>
<td>Motion Tracking</td>
<td>373</td>
</tr>
<tr>
<td>SVM</td>
<td>393</td>
</tr>
<tr>
<td>K-Means</td>
<td>144</td>
</tr>
<tr>
<td>GEO MEAN</td>
<td>161</td>
</tr>
</tbody>
</table>

Legend:
- Crypto
- GhostRider Kernels
- Graph Kernels
- Machine Learning
- Utility
- K-Means
Our Solution: All Digital Side Channels
Our Solution: **Timing** Side Channel

- Crypto
- GhostRider Kernels
- Graph Kernels
- Machine Learning
- Utility
- Geometric Mean

![Graph showing slowdown across different categories](image-url)
Our Solution: **Microarch. Power Side Channel**

![Graph showing slowdown against various categories like Crypto, GhostRider Kernels, Graph Kernels, Machine Learning, Utility, and Geometric Mean.]
Outline

- Our Solution’s Design
- Core Principles that Enable Generalization
- Performance Comparison
- Future Work
Our Solution: **Microarch. Power** Side Channel

![Graph](image.png)

- **Crypto**
- **GhostRider Kernels**
- **Graph Kernels**
- **Machine Learning**
- **Utility**
- **Geometric Mean**
Our Solution: All Digital Side Channels

- Crypto
- GhostRider Kernels
- Graph Kernels
- Machine Learning
- Utility
- Geometric Mean

Slowdown (X)

1. Lattice Crypto
2. Curve 25519
3. Poly 1305
4. Binary Search
5. Heap Add
6. Matrix Mul
7. Top-K Search
8. Bellman Ford
9. Pagerank
10. Tax Solver
11. Hash Table
12. Bloom Filter
13. Motion Tracking
14. SVM
15. K-Means
16. GEO MEAN
Our Solution: **Timing** Side Channel
Our Solution: All Digital Side Channels

![Graph showing slowdown across different categories including Crypto, GhostRider Kernels, Graph Kernels, Machine Learning, Utility, and K-Means with a geometric mean. The graph indicates that for Binary Search, there is a slowdown of 65.]
Our Solution: **Timing** Side Channel

![Graph showing slowdown across different categories]

- Crypto
- GhostRider Kernels
- Graph Kernels
- Machine Learning
- Utility
- Geometric Mean
Outline

- Our Solution’s Design
- Core Principles that Enable Generalization
- Performance Comparison
- Future Work
Future Work in Side-Channel Defenses

- **Automation**: Synthesizing program transformations
- **Precision**: Integrating better models of information leakage
- **Performance**: Aggressive compiler optimizations and modest microarch. modifications
\[ f : I \rightarrow O \]

desired computation

secret input data

output results
\( f' : I \rightarrow O \)

transformed computation  
secret input data  
output results

such that \( f' \) does not leak information through side channels.
Synthesizing Side-Channel Defenses

\[ f' : I \rightarrow O \]

- transformed computation
- secret input data
- output results

such that \( f' \) does not leak information through side channels
Synthesizing Side-Channel Defenses

\[ f' : I \rightarrow O \]

transformed computation
secret input data
output results

such that \( f' \) does not leak information through side channels

Goal: Adapt research in superoptimizers and program synthesis
Synthesis for **Stronger Guarantees**

Microarchitectural Specification

Domain-Specific Language for Compiler Transformations

Executable Code for Program Transformations
Synthesis for **Stronger Guarantees**

1. Microarchitectural Specification
2. Domain-Specific Language for Compiler Transformations
   - Type Analysis
   - Solver-Based Verification
3. Executable Code for Program Transformations
Precision of Side-Channel Defenses
Precision of Side-Channel Defenses

We need to tell compilers about potential side channels.
Precision of Side-Channel Defenses

We need to tell compilers about potential side channels.

Our current approach is an ad-hoc mix of program analysis, statistics, and manual inspection.
Precision of Side-Channel Defenses

We need to tell compilers about potential side channels.

Our current approach is an ad-hoc mix of program analysis, statistics, and manual inspection.

Goal: Precise abstractions of underlying layers.
Performance of Side-Channel Defenses
Compilers currently are at the mercy of the ISA.
Performance of Side-Channel Defenses

Compilers currently are at the mercy of the ISA.

We need more control of the microarchitecture and hardware.
Performance of Side-Channel Defenses

Compilers currently are at the mercy of the ISA.

We need more control of the microarchitecture and hardware.

Goal: Broaden the definition of the ISA beyond just a functional interface.
Layers of Abstraction as a Liability

Layer #1

Layer #2

Layer #3
Layers of Abstraction as a Liability

Debugging for security sometimes requires knowing a little about the implementation.

Layer #1

Layer #2

Layer #3
Layers of Abstraction as a Liability

Debugging for security sometimes requires knowing a little about the implementation.

But abstractions explicitly disable peeking into the implementation!
Layers of Abstraction as a Liability

Debugging for security sometimes requires knowing a little about the implementation.

But abstractions explicitly disable peeking into the implementation!

This problem affects performance debugging as well.
Peeking Inside the Hardware Implementation

- Instruction Set Architecture
- Microarchitecture
- Physical Hardware
Peeking Inside the Hardware Implementation

Can assist in:

- Discovering Denial-of-Service attacks
- Locating Confused-Deputy problems
- Tuning performance and energy consumption
Program Analysis for Hardware Design

- Instruction Set Architecture
- Microarchitecture
- Physical Hardware
Peeking Inside the Hardware Implementation

Can assist in:

- Discovering Denial-of-Service attacks
- Locating Confused-Deputy problems
- Tuning performance and energy consumption
Program Analysis for Hardware Design

- Instruction Set Architecture
- Microarchitecture
- Physical Hardware
Program Analysis for Hardware Design

Functional programming for hardware design [e.g. Cλash, Floh, SHard, and FLaSH]
Program Analysis for Hardware Design

Research Questions:

- How can we quantify security?
- How can we transform programs for energy efficiency?

Functional programming for hardware design [e.g. Cλash, Floh, SHard, and FLaSH]
Thanks to My Collaborators and Sponsors

Calvin Lin, PhD Advisor, UT Austin
Mohit Tiwari, PhD Advisor, UT Austin
K Rustan M. Leino, Amazon
Chris Hawblitzel, Microsoft Research
Laure Thompson, Cornell University
Joshua Eversmann, Civitas Learning
Greg McDonald, HBK Investments

Bryan Parno, Carnegie Mellon University
Jacob R. Lorch, Microsoft Research
Srinath Setty, Microsoft Research
Manos Kapritsos, University of Michigan
Barry Bond, Microsoft Research
Raymond Chee, Carnegie Mellon University
Varun Adiga and Kasra Sadeghi, UT Austin